

LibCapy documentation

Pascal Baillehache

July 29, 2025

Contents

1	Unit argParser.h	25
1.1	Macros	25
1.2	Enumerations	25
1.3	Typedefs	26
1.4	Struct CapyArg	26
1.4.1	Struct CapyArg's properties	26
1.4.2	Struct CapyArg's methods	26
1.5	Struct CapyArgParser	26
1.5.1	Struct CapyArgParser's properties	26
1.5.2	Struct CapyArgParser's methods	27
1.6	Functions	29
2	Unit array.h	31
2.1	Macros	31
2.2	Enumerations	49
2.3	Typedefs	49
2.4	Functions	49
3	Unit bezier.h	50
3.1	Macros	50
3.2	Enumerations	50
3.3	Typedefs	50
3.4	Struct CapyBezierPosition	51
3.4.1	Struct CapyBezierPosition's properties	51
3.4.2	Struct CapyBezierPosition's methods	51
3.5	Struct CapyBezierIterator	51
3.5.1	Struct CapyBezierIterator's properties	51
3.5.2	Struct CapyBezierIterator's methods	52

3.6	Functions	52
4	Unit <code>bitarray.h</code>	56
4.1	Macros	56
4.2	Enumerations	56
4.3	Typedefs	56
4.4	Struct <code>CapyBitArray</code>	56
4.4.1	Struct <code>CapyBitArray</code> 's properties	56
4.4.2	Struct <code>CapyBitArray</code> 's methods	56
4.5	Functions	58
5	Unit <code>bnoise.h</code>	58
5.1	Macros	58
5.2	Enumerations	59
5.3	Typedefs	59
5.4	Struct <code>CapyBNoise</code>	59
5.4.1	Struct <code>CapyBNoise</code> 's properties	59
5.4.2	Struct <code>CapyBNoise</code> 's methods	59
5.5	Functions	60
6	Unit <code>bresenham.h</code>	61
6.1	Macros	61
6.2	Enumerations	61
6.3	Typedefs	61
6.4	Struct <code>CapyBresenham</code>	61
6.4.1	Struct <code>CapyBresenham</code> 's properties	61
6.4.2	Struct <code>CapyBresenham</code> 's methods	62
6.5	Functions	62
7	Unit <code>btree.h</code>	63
7.1	Macros	63
7.2	Enumerations	76
7.3	Typedefs	76
7.4	Functions	76
8	Unit <code>burrowswheelertransform.h</code>	76
8.1	Macros	76
8.2	Enumerations	76
8.3	Typedefs	76
8.4	Struct <code>CapyBurrowsWheelerTransformData</code>	76
8.4.1	Struct <code>CapyBurrowsWheelerTransformData</code> 's properties	76

8.4.2	Struct CpyBurrowsWheelerTransformData's methods	77
8.5	Struct CpyBurrowsWheelerTransform	77
8.5.1	Struct CpyBurrowsWheelerTransform's properties	77
8.5.2	Struct CpyBurrowsWheelerTransform's methods	77
8.6	Functions	78
9	Unit camera.h	78
9.1	Macros	78
9.2	Enumerations	79
9.3	Typedefs	79
9.4	Struct CpyCamera	79
9.4.1	Struct CpyCamera's properties	79
9.4.2	Struct CpyCamera's methods	80
9.5	Functions	83
10	Unit capy.h	83
10.1	Macros	83
10.2	Enumerations	84
10.3	Typedefs	84
10.4	Functions	84
11	Unit capymath.h	84
11.1	Macros	85
11.2	Enumerations	85
11.3	Typedefs	86
11.4	Struct CpyPeasantMulDivRes	86
11.4.1	Struct CpyPeasantMulDivRes's properties	86
11.4.2	Struct CpyPeasantMulDivRes's methods	86
11.5	Struct CpyVec	86
11.5.1	Struct CpyVec's properties	86
11.5.2	Struct CpyVec's methods	86
11.6	Struct CpyMat	86
11.6.1	Struct CpyMat's properties	86
11.6.2	Struct CpyMat's methods	87
11.7	Struct CpyFiboLattice	87
11.7.1	Struct CpyFiboLattice's properties	87
11.7.2	Struct CpyFiboLattice's methods	87
11.8	Struct CpyPiecewiseGaussian	87
11.8.1	Struct CpyPiecewiseGaussian's properties	87
11.8.2	Struct CpyPiecewiseGaussian's methods	87
11.9	Functions	88

12 Unit cext.h	118
12.1 Macros	119
12.2 Enumerations	125
12.3 Typedefs	125
12.4 Functions	126
13 Unit chrono.h	128
13.1 Macros	128
13.2 Enumerations	128
13.3 Typedefs	128
13.4 Struct CapyChrono	129
13.4.1 Struct CapyChrono's properties	129
13.4.2 Struct CapyChrono's methods	129
13.5 Functions	130
14 Unit collisionDetection.h	131
14.1 Macros	131
14.2 Enumerations	131
14.3 Typedefs	131
14.4 Struct CapyCollisionDetection	131
14.4.1 Struct CapyCollisionDetection's properties	131
14.4.2 Struct CapyCollisionDetection's methods	131
14.5 Functions	133
15 Unit colorChart.h	134
15.1 Macros	134
15.2 Enumerations	134
15.3 Typedefs	134
15.4 Struct CapyColorChartDetectOpt	134
15.4.1 Struct CapyColorChartDetectOpt's properties	134
15.4.2 Struct CapyColorChartDetectOpt's methods	136
15.5 Struct CapyColorChart	136
15.5.1 Struct CapyColorChart's properties	136
15.5.2 Struct CapyColorChart's methods	137
15.6 Functions	140
16 Unit colorCorrectionBezier.h	142
16.1 Macros	142
16.2 Enumerations	142
16.3 Typedefs	142
16.4 Struct CapyColorCorrBezier	142

16.4.1	Struct CpyColorCorrBezier's properties	142
16.4.2	Struct CpyColorCorrBezier's methods	143
16.5	Functions	143
17	Unit colorCorrection.h	143
17.1	Macros	144
17.2	Enumerations	146
17.3	Typedefs	146
17.4	Functions	146
18	Unit colorCorrectionMatrix.h	146
18.1	Macros	146
18.2	Enumerations	146
18.3	Typedefs	146
18.4	Struct CpyColorCorrMat	147
18.4.1	Struct CpyColorCorrMat's properties	147
18.4.2	Struct CpyColorCorrMat's methods	147
18.5	Functions	147
19	Unit color.h	148
19.1	Macros	148
19.2	Enumerations	148
19.3	Typedefs	148
19.4	Struct CpyColor	149
19.4.1	Struct CpyColor's properties	149
19.4.2	Struct CpyColor's methods	149
19.5	Struct CpyColorPalette	152
19.5.1	Struct CpyColorPalette's properties	152
19.5.2	Struct CpyColorPalette's methods	152
19.6	Functions	154
20	Unit colorHisto.h	155
20.1	Macros	156
20.2	Enumerations	156
20.3	Typedefs	156
20.4	Struct CpyColorHisto	156
20.4.1	Struct CpyColorHisto's properties	156
20.4.2	Struct CpyColorHisto's methods	156
20.5	Functions	157

21 Unit colorPatch.h	158
21.1 Macros	158
21.2 Enumerations	158
21.3 Typedefs	158
21.4 Struct CappyColorPatch	159
21.4.1 Struct CappyColorPatch's properties	159
21.4.2 Struct CappyColorPatch's methods	159
21.5 Struct CappyColorPatches	160
21.5.1 Struct CappyColorPatches's properties	160
21.5.2 Struct CappyColorPatches's methods	160
21.6 Functions	161
 22 Unit comparator.h	 162
22.1 Macros	162
22.2 Enumerations	162
22.3 Typedefs	163
22.4 Functions	163
 23 Unit compressor.h	 163
23.1 Macros	163
23.2 Enumerations	164
23.3 Typedefs	164
23.4 Struct CappyCompressorData	164
23.4.1 Struct CappyCompressorData's properties	164
23.4.2 Struct CappyCompressorData's methods	165
23.5 Struct CappyRLECompressor	165
23.5.1 Struct CappyRLECompressor's properties	165
23.5.2 Struct CappyRLECompressor's methods	165
23.6 Struct CappyBWTRLECompressor	165
23.6.1 Struct CappyBWTRLECompressor's properties	165
23.6.2 Struct CappyBWTRLECompressor's methods	165
23.7 Struct CappyHuffmanCompressor	165
23.7.1 Struct CappyHuffmanCompressor's properties	165
23.7.2 Struct CappyHuffmanCompressor's methods	166
23.8 Functions	166
 24 Unit dataset.h	 168
24.1 Macros	168
24.2 Enumerations	169
24.3 Typedefs	169
24.4 Struct CappyDatasetRow	169

24.4.1	Struct CapyDatasetRow's properties	169
24.4.2	Struct CapyDatasetRow's methods	170
24.5	Struct CapyDatasetFieldDesc	170
24.5.1	Struct CapyDatasetFieldDesc's properties	170
24.5.2	Struct CapyDatasetFieldDesc's methods	171
24.6	Struct CapyDataset	171
24.6.1	Struct CapyDataset's properties	171
24.6.2	Struct CapyDataset's methods	172
24.7	Functions	180
25	Unit date.h	181
25.1	Macros	182
25.2	Enumerations	182
25.3	Typedefs	182
25.4	Functions	182
26	Unit dict.h	183
26.1	Macros	183
26.2	Enumerations	194
26.3	Typedefs	194
26.4	Functions	194
27	Unit diffevo.h	194
27.1	Macros	194
27.2	Enumerations	194
27.3	Typedefs	195
27.4	Struct CapyDiffEvo	195
27.4.1	Struct CapyDiffEvo's properties	195
27.4.2	Struct CapyDiffEvo's methods	197
27.5	Struct CapyDiffEvoThreadData	199
27.5.1	Struct CapyDiffEvoThreadData's properties	199
27.5.2	Struct CapyDiffEvoThreadData's methods	199
27.6	Functions	199
28	Unit display.h	200
28.1	Macros	200
28.2	Enumerations	200
28.3	Typedefs	201
28.4	Struct CapyDisplayKeyEvt	201
28.4.1	Struct CapyDisplayKeyEvt's properties	201
28.4.2	Struct CapyDisplayKeyEvt's methods	201

28.5	Struct CappyDisplayMouseEvent	201
28.5.1	Struct CappyDisplayMouseEvent's properties	201
28.5.2	Struct CappyDisplayMouseEvent's methods	202
28.6	Struct CappyDisplayCom	202
28.6.1	Struct CappyDisplayCom's properties	202
28.6.2	Struct CappyDisplayCom's methods	203
28.7	Struct CappyDisplay	203
28.7.1	Struct CappyDisplay's properties	203
28.7.2	Struct CappyDisplay's methods	204
28.8	Functions	207
29	Unit displayMagnifier.h	208
29.1	Macros	208
29.2	Enumerations	208
29.3	Typedefs	208
29.4	Struct CappyDisplayMagnifier	209
29.4.1	Struct CappyDisplayMagnifier's properties	209
29.4.2	Struct CappyDisplayMagnifier's methods	209
29.5	Functions	211
30	Unit distribution.h	211
30.1	Macros	211
30.2	Enumerations	213
30.3	Typedefs	214
30.4	Struct CappyDistNormal	214
30.4.1	Struct CappyDistNormal's properties	214
30.4.2	Struct CappyDistNormal's methods	214
30.5	Struct CappyDistDiscreteOccurence	215
30.5.1	Struct CappyDistDiscreteOccurence's properties	215
30.5.2	Struct CappyDistDiscreteOccurence's methods	215
30.6	Struct CappyDistDiscrete	215
30.6.1	Struct CappyDistDiscrete's properties	215
30.6.2	Struct CappyDistDiscrete's methods	215
30.7	Functions	216
31	Unit elo.h	219
31.1	Macros	219
31.2	Enumerations	219
31.3	Typedefs	219
31.4	Struct CappyElo	220
31.4.1	Struct CappyElo's properties	220

31.4.2	Struct CopyElo's methods	220
31.5	Functions	220
32	Unit externalHeaders.h	221
32.1	Macros	221
32.2	Enumerations	221
32.3	Typedefs	221
32.4	Functions	222
33	Unit feistelCipher.h	222
33.1	Macros	222
33.2	Enumerations	222
33.3	Typedefs	223
33.4	Struct CopyFeistelCipher	223
33.4.1	Struct CopyFeistelCipher's properties	223
33.4.2	Struct CopyFeistelCipher's methods	223
33.5	Functions	225
34	Unit fft.h	226
34.1	Macros	226
34.2	Enumerations	226
34.3	Typedefs	226
34.4	Struct CopyDFTCoeffs	226
34.4.1	Struct CopyDFTCoeffs's properties	226
34.4.2	Struct CopyDFTCoeffs's methods	227
34.5	Struct CopyDFT	227
34.5.1	Struct CopyDFT's properties	227
34.5.2	Struct CopyDFT's methods	228
34.6	Struct CopyDFT2DCoeffs	229
34.6.1	Struct CopyDFT2DCoeffs's properties	229
34.6.2	Struct CopyDFT2DCoeffs's methods	230
34.7	Struct CopyDFT2D	230
34.7.1	Struct CopyDFT2D's properties	230
34.7.2	Struct CopyDFT2D's methods	231
34.8	Functions	231
35	Unit fileFormat.h	234
35.1	Macros	234
35.2	Enumerations	235
35.3	Typedefs	235
35.4	Functions	235

36 Unit floodFill.h	235
36.1 Macros	236
36.2 Enumerations	236
36.3 Typedefs	236
36.4 Struct CapyFloodFill	236
36.4.1 Struct CapyFloodFill's properties	236
36.4.2 Struct CapyFloodFill's methods	236
36.5 Functions	237
37 Unit font.h	237
37.1 Macros	237
37.2 Enumerations	237
37.3 Typedefs	238
37.4 Struct CapyFont	238
37.4.1 Struct CapyFont's properties	238
37.4.2 Struct CapyFont's methods	238
37.5 Functions	239
38 Unit frequentistHypothesisTesting.h	239
38.1 Macros	239
38.2 Enumerations	239
38.3 Typedefs	240
38.4 Struct CapyFHT	240
38.4.1 Struct CapyFHT's properties	240
38.4.2 Struct CapyFHT's methods	240
38.5 Functions	241
39 Unit galeshapleypairing.h	242
39.1 Macros	242
39.2 Enumerations	242
39.3 Typedefs	242
39.4 Struct CapyGaleShapleyPairing	242
39.4.1 Struct CapyGaleShapleyPairing's properties	242
39.4.2 Struct CapyGaleShapleyPairing's methods	243
39.5 Functions	244
40 Unit geomap.h	244
40.1 Macros	244
40.2 Enumerations	244
40.3 Typedefs	245
40.4 Struct CapyGeoMapCoord	245

40.4.1	Struct CapiGeoMapCoord's properties	245
40.4.2	Struct CapiGeoMapCoord's methods	245
40.5	Struct CapiGeoMap	245
40.5.1	Struct CapiGeoMap's properties	245
40.5.2	Struct CapiGeoMap's methods	246
40.6	Functions	247
41	Unit geometricShape.h	248
41.1	Macros	248
41.2	Enumerations	249
41.3	Typedefs	249
41.4	Struct CapiPoint2D	249
41.4.1	Struct CapiPoint2D's properties	249
41.4.2	Struct CapiPoint2D's methods	250
41.5	Struct CapiSegment	250
41.5.1	Struct CapiSegment's properties	250
41.5.2	Struct CapiSegment's methods	250
41.6	Struct CapiTriangle	250
41.6.1	Struct CapiTriangle's properties	250
41.6.2	Struct CapiTriangle's methods	250
41.7	Struct CapiQuadrilateral	251
41.7.1	Struct CapiQuadrilateral's properties	251
41.7.2	Struct CapiQuadrilateral's methods	251
41.8	Struct CapiRectangle	252
41.8.1	Struct CapiRectangle's properties	252
41.8.2	Struct CapiRectangle's methods	252
41.9	Struct CapiCircle	252
41.9.1	Struct CapiCircle's properties	252
41.9.2	Struct CapiCircle's methods	253
41.10	Struct CapiGeometry2DIntersection	253
41.10.1	Struct CapiGeometry2DIntersection's properties	253
41.10.2	Struct CapiGeometry2DIntersection's methods	253
41.11	Functions	255
42	Unit gradientDescent.h	258
42.1	Macros	258
42.2	Enumerations	258
42.3	Typedefs	259
42.4	Struct CapiGradientDescent	259
42.4.1	Struct CapiGradientDescent's properties	259
42.4.2	Struct CapiGradientDescent's methods	260

42.5	Functions	260
43	Unit graph.h	261
43.1	Macros	261
43.2	Enumerations	261
43.3	Typedefs	262
43.4	Struct CopyGraphNode	262
43.4.1	Struct CopyGraphNode's properties	262
43.4.2	Struct CopyGraphNode's methods	262
43.5	Struct CopyGraphLink	262
43.5.1	Struct CopyGraphLink's properties	262
43.5.2	Struct CopyGraphLink's methods	262
43.6	Struct CopyGraph	263
43.6.1	Struct CopyGraph's properties	263
43.6.2	Struct CopyGraph's methods	263
43.7	Functions	266
44	Unit graphplotter.h	267
44.1	Macros	267
44.2	Enumerations	267
44.3	Typedefs	267
44.4	Struct CopyGraphPlotterLegendData	268
44.4.1	Struct CopyGraphPlotterLegendData's properties	268
44.4.2	Struct CopyGraphPlotterLegendData's methods	268
44.5	Struct CopyGraphPlotter	268
44.5.1	Struct CopyGraphPlotter's properties	268
44.5.2	Struct CopyGraphPlotter's methods	269
44.6	Functions	273
45	Unit greedy.h	273
45.1	Macros	274
45.2	Enumerations	274
45.3	Typedefs	274
45.4	Struct CopyGreedyObject	274
45.4.1	Struct CopyGreedyObject's properties	274
45.4.2	Struct CopyGreedyObject's methods	274
45.5	Struct CopyGreedyResult	274
45.5.1	Struct CopyGreedyResult's properties	274
45.5.2	Struct CopyGreedyResult's methods	275
45.6	Struct CopyGreedy	275
45.6.1	Struct CopyGreedy's properties	275

45.6.2	Struct CapyGreedy's methods	275
45.7	Functions	276
46	Unit hashFun.h	276
46.1	Macros	276
46.2	Enumerations	277
46.3	Typedefs	277
46.4	Struct CapyFNV1aHashFun	277
46.4.1	Struct CapyFNV1aHashFun's properties	277
46.4.2	Struct CapyFNV1aHashFun's methods	277
46.5	Functions	277
47	Unit idxCombination.h	279
47.1	Macros	279
47.2	Enumerations	279
47.3	Typedefs	279
47.4	Struct CapyIdxCombination	279
47.4.1	Struct CapyIdxCombination's properties	279
47.4.2	Struct CapyIdxCombination's methods	280
47.5	Functions	280
48	Unit image.h	281
48.1	Macros	281
48.2	Enumerations	283
48.3	Typedefs	283
48.4	Struct CapyImgPixel	284
48.4.1	Struct CapyImgPixel's properties	284
48.4.2	Struct CapyImgPixel's methods	284
48.5	Struct CapyImgIterator	284
48.5.1	Struct CapyImgIterator's properties	284
48.5.2	Struct CapyImgIterator's methods	285
48.6	Functions	286
49	Unit imgKernel.h	289
49.1	Macros	289
49.2	Enumerations	289
49.3	Typedefs	290
49.4	Struct CapyImgKernel	290
49.4.1	Struct CapyImgKernel's properties	290
49.4.2	Struct CapyImgKernel's methods	290
49.5	Functions	292

50 Unit kfoldCrossValid.h	293
50.1 Macros	293
50.2 Enumerations	293
50.3 Typedefs	293
50.4 Struct CapiKfoldCrossValidResPredictor	293
50.4.1 Struct CapiKfoldCrossValidResPredictor's properties	293
50.4.2 Struct CapiKfoldCrossValidResPredictor's methods	295
50.5 Struct CapiKfoldCrossValid	295
50.5.1 Struct CapiKfoldCrossValid's properties	295
50.5.2 Struct CapiKfoldCrossValid's methods	295
50.6 Functions	296
 51 Unit kmeans.h	 297
51.1 Macros	297
51.2 Enumerations	297
51.3 Typedefs	297
51.4 Struct CapiKMeansClusterOfPoint	298
51.4.1 Struct CapiKMeansClusterOfPoint's properties	298
51.4.2 Struct CapiKMeansClusterOfPoint's methods	298
51.5 Struct CapiKMeans	298
51.5.1 Struct CapiKMeans's properties	298
51.5.2 Struct CapiKMeans's methods	298
51.6 Functions	299
 52 Unit lightray.h	 300
52.1 Macros	300
52.2 Enumerations	300
52.3 Typedefs	301
52.4 Struct CapiRefractiveCoeff	301
52.4.1 Struct CapiRefractiveCoeff's properties	301
52.4.2 Struct CapiRefractiveCoeff's methods	301
52.5 Struct CapiRefractionValue	301
52.5.1 Struct CapiRefractionValue's properties	301
52.5.2 Struct CapiRefractionValue's methods	301
52.6 Functions	302
 53 Unit list.h	 304
53.1 Macros	304
53.2 Enumerations	319
53.3 Typedefs	319
53.4 Functions	320

54 Unit lsystem.h	320
54.1 Macros	320
54.2 Enumerations	322
54.3 Typedefs	322
54.4 Struct CapyLSysLindenMayerAlgaeState	322
54.4.1 Struct CapyLSysLindenMayerAlgaeState's properties	322
54.4.2 Struct CapyLSysLindenMayerAlgaeState's methods	322
54.5 Struct CapyLSysKochSnowflakeState	322
54.5.1 Struct CapyLSysKochSnowflakeState's properties	322
54.5.2 Struct CapyLSysKochSnowflakeState's methods	322
54.6 Struct CapyLSysFractalBinaryTreeState	323
54.6.1 Struct CapyLSysFractalBinaryTreeState's properties	323
54.6.2 Struct CapyLSysFractalBinaryTreeState's methods	323
54.7 Struct CapyLSysHilbertCurveState	323
54.7.1 Struct CapyLSysHilbertCurveState's properties	323
54.7.2 Struct CapyLSysHilbertCurveState's methods	323
54.8 Struct CapyLSysHeighwayDragonCurveState	323
54.8.1 Struct CapyLSysHeighwayDragonCurveState's properties	323
54.8.2 Struct CapyLSysHeighwayDragonCurveState's methods	324
54.9 Struct CapyLSysGosperCurveState	324
54.9.1 Struct CapyLSysGosperCurveState's properties	324
54.9.2 Struct CapyLSysGosperCurveState's methods	324
54.10 Functions	324
 55 Unit markovdecisionprocess.h	 324
55.1 Macros	324
55.2 Enumerations	326
55.3 Typedefs	326
55.4 Struct CapyMDPPolicyEpsilonSoft	326
55.4.1 Struct CapyMDPPolicyEpsilonSoft's properties	326
55.4.2 Struct CapyMDPPolicyEpsilonSoft's methods	327
55.5 Struct CapyMDPTransition	327
55.5.1 Struct CapyMDPTransition's properties	327
55.5.2 Struct CapyMDPTransition's methods	327
55.6 Struct CapyMDPTransitionRecorder	328
55.6.1 Struct CapyMDPTransitionRecorder's properties	328
55.6.2 Struct CapyMDPTransitionRecorder's methods	328
55.7 Struct CapyMarkovDecisionProcess	329
55.7.1 Struct CapyMarkovDecisionProcess's properties	329
55.7.2 Struct CapyMarkovDecisionProcess's methods	330

55.8	Functions	335
56	Unit mathfun.h	339
56.1	Macros	339
56.2	Enumerations	340
56.3	Typedefs	340
56.4	Struct CapyHyperplane	340
56.4.1	Struct CapyHyperplane's properties	340
56.4.2	Struct CapyHyperplane's methods	340
56.5	Functions	340
57	Unit maze.h	341
57.1	Macros	341
57.2	Enumerations	342
57.3	Typedefs	342
57.4	Struct CapyMaze2D	342
57.4.1	Struct CapyMaze2D's properties	342
57.4.2	Struct CapyMaze2D's methods	342
57.5	Functions	343
58	Unit memoryPool.h	344
58.1	Macros	344
58.2	Enumerations	348
58.3	Typedefs	349
58.4	Functions	349
59	Unit minimax.h	349
59.1	Macros	349
59.2	Enumerations	349
59.3	Typedefs	350
59.4	Struct CapyMiniMax	350
59.4.1	Struct CapyMiniMax's properties	350
59.4.2	Struct CapyMiniMax's methods	351
59.5	Functions	352
60	Unit mirrorballcamera.h	353
60.1	Macros	353
60.2	Enumerations	353
60.3	Typedefs	353
60.4	Struct CapyMirrorBallCamera	353
60.4.1	Struct CapyMirrorBallCamera's properties	353

60.4.2	Struct CapyMirrorBallCamera's methods	354
60.5	Functions	354
61	Unit neuralNetwork.h	355
61.1	Macros	355
61.2	Enumerations	356
61.3	Typedefs	356
61.4	Struct CapyNNLink	356
61.4.1	Struct CapyNNLink's properties	356
61.4.2	Struct CapyNNLink's methods	357
61.5	Struct CapyNNNode	357
61.5.1	Struct CapyNNNode's properties	357
61.5.2	Struct CapyNNNode's methods	358
61.6	Struct CapyNNLayer	358
61.6.1	Struct CapyNNLayer's properties	358
61.6.2	Struct CapyNNLayer's methods	358
61.7	Struct CapyNNModel	358
61.7.1	Struct CapyNNModel's properties	358
61.7.2	Struct CapyNNModel's methods	358
61.8	Struct CapyNeuralNetwork	359
61.8.1	Struct CapyNeuralNetwork's properties	359
61.8.2	Struct CapyNeuralNetwork's methods	359
61.9	Struct CapyNNActivationLinear	360
61.9.1	Struct CapyNNActivationLinear's properties	360
61.9.2	Struct CapyNNActivationLinear's methods	360
61.10	Struct CapyNNActivationStep	361
61.10.1	Struct CapyNNActivationStep's properties	361
61.10.2	Struct CapyNNActivationStep's methods	361
61.11	Struct CapyNNActivationSigmoid	361
61.11.1	Struct CapyNNActivationSigmoid's properties	361
61.11.2	Struct CapyNNActivationSigmoid's methods	361
61.12	Struct CapyNNActivationHyperTangent	361
61.12.1	Struct CapyNNActivationHyperTangent's properties	361
61.12.2	Struct CapyNNActivationHyperTangent's methods	361
61.13	Struct CapyNNActivationReLU	361
61.13.1	Struct CapyNNActivationReLU's properties	361
61.13.2	Struct CapyNNActivationReLU's methods	361
61.14	Struct CapyNNActivationSiLU	362
61.14.1	Struct CapyNNActivationSiLU's properties	362
61.14.2	Struct CapyNNActivationSiLU's methods	362
61.15	Functions	362

62 Unit nnPredictor.h	367
62.1 Macros	367
62.2 Enumerations	367
62.3 Typedefs	367
62.4 Struct CapyNNPredictor	367
62.4.1 Struct CapyNNPredictor's properties	367
62.4.2 Struct CapyNNPredictor's methods	369
62.5 Functions	370
63 Unit noise.h	371
63.1 Macros	372
63.2 Enumerations	372
63.3 Typedefs	372
63.4 Functions	372
64 Unit pathfinder.h	373
64.1 Macros	373
64.2 Enumerations	373
64.3 Typedefs	374
64.4 Struct CapyPath	374
64.4.1 Struct CapyPath's properties	374
64.4.2 Struct CapyPath's methods	374
64.5 Struct CapyPathFinder	374
64.5.1 Struct CapyPathFinder's properties	374
64.5.2 Struct CapyPathFinder's methods	374
64.6 Functions	375
65 Unit pen.h	376
65.1 Macros	376
65.2 Enumerations	376
65.3 Typedefs	376
65.4 Struct CapyPen	376
65.4.1 Struct CapyPen's properties	376
65.4.2 Struct CapyPen's methods	377
65.5 Functions	380
66 Unit plyFormat.h	381
66.1 Macros	381
66.2 Enumerations	381
66.3 Typedefs	381
66.4 Struct CapyPlyFormat	382

66.4.1	Struct CappyPlyFormat's properties	382
66.4.2	Struct CappyPlyFormat's methods	382
66.5	Functions	382
67	Unit pngFormat.h	383
67.1	Macros	383
67.2	Enumerations	383
67.3	Typedefs	383
67.4	Struct CappyPngFormat	383
67.4.1	Struct CappyPngFormat's properties	383
67.4.2	Struct CappyPngFormat's methods	383
67.5	Functions	384
68	Unit pointCloud.h	384
68.1	Macros	384
68.2	Enumerations	385
68.3	Typedefs	385
68.4	Struct CappyPointCloudLink	385
68.4.1	Struct CappyPointCloudLink's properties	385
68.4.2	Struct CappyPointCloudLink's methods	385
68.5	Struct CappyPointCloud	385
68.5.1	Struct CappyPointCloud's properties	385
68.5.2	Struct CappyPointCloud's methods	386
68.6	Struct CappyPointCloudNearestNeighbourRes	389
68.6.1	Struct CappyPointCloudNearestNeighbourRes's proper- ties	389
68.6.2	Struct CappyPointCloudNearestNeighbourRes's methods	389
68.7	Struct CappyPointCloudNearestNeighbour	389
68.7.1	Struct CappyPointCloudNearestNeighbour's properties .	389
68.7.2	Struct CappyPointCloudNearestNeighbour's methods . .	389
68.8	Functions	390
69	Unit poissonSampling.h	392
69.1	Macros	392
69.2	Enumerations	392
69.3	Typedefs	392
69.4	Struct CappyPoissonSampling	393
69.4.1	Struct CappyPoissonSampling's properties	393
69.4.2	Struct CappyPoissonSampling's methods	393
69.5	Functions	393

70 Unit policygradient.h	394
70.1 Macros	394
70.2 Enumerations	398
70.3 Typedefs	398
70.4 Struct CapiPGTransition	398
70.4.1 Struct CapiPGTransition's properties	398
70.4.2 Struct CapiPGTransition's methods	399
70.5 Struct CapiPGTransitionRecorder	399
70.5.1 Struct CapiPGTransitionRecorder's properties	399
70.5.2 Struct CapiPGTransitionRecorder's methods	399
70.6 Struct CapiPolicyGradient	400
70.6.1 Struct CapiPolicyGradient's properties	400
70.6.2 Struct CapiPolicyGradient's methods	401
70.7 Functions	401
71 Unit polynomial.h	404
71.1 Macros	404
71.2 Enumerations	404
71.3 Typedefs	404
71.4 Struct CapiPolynomial1D	404
71.4.1 Struct CapiPolynomial1D's properties	404
71.4.2 Struct CapiPolynomial1D's methods	405
71.5 Functions	405
72 Unit predictor.h	406
72.1 Macros	406
72.2 Enumerations	411
72.3 Typedefs	411
72.4 Struct CapiPredictorPrediction	412
72.4.1 Struct CapiPredictorPrediction's properties	412
72.4.2 Struct CapiPredictorPrediction's methods	412
72.5 Functions	412
73 Unit quaternion.h	414
73.1 Macros	414
73.2 Enumerations	414
73.3 Typedefs	414
73.4 Functions	414

74 Unit random.h	415
74.1 Macros	416
74.2 Enumerations	416
74.3 Typedefs	416
74.4 Struct CopyRandom	416
74.4.1 Struct CopyRandom's properties	416
74.4.2 Struct CopyRandom's methods	417
74.5 Functions	421
75 Unit range.h	422
75.1 Macros	422
75.2 Enumerations	425
75.3 Typedefs	425
75.4 Functions	425
76 Unit ratio.h	425
76.1 Macros	426
76.2 Enumerations	426
76.3 Typedefs	426
76.4 Struct CopyRatio	426
76.4.1 Struct CopyRatio's properties	426
76.4.2 Struct CopyRatio's methods	426
76.5 Functions	427
77 Unit rsacipher.h	433
77.1 Macros	433
77.2 Enumerations	433
77.3 Typedefs	433
77.4 Struct CopyRSACipherKey	433
77.4.1 Struct CopyRSACipherKey's properties	433
77.4.2 Struct CopyRSACipherKey's methods	433
77.5 Struct CopyRSACipherData	433
77.5.1 Struct CopyRSACipherData's properties	433
77.5.2 Struct CopyRSACipherData's methods	434
77.6 Struct CopyRSACipher	434
77.6.1 Struct CopyRSACipher's properties	434
77.6.2 Struct CopyRSACipher's methods	434
77.7 Functions	435

78 Unit rulerandcompass.h	436
78.1 Macros	436
78.2 Enumerations	436
78.3 Typedefs	436
78.4 Struct CapyRulerAndCompass	437
78.4.1 Struct CapyRulerAndCompass's properties	437
78.4.2 Struct CapyRulerAndCompass's methods	437
78.5 Functions	440
79 Unit rungekutta.h	441
79.1 Macros	441
79.2 Enumerations	441
79.3 Typedefs	441
79.4 Struct CapyRungeKutta	442
79.4.1 Struct CapyRungeKutta's properties	442
79.4.2 Struct CapyRungeKutta's methods	442
79.5 Functions	443
80 Unit sampling.h	444
80.1 Macros	444
80.2 Enumerations	444
80.3 Typedefs	444
80.4 Struct CapySample	445
80.4.1 Struct CapySample's properties	445
80.4.2 Struct CapySample's methods	445
80.5 Functions	445
81 Unit sdfcsg.h	446
81.1 Macros	446
81.2 Enumerations	447
81.3 Typedefs	447
81.4 Struct CapySdfTransform	447
81.4.1 Struct CapySdfTransform's properties	447
81.4.2 Struct CapySdfTransform's methods	448
81.5 Struct CapyRayMarchingRes	448
81.5.1 Struct CapyRayMarchingRes's properties	448
81.5.2 Struct CapyRayMarchingRes's methods	449
81.6 Functions	449

82 Unit slidingAverage.h	458
82.1 Macros	458
82.2 Enumerations	458
82.3 Typedefs	458
82.4 Struct CpySlidingAverage	458
82.4.1 Struct CpySlidingAverage's properties	458
82.4.2 Struct CpySlidingAverage's methods	459
82.5 Functions	459
 83 Unit sort.h	 460
83.1 Macros	460
83.2 Enumerations	460
83.3 Typedefs	460
83.4 Functions	460
 84 Unit strDecorator.h	 461
84.1 Macros	461
84.2 Enumerations	461
84.3 Typedefs	461
84.4 Struct CpyStrDecorator	461
84.4.1 Struct CpyStrDecorator's properties	461
84.4.2 Struct CpyStrDecorator's methods	462
84.5 Functions	463
 85 Unit streamIo.h	 464
85.1 Macros	464
85.2 Enumerations	464
85.3 Typedefs	464
85.4 Struct CpyStreamIo	465
85.4.1 Struct CpyStreamIo's properties	465
85.4.2 Struct CpyStreamIo's methods	465
85.5 Functions	468
 86 Unit supportVectorMachine.h	 468
86.1 Macros	468
86.2 Enumerations	469
86.3 Typedefs	469
86.4 Struct CpySVMKernelLinear	469
86.4.1 Struct CpySVMKernelLinear's properties	469
86.4.2 Struct CpySVMKernelLinear's methods	469
86.5 Struct CpySVMKernelPolynomial	469

86.5.1	Struct CapySVMKernelPolynomial's properties	469
86.5.2	Struct CapySVMKernelPolynomial's methods	470
86.6	Struct CapySVMKernelGaussian	470
86.6.1	Struct CapySVMKernelGaussian's properties	470
86.6.2	Struct CapySVMKernelGaussian's methods	470
86.7	Struct CapySVMEvaluation	470
86.7.1	Struct CapySVMEvaluation's properties	470
86.7.2	Struct CapySVMEvaluation's methods	471
86.8	Struct CapySupportVectorMachine	471
86.8.1	Struct CapySupportVectorMachine's properties	471
86.8.2	Struct CapySupportVectorMachine's methods	472
86.9	Functions	473
87	Unit tree.h	477
87.1	Macros	477
87.2	Enumerations	487
87.3	Typedefs	487
87.4	Functions	487
88	Unit trycatch.h	487
88.1	Macros	487
88.2	Enumerations	489
88.3	Typedefs	490
88.4	Functions	490
89	Unit turtlegraphic.h	493
89.1	Macros	493
89.2	Enumerations	493
89.3	Typedefs	493
89.4	Struct CapyTurtleGraphicState	493
89.4.1	Struct CapyTurtleGraphicState's properties	493
89.4.2	Struct CapyTurtleGraphicState's methods	493
89.5	Struct CapyTurtleGraphic	493
89.5.1	Struct CapyTurtleGraphic's properties	493
89.5.2	Struct CapyTurtleGraphic's methods	494
89.6	Functions	495
90	Unit voting.h	496
90.1	Macros	496
90.2	Enumerations	496
90.3	Typedefs	496

90.4	Struct CapiRankedChoiceVote	496
90.4.1	Struct CapiRankedChoiceVote's properties	496
90.4.2	Struct CapiRankedChoiceVote's methods	496
90.5	Struct CapiRankedChoiceVotingResult	496
90.5.1	Struct CapiRankedChoiceVotingResult's properties	496
90.5.2	Struct CapiRankedChoiceVotingResult's methods	497
90.6	Struct CapiRankedChoiceVoting	497
90.6.1	Struct CapiRankedChoiceVoting's properties	497
90.6.2	Struct CapiRankedChoiceVoting's methods	497
90.7	Functions	498
91	Unit x11display.h	499
91.1	Macros	499
91.2	Enumerations	499
91.3	Typedefs	500
91.4	Struct CapiX11DisplayEvt	500
91.4.1	Struct CapiX11DisplayEvt's properties	500
91.4.2	Struct CapiX11DisplayEvt's methods	501
91.5	Struct CapiX11RGB	501
91.5.1	Struct CapiX11RGB's properties	501
91.5.2	Struct CapiX11RGB's methods	501

1 Unit argParser.h

CLI arguments parser class.

1.1 Macros

```
#define CAPY_ARGPARSER_H
```

```
#define CopyArgParserCreate(argc, argv, args) \
    CopyArgParserCreate_(argc, argv, args, sizeof(args)/sizeof(CapyArg), false)
)
```

Helper macro to automatically deduce the number of argument in the definition list

```
#define CopyArgParserCreateSilent(argc, argv, args) \
    CopyArgParserCreate_(argc, argv, args, sizeof(args)/sizeof(CapyArg), true)
```

Helper macro to automatically deduce the number of argument in the definition list (silent version for unit tests)

```
#define CopyArgParserAlloc(argc, argv, args) \
    CopyArgParserAlloc_(argc, argv, args, sizeof(args)/sizeof(CapyArg), false)
```

Helper macro to automatically deduce the number of argument in the definition list

```
#define CopyArgParserAllocSilent(argc, argv, args) \
    CopyArgParserAlloc_(argc, argv, args, sizeof(args)/sizeof(CapyArg), true)
```

Helper macro to automatically deduce the number of argument in the definition list (silent version for unit tests)

1.2 Enumerations

```
typedef enum CapyArgNbVal {
    capyArg_variableNbVal = -1,
    capyArg_noVal,
} CapyArgNbVal;
```

Enumeration of the possible number of values for one argument in the definition list

1.3 Typedefs

None.

1.4 Struct CopyArg

1.4.1 Struct CopyArg's properties

```
char const* lbl;
```

The label of the argument (e.g. '-argument')

```
char const* shortLbl;
```

The short label of the argument (e.g. '-a')

```
CopyArgNbVal nbVal;
```

The number or values following this argument

```
CopyPad(CopyArgNbVal, 0);
```

```
char const* help;
```

The description of this argument

```
CopyListPtrChar* vals;
```

The list of values of this argument

1.4.2 Struct CopyArg's methods

None.

1.5 Struct CopyArgParser

1.5.1 Struct CopyArgParser's properties

```
CopyArg* args;
```

The arguments definition


```
size_t nbArgs;
```

The number of arguments in the definition

```
char* const* argv;
```

The argc and argv argument of the main() function

```
int argc;
```

```
CapyPad(int, 1);
```

```
bool silent;
```

Silent flag (default: false)

```
CapyPad(bool, 2);
```

```
CapyListPtrChar* vals;
```

The orphan values (values in argv not associated to an argument in the arguments definition)

1.5.2 Struct CapyArgParser's methods

```
void (*destruct)(void);
```

Destructor

```
void (*help)(void);
```

Print the help message describing the argument definition of the parser

```
size_t (*getNbVal)(char const* const arg);
```

Get the number of values for a given argument (number given via the command line, not the number in the definition)

Input argument(s):

- arg: the argument, must be in the argument definitions, may be
- the label or the shortcut

Output and side effect(s):

- Return the number of values

```
bool (*isSet)(char const* const arg);
```

Check if an optional argument is used

Input argument(s):

- arg: the argument to check, must be in the argument definitions, may be
- the label or the shortcut

Output and side effect(s):

- Return true if the argument is used, else false

```
int64_t (*getAsInt)(  
    char const* const arg,  
    size_t const iVal);
```

Get a value decoded as int for an optional argument

Input argument(s):

- arg: the argument to check, must be in the argument definitions, may be
- the label or the shortcut
- iVal: the index of the value

Output and side effect(s):

- Return the requested value, or 0 if the value was missing

```
double (*getAsDouble)(  
    char const* const arg,  
    size_t const iVal);
```

Get a value decoded as double for an optional argument

Input argument(s):

- arg: the argument to check, must be in the argument definitions, may be

- the label or the shortcut
- iVal: the index of the value

Output and side effect(s):

- Return the requested value, or 0.0 if the value was missing

```
char* (*getAsStr)(
    char const* const arg,
    size_t const iVal);
```

Get a value as a string for an optional argument

Input argument(s):

- arg: the argument to check, must be in the argument definitions, may be
- the label or the shortcut
- iVal: the index of the value

Output and side effect(s):

- Return the requested value, or NULL if the value was missing

1.6 Functions

```
CopyArgParser CopyArgParserCreate_(
    int const argc,
    char** const argv,
    CopyArg* const args,
    size_t const nbArgs,
    bool const silent);
```

Create a CopyArgParser, check the argument definitions and parse the arguments from the command line.

Input argument(s):

- argc: the argc argument of the main() function
- argv: the argv argument of the main() function
- args: the argument definition list
- nbArgs: the number of arguments in the definition list

- silent: silent flag

Output and side effect(s):

- Return a CpyArgParser
- Exceptions:
- May raise CpyExc_InvalidCLIArg, CpyExc_MallocFailed

```
CpyArgParser* CpyArgParserAlloc_(
    int const argc,
    char** const argv,
    CpyArg* const args,
    size_t const nbArgs,
    bool const silent);
```

Allocate memory for a CpyArgParser, create it, check the argument definitions and parse the arguments from the command line.

Input argument(s):

- argc: the argc argument of the main() function
- argv: the argv argument of the main() function
- args: the argument definition list
- nbArgs: the number of arguments in the definition list
- silent: silent flag

Output and side effect(s):

- Return a CpyArgParser
- Exceptions:
- May raise CpyExc_InvalidCLIArg, CpyExc_MallocFailed

```
void CpyArgParserFree(CpyArgParser** const that);
```

Free the memory used by a CpyArgParser and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyArgParser to free

2 Unit array.h

Generic array class.

2.1 Macros

```
#define CAPY_ARRAY_H
```

```
#define CapyDecArrayIterator(name, type_) \
typedef struct name name; \
typedef struct name ## Iterator { \
    size_t idx; \
    name* arr; \
    type_ datatype; \
    CapyPad(type_, datatype); \
    CapyArrayIteratorType type; \
    CapyPad(CapyArrayIteratorType, type); \
    void (*destruct)(void); \
    type_* (*reset)(void); \
    type_* (*prev)(void); \
    type_* (*next)(void); \
    bool (*isActive)(void); \
    type_* (*get)(void); \
    void (*setType)(CapyArrayIteratorType type); \
    void (*toLast)(void); \
} name ## Iterator; \
name ## Iterator name ## IteratorCreate( \
    name* const arr, CapyArrayIteratorType const type); \
name ## Iterator* name ## IteratorAlloc( \
    name* const arr, CapyArrayIteratorType const type); \
void name ## IteratorDestruct(void); \
void name ## IteratorFree(name ## Iterator** const that); \
type_* name ## IteratorReset(void); \
type_* name ## IteratorPrev(void); \
type_* name ## IteratorNext(void); \
bool name ## IteratorIsActive(void); \
void name ## IteratorToLast(void); \
type_* name ## IteratorGet(void); \
void name ## IteratorSetType(CapyArrayIteratorType type);
```

Iterator for generic array structure. Declaration macro for an iterator structure named 'name' ## Iterator, associated to an array named 'name' containing elements of type 'type'

```
#define CapyDefArrayIteratorCreate(name, type_) \
name ## Iterator name ## IteratorCreate( \
    name* const arr, \
    CapyArrayIteratorType const type) { \
    name ## Iterator that = { \
        .idx = 0, \
        .arr = arr, \
        .type = type, \
        .destruct = name ## IteratorDestruct, \
```



```

        .reset = name ## IteratorReset,          \
        .prev = name ## IteratorPrev,            \
        .next = name ## IteratorNext,            \
        .toLast = name ## IteratorToLast,        \
        .isActive = name ## IteratorIsActive,    \
        .get = name ## IteratorGet,              \
        .setType = name ## IteratorSetType,       \
    };                                           \
    $(&that, reset)();                          \
    return that;                                \
}

```

Create an iterator on a generic array

Input argument(s):

- arr: the generic array on which to iterate
- type: the type of iterator

Output and side effect(s):

- Return the iterator

```

#define CopyDefArrayIteratorAlloc(name, type_) \
name ## Iterator* name ## IteratorAlloc(      \
    name* const arr,                          \
    CopyArrayIteratorType const type) {       \
    name ## Iterator* that = NULL;            \
    safeMalloc(that, 1);                      \
    if(!that) return NULL;                   \
    name ## Iterator i = name ## IteratorCreate(arr, type); \
    memcpy(that, &i, sizeof(name ## Iterator)); \
    return that;                              \
}

```

Allocate memory and create an iterator on a generic array

Input argument(s):

- arr: the generic array on which to iterate
- type: the type of iterator

Output and side effect(s):

- Return the iterator

```

#define CopyDefArrayIteratorDestruct(name, type) \
void name ## IteratorDestruct(void) {          \
    return;                                     \
}

```


Free the memory used by an iterator.

Input argument(s):

- that: the iterator to free

```
#define CopyDefArrayIteratorFree(name, type_) \
void name ## IteratorFree(name ## Iterator** const that) { \
    if(that == NULL || *that == NULL) return; \
    $(*that, destruct()); \
    free(*that); \
    *that = NULL; \
}
```

Free the memory used by a pointer to an iterator and reset '*that' to NULL

Input argument(s):

- that: a pointer to the iterator to free

```
#define CopyDefArrayIteratorReset(name, type_) \
type_ * name ## IteratorReset(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    assert(that->arr && "Array iterator uninitialised"); \
    that->idx = 0; \
    if(name ## IteratorIsActive()) \
        return name ## IteratorGet(); \
    else \
        return NULL; \
}
```

Reset the iterator

Output and side effect(s):

- Return the first element of the iteration

```
#define CopyDefArrayIteratorPrev(name, type_) \
type_ * name ## IteratorPrev(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    --(that->idx); \
    if(name ## IteratorIsActive()) \
        return name ## IteratorGet(); \
    else \
        return NULL; \
}
```

Move the iterator to the previous element

Output and side effect(s):

- Return the previous element of the iteration


```

#define CopyDefArrayIteratorNext(name, type_) \
type_* name ## IteratorNext(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    ++(that->idx); \
    if(name ## IteratorIsActive()) \
        return name ## IteratorGet(); \
    else \
        return NULL; \
}

```

Move the iterator to the next element
Output and side effect(s):

- Return the next element of the iteration

```

#define CopyDefArrayIteratorToLast(name, type_) \
void name ## IteratorToLast(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    that->idx = that->arr->size - 1; \
}

```

Move the iterator to the last element

```

#define CopyDefArrayIteratorIsActive(name, type_) \
bool name ## IteratorIsActive(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    return (that->idx < that->arr->size); \
}

```

Check if the iterator is on a valid element
Output and side effect(s):

- Return true if the iterator is on a valid element, else false

```

#define CopyDefArrayIteratorGet(name, type_) \
type_* name ## IteratorGet(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    if($(that, isActive)() == false) return NULL; \
    switch(that->type) { \
        case copyArrayIteratorType_forward: \
            return $(that->arr, getPtr)(that->idx); \
        case copyArrayIteratorType_backward: \
            return $(that->arr, getPtr)(that->arr->size - 1 - that->idx); \
    } \
    return NULL; \
}

```

Get the current element of the iteration
Output and side effect(s):

- Return a pointer to the current element


```

#define CopyDefArrayIteratorSetType(name, type_) \
void name ## IteratorSetType(CopyArrayIteratorType type) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    that->type = type; \
    name ## IteratorReset(); \
}

```

Set the type of the iterator and reset it

Input argument(s):

- type: the new type of the iterator

```

#define CopyDefArrayIterator(name, type) \
    CopyDefArrayIteratorCreate(name, type) \
    CopyDefArrayIteratorAlloc(name, type) \
    CopyDefArrayIteratorDestruct(name, type) \
    CopyDefArrayIteratorFree(name, type) \
    CopyDefArrayIteratorReset(name, type) \
    CopyDefArrayIteratorPrev(name, type) \
    CopyDefArrayIteratorNext(name, type) \
    CopyDefArrayIteratorToLast(name, type) \
    CopyDefArrayIteratorIsActive(name, type) \
    CopyDefArrayIteratorGet(name, type) \
    CopyDefArrayIteratorSetType(name, type)

```

Definition macro calling all the submacros at once for an iterator on an array structure named 'name', containing elements of type 'type'

```

#define CopyDecSizedArray(name, type, size_) \
CopyDecArrayIterator(name, type) \
typedef struct name { \
    type data[size_]; \
    CopyPad(type[size_], data); \
    size_t size; \
    size_t sizeData; \
    name ## Iterator iter; \
    void (*destruct)(void); \
    size_t (*getSize)(void); \
    void (*set)(size_t idx, type const* const val); \
    type (*get)(size_t const idx); \
    type* (*getPtr)(size_t const idx); \
    void (*initIterator)(void); \
    name* (*clone)(void); \
    void (*shuffle)(CopyRandom* const rnd); \
    void (*shuffleLemire)(CopyRandom* const rnd); \
    void (*qsort)(CopyComparator* const cmp); \
    type* (*getMin)(CopyComparator* const cmp); \
    type* (*getMax)(CopyComparator* const cmp); \
    size_t (*search)( \
        type const* const val, \
        CopyComparator* const cmp); \
} name; \
name name ## Create(void); \
name* name ## Alloc(void); \
void name ## Destruct(void); \
void name ## Free(name** const that); \

```



```

size_t name ## GetSize(void); \
void name ## Set(size_t const idx, type const* const val); \
type name ## Get(size_t const idx); \
type* name ## GetPtr(size_t const idx); \
void name ## InitIterator(void); \
void name ## Shuffle(CapyRandom* const rnd); \
void name ## ShuffleLemire(CapyRandom* const rnd); \
void name ## QSort(CapyComparator* const cmp); \
type* name ## GetMin(CapyComparator* const cmp); \
type* name ## GetMax(CapyComparator* const cmp); \
size_t name ## Search( \
    type const* const val, CapyComparator* const cmp); \
name* name ## Clone(void);

```

Fixed size generic array structure. Declaration macro for an array structure named 'name', containing 'size_' elements of type 'type'

```

#define CapyDefSizedArrayCreate(name, type, size_) \
name name ## Create(void) { \
    return (name){ \
        .size = size_, \
        .sizeData = sizeof(type), \
        .destruct = name ## Destruct, \
        .getSize = name ## GetSize, \
        .set = name ## Set, \
        .get = name ## Get, \
        .initIterator = name ## InitIterator, \
        .getPtr = name ## GetPtr, \
        .clone = name ## Clone, \
        .shuffle = name ## Shuffle, \
        .shuffleLemire = name ## ShuffleLemire, \
        .qsort = name ## QSort, \
        .getMin = name ## GetMin, \
        .getMax = name ## GetMax, \
        .search = name ## Search, \
        .iter = {.arr = NULL}, \
    }; \
}

```

Fixed size generic array structure. Definition macro and submacros for an array structure named 'name', containing 'size_' elements of type 'type' Create an array

Output and side effect(s):

- Return an array containing 'size_' elements of type 'type'

```

#define CapyDefSizedArrayAlloc(name, type, size_) \
name* name ## Alloc(void) { \
    name* that = NULL; \
    safeMalloc(that, 1); \
    if(!that) return NULL; \
    *that = name ## Create(); \
    that->iter = name ## IteratorCreate( \
        that, capyArrayIteratorType_forward); \
    return that; \
}

```


Allocate memory for a new array and create it

Output and side effect(s):

- Return an array containing 'size_' elements of type 'type'

Exception(s):

- May raise CopyExc_MallocFailed.

```
#define CopyDefSizedArrayClone(name, type, size_) \
name* name ## Clone(void) {                      \
    name* that = (name*)copyThat;                \
    name* clone = NULL;                          \
    safeMalloc(clone, 1);                        \
    if(!clone) return NULL;                      \
    *clone = *that;                             \
    clone->iter = name ## IteratorCreate(         \
        clone, copyArrayIteratorType_forward);  \
    return clone;                                \
}
```

Clone an array

Output and side effect(s):

- Return a clone of the array

Exception(s):

- May raise CopyExc_MallocFailed.

```
#define CopyDefSizedArrayDestruct(name, type, size_) \
void name ## Destruct(void) {return;}                \
```

Free the memory used by an array.

Input argument(s):

- that: the array to free

```
#define CopyDefSizedArrayFree(name, type, size_) \
void name ## Free(name** const that) {            \
    if(that == NULL || *that == NULL) return;    \
    $(*that, destruct)();                        \
    free(*that);                                 \
    *that = NULL;                               \
}
```

Free the memory used by a pointer to an array and reset '*that' to NULL

Input argument(s):

- that: a pointer to the array to free

```
#define CopyDefSizedArrayGetSize(name, type, size_) \
size_t name ## GetSize(void) {                      \
    return size_;                                   \
}
```

Get the size of the array

Output and side effect(s):

- Return the size

```
#define CopyDefSizedArraySet(name, type, size_) \
void name ## Set(size_t const idx, type const* const val) { \
    assert (val != NULL);                                \
    name* that = (name*)copyThat;                        \
    that->data[idx] = *val;                               \
}
```

Set an element of the array

Input argument(s):

- idx: the index of the element to set
- val: the value to set

```
#define CopyDefSizedArrayGet(name, type, size_) \
type* name ## Get(size_t const idx) {          \
    name* that = (name*)copyThat;              \
    return that->data[idx];                     \
}
```

Get an element of the array

Input argument(s):

- idx: the index of the element to get

Output and side effect(s):

- Return the element value

```
#define CopyDefSizedArrayGetPtr(name, type, size_) \
type* name ## GetPtr(size_t const idx) {          \
    name* that = (name*)copyThat;                  \
    return &(that->data[idx]);                      \
}
```


Get a pointer to an element of the array

Input argument(s):

- idx: the index of the element to get

Output and side effect(s):

- Return the pointer to the element

```
#define CopyDefSizedArrayInitIterator(name, type, size_) \
void name ## InitIterator(void) { \
    name* that = (name*)copyThat; \
    that->iter = \
        name ## IteratorCreate(that, copyArrayIteratorType_forward); \
}
```

Initialise the iterator of the array, must be called after the creation of the array when it is created with Create(), Alloc() automatically initialise the iterator.

```
#define CopyDefSizedArrayShuffle(name, type, size_) \
void name ## Shuffle(CapyRandom* const rnd) { \
    name* that = (name*)copyThat; \
    CapyRangeSize range = {.max = size_ - 1}; \
    for(range.min = 0; range.min < size_ - 1; ++range.min) { \
        size_t jElem = $(rnd, getSizeRange)(&range); \
        if(jElem != range.min) { \
            type tmp = that->data[range.min]; \
            that->data[range.min] = that->data[jElem]; \
            that->data[jElem] = tmp; \
        } \
    } \
}
```

Shuffle the array given a pseudo-random number generator. Fisher-Yates algorithm.

```
#define CopyDefSizedArrayShuffleLemire(name, type, size_) \
void name ## ShuffleLemire(CapyRandom* const rnd) { \
    name* that = (name*)copyThat; \
    CapyRangeUInt32 range = {.max = size_ - 1}; \
    for(range.min = 0; range.min < size_ - 1; ++range.min) { \
        size_t jElem = $(rnd, getUInt32RangeLemire)(&range); \
        if(jElem != range.min) { \
            type tmp = that->data[range.min]; \
            that->data[range.min] = that->data[jElem]; \
            that->data[jElem] = tmp; \
        } \
    } \
}
```


Shuffle the array given a pseudo-random number generator. Fisher-Yates and Lemire algorithm. The size of the array must be less than 2^{32} . Faster than Shuffle.

```
#define CopyDefSizedArrayQSort(name, type, size_) \
void name ## QSort(CapyComparator* const cmp) { \
    name* that = (name*)copyThat; \
    if(that->size < 2) return; \
    CapyQuickSort(that->data, sizeof(type), size_, cmp); \
}
```

Sort the array using the qsort function given a comparator.

```
#define CopyDefSizedArrayGetMin(name, type, size_) \
type* name ## GetMin(CapyComparator* const cmp) { \
    name* that = (name*)copyThat; \
    if(that->size == 0) return NULL; \
    type* min = $(that, getPtr)(0); \
    loop(i, that->size) { \
        type* val = $(that, getPtr)(i); \
        if($(cmp, eval)(min, val) > 0) min = val; \
    } \
    return min; \
}
```

Return the minimum element given a comparator

```
#define CopyDefSizedArrayGetMax(name, type, size_) \
type* name ## GetMax(CapyComparator* const cmp) { \
    name* that = (name*)copyThat; \
    if(that->size == 0) return NULL; \
    type* max = $(that, getPtr)(0); \
    loop(i, that->size) { \
        type* val = $(that, getPtr)(i); \
        if($(cmp, eval)(max, val) < 0) max = val; \
    } \
    return max; \
}
```

Return the maximum element given a comparator

```
#define CopyDefSizedArraySearch(name, type, size_) \
size_t name ## Search( \
    type const* const val, CapyComparator* const cmp) { \
    name* that = (name*)copyThat; \
    if(that->size == 0) raiseExc(CapyExc_InvalidElemIdx); \
    CapyRangeSize range = {.min = 0, .max = that->size - 1}; \
    while(range.min < range.max) { \
        size_t idx = range.min + (range.max - range.min) / 2; \
        if(idx == range.min) idx = range.max; \
        type const* const v = $(that, getPtr)(idx); \
        int ret = $(cmp, eval)(v, val); \
        if(ret < 0) range.min = idx + 1; \
        else if(ret > 0) range.max = idx - 1; \
        else return idx; \
    }
```



```

}
type const* const v = $(that, getPtr)(range.min);
int ret = $(cmp, eval)(v, val);
if(ret == 0) return range.min;
raiseExc(CapyExc_InvalidElemIdx);
return 0;
}

```

Return the index of the value, searched using binary search, or raise `CapyExc_InvalidElemIdx` if the value is not found

```

#define CapyDefSizedArray(name, type, size_) \
    CapyDefSizedArrayCreate(name, type, size_) \
    CapyDefSizedArrayAlloc(name, type, size_) \
    CapyDefSizedArrayClone(name, type, size_) \
    CapyDefSizedArrayDestruct(name, type, size_) \
    CapyDefSizedArrayFree(name, type, size_) \
    CapyDefSizedArrayGetSize(name, type, size_) \
    CapyDefSizedArraySet(name, type, size_) \
    CapyDefSizedArrayGet(name, type, size_) \
    CapyDefSizedArrayGetPtr(name, type, size_) \
    CapyDefSizedArrayInitIterator(name, type, size_) \
    CapyDefSizedArrayShuffle(name, type, size_) \
    CapyDefSizedArrayShuffleLemire(name, type, size_) \
    CapyDefSizedArrayQSort(name, type, size_) \
    CapyDefSizedArrayGetMin(name, type, size_) \
    CapyDefSizedArrayGetMax(name, type, size_) \
    CapyDefSizedArraySearch(name, type, size_) \
    CapyDefArrayIterator(name, type)

```

Definition macro calling all the submacros at once for an array structure named 'name', containing 'size_' elements of type 'type'

```

#define CapyDecArray(name, type) \
    CapyDecArrayIterator(name, type) \
    typedef struct name { \
        type* data; \
        size_t size; \
        size_t sizeData; \
        name ## Iterator iter; \
        void (*destruct)(void); \
        size_t (*getSize)(void); \
        void (*set)(size_t idx, type const* const val); \
        type (*get)(size_t const idx); \
        type* (*getPtr)(size_t const idx); \
        void (*resize)(size_t const size); \
        void (*initIterator)(void); \
        name* (*clone)(void); \
        void (*shuffle)(CapyRandom* const rnd); \
        void (*shuffleLemire)(CapyRandom* const rnd); \
        void (*qsort)(CapyComparator* const cmp); \
        type* (*getMin)(CapyComparator* const cmp); \
        type* (*getMax)(CapyComparator* const cmp); \
        size_t (*search)( \
            type const* const val, \
            CapyComparator* const cmp); \
    } name; \
    name name ## Create(size_t const size);

```



```

name* name ## Alloc(size_t const size);           \
name* name ## Clone(void);                       \
void name ## Destruct(void);                     \
void name ## Free(name** const that);             \
size_t name ## GetSize(void);                    \
void name ## Set(size_t const idx, type const* const val); \
type name ## Get(size_t const idx);              \
type name ## NoGet(size_t const idx);            \
type* name ## GetPtr(size_t const idx);          \
void name ## Resize(size_t const size);          \
void name ## Shuffle(CapyRandom* const rnd);      \
void name ## ShuffleLemire(CapyRandom* const rnd); \
void name ## QSort(CapyComparator* const cmp);   \
type* name ## GetMin(CapyComparator* const cmp); \
type* name ## GetMax(CapyComparator* const cmp); \
size_t name ## Search(                           \
    type const* const val, CapyComparator* const cmp); \
void name ## InitIterator(void);

```

----- Resizable generic array structure. Declaration macro for an array structure named 'name', containing elements of type 'type'

```

#define CapyDefArrayCreate(name, type)           \
name name ## Create_(size_t size, size_t sizeData) { \
    name that = {                                \
        .data = NULL,                          \
        .size = size,                          \
        .sizeData = sizeData,                  \
        .destruct = name ## Destruct,          \
        .getSize = name ## GetSize,            \
        .set = name ## Set,                    \
        .get = name ## Get,                    \
        .getPtr = name ## GetPtr,              \
        .initIterator = name ## InitIterator,  \
        .resize = name ## Resize,              \
        .clone = name ## Clone,                \
        .shuffle = name ## Shuffle,            \
        .shuffleLemire = name ## ShuffleLemire, \
        .qsort = name ## QSort,                \
        .getMin = name ## GetMin,              \
        .getMax = name ## GetMax,              \
        .search = name ## Search,              \
        .iter = {.arr = NULL},                 \
    };                                           \
    if(size > 0) {                              \
        that.data = malloc(sizeData * size);    \
        if(that.data == NULL)                  \
            raiseExc(CapyExc_MallocFailed);     \
    }                                           \
    if(sizeData != sizeof(type))                \
        that.get = name ## NoGet;              \
    return that;                                \
}                                               \
name name ## Create(size_t size) {             \
    return name ## Create_(size, sizeof(type)); \
}

```

Resizable generic array structure. Definition macro and submacros for an

array structure named 'name', containing elements of type 'type' Create an array

Input argument(s):

- size: the new size of the array, may be zero

Output and side effect(s):

- Return an array containing elements of type 'type'

Exception(s):

- May raise CpyExc_MallocFailed.

```
#define CpyDefArrayAlloc(name, type) \
name* name ## Alloc_(size_t size, size_t sizeData) { \
    name* that = NULL; \
    safeMalloc(that, 1); \
    if(!that) return NULL; \
    name a = name ## Create_(size, sizeData); \
    memcpy(that, &a, sizeof(name)); \
    name ## Iterator i = name ## IteratorCreate( \
        that, cpyArrayIteratorType_forward); \
    memcpy(&(that->iter), &i, sizeof(name ## Iterator)); \
    return that; \
} \
name* name ## Alloc(size_t size) { \
    return name ## Alloc_(size, sizeof(type)); \
}
```

Allocate memory for a new array and create it

Input argument(s):

- size: the new size of the array, may be zero

Output and side effect(s):

- Return an array containing elements of type 'type'

Exception(s):

- May raise CpyExc_MallocFailed.


```

#define CopyDefArrayClone(name, type) \
name* name ## Clone(void) { \
    name* that = (name*)copyThat; \
    name* clone = NULL; \
    safeMalloc(clone, 1); \
    if(!clone) return NULL; \
    name a = name ## Create_(that->size, that->sizeData); \
    memcpy(clone, &a, sizeof(name)); \
    name ## Iterator i = name ## IteratorCreate( \
        that, copyArrayIteratorType_forward); \
    memcpy(&(that->iter), &i, sizeof(name ## Iterator)); \
    clone->data = malloc(that->sizeData * that->size); \
    if(clone->data == NULL) \
        raiseExc(CapyExc_MallocFailed); \
    else \
        memcpy(clone->data, that->data, that->sizeData * that->size); \
    return clone; \
}

```

Clone an array

Output and side effect(s):

- Return a clone of the array

Exception(s):

- May raise CapyExc_MallocFailed.

```

#define CopyDefArrayDestruct(name, type) \
void name ## Destruct(void) { \
    name* that = (name*)copyThat; \
    free(that->data); \
    that->data = NULL; \
    that->size = 0; \
}

```

Free the memory used by an array.

Input argument(s):

- that: the array to free

```

#define CopyDefArrayFree(name, type) \
void name ## Free(name** const that) { \
    if(that == NULL || *that == NULL) return; \
    $(*that, destruct)(); \
    free(*that); \
    *that = NULL; \
}

```

Free the memory used by a pointer to an array and reset '*that' to NULL

Input argument(s):

- that: a pointer to the array to free

```
#define CopyDefArrayGetSize(name, type) \
size_t name ## GetSize(void) { \
    return ((name*)copyThat)->size; \
}
```

Get the size of the array

Output and side effect(s):

- Return the size

```
#define CopyDefArraySet(name, type) \
void name ## Set(size_t const idx, type const* const val) { \
    assert(val != NULL); \
    name* that = (name*)copyThat; \
    memcpy( \
        (char*)(that->data) + idx * that->sizeData, \
        val, \
        that->sizeData); \
}
```

Set an element of the array

Input argument(s):

- idx: the index of the element to set
- val: the value to set

```
#define CopyDefArrayGet(name, type) \
type name ## Get(size_t const idx) { \
    return *((type*)((char*)((name*)copyThat)->data) + \
        idx * ((name*)copyThat)->sizeData); \
} \
type name ## NoGet(size_t const idx) { \
    assert(false && \
        "Cannot return element of type " \
        CAPY_MACRO_TO_STR(type) \
        ", use getPtr instead."); \
    (void)idx; \
    type nothing; \
    memset(&nothing, 0, sizeof(nothing)); \
    return nothing; \
}
```

Get an element of the array

Input argument(s):

- idx: the index of the element to get

Output and side effect(s):

- Return the element value

```
#define CopyDefArrayGetPtr(name, type) \
type* name ## GetPtr(size_t const idx) { \
    return (type*)( \
        (char*)((name*)copyThat)->data) + idx * ((name*)copyThat)->sizeData); \
}
```

Get a pointer to an element of the array

Input argument(s):

- idx: the index of the element to get

Output and side effect(s):

- Return the pointer to the element

```
#define CopyDefArrayResize(name, type) \
void name ## Resize(size_t const size) { \
    name* that = (name*)copyThat; \
    if(size == 0) \
        $(that, destruct)(); \
    else \
        safeRealloc(that->data, size * that->sizeData); \
    that->size = size; \
}
```

Resize the array. If the new size of the array is bigger than the current size, new element are not initialised. If the new size of the array is smaller than the old size, elements after the new size are lost.

Input argument(s):

- size: the new size of the array, may be zero, in number of element

Exception(s):

- May raise CopyExc_MallocFailed.

```
#define CopyDefArrayInitIterator(name, type) \
void name ## InitIterator(void) { \
    name* that = (name*)copyThat; \
    name ## Iterator i = name ## IteratorCreate( \
        that, copyArrayIteratorType_forward); \
    memcpy(&(that->iter), &i, sizeof(name ## Iterator)); \
}
```


Initialise the iterator of the array, must be called after the creation of the array when it is created with Create(), Alloc() automatically initialise the iterator.

```
#define CopyDefArrayShuffle(name, type) \
void name ## Shuffle(CopyRandom* const rnd) { \
    name* that = (name*)copyThat; \
    CopyRangeSize range = {.min = 0, .max = that->size - 1}; \
    for(range.min = 0; range.min < that->size - 1; ++range.min) { \
        size_t jElem = $(rnd, getSizeRange)(&range); \
        if(range.min != jElem) { \
            char tmp[that->sizeData]; \
            memcpy( \
                tmp, \
                (char*)(that->data) + range.min * that->sizeData, \
                that->sizeData); \
            memcpy( \
                (char*)(that->data) + range.min * that->sizeData, \
                (char*)(that->data) + jElem * that->sizeData, \
                that->sizeData); \
            memcpy( \
                (char*)(that->data) + jElem * that->sizeData, \
                tmp, \
                that->sizeData); \
        } \
    } \
}
```

Shuffle the array given a pseudo-random number generator. Fisher-Yates algorithm.

```
#define CopyDefArrayShuffleLemire(name, type) \
void name ## ShuffleLemire(CopyRandom* const rnd) { \
    name* that = (name*)copyThat; \
    CopyRangeSize range = {.min = 0, .max = that->size - 1}; \
    for(range.min = 0; range.min < that->size - 1; ++range.min) { \
        size_t jElem = $(rnd, getSizeRange)(&range); \
        if(range.min != jElem) { \
            char tmp[that->sizeData]; \
            memcpy( \
                tmp, \
                (char*)(that->data) + range.min * that->sizeData, \
                that->sizeData); \
            memcpy( \
                (char*)(that->data) + range.min * that->sizeData, \
                (char*)(that->data) + jElem * that->sizeData, \
                that->sizeData); \
            memcpy( \
                (char*)(that->data) + jElem * that->sizeData, \
                tmp, \
                that->sizeData); \
        } \
    } \
}
```

Shuffle the array given a pseudo-random number generator. Fisher-Yates and Lemire algorithm. The array must be less than 2³² in size. Faster than

shuffle.

```
#define CopyDefArrayQSort(name, type) \
void name ## QSort(CapyComparator* const cmp) { \
    name* that = (name*)copyThat; \
    if(that->size < 2) return; \
    CapyQuickSort( \
        that->data, that->sizeData, that->size, cmp); \
}
```

Sort the array using the qsort function given a comparator.

```
#define CopyDefArrayGetMin(name, type) \
type* name ## GetMin(CapyComparator* const cmp) { \
    name* that = (name*)copyThat; \
    if(that->size == 0) return NULL; \
    type* min = $(that, getPtr)(0); \
    loop(i, that->size) { \
        type* val = $(that, getPtr)(i); \
        if($(cmp, eval)(min, val) > 0) min = val; \
    } \
    return min; \
}
```

Return the minimum element given a comparator

```
#define CopyDefArrayGetMax(name, type) \
type* name ## GetMax(CapyComparator* const cmp) { \
    name* that = (name*)copyThat; \
    if(that->size == 0) return NULL; \
    type* max = $(that, getPtr)(0); \
    loop(i, that->size) { \
        type* val = $(that, getPtr)(i); \
        if($(cmp, eval)(max, val) < 0) max = val; \
    } \
    return max; \
}
```

Return the maximum element given a comparator

```
#define CopyDefArraySearch(name, type) \
size_t name ## Search( \
    type const* const val, CapyComparator* const cmp) { \
    name* that = (name*)copyThat; \
    if(that->size == 0) raiseExc(CapyExc_InvalidElemIdx); \
    CapyRangeSize range = {.min = 0, .max = that->size - 1}; \
    while(range.min < range.max) { \
        size_t idx = range.min + (range.max - range.min) / 2; \
        if(idx == range.min) idx = range.max; \
        type const* const v = $(that, getPtr)(idx); \
        int ret = $(cmp, eval)(v, val); \
        if(ret < 0) range.min = idx + 1; \
        else if(ret > 0) range.max = idx - 1; \
        else return idx; \
    } \
    type const* const v = $(that, getPtr)(range.min); \
    int ret = $(cmp, eval)(v, val); \
}
```



```

    if(ret == 0) return range.min;
    raiseExc(CapyExc_InvalidElemIdx);
    return 0;
}

```

\
 \
 \

Return the index of the value, searched using binary search, or raise `CapyExc_InvalidElemIdx` if the value is not found

```

#define CapyDefArray(name, type) \
    CapyDefArrayCreate(name, type) \
    CapyDefArrayAlloc(name, type) \
    CapyDefArrayClone(name, type) \
    CapyDefArrayDestruct(name, type) \
    CapyDefArrayFree(name, type) \
    CapyDefArrayGetSize(name, type) \
    CapyDefArraySet(name, type) \
    CapyDefArrayGet(name, type) \
    CapyDefArrayGetPtr(name, type) \
    CapyDefArrayResize(name, type) \
    CapyDefArrayInitIterator(name, type) \
    CapyDefArrayShuffle(name, type) \
    CapyDefArrayShuffleLemire(name, type) \
    CapyDefArrayQSort(name, type) \
    CapyDefArrayGetMin(name, type) \
    CapyDefArrayGetMax(name, type) \
    CapyDefArraySearch(name, type) \
    CapyDefArrayIterator(name, type)

```

Definition macro calling all the submacros at once for an array structure named 'name', containing elements of type 'type'

2.2 Enumerations

```

typedef enum CapyArrayIteratorType {
    capyArrayIteratorType_forward,
    capyArrayIteratorType_backward,
} CapyArrayIteratorType;

```

Enumeration for the types of iterator on generic array

2.3 Typedefs

```

typedef struct CapyRandom CapyRandom;

```

Predeclaration of the Random class used by the shuffle method (can't include due to loop dependency with `DistributionDiscrete`)

2.4 Functions

None.

3 Unit bezier.h

Bezier surface class.

3.1 Macros

```
#define CAPY_BEZIER_H
```

```
#define CAPY_BEZIER_MAX_DIMIN 254
```

Maximum order and dimension for a CapyBezier

```
#define CAPY_BEZIER_MAX_DIMOUT 254
```

3.2 Enumerations

None.

3.3 Typedefs

```
typedef uint8_t CapyBezierOrder_t;
```

Type of the order of a CapyBezier

```
typedef struct CapyBezier CapyBezier;
```

Bezier curve object

```
typedef struct CapyBezierIteratorWindow CapyBezierIteratorWindow;
```

Iteration windows for CapyBezierIterator

```
typedef struct CapyBezierSpline CapyBezierSpline;
```

Bezier spline object (composite of Bezier curves)

3.4 Struct CpyBezierPosition

3.4.1 Struct CpyBezierPosition's properties

```
double in[CAPY_BEZIER_MAX_DIMIN];
```

```
double out[CAPY_BEZIER_MAX_DIMOUT];
```

3.4.2 Struct CpyBezierPosition's methods

None.

3.5 Struct CpyBezierIterator

3.5.1 Struct CpyBezierIterator's properties

```
size_t idx;
```

Index of the current step in the iteration

```
CpyBezierPosition datatype;
```

Returned data type

```
CpyBezierIteratorWindow* windows;
```

List of windows

```
CpyBezier const* bezier;
```

Bezier associated to the iteration

```
double epsilon;
```

Epsilon for the step of iteration using euclidean distance along the Bezier curve (default: 0.1)

```
CpyMemPoolBezierIteratorWindow memPool;
```

Memory pool for the windows

3.5.2 Struct CpyBezierIterator's methods

```
void (*destruct)(void);
```

Destructor

```
CpyBezierPosition* (*reset)(void);
```

Reset the iterator

Output and side effect(s):

- Return the first position of the iteration

```
CpyBezierPosition* (*next)(void);
```

Move the iterator to the next position

Output and side effect(s):

- Return the next position of the iteration

```
bool (*isActive)(void);
```

Check if the iterator is on a valid position

Output and side effect(s):

- Return true if the iterator is on a valid position, else false

```
CpyBezierPosition* (*get)(void);
```

Get the current position of the iteration

Output and side effect(s):

- Return the current position

3.6 Functions

```
CpyBezier CpyBezierCreate(  
    CpyBezierOrder_t const order,  
    size_t const dimIn,  
    size_t const dimOut);
```


Create a CpyBezier

Input argument(s):

- order: the order of the Bezier object
- dimIn: number of inputs
- dimOut: number of outputs

Output and side effect(s):

- Return a CpyBezier

```
CpyBezier* CpyBezierAlloc(  
    CpyBezierOrder_t const order,  
    size_t const dimIn,  
    size_t const dimOut);
```

Allocate memory for a new CpyBezier and create it

Input argument(s):

- order: the order of the Bezier object
- dimIn: number of inputs
- dimOut: number of outputs

Output and side effect(s):

- Return a CpyBezier

Exception(s):

- May raise CpyExc_MallocFailed.

```
CpyBezier* CpyBezierLoad(FILE* const stream);
```

Allocate memory for a new CpyBezier and load it from a binary stream

Input argument(s):

- stream: the binary stream to load the Bezier from

Output and side effect(s):

- Return a CpyBezier

Exception(s):

- May raise CpyExc_MallocFailed, CpyExc_StreamReadError.

```
void CpyBezierFree(CpyBezier** const that);
```

Free the memory used by a CpyBezier* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyBezier to free

```
CpyBezierIterator CpyBezierIteratorCreate(CpyBezier const* const bezier);
```

Create an iterator on a CpyBezier

Input argument(s):

- bezier: the bezier on which to iterate

Output and side effect(s):

- Return the iterator

```
CpyBezierIterator* CpyBezierIteratorAlloc(CpyBezier const* const bezier);
```

Allocate memory and create an iterator on a CpyBezier

Input argument(s):

- bezier: the bezier on which to iterate

Output and side effect(s):

- Return the iterator

```
void CpyBezierIteratorFree(CpyBezierIterator** const that);
```

Free the memory used by a pointer to an iterator and reset '*that' to NULL

Input argument(s):

- that: a pointer to the iterator to free


```
CapyBezierSpline CapyBezierSplineCreate(
    size_t const* const nbSegment,
        size_t const dimIn,
        size_t const dimOut);
```

Create a CapyBezierSpline

Input argument(s):

- nbSegment: array of dimIn integer, number of segments per input dimension
- dimIn: number of inputs
- dimOut: number of outputs

Output and side effect(s):

- Return a CapyBezierSpline

```
CapyBezierSpline* CapyBezierSplineAlloc(
    size_t const* const nbSegment,
        size_t const dimIn,
        size_t const dimOut);
```

Allocate memory for a new CapyBezierSpline and create it

Input argument(s):

- nbSegment: array of dimIn integer, number of segments per input dimension
- dimIn: number of inputs
- dimOut: number of outputs

Output and side effect(s):

- Return a CapyBezierSpline

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyBezierSplineFree(CapyBezierSpline** const that);
```

Free the memory used by a CapyBezierSpline* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyBezierSpline to free

4 Unit bitarray.h

Bit array manipulation class.

4.1 Macros

```
#define CAPY_BITARRAY_H
```

4.2 Enumerations

None.

4.3 Typedefs

None.

4.4 Struct CopyBitArray

4.4.1 Struct CopyBitArray's properties

```
size_t size;
```

Size in bits of the array

```
uint8_t* data;
```

Data

4.4.2 Struct CopyBitArray's methods

```
void (*destruct)(void);
```

Destructor

```
void (*set)(  
    uint8_t const* const bytes,  
    size_t const size);
```

Initialised the array with an array of bytes

Input argument(s):

- bytes: the bytes used to initialise
- size: the number of bytes in 'bytes'

Output and side effect(s):

- 'that->data' is freed if necessary and replaced with a copy of 'bytes',
- and 'that->size' is updated

```
bool (*getBit)(size_t const iBit);
```

Get one bit in the array

Input argument(s):

- iBit: the index of the bit

Output and side effect(s):

- Return the bit at the requested position

```
void (*resize)(size_t const size);
```

Resize the array

Input argument(s):

- size: the new size in bits

Output and side effect(s):

- Memory is allocated for 'that->data' and 'that->size' is updated. The
- content of 'that->data' is undefined.

```
void (*setBit)(
    size_t const iBit,
    bool const bit);
```

Set a bit in the array

Input argument(s):

- iBit: the index of the bit
- bit: the value of the bit

Output and side effect(s):

- The bit is updated.

4.5 Functions

```
CapyBitArray CapyBitArrayCreate(void);
```

Create a CapyBitArray

Output and side effect(s):

- Return an empty CapyBitArray

```
CapyBitArray* CapyBitArrayAlloc(void);
```

Allocate memory for a new CapyBitArray and create it

Output and side effect(s):

- Return an empty CapyBitArray

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyBitArrayFree(CapyBitArray** const that);
```

Free the memory used by a CapyBitArray* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyBitArray to free

5 Unit bnoise.h

B-Noise generator class.

5.1 Macros

```
#define CAPY_BNOISE_H
```

```
#define CAPY_BNOISE_NB_DIM_IN 2
```

Dimensions of noise: 2D->1D

```
#define CAPY_BNOISE_NB_DIM_OUT 1
```


5.2 Enumerations

None.

5.3 Typedefs

```
typedef uint8_t CapyBNoiseOrder_t;
```

Type of the order of a BNoise

5.4 Struct CapyBNoise

5.4.1 Struct CapyBNoise's properties

```
CapyNoiseDef;
```

Inherits CapyNoise

```
CapyBNoiseOrder_t order;
```

Order of the noise

```
CapyPad(CapyBNoiseOrder_t, order);
```

```
size_t dim;
```

Dimension of the grid (square) in number of vertices

```
double fDim;
```

Cast to float of the dimension of the grid

```
double* map;
```

Noise data

```
double* curve[CAPY_BNOISE_NB_DIM_IN][CAPY_BNOISE_NB_DIM_IN];
```

5.4.2 Struct CapyBNoise's methods

```
void (*destructCapyNoise)(void);
```

Destructor

5.5 Functions

```
CapyBNoise CapyBNoiseCreate(  
    CapyBNoiseOrder_t const order,  
    CapyRandomSeed_t const seed);
```

Create a CapyBNoise

Input argument(s):

- order: order of the noise
- seed: seed of the noise

Output and side effect(s):

- Return a CapyBNoise

```
CapyBNoise* CapyBNoiseAlloc(  
    CapyBNoiseOrder_t const order,  
    CapyRandomSeed_t const seed);
```

Allocate memory for a new CapyBNoise and create it

Input argument(s):

- order: order of the noise
- seed: seed of the noise

Output and side effect(s):

- Return a CapyBNoise

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyBNoiseFree(CapyBNoise** const that);
```

Free the memory used by a CapyBNoise* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyBNoise to free

6 Unit bresenham.h

Bresenham class.

6.1 Macros

```
#define CAPY_BRESENHAM_H
```

6.2 Enumerations

None.

6.3 Typedefs

None.

6.4 Struct CapyBresenham

6.4.1 Struct CapyBresenham's properties

```
int64_t* pos;
```

Current position

```
int64_t* startPos;
```

Start position (default: 0,0)

```
int64_t* endPos;
```

End position (default: 1,1)

```
int64_t* datatype;
```

Fields for the forEach macro

```
size_t idx;
```

Index in iteration

```
int64_t d[2], s[2], err[2];
```

Internal variables of the Bresenham algorithm

6.4.2 Struct CpyBresenham's methods

```
int64_t** (*reset)(void);
```

Methods for the forEach macro

```
bool (*isActive)(void);
```

```
int64_t** (*next)(void);
```

```
void (*destruct)(void);
```

Destructor

6.5 Functions

```
CpyBresenham CpyBresenhamCreate(void);
```

Create a CpyBresenham

Output and side effect(s):

- Return a CpyBresenham

```
CpyBresenham* CpyBresenhamAlloc(void);
```

Allocate memory for a new CpyBresenham and create it

Output and side effect(s):

- Return a CpyBresenham

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyBresenhamFree(CpyBresenham** const that);
```

Free the memory used by a CpyBresenham* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyBresenham to free

7 Unit btree.h

BTree class.

7.1 Macros

```
#define CAPY_BTREE_H
```

```
#define CopyDecBTree(name, type) \
typedef struct name name; \
struct name { \
    type* elems; \
    size_t nbMaxElem; \
    size_t nbElem; \
    name* parent; \
    name** childs; \
    void (*destruct)(void); \
    int (*cmp)( \
        type const* const elemA, \
        type const* const elemB); \
    void (*add)(type const elem); \
    void (*remove)(type const elem); \
    void (*removeElem)(size_t const iElem); \
    type* (*find)(type const elem); \
    name* (*findNode)(type const elem); \
    name* (*getLeafNodeForElem)(type const elem); \
    void (*split)(void); \
    name* (*insertNode)(name const node); \
    void (*mergeChildren)(void); \
    void (*rebalance)(void); \
}; \
name name ## Create( \
    size_t const nb, \
    int(*cmp)(type const*, type const*)); \
name* name ## Alloc( \
    size_t const nb, \
    int(*cmp)(type const*, type const*)); \
void name ## Destruct(void); \
void name ## Free(name** const that); \
name* name ## GetLeafNodeForElem(type const elem); \
void name ## Split(void); \
void name ## Add(type const elem); \
void name ## Remove(type const elem); \
void name ## RemoveElem(size_t const iElem); \
name* name ## InsertNode(name const node); \
type* name ## Find(type const elem); \
name* name ## FindNode(type const elem); \
void name ## MergeChildren(void); \
void name ## Rebalance(void);
```

Generic b-tree structure. Declaration macro for a b-tree structure named 'name', containing elem of type 'type'.


```

#define CopyDefBTreeCreate(name, type) \
name name ## Create( \
    size_t const nb, \
        int(*cmp)(type const*, type const*)) { \
    name that = { \
        .nbMaxElem = nb, \
        .destruct = name ## Destruct, \
        .cmp = cmp, \
        .add = name ## Add, \
        .remove = name ## Remove, \
        .removeElem = name ## RemoveElem, \
        .find = name ## Find, \
        .findNode = name ## FindNode, \
        .getLeafNodeForElem = name ## GetLeafNodeForElem, \
        .split = name ## Split, \
        .insertNode = name ## InsertNode, \
        .mergeChildren = name ## MergeChildren, \
        .rebalance = name ## Rebalance, \
    }; \
    safeMalloc(that.elems, nb); \
    loop(i, nb) that.elems[i] = (type){0}; \
    safeMalloc(that.childs, nb + 1); \
    loop(i, nb + 1) that.childs[i] = NULL; \
    return that; \
}

```

Generic b-tree structure. Definition macro for a b-tree structure named 'name', containing elem of type 'type' Create a b-tree

Input argument(s):

- nb: max number of element per node
- cmp: comparison function to sort elem

Output and side effect(s):

- Return a b-tree containing elem of type 'type'

```

#define CopyDefBTreeAlloc(name, type) \
name* name ## Alloc( \
    size_t const nb, \
        int(*cmp)(type const*, type const*)) { \
    name* that = NULL; \
    safeMalloc(that, 1); \
    if(!that) return NULL; \
    *that = name ## Create(nb, cmp); \
    return that; \
}

```

Allocate memory for a new b-tree and create it

Input argument(s):

- nb: max number of element per node

- cmp: comparison function to sort elem

Output and side effect(s):

- Return a b-tree containing elem of type 'type'

Exception(s):

- May raise CopyExc_MallocFailed.

```
#define CopyDefBTreeDestruct(name, type) \
void name ## Destruct(void) { \
    name* that = (name*)copyThat; \
    loop(iElem, that->nbElem) { \
        if(that->elems[iElem].destruct != NULL) { \
            $(that->elems + iElem, destruct)(); \
        } \
    } \
    loop(iChild, that->nbElem + 1) { \
        if(that->childs[iChild] != NULL) { \
            name ## Free(that->childs + iChild); \
        } \
    } \
    free(that->elems); \
    free(that->childs); \
    *that = (name){0}; \
}
```

Free the memory used by a b-tree.

Input argument(s):

- that: the b-tree to free

```
#define CopyDefBTreeFree(name, type) \
void name ## Free(name** const that) { \
    if(that == NULL || *that == NULL) return; \
    $(*that, destruct)(); \
    free(*that); \
    *that = NULL; \
}
```

Free the memory used by a pointer to a b-tree and reset '*that' to NULL

Input argument(s):

- that: a pointer to the b-tree to free


```

#define CopyDefBTreeGetLeafNodeForElem(name, type) \
name* name ## GetLeafNodeForElem(type const elem) { \
    name* that = (name*)copyThat; \
    size_t iElem = 0; \
    while( \
        iElem < that->nbElem && \
        (that->cmp)(that->elems + iElem, &elem) < 0 \
    ) { \
        iElem += 1; \
    } \
    if(that->childs[iElem] != NULL) { \
        return $(that->childs[iElem], getLeafNodeForElem)(elem); \
    } else { \
        return that; \
    } \
}

```

Get the leaf node where an element should be

Input argument(s):

- elem: the element

Output and side effect(s):

- Traverse the BTree and return the first leaf that accomodate the element.

```

#define CopyDefBTreeSplit(name, type) \
void name ## Split(void) { \
    name* that = (name*)copyThat; \
    if(that->nbElem < 3) return; \
    size_t iMedian = that->nbElem / 2; \
    name* priorNode = name ## Alloc(that->nbMaxElem, that->cmp); \
    name* posteriorNode = name ## Alloc(that->nbMaxElem, that->cmp); \
    priorNode->parent = that; \
    posteriorNode->parent = that; \
    loop(iElem, iMedian) { \
        priorNode->elems[iElem] = that->elems[iElem]; \
        priorNode->childs[iElem] = that->childs[iElem]; \
        that->childs[iElem] = NULL; \
        if(priorNode->childs[iElem] != NULL) { \

```



```

    priorNode->childs[iElem]->parent = priorNode;
    }
    \
    }
    \
    priorNode->childs[iMedian] = that->childs[iMedian];
    that->childs[iMedian] = NULL;
    \
    if(priorNode->childs[iMedian] != NULL) {
        \
        priorNode->childs[iMedian]->parent = priorNode;
    }
    \
    priorNode->nbElem = iMedian;
    \
    loop(iElem, that->nbElem - iMedian - 1) {
        \
        posteriorNode->elems[iElem] = that->elems[iMedian + 1 + iElem];
        \
        posteriorNode->childs[iElem] = that->childs[iMedian + 1 + iElem];
        \
        that->childs[iMedian + 1 + iElem] = NULL;
        \
        if(posteriorNode->childs[iElem] != NULL) {
            \
            posteriorNode->childs[iElem]->parent = posteriorNode;
        }
        \
    }
    \
    posteriorNode->childs[that->nbElem - iMedian - 1] =
        \
        that->childs[that->nbElem];
    \
    that->childs[that->nbElem] = NULL;
    \
    if(posteriorNode->childs[that->nbElem - iMedian - 1] != NULL) {
        \
        posteriorNode->childs[that->nbElem - iMedian - 1]->parent =
            \
            posteriorNode;
    }
    \
    posteriorNode->nbElem = that->nbElem - iMedian - 1;
    \
    that->elems[0] = that->elems[iMedian];
    \
    that->nbElem = 1;
    \
    that->childs[0] = priorNode;
    \

```



```

that->childs[1] = posteriorNode;
}

```

Split the root node in two
Output and side effect(s):

- If the root node has less than three elements nothing happens. Else,
- the elements of the root node are split into two new nodes, one for
- those lower than the median element in the root and the other for those
- greater than the median element in the root. The median element remains
- in the root and the new nodes become its prior and posterior childs.

```

#define CopyDefBTreeInsertNode(name, type) \
name* name ## InsertNode(name const node) { \
    name* that = (name*)copyThat; \
    if(that->nbElem >= that->nbMaxElem) { \
        $(that, split()); \
        if((that->cmp)(that->elems, node.elems) > 0) { \
            $(that->childs[0], insertNode)(node); \
        } else { \
            $(that->childs[1], insertNode)(node); \
        } \
        if(that->parent != NULL) { \
            size_t iChild = 0; \
            while(iChild <= that->parent->nbElem) { \
                if(that->parent->childs[iChild] == that) { \
                    that->parent->childs[iChild] = NULL; \
                } \
                iChild += 1; \
            } \
            name* newThat = $(that->parent, insertNode)(*that); \
            free(that->childs); \
            free(that->elems); \
            free(that); \
            that = newThat; \
        } \
    } else { \
        size_t iElem = 0; \
        while( \
            iElem < that->nbElem && \
            (that->cmp)(that->elems + iElem, node.elems) < 0 \
        ) { \
            iElem += 1; \
        } \
        if(that->childs[iElem] == NULL) { \
            size_t jElem = that->nbElem; \
            while(jElem > iElem) { \
                that->childs[jElem + 1] = that->childs[jElem]; \
                that->childs[jElem] = that->childs[jElem - 1]; \
            } \
        } \
    } \
}

```



```

        that->elems[jElem] = that->elems[jElem - 1];
        jElem -= 1;
    }
    that->elems[iElem] = node.elems[0];
    that->childs[iElem] = node.childs[0];
    if(that->childs[iElem] != NULL) {
        that->childs[iElem]->parent = that;
    }
    that->childs[iElem + 1] = node.childs[1];
    if(that->childs[iElem + 1] != NULL) {
        that->childs[iElem + 1]->parent = that;
    }
    that->nbElem += 1;
} else {
    that = $(that->childs[iElem], insertNode)(node);
}
return that;
}

```

Insert a node into the elements of the tree

Input argument(s):

- node: the node to insert

Output and side effect(s):

- The node is inserted. This may trigger a restructuration of the BTree
- which invalidate the 'that' pointer. Then, this method returns 'that' if
- it's still valid, or its parent if 'that' doesn't exist any more.

```

#define CopyDefBTreeAdd(name, type)
void name ## Add(type const elem) {
    name* that = (name*)copyThat;
    name node = name ## Create(that->nbMaxElem, that->cmp);
    node.elems[0] = elem;
    node.nbElem = 1;
    name* leaf = $(that, getLeafNodeForElem)(elem);
    leaf = $(leaf, insertNode)(node);
    $(&node, destruct)();
}

```

Add an element to the b-tree

Input argument(s):

- elem: the element to add


```

#define CopyDefBTreeRebalance(name, type) \
void name ## Rebalance(void) { \
    name* that = (name*)copyThat; \
    if(that->parent == NULL) return; \
    size_t iChildOfThatInParent = 0; \
    while( \
        iChildOfThatInParent <= that->parent->nbElem && \
        that->parent->childs[iChildOfThatInParent] != that \
    ) { \
        iChildOfThatInParent += 1; \
    } \
    size_t nbElemInSiblings[2] = {0, 0}; \
    if( \
        iChildOfThatInParent > 0 && \
        that->parent->childs[iChildOfThatInParent - 1] != NULL \
    ) { \
        nbElemInSiblings[0] = \
            that->parent->childs[iChildOfThatInParent - 1]->nbElem; \
    } \
    if( \
        iChildOfThatInParent < that->parent->nbElem && \
        that->parent->childs[iChildOfThatInParent + 1] != NULL \
    ) { \
        nbElemInSiblings[1] = \
            that->parent->childs[iChildOfThatInParent + 1]->nbElem; \
    } \
    size_t iGiver = 0; \
    if(nbElemInSiblings[1] > nbElemInSiblings[0]) iGiver = 1; \
    if(nbElemInSiblings[iGiver] > that->nbMaxElem / 2) { \
        size_t iElemTaken = iChildOfThatInParent + iGiver - 1; \
        type valTaken = that->parent->elems[iElemTaken]; \
        size_t iSiblingGiver = \
            iChildOfThatInParent + 2 * iGiver - 1; \
        name* siblingGiver = that->parent->childs[iSiblingGiver]; \
        if(iGiver == 0) { \
            size_t iElem = that->nbElem; \
            while(iElem > 0) { \
                that->elems[iElem] = that->elems[iElem - 1]; \
                that->childs[iElem + 1] = that->childs[iElem]; \
                iElem -= 1; \
            } \
            that->childs[1] = that->childs[0]; \
            that->nbElem += 1; \
            that->elems[0] = valTaken; \
            that->childs[0] = siblingGiver->childs[siblingGiver->nbElem]; \
            if(that->childs[0] != NULL) { \
                that->childs[0]->parent = that; \
            } \
            that->parent->elems[iElemTaken] = \
                siblingGiver->elems[siblingGiver->nbElem - 1]; \
            siblingGiver->childs[siblingGiver->nbElem] = NULL; \
            siblingGiver->nbElem -= 1; \
        } else { \
            that->elems[that->nbElem] = valTaken; \
            that->childs[that->nbElem + 1] = siblingGiver->childs[0]; \
            if(that->childs[that->nbElem + 1] != NULL) { \
                that->childs[that->nbElem + 1]->parent = that; \
            } \
            that->nbElem += 1; \
            that->parent->elems[iElemTaken] = siblingGiver->elems[0]; \
            siblingGiver->nbElem -= 1; \
        } \
    } \
}

```



```

        loop(iElem, siblingGiver->nbElem) {
            siblingGiver->elems[iElem] = siblingGiver->elems[iElem + 1];
            siblingGiver->childs[iElem] = siblingGiver->childs[iElem + 1];
        }
        siblingGiver->childs[siblingGiver->nbElem] =
            siblingGiver->childs[siblingGiver->nbElem + 1];
        siblingGiver->childs[siblingGiver->nbElem + 1] = NULL;
    }
} else {
    if(0) {
        printf(
            "TODO: 'that' and 'that->parent' and 'siblinGiver' should be "
            "merged together into one single node to keep the minimum "
            "number of element in a node. Cf issue #36.\n");
    }
}
}
}

```

Rebalance the tree to ensure it has the required minimum number of elements
Output and side effect(s):

- The tree is rebalanced.

```

#define CopyDefBTreeMergeChildren(name, type)
void name ## MergeChildren(void) {
    name* that = (name*)copyThat;
    size_t nbElemInChildren = 0;
    size_t iChild = 0;
    while(iChild <= that->nbElem) {
        if(that->childs[iChild] != NULL) {
            nbElemInChildren += that->childs[iChild]->nbElem;
        }
        iChild += 1;
    }
    if(
        nbElemInChildren + that->nbElem > that->nbMaxElem ||
        nbElemInChildren == 0
    ) {
        return;
    }
    name newNode = name ## Create(that->nbMaxElem, that->cmp);
    newNode.parent = that->parent;
    iChild = 0;
    while(iChild <= that->nbElem) {
        name* child = that->childs[iChild];
        if(child != NULL) {
            loop(iElem, child->nbElem) {
                newNode.elems[newNode.nbElem] = child->elems[iElem];
                newNode.childs[newNode.nbElem] = child->childs[iElem];
                if(newNode.childs[newNode.nbElem] != NULL) {
                    newNode.childs[newNode.nbElem]->parent = that;
                }
                newNode.nbElem += 1;
            }
            newNode.childs[newNode.nbElem] = child->childs[child->nbElem];
            if(newNode.childs[newNode.nbElem] != NULL) {
                newNode.childs[newNode.nbElem]->parent = that;
            }
        }
    }
}

```



```

        free(that->childs[iChild]->elems);
        free(that->childs[iChild]->childs);
        free(that->childs[iChild]);
    }
    if(iChild < that->nbElem) {
        newNode.elems[newNode.nbElem] = that->elems[iChild];
        newNode.nbElem += 1;
    }
    iChild += 1;
}
*that = newNode;
}

```

Merge the children of a node into that node.

Output and side effect(s):

- The children are merged into the node if possible. If not, nothing happen.

```

#define CopyDefBTreeRemoveElem(name, type)
void name ## RemoveElem(size_t const iElem) {
    name* that = (name*)copyThat;
    if(iElem >= that->nbElem) return;
    if(that->childs[iElem] != NULL) {
        name* child = that->childs[iElem];
        while(child->childs[child->nbElem] != NULL) {
            child = child->childs[child->nbElem];
        }
        that->elems[iElem] = child->elems[child->nbElem - 1];
        $(child, removeElem)(child->nbElem - 1);
    } else if(that->childs[iElem + 1] != NULL) {
        name* child = that->childs[iElem + 1];
        while(child->childs[0] != NULL) child = child->childs[0];
        that->elems[iElem] = child->elems[0];
        $(child, removeElem)(0);
    } else {
        size_t jElem = iElem;
        while(jElem < that->nbElem - 1) {
            that->elems[jElem] = that->elems[jElem + 1];
            that->childs[jElem + 1] = that->childs[jElem + 2];
            jElem += 1;
        }
        that->nbElem -= 1;
        if(that->nbElem < that->nbMaxElem / 2) {
            $(that, rebalance)();
        }
        if(that->nbElem == 0 && that->parent != NULL) {
            size_t iChildThatInParent = 0;
            while(that->parent->childs[iChildThatInParent] != that) {
                iChildThatInParent += 1;
            }
            name ## Free(that->parent->childs + iChildThatInParent);
            that = NULL;
        }
    }
}

```


Remove an element at a given position in a node of a b-tree

Input argument(s):

- iElem: the position of the element to remove

Output and side effect(s):

- The element is removed.

```
#define CopyDefBTreeRemove(name, type)
\
void name ## Remove(type const val) {
\
    name* that = (name*)copyThat;
\
    if(that->nbElem == 0) return;
\
    size_t iElem = 0;
\
    while(iElem < that->nbElem) {
\
        int cmpVal = (that->cmp)(that->elems + iElem, &val);
\
        if(cmpVal == 0) {
\
            if(that->elems[iElem].destruct != NULL) {
\
                $(that->elems + iElem, destruct)();
\
            }
\
            $(that, removeElem)(iElem);
\
            return;
\
        } else if(cmpVal > 0) {
\
            if(that->childs[iElem] != NULL) {
\
                $(that->childs[iElem], remove)(val);
\
            }
\
            return;
\
        } else if(cmpVal < 0) {
\
            iElem += 1;
\
        }
\
    }
\
    if(that->childs[iElem] != NULL) {
\
        $(that->childs[iElem], remove)(val);
\
    }
```



```

    }
    \
}

```

Remove an element from the b-tree

Input argument(s):

- elem: the element to remove

Output and side effect(s):

- The element is removed if it exists, else nothing happens. If the
- element is removed it is destruct

```

#define CopyDefBTreeFind(name, type) \
type* name ## Find(type const val) { \
    name* that = (name*)copyThat; \
    if(that->nbElem == 0) return NULL; \
    size_t iElem = 0; \
    while(iElem < that->nbElem) { \
        int cmpVal = (that->cmp)(that->elems + iElem, &val); \
        if(cmpVal == 0) { \
            return that->elems + iElem; \
        } else if(cmpVal > 0) { \
            if(that->childs[iElem] != NULL) { \
                return $(that->childs[iElem], find)(val); \
            } else { \
                return NULL; \
            } \
        } else if(cmpVal < 0) { \
            iElem += 1; \
        } \
    } \
    if(that->childs[iElem] != NULL) { \
        return $(that->childs[iElem], find)(val); \
    } else { \
        return NULL; \
    } \
}

```

Find an element in the b-tree

Input argument(s):

- elem: the element to find

Output and side effect(s):

- If the element is found return a pointer to it, else return null.


```

#define CopyDefBTreeFindNode(name, type) \
name* name ## FindNode(type const val) { \
    name* that = (name*)copyThat; \
    if(that->nbElem == 0) return NULL; \
    size_t iElem = 0; \
    while(iElem < that->nbElem) { \
        int cmpVal = (that->cmp)(that->elems + iElem, &val); \
        if(cmpVal == 0) { \
            return that; \
        } else if(cmpVal > 0) { \
            if(that->childs[iElem] != NULL) { \
                return $(that->childs[iElem], findNode)(val); \
            } else { \
                return NULL; \
            } \
        } else if(cmpVal < 0) { \
            iElem += 1; \
        } \
    } \
    if(that->childs[iElem] != NULL) { \
        return $(that->childs[iElem], findNode)(val); \
    } else { \
        return NULL; \
    } \
}

```

Find a node containing an element in the b-tree

Input argument(s):

- elem: the element to find

Output and side effect(s):

- If the element is actually in the tree, return the node containing the
- element, else return NULL.

```

#define CopyDefBTree(name, type) \
    CopyDefBTreeCreate(name, type) \
    CopyDefBTreeAlloc(name, type) \
    CopyDefBTreeDestruct(name, type) \
    CopyDefBTreeFree(name, type) \
    CopyDefBTreeAdd(name, type) \
    CopyDefBTreeRemove(name, type) \
    CopyDefBTreeRemoveElem(name, type) \
    CopyDefBTreeGetLeafNodeForElem(name, type) \
    CopyDefBTreeSplit(name, type) \
    CopyDefBTreeInsertNode(name, type) \
    CopyDefBTreeFind(name, type) \
    CopyDefBTreeFindNode(name, type) \
    CopyDefBTreeMergeChildren(name, type) \
    CopyDefBTreeRebalance(name, type)

```

Definition macro calling all the submacros at once for a BTree containing elements of type 'type'.

7.2 Enumerations

None.

7.3 Typedefs

None.

7.4 Functions

None.

8 Unit burrowswheelertransform.h

BurrowsWheelerTransform class.

8.1 Macros

```
#define CAPY_BURROWSWHEELERTRANSFORM_H
```

8.2 Enumerations

None.

8.3 Typedefs

None.

8.4 Struct CapyBurrowsWheelerTransformData

8.4.1 Struct CapyBurrowsWheelerTransformData's properties

```
uint8_t* bytes;
```

Bytes of data

```
size_t size;
```

Size in byte


```
size_t idxFirstByte;
```

Index of the first byte in transformed data

8.4.2 Struct `CapyBurrowsWheelerTransformData`'s methods

None.

8.5 Struct `CapyBurrowsWheelerTransform`

8.5.1 Struct `CapyBurrowsWheelerTransform`'s properties

None.

8.5.2 Struct `CapyBurrowsWheelerTransform`'s methods

```
void (*destruct)(void);
```

Destructor

```
CapyBurrowsWheelerTransformData (*transform)(  
    CapyBurrowsWheelerTransformData const data);
```

Transform the data

Input argument(s):

- data: the data to transform

Output and side effect(s):

- Return the transformed data

```
CapyBurrowsWheelerTransformData (*untransform)(  
    CapyBurrowsWheelerTransformData const data);
```

Untransform the data

Input argument(s):

- data: the data to untransform

Output and side effect(s):

- Return the untransformed data

8.6 Functions

```
CapyBurrowsWheelerTransform CapyBurrowsWheelerTransformCreate(void);
```

Create a CapyBurrowsWheelerTransform

Output and side effect(s):

- Return a CapyBurrowsWheelerTransform

```
CapyBurrowsWheelerTransform* CapyBurrowsWheelerTransformAlloc(void);
```

Allocate memory for a new CapyBurrowsWheelerTransform and create it

Output and side effect(s):

- Return a CapyBurrowsWheelerTransform

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyBurrowsWheelerTransformFree(CapyBurrowsWheelerTransform** const  
    that);
```

Free the memory used by a CapyBurrowsWheelerTransform* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyBurrowsWheelerTransform to free

9 Unit camera.h

Camera class. Assuming a left handed world coordinates system where +x/right, +y/up, +z/forward. All unit in meters.

9.1 Macros

```
#define CAPY_CAMERA_H
```


9.2 Enumerations

```
typedef enum CapyCameraType {  
    capyCameraType_pinhole,  
    capyCameraType_nb,  
} CapyCameraType;
```

Types of camera

9.3 Typedefs

None.

9.4 Struct CapyCamera

9.4.1 Struct CapyCamera's properties

```
CapyCameraType type;
```

Type of camera (default: pinhole)

```
CapyPad(CapyCameraType, type);
```

```
CapyVec pos;
```

Position (center of the lens, in world coordinates system, default: 0)

```
CapyMat pose;
```

Pose matrix of the camera (in world coordinates system, default: identity)
Per row: right, up, front normalised vector

```
CapyVec right;
```

Right direction from the camera (in world coordinates system, reference to the pose matrix values, default: x)

```
CapyVec up;
```

Up direction from the camera (in world coordinates system, reference to the pose matrix values, default: y)

```
CapyVec front;
```


Forward direction from the camera (in world coordinates system, reference to the pose matrix values, default: z)

```
double focalLength;
```

Focal length, the smaller the larger the field of view (default: 0.017)

```
double sensorResolution;
```

Sensor resolution (pixels/meter, default: 100000)

9.4.2 Struct CappyCamera's methods

```
void (*destruct)(void);
```

Destructor

```
void (*project)(  
    double const* const in,  
    double* const out);
```

Project a 3D point in world coordinates to a 3D point in screen coordinates

Input argument(s):

- in: the coordinates of the point to project
- out: the result projected coordinates, can be same as 'in'

Output and side effect(s):

- 'out' is updated. The projection is calculated according to the type
- of camera

```
void (*projectInv)(  
    double const* const in,  
    double* const out);
```

Project a 3D point in screen coordinates to a 3D point in world coordinates

Input argument(s):

- in: the coordinates of the point to inverse project
- out: the result inverse projected coordinates, can be same as 'in'

Output and side effect(s):

- 'out' is updated. The inverse projection is calculated according to the
- type of camera

```
void (*toImgCoord)(  
    double const* const posScreen,  
    CapyImgPos_t const* const posOrig,  
    CapyImgPos_t* const posImg);
```

Convert screen coordinates into image coordinates

Input argument(s):

- posScreen: the screen coordinates
- posOrig: the position in the image corresponding to the origin of the
- screen
- posImg: the result image coordinates

Output and side effect(s):

- 'posImg' is updated

```
void (*fromImgCoord)(  
    CapyImgPos_t* const posImg,  
    CapyImgPos_t const* const posOrig,  
    double* const out);
```

Convert image coordinates to a direction vector from the camera position

Input argument(s):

- posImg: the image coordinates
- posOrig: the position in the image corresponding to the origin of the
- screen
- out: the result direction vector

Output and side effect(s):

- 'out' is updated and normalised


```
void (*translate)(double const* const u);
```

Translate the camera

Input argument(s):

- u: the translation vector

Output and side effect(s):

- The camera position is updated

```
void (*rotate)(CapyQuaternion const* const q);
```

Rotate the camera

Input argument(s):

- q: the quaternion equivalent to the rotation

Output and side effect(s):

- The camera pose is updated

```
double (*getProjectedLength)(  
    double const* const posA,  
    double const* const posB);
```

Get the length in pixel of a projected segment

Input argument(s):

- posA: position of one end of the segment
- posB: position of the other end of the segment

Output and side effect(s):

- Return the length

9.5 Functions

```
CapyCamera CapyCameraCreate(void);
```

Create a CapyCamera

Output and side effect(s):

- Return a CapyCamera

```
CapyCamera* CapyCameraAlloc(void);
```

Allocate memory for a new CapyCamera and create it

Output and side effect(s):

- Return a CapyCamera

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyCameraFree(CapyCamera** const that);
```

Free the memory used by a CapyCamera* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyCamera to free

10 Unit capy.h

LibCapy related function and headers include.

10.1 Macros

```
#define CAPY_H
```


10.2 Enumerations

```
typedef enum Capy_BuildMode {  
    Capy_BuildMode_Development,  
    Capy_BuildMode_Production,  
} Capy_BuildMode;
```

Build modes

10.3 Typedefs

None.

10.4 Functions

```
char const* CapyGetVersion(void);
```

Getter for the version of the library

Output and side effect(s):

- Return the version as a char*

```
char const* CapyGetCommitId(void);
```

Getter for the commit id of the library

Output and side effect(s):

- Return the commit as a char*

```
Capy_BuildMode CapyGetBuildMode(void);
```

Getter for the build mode of the library

Output and side effect(s):

- Return the build mode as an enum Capy_BuildMode

11 Unit capymath.h

Mathematical functions.

11.1 Macros

```
#define CAPY_MATH_H
```

```
#define CopyVecDef(S, ...) \
    struct { \
        size_t dim; \
        union {double vals[S]; struct __attribute__((packed)){ \
            double __VA_ARGS__;};}; \
    }
```

```
#define CopyVecCreateLocal3D \
    (CopyVec){.dim = 3, .vals = (double[3]){0., 0., 0.}}
```

Macro to define a local CopyVec of dimension 3 and avoid memory allocation

```
#define CopyVecCreateLocal2D \
    (CopyVec){.dim = 2, .vals = (double[2]){0., 0.}}
```

Macro to define a local CopyVec of dimension 2 and avoid memory allocation

```
#define CopyMatCreateLocal3x3 \
    (CopyMat){ \
        .dims = {3, 3}, \
        .vals = (double[9]){0., 0., 0., 0., 0., 0., 0., 0., 0.}, \
    }
```

Macro to define a local CopyMat of dimension 3x3 and avoid memory allocation

```
#define CopyMatCreateLocal2x2 \
    (CopyMat){ \
        .dims = {2, 2}, \
        .vals = (double[4]){0., 0., 0., 0.}, \
    }
```

Macro to define a local CopyMat of dimension 2x2 and avoid memory allocation

11.2 Enumerations

None.

11.3 Typedefs

None.

11.4 Struct CopyPeasantMulDivRes

11.4.1 Struct CopyPeasantMulDivRes's properties

```
uint64_t base;
```

```
CopyRatio frac;
```

11.4.2 Struct CopyPeasantMulDivRes's methods

None.

11.5 Struct CopyVec

11.5.1 Struct CopyVec's properties

```
size_t dim;
```

```
double* vals;
```

11.5.2 Struct CopyVec's methods

None.

11.6 Struct CopyMat

11.6.1 Struct CopyMat's properties

```
double* vals;
```

```
union {
```

```
size_t dims[2];
```



```
struct __attribute__((packed)) {size_t nbCol; size_t nbRow;};
```

```
};
```

11.6.2 Struct CopyMat's methods

None.

11.7 Struct CopyFiboLattice

11.7.1 Struct CopyFiboLattice's properties

```
size_t nbPoints;
```

Number of points in the lattice

```
double* points;
```

Array of points. For grid lattice, the points are in $[0,1],[0,1]$. For polar lattice, the points are in $[0,1],[0,2\pi]$ For lattice on sphere and demi-sphere the points are in $[0,\pi],[0,2\pi]$

11.7.2 Struct CopyFiboLattice's methods

None.

11.8 Struct CopyPiecewiseGaussian

11.8.1 Struct CopyPiecewiseGaussian's properties

```
double base, amp, mean, sig1, sig2;
```

11.8.2 Struct CopyPiecewiseGaussian's methods

None.

11.9 Functions

```
uint64_t CpyGcd(  
    uint64_t a,  
    uint64_t b);
```

Get the GCD of two positive integers

Input argument(s):

- a, b: the two integers

Output and side effect(s):

- Return the greatest common divisor using the Stein's algorithm

```
int64_t CpyGcdDecomposition(  
    int64_t a,  
    int64_t b,  
    int64_t* x,  
    int64_t* y);
```

Get the GCD decomposition of two integers

Input argument(s):

- a, b: the two integers
- x, y: the result two integers

Output and side effect(s):

- Update x and y with the result of $ax+by=\text{gcd}(a,b)$ using the extended
- Euclidian algorithm, and return $\text{gcd}(a,b)$

```
uint64_t CpyGcdDecompositionUnsignedInput(  
    uint64_t a,  
    uint64_t b,  
    int64_t* x,  
    int64_t* y);
```

Get the GCD decomposition of two integers (version for positive inputs)

Input argument(s):

- a, b: the two integers
- x, y: the result two integers

Output and side effect(s):

- Update x and y with the result of $ax+by=\text{gcd}(a,b)$ using the extended
- Euclidian algorithm, and return $\text{gcd}(a,b)$

```
uint64_t CopyLcm(  
    uint64_t a,  
    uint64_t b);
```

Get the LCM of two positive integers

Input argument(s):

- a, b: the two integers

Output and side effect(s):

- Return the lowest common multiple using the formula
- $\text{lcm}(a, b) = \frac{ab}{\text{gcd}(a, b)}$

```
uint64_t CopyPeasantMul(  
    uint64_t a,  
    uint64_t b);
```

Peasant multiplication of two integers

Input argument(s):

- a, b: the two integers to multiply

Output and side effect(s):

- Return the multiplication of two integers using the Peasant method.

```
CopyPeasantMulDivRes CopyPeasantMulDiv(  
    uint64_t a,  
    uint64_t b,  
    uint64_t c);
```

```
double CopySmoothStep(  
    double const x,  
    double const a);
```

Generic smooth step function.

Input argument(s):

- x: input, in $[0.0, 1.0]$
- a: smoothing coefficient in $]0.0, +\infty]$

Output and side effect(s):

- Return the smoothed value of x. If a equals 1.0, it's x itself. As
- a gets lower than 1.0, the smoothed value varies following the
- pattern fast-slow-fast. As a gets greater than 1.0, the smoothed
- value varies following the pattern slow-fast-slow. Continuous but
- not necessary derivable at $x=0.0$ or $x=1.0$.

```
double CpySmootherStep(double const x);
```

Smoother step function Inputs: x: the input value in $[0,1]$

Output and side effect(s):

- Return the smoothed value, in $[0, 1]$

```
int64_t CpyPowi(
    int64_t const x,
    uint64_t const n);
```

Power function for an integer value and integer exponent

Input argument(s):

- x: the value
- n: the power

Output and side effect(s):

- Return x^n

```
double CpyPowf(
    double const x,
    uint64_t const n);
```

Power function for a real value and integer exponent

Input argument(s):

- x: the value
- n: the power

Output and side effect(s):

- Return x^n

```
double CopyLerp(
    double const x,
    CopyRangeDouble const* const from,
    CopyRangeDouble const* const to);
```

LERP function, map a double value linearly from a range to another

Input argument(s):

- x: the input
- from: range of the input
- to: range of the output

Output and side effect(s):

- Return the mapped input

```
double CopyLerpNorm2Arr(
    double const x,
    double const to[2]);
```

LERP function, map a double value linearly from $[0, 1]$ to an array

Input argument(s):

- x: the input
- to: array of two values

Output and side effect(s):

- Return the mapped input

```
int CopyCmpCharInc(
    char const* a,
    char const* b);
```



```
int CmpInt8Inc(  
    int8_t const* a,  
    int8_t const* b);
```

```
int CmpUInt8Inc(  
    uint8_t const* a,  
    uint8_t const* b);
```

```
int CmpInt16Inc(  
    int16_t const* a,  
    int16_t const* b);
```

```
int CmpUInt16Inc(  
    uint16_t const* a,  
    uint16_t const* b);
```

```
int CmpInt32Inc(  
    int32_t const* a,  
    int32_t const* b);
```

```
int CmpUInt32Inc(  
    uint32_t const* a,  
    uint32_t const* b);
```

```
int CmpInt64Inc(  
    int64_t const* a,  
    int64_t const* b);
```

```
int CmpUInt64Inc(  
    uint64_t const* a,  
    uint64_t const* b);
```

```
int CmpFloatInc(  
    float const* a,  
    float const* b);
```

```
int CmpDoubleInc(  
    double const* a,  
    double const* b);
```

```
int CmpSizeInc(  
    size_t const* a,  
    size_t const* b);
```



```
int CmpCharDec(  
    char const* a,  
    char const* b);
```

```
int CmpInt8Dec(  
    int8_t const* a,  
    int8_t const* b);
```

```
int CmpUInt8Dec(  
    uint8_t const* a,  
    uint8_t const* b);
```

```
int CmpInt16Dec(  
    int16_t const* a,  
    int16_t const* b);
```

```
int CmpUInt16Dec(  
    uint16_t const* a,  
    uint16_t const* b);
```

```
int CmpInt32Dec(  
    int32_t const* a,  
    int32_t const* b);
```

```
int CmpUInt32Dec(  
    uint32_t const* a,  
    uint32_t const* b);
```

```
int CmpInt64Dec(  
    int64_t const* a,  
    int64_t const* b);
```

```
int CmpUInt64Dec(  
    uint64_t const* a,  
    uint64_t const* b);
```

```
int CmpFloatDec(  
    float const* a,  
    float const* b);
```

```
int CmpDoubleDec(  
    double const* a,  
    double const* b);
```



```
CapyVec* CapyVecAllocArr(  
    size_t dim,  
    size_t nb);
```

Allocate an array of CapyVec

Input argument(s):

- dim: the dimension of the vectors
- nb: the size of the array

Output and side effect(s):

- Return a newly allocated array of CapyVec (values initialised to 0.0)
- Exceptions:
- May raise CapyExc_MallocFailed

```
void CapyVecFreeArr(  
    CapyVec** const that,  
    size_t nb);
```

Free an array of CapyVec

Input argument(s):

- that: the array of CapyVec
- nb: the size of the array

```
CapyVec CapyVecCreate(size_t dim);
```

Create a CapyVec

Input argument(s):

- dim: the dimension of the vector

Output and side effect(s):

- Return a CapyVec (values initialised to 0.0)
- Exceptions:
- May raise CapyExc_MallocFailed


```
void CpyVecDestruct(CpyVec* const that);
```

Free a CpyVec

Input argument(s):

- that: the CpyVec to free

```
void CpyVec3DGetOrtho(  
    double const* const u,  
    double* const v);
```

Get a 3D vector orthogonal to another 3D vector

Input argument(s):

- u: the input vector
- v: the output vector orthonormal to u

Output and side effect(s):

- v is updated

```
CpyMat CpyMatCreate(  
    size_t const nbCol,  
    size_t const nbRow);
```

Create a CpyMat

Input argument(s):

- nbCol: the number of columns of the matrix
- nbRow: the number of rows of the matrix

Output and side effect(s):

- Return a CpyVec (values initialised to 0.0)
- Exceptions:
- May raise CpyExc_MallocFailed

```
CpyMat CpyMatCreateRotMat(  
    size_t const iAxis,  
    double const theta);
```


Create the 3x3 CpyMat for the rotation matrix around the i-th axis and given angle (same handness for the rotation as for the coordinates system)

Input argument(s):

- iAxis: the axis index (0: x, 1: y, 2: z)
- theta: the angle in radians

Output and side effect(s):

- Return a CpyVec
- Exceptions:
- May raise CpyExc_MallocFailed, CpyExc_UndefinedExecution

```
void CpyMatDestruct(CpyMat* const that);
```

Free a CpyMat

Input argument(s):

- that: the CpyMat to free

```
void CpyVecAdd(  
    CpyVec const* const a,  
    CpyVec const* const b,  
    CpyVec* const c);
```

Add two vectors

Input argument(s):

- a: first vector
- b: second vector
- c: result vector, $c=a+b$ (c can be a or b)

```
void CpyVecSub(  
    CpyVec const* const a,  
    CpyVec const* const b,  
    CpyVec* const c);
```

Subtract two vectors

Input argument(s):

- a: first vector

- b: second vector
- c: result vector, $c=a-b$

```
void CapyVecDot(
    CapyVec const* const a,
    CapyVec const* const b,
    double* const c);
```

Dot product of two vectors

Input argument(s):

- a: first vector
- b: second vector
- c: result vector, $c=a.b$

```
void CapyVecCross(
    CapyVec const* const a,
    CapyVec const* const b,
    CapyVec* const c);
```

Cross product of two vectors of dimension 3

Input argument(s):

- a: first vector
- b: second vector
- c: result vector, $c=a*b$

```
void CapyVecMul(
    CapyVec const* const a,
    double const b,
    CapyVec* const c);
```

Product vector scalar

Input argument(s):

- a: vector
- b: scalar
- c: result vector, $c=a*b$


```
void CopyVecCopy(  
    CopyVec const* const a,  
    CopyVec* const b);
```

Copy a vector to another

Input argument(s):

- a: from vector
- b: to vector

```
void CopyVecNormalise(CopyVec* const a);
```

Normalise the vector

Input argument(s):

- a: the vector

Output and side effect(s):

- The vector is normalised.

```
void CopyVecNormaliseFast(CopyVec* const a);
```

Normalise the vector using the fast inverse square root

Input argument(s):

- a: the vector

Output and side effect(s):

- The vector is normalised.

```
double CopyVecGetNorm(CopyVec const* const a);
```

Get the norm of the vector

Input argument(s):

- a: the vector

Output and side effect(s):

- Return the norm of the vector


```
double CpyVecGetCosineSimilarity(  
    CpyVec const* const a,  
    CpyVec const* const b);
```

Get the cosine-similarity of two vectors

Input argument(s):

- a: the first vector
- b: the second vector

Output and side effect(s):

- Return the dot product of normalised vectors

```
double CpyVec2DApproxNorm(CpyVec* const u);
```

Get the approximated norm of a 2D vector

Input argument(s):

- u: the 2D vector

Output and side effect(s):

- Return the approx norm of the vector (equals to $0.96x + 0.4y$ where $x \geq y \geq 0$)
- accurate within 4

```
double CpyVec2DGetAngle(  
    CpyVec const* const u,  
    CpyVec const* const v);
```

Get the angle between two 2D vectors

Input argument(s):

- u: the first 2D vector
- v: the second 2D vector

Output and side effect(s):

- Return the angle between the 2 vectors in $[-M_PI, M_PI]$


```
void CapyVec2DSetAngle(  
    CapyVec const* const a,  
        double const b,  
    CapyVec* const c);
```

Set the angle of a 2D vector

Input argument(s):

- a: vector
- b: angle in radians
- c: result vector, $c=a*b$

Output and side effect(s):

- Set the angle to 'b' (relative to x, ccw) while conserving the distance
- of 'a' and store the result in 'c' (which can be 'a')

```
void CapyVec2DRotate(  
    CapyVec const* const a,  
        double const theta,  
    CapyVec* const b);
```

Rotate a 2D vector by a given angle

Input argument(s):

- a: vector to rotate
- theta: angle in radians
- b: result vector

Output and side effect(s):

- Rotate CCW the vector 'a' by 'theta' and store the result in 'c'
- (which can be 'a')

```
double CapyVecGetSquaredNorm(CapyVec* const a);
```

Get the squared norm of the vector

Input argument(s):

- a: the vector

Output and side effect(s):

- Return the squared norm of the vector

```
double CpyVecGetMoment(  
    CpyVec const* const u,  
    double const c,  
    uint8_t const n);
```

Get the moment of a vector's values

Input argument(s):

- u: the vector
- c: the center of the moment
- n: the order of the moment

Output and side effect(s):

- Return the moment. `CpyVecGetMoment(u, 0, 0)` is the sum of u's values, aka
- total mass. `CpyVecGetMoment(u, 0, 1)` is the mean of u's values, aka
- first raw moment. `CpyVecGetMoment(u, mean(u), 2)` is the variance of u's
- values, aka second centered moment, or square of the standard deviation
- sigma. `CpyVecGetMoment(u, mean(u), 3)` is the skewness, aka third centered
- moment. `CpyVecGetMoment(u, mean(u), 4)` is the kurtosis, aka fourth
- centered moment.
- Standardized moment of order n (aka normalized n-th central moment) is
- equal to:
- $\text{CpyVecMoment}(u, \text{mean}(u), n) / \sqrt[n]{\text{CpyVecMoment}(u, \text{mean}(u), 2)}$


```
double CpyVecQuickSelect(  
    CpyVec const* const u,  
    size_t const k);
```

Get the k-th largest number in a CpyVec (using quickselect)

Input argument(s):

- u: the vector
- k: the index of the element to find

Output and side effect(s):

- Return the k-th largest number. Can also be used to get the median value
- without sorting (CpyVecQuickSelect(u, u->dim/2))

```
double CpyVecGetMedian(CpyVec const* const u);
```

Get the median value of vector's values (using quickselect)

Input argument(s):

- u the vector

Output and side effect(s):

- Return the median value (threshold which split the vector values into
- two sets ('smaller than threshold' and 'larger than threshold') of same
- size)

```
double CpyVecGetCovariance(  
    CpyVec const* const u,  
    CpyVec const* const v);
```

Get the covariance of two vectors' values. The two vectors must have same dimension.

Input argument(s):

- u: the first vector
- v: the second vector

Output and side effect(s):

- Return the covariance, equal to $E[(u-E(u))(v-E(v))]$.

Exception(s):

- May raise `CapyExc_InvalidParameters`.

```
double CapyVecGetPearsonCorrelation(  
    CapyVec const* const u,  
    CapyVec const* const v);
```

Get the Pearson correlation of two vector's values. The two vectors must have same dimension.

Input argument(s):

- u: the first vector
- v: the second vector

Output and side effect(s):

- Return the covariance (in $[-1,1]$), equal to $\text{cov}(u, v)/(\text{sigma}(u)*\text{sigma}(v))$.

Exception(s):

- May raise `CapyExc_InvalidParameters`.

```
double CapyVecGetDistanceCovariance(  
    CapyVec const* const u,  
    CapyVec const* const v);
```

Get the distance covariance of two vector's values (seen as univariate variables). The two vectors must have same dimension.

Input argument(s):

- u: the first vector
- v: the second vector

Output and side effect(s):

- Return the distance covariance.

Exception(s):

- May raise `CapyExc_InvalidParameters`.

```
double CapyVecGetDistanceCorrelation(
    CapyVec const* const u,
    CapyVec const* const v);
```

Get the distance correlation of two vector's values (seen as univariate variables). The two vectors must have same dimension.

Input argument(s):

- u: the first vector
- v: the second vector

Output and side effect(s):

- Return the covariance (in $[0,1]$).

Exception(s):

- May raise `CapyExc_InvalidParameters`.

```
void CapyVecSoftmax(
    CapyVec* const u,
    double const t);
```

Apply the softmax function to a vector

Input argument(s):

- u: the vector
- t: 'temperature'

Output and side effect(s):

- The vector is updated. The temperature must be >0.0 . A temperature of
- 1.0 gives the standard softmax function. The higher the temperature the
- more uniformly distributed the result vector is. A temperature value
- infinitely small produces a vector with value 1.0 for the max value of

- the input, and 0.0 for all other values.

```
double CpyVecGetDistance(
    CpyVec const* const u,
    CpyVec const* const v);
```

Get the distance between two vectors. The two vectors must have same dimension.

Input argument(s):

- u: the first vector
- v: the second vector

Output and side effect(s):

- Return the norm of the difference of the two vectors.

Exception(s):

- May raise `CpyExc_InvalidParameters`.

```
void CpyVecLerp(
    CpyVec const* const start,
    CpyVec const* const end,
    CpyVec* const res,
    double const t);
```

Apply lerp to a vector components

Input argument(s):

- from: start vector
- to: end vector
- res: result vector (can same as 'start' or 'end')
- t: the coefficient (in [0,1])

Output and side effect(s):

- 'res' is set to $\text{start} + t * (\text{end} - \text{start})$


```
void CopyVecEasing(
    CopyVec const* const start,
    CopyVec const* const end,
    CopyVec* const res,
    double const t,
    CopyVecEasingFun easing);
```

Apply easing to a vector components

Input argument(s):

- from: start vector
- to: end vector
- res: result vector (can same as 'start' or 'end')
- t: the coefficient (in [0,1])
- easing: the easing function

Output and side effect(s):

- 'res' is updated

```
void CopyMatProdVec(
    CopyMat const* const a,
    CopyVec const* const b,
    CopyVec* const c);
```

Product matrix vector

Input argument(s):

- a: matrix
- b: vector
- c: result vector, $c=a*b$

```
void CopyMatTransProdVec(
    CopyMat const* const a,
    CopyVec const* const b,
    CopyVec* const c);
```

Product transposed matrix vector

Input argument(s):

- a: matrix

- b: vector
- c: result vector, $c = \hat{a} * b$

```
void CopyMatProdMat(
    CopyMat const* const a,
    CopyMat const* const b,
    CopyMat* const c);
```

Product matrix matrix

Input argument(s):

- a: matrix
- b: matrix
- c: result matrix, $c = a * b$

```
void CopyMatProdScalar(
    CopyMat const* const a,
    double const b,
    CopyMat* const c);
```

Product scalar matrix

Input argument(s):

- a: matrix
- b: scalar
- c: result matrix, $c = a * b$

```
void CopyMatAddMat(
    CopyMat const* const a,
    CopyMat const* const b,
    CopyMat* const c);
```

Add matrix matrix

Input argument(s):

- a: matrix
- b: matrix
- c: result matrix, $c = a + b$


```
void CopyMatTransp(  
    CopyMat const* const a,  
    CopyMat* const b);
```

Transpose matrix

Input argument(s):

- a: matrix
- b: result transpose matrix

```
void CopyMatDet(  
    CopyMat const* const a,  
    double* const b);
```

Get the determinant of a matrix a: matrix b: result determinant

```
void CopyMatPseudoInv(  
    CopyMat const* const a,  
    CopyMat* const b);
```

Pseudo inverse matrix (Moore-Penrose inverse)

Input argument(s):

- a: matrix
- b: result pseudo inverse matrix
- Exceptions:
- May raise CopyExc_MatrixInversionFailed, CopyExc_MallocFailed

```
void CopyMatInv(  
    CopyMat const* const a,  
    CopyMat* const b);
```

Inverse matrix (if the matrix is not square the result is the pseudo inverse)

Input argument(s):

- a: matrix
- b: result inverse matrix
- Exceptions:
- May raise CopyExc_MatrixInversionFailed, CopyExc_MallocFailed


```
void CpyMatGetQR(
    CpyMat const* const m,
    CpyMat* const q,
    CpyMat* const r);
```

Get the QR decomposition of a matrix cf <http://www.seas.ucla.edu/vandenbe/133A/lectures/qr.pdf>

Input argument(s):

- m: the matrix to decompose (nbRow \geq nbCol)
- q: the result Q matrix (same dimensions as m)
- r: the result R matrix (dimensions: m.nbCol, m.nbCol)

Exception(s):

- May raise CpyExc_QRDecompositionFailed, CpyExc_MallocFailed.

```
void CpyMatGetEigen(
    CpyMat const* const m,
    CpyVec* const eigenVal,
    CpyMat* const eigenVec);
```

Get the Eigen values and vectors of a matrix cf <http://madrury.github.io/jekyll/update/statistics/2017/10/04/qr-algorithm.html>

Input argument(s):

- m: the matrix (nbRow==nbCol)
- eigenVal: the Eigen values (from largest to smallest in absolute value, dim==m.nbCol)
- eigenVec: the Eigen vectors (same order as Eigen values, same dimension as m, one per column)

Output and side effect(s):

- eigenVec and eigenVal are updated with the result.

Exception(s):

- May raise CpyExc_QRDecompositionFailed, CpyExc_MallocFailed.


```
void CpyMatSetToIdentity(CpyMat* const m);
```

Set a matrix to the identity.

Input argument(s):

- m: the matrix (may be rectangular)

```
void CpyMatCopy(  
    CpyMat const* const src,  
    CpyMat* const dest);
```

Copy a matrix to another.

Input argument(s):

- src: the matrix to be copied
- dest: the matrix updated

```
double CpyMatGetMomentCol(  
    CpyMat const* const m,  
    size_t const iCol,  
    double const c,  
    uint8_t const n);
```

Get the moment of a column of a matrix

Input argument(s):

- u: the vector
- iCol: the column index
- c: the center of the moment
- n: the order of the moment

Output and side effect(s):

- Return the moment (cf CpyVecGetMoment)

```
uint64_t CpyMultMod(  
    uint64_t a,  
    uint64_t b,  
    uint64_t c);
```

Calculate (a*b)

Input argument(s):

- a,b,c: the value of a, b and c in $(a*b)$
- Ouput:
- Return $(a*b)$

```
uint64_t CapyPowMod(
    uint64_t a,
    uint64_t b,
    uint64_t c);
```

Calculate (a^b)

Input argument(s):

- a,b,c: the value of a, b and c in (a^b)
- Ouput:
- Return (a^b)

```
uint64_t CapyGetFibonacciNumber(uint64_t const n);
```

Return the n-th Fibonacci number.

Input argument(s):

- n: the index of the element (starting from 0)

Output and side effect(s):

- Return the n-th Fibonnaci number.

```
uint64_t* CapyFibonacciSeq(uint64_t const n);
```

Return the Fibonacci sequence up to the 'n'-th element.

Input argument(s):

- n: the number of elements

Output and side effect(s):

- Return the Fibonnaci sequence in newly allocated array.

```
uint64_t CapyFibonacciIdx(uint64_t const val);
```


Return the index in the Fibonacci sequence of the smallest value greater or equal than a given value.

Input argument(s):

- val: the value

Output and side effect(s):

- Return the index in the Fibonacci sequence. (i.e: `CapyFibonacciIdx(10)=6`)

```
void CapyFiboLatticeDestruct(CapyFiboLattice* const that);
```

Free a `CapyFiboLattice`

Input argument(s):

- that: the `CapyFiboLattice` to free

```
CapyFiboLattice CapyFibonacciGridLattice(uint64_t const n);
```

Return the Fibonacci grid lattice for the 'n'-th Fibonacci number

Input argument(s):

- n: the Fibonacci number

Output and side effect(s):

- Return the lattice.

```
CapyFiboLattice CapyFibonacciPolarLattice(uint64_t const n);
```

Return the Fibonacci polar lattice for the 'n'-th Fibonacci number

Input argument(s):

- n: the Fibonacci number

Output and side effect(s):

- Return the lattice.

```
CapyFiboLattice CapyFibonacciDemiSphereLattice(size_t const n);
```


Get the polar coordinates of n points uniformly distributed on a demi-sphere using the Fibonacci sequence

Input argument(s):

- n: the number of points

Output and side effect(s):

- Return the lattice.
- points[2i] in [0,pi] and points[2i+1] in [0,2pi]

```
CapyFiboLattice  CapyFibonacciSphereLattice(size_t const n);
```

Get the polar coordinates of n points uniformly distributed on a sphere using the Fibonacci sequence

Input argument(s):

- n: the number of points

Output and side effect(s):

- Return the lattice.
- points[2i] in [0,pi] and points[2i+1] in [0,2pi]

```
bool CapySolveQuadratic(  
    double const* const coeffs,  
    double* const roots);
```

Solve the quadratic equation $a+bx+cx^2=0$

Input argument(s):

- coeffs: the coefficients of the equation (in order a,b,...)
- roots: array of size 2 to memorise the roots

Output and side effect(s):

- Return true and update 'roots' (sorted by increasing values) if there is a solution, else return false and leave 'roots' unchanged. If there are less

- roots than the maximum possible number, the smallest root is repeated to
- fill in 'roots'.

```
bool CappySolveCubic(
    double const* const coeffs,
    double* const roots);
```

Solve the cubic equation $a+bx+cx^2+dx^3=0$

Input argument(s):

- coeffs: the coefficients of the equation (in order a,b,...)
- roots: array of size 3 to memorise the roots

Output and side effect(s):

- Return true and update 'roots' (sorted by increasing values) if there is a
- solution, else return false and leave 'roots' unchanged. If there are less
- roots than the maximum possible number, the smallest root is repeated to
- fill in 'roots'.

```
bool CappySolveQuartic(
    double const* const coeffs,
    double* const roots);
```

Solve the quartic equation $a+bx+cx^2+dx^3+ex^4=0$

Input argument(s):

- coeffs: the coefficients of the equation (in order a,b,...)
- roots: array of size 4 to memorise the roots

Output and side effect(s):

- Return true and update 'roots' (sorted by increasing values) if there is a
- solution, else return false and leave 'roots' unchanged. If there are less

- roots than the maximum possible number, the smallest root is repeated to
- fill in 'roots'.

```
float CapyFastInverseSquareRoot(float x);
```

Get the approximated inverse square root of a number using the Quake algorithm (cf https://en.wikipedia.org/wiki/Fast_inverse_square_root)

Input argument(s):

- x: the number

Output and side effect(s):

- Return $1/\sqrt{x}$.

```
float CapyFastExponential(float x);
```

Get the approximated exponential of a number using the Quake algorithm (cf <https://specbranch.com/posts/fast-exp/>)

Input argument(s):

- x: the number

Output and side effect(s):

- Return $\exp(x)$.

```
double CapyDegToRad(double const theta);
```

Convert from degree to radians

Input argument(s):

- theta: the angle in degree

Output and side effect(s):

- Return the angle in radians.

```
double CapyRadToDeg(double const theta);
```


Convert from radians to degree

Input argument(s):

- theta: the angle in radians

Output and side effect(s):

- Return the angle in degree.

```
double CpyPiecewiseGaussianEval(  
    CpyPiecewiseGaussian const* const gauss,  
    double const x);
```

Piecewise Gaussian evaluation

Input argument(s):

- x: argument of the Gaussian
- gauss: the Gaussian

Output and side effect(s):

- Return the value of the piecewise Gaussian at the requested argument

```
void CpyAckley(  
    double const* const in,  
    double* const out);
```

Ackley's function

Input argument(s):

- in: 2D input
- out: 1D output

Output and side effect(s):

- 'out' is updated. Cf https://en.wikipedia.org/wiki/Ackley_function

```
void CpyHimmelblau(  
    double const* const in,  
    double* const out);
```

Himmelblau's function

Input argument(s):

- in: 2D input
- out: 1D output

Output and side effect(s):

- 'out' is updated. Cf <https://en.wikipedia.org/wiki/Himmelblau>

```
bool CappyIsInsideEllipse(
    double const* const pos,
    double const* const center,
    double const* const dims);
```

Check if a position is inside an ellipse (aligned with coordinate system)

Input argument(s):

- pos: the position to check
- center: the center of the ellipse
- dims: the dimensions of the ellipse

Output and side effect(s):

- Return true if pos is inside the ellipse, false else

```
double CappyCubicBezierEval(
    double const t,
    double const params[4]);
```

Calculate the value of a cubic Bezier curve Inputs: t: argument of the function, in $[0, 1]$ params: the 4 control values of the Bezier

Output and side effect(s):

- Return the value of the Bezier

```
double CappyEaseQuadraticAccel(
    double const from,
    double const to,
    double const t);
```

Quadratic easing acceleration

Input argument(s):

- from: start value

- to: end value
- t: control (in [0,1])

Output and side effect(s):

- Return the eased value

```
double CopyEaseQuadraticDecel(
    double const from,
    double const to,
    double const t);
```

Quadratic easing deceleration

Input argument(s):

- from: start value
- to: end value
- t: control (in [0,1])

Output and side effect(s):

- Return the eased value

```
double CopyEaseQuadraticAccelDecel(
    double const from,
    double const to,
    double const t);
```

Quadratic easing acceleration first half deceleration second half

Input argument(s):

- from: start value
- to: end value
- t: control (in [0,1])

Output and side effect(s):

- Return the eased value


```
double CappyEaseQuadraticDecelAccel(  
    double const from,  
    double const to,  
    double const t);
```

Quadratic easing deceleration first half acceleration second half

Input argument(s):

- from: start value
- to: end value
- t: control (in [0,1])

Output and side effect(s):

- Return the eased value

```
double CappyConvAcreSquareMeter(double const areaAcre);
```

Conversion from acre to square meter

Input argument(s):

- areaAcre: the area in acre

Output and side effect(s):

- Return the area in square meter

```
double CappyConvSquareMeterAcre(double const areaSquareMeter);
```

Conversion from square meter to acre

Input argument(s):

- areaSquareMeter: the area in square meter

Output and side effect(s):

- Return the area in acre

12 Unit catmullromspline.h

CatmullRom class.

12.1 Macros

```
#define CAPY_CATMULLROMSPLINE_H
```

```
#define CAPY_CATMULLROM_MAX_DIMOUT 254
```

Maximum dimension for a CappyCatmullRom

12.2 Enumerations

None.

12.3 Typedefs

```
typedef struct CappyCatmullRomIteratorWindow CappyCatmullRomIteratorWindow;
```

Iteration windows for CappyCatmullRomIterator

12.4 Struct CappyCatmullRom

12.4.1 Struct CappyCatmullRom's properties

```
struct CappyMathFunDef;
```

Inherits CappyMathFun

```
double alpha;
```

Alpha parameter defining the type of spline (default: 0.5, in [0,1], 0: uniform, 0.5: centripetal, 1: chordal)

```
CapyVec ctrls[4];
```

Control points.

```
double knots[4];
```

Knots value (automatically updated when setting the control points)

12.4.2 Struct CappyCatmullRom's methods

```
void (*destructCapyMathFun)(void);
```

Destructor

```
CapyVec const* (*getCtrl)(size_t const id);
```

Get a control point.

Input argument(s):

- id: the index of the control point, in $[0,3]$

Output and side effect(s):

- Return the control point

```
void (*setCtrl)(
    size_t const id,
    double const* const vals);
```

Set the values of a control point

Input argument(s):

- id: the index of the control point
- vals: the value to set, array of 'that->dimOut' double

Output and side effect(s):

- The control point values and the knot values are updated.

12.5 Struct CappyCatmullRomPosition

12.5.1 Struct CappyCatmullRomPosition's properties

```
double in[1];
```

```
double out[CAPY_CATMULLROM_MAX_DIMOUT];
```

12.5.2 Struct CappyCatmullRomPosition's methods

None.

12.6 Struct CappyCatmullRomIterator

12.6.1 Struct CappyCatmullRomIterator's properties

```
size_t idx;
```

Index of the current step in the iteration

```
CappyCatmullRomPosition datatype;
```

Returned data type

```
CappyCatmullRomIteratorWindow* windows;
```

List of windows

```
CappyCatmullRom const* spline;
```

CatmullRom associated to the iteration

```
double epsilon;
```

Epsilon for the step of iteration using euclidean distance along the Catmull-Rom curve (default: 0.1)

```
CappyMemPoolCatmullRomIteratorWindow memPool;
```

Memory pool for the windows

12.6.2 Struct CappyCatmullRomIterator's methods

```
void (*destruct)(void);
```

Destructor

```
CappyCatmullRomPosition* (*reset)(void);
```

Reset the iterator

Output and side effect(s):

- Return the first position of the iteration

```
CappyCatmullRomPosition* (*next)(void);
```


Move the iterator to the next position

Output and side effect(s):

- Return the next position of the iteration

```
bool (*isActive)(void);
```

Check if the iterator is on a valid position

Output and side effect(s):

- Return true if the iterator is on a valid position, else false

```
CapyCatmullRomPosition* (*get)(void);
```

Get the current position of the iteration

Output and side effect(s):

- Return the current position

12.7 Functions

```
CapyCatmullRom CapyCatmullRomCreate(size_t const dim);
```

Create a CapyCatmullRom

Input argument(s):

- dim: dimension of the control points

Output and side effect(s):

- Return a CapyCatmullRom

```
CapyCatmullRom* CapyCatmullRomAlloc(size_t const dim);
```

Allocate memory for a new CapyCatmullRom and create it

Input argument(s):

- dim: dimension of the control points

Output and side effect(s):

- Return a CpyCatmullRom

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyCatmullRomFree(CpyCatmullRom** const that);
```

Free the memory used by a CpyCatmullRom* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyCatmullRom to free

```
CpyCatmullRomIterator CpyCatmullRomIteratorCreate(
    CpyCatmullRom const* const bezier);
```

Create an iterator on a CpyCatmullRom

Input argument(s):

- bezier: the bezier on which to iterate

Output and side effect(s):

- Return the iterator

```
CpyCatmullRomIterator* CpyCatmullRomIteratorAlloc(
    CpyCatmullRom const* const bezier);
```

Allocate memory and create an iterator on a CpyCatmullRom

Input argument(s):

- bezier: the bezier on which to iterate

Output and side effect(s):

- Return the iterator

```
void CpyCatmullRomIteratorFree(
    CpyCatmullRomIterator** const that);
```

Free the memory used by a pointer to an iterator and reset '*that' to NULL

Input argument(s):

- that: a pointer to the iterator to free

13 Unit cext.h

Functions and macros extending the C language.

13.1 Macros

```
#define CAPY_CEXT_H
```

```
#define CAPY_MACRO_TO_STR_(x) #x
```

Conversion of a macro into a char* containing its value

```
#define CAPY_MACRO_TO_STR(x) CAPY_MACRO_TO_STR_(x)
```

```
#define safeMalloc(ptr, size) \
do { \
    ptr = malloc(sizeof(*(ptr)) * (size)); \
    if(ptr == NULL) { \
        raiseExc(CapyExc_MallocFailed); \
        assert(false && "malloc failed uncaught"); \
    } \
} while(false)
```

Safe malloc, allocates 'size*sizeof(*ptr)' bytes of memory and raises CapyExc_MallocFailed if the allocation fails.

```
#define safeRealloc(ptr, size) \
do { \
    void* ptr_ = realloc(ptr, sizeof(*ptr) * (size)); \
    if(ptr_ == NULL) raiseExc(CapyExc_MallocFailed); \
    else ptr = ptr_; \
} while(false)
```

Safe realloc, raises CapyExc_MallocFailed if the reallocation fails and leaves 'ptr' unchanged if it fails

```
#define loop(varName, nbIter) \
for(__typeof__((nbIter) + 0) (varName) = 0; \
    (varName) < (nbIter); ++(varName))
```

Shortcut equivalent to a for loop iterating from 0 to (nbIter-1) and storing the current iteration index in varName

```
#define CapyPad(T, I) char padding ## I \
[ sizeof(void*) * ((sizeof(void*) + sizeof(T)) / sizeof(void*)) - sizeof(T) ]
```


Macro used to create padding fields

Input argument(s):

- T: the type of the field to padd
- I: index to differentiate several padding field in the same structure

```
#define loopRange(varName, range) \
    for(__typeof__((range).min) (varName) = (range).min; \
        (varName) <= (range).max; ++(varName))
```

```
#define println(...) \
    ((copyRetInt[0] = printf(__VA_ARGS__)) < 0 ? copyRetInt[0]: \
     ((copyRetInt[1] = printf("\n")) < 0 ? copyRetInt[1] : \
      copyRetInt[0] + copyRetInt[1]))
```

Macros equivalent to printf and fprintf where a 'n' would have been added at the end of the formatting string

```
#define fprintfln(stream, ...) \
    ((copyRetInt[0] = fprintf(stream, __VA_ARGS__)) < 0 ? copyRetInt[0]: \
     ((copyRetInt[1] = fprintf(stream, "\n")) < 0 ? copyRetInt[1] : \
      copyRetInt[0] + copyRetInt[1]))
```

```
#define methodOf(C) C* that = (C*)copyThat
```

Macro to define a class method. To be added as the beginning of a function 'f' used as a method of class 'C'.

```
#define $(instance, method) \
    ((__typeof__(instance))(copyThat = (instance)))->method
```

Macro to use OOP methods. Given a structure 'S' with a method defined as pointer to function as field 'm', an instance 's' of 'S' can execute this method with the command \$(s, m)() (where 's' must be a pointer to 'S') The function pointed to by 'm' can access the instance at the origin of the call by defining the macro #define CopyThatS struct S* that = (struct S*)copyThat and calling this macro at the head of the function, before any call to another \$(..., ...). A pointer to the instance is then available in the body of function through the variable 'that'. The macro CopyThat... must be defined for each structure using OOP methods. LibCapy provide this macro for each of the structure it defines.

```
#define $$$(instance, actor, method) \
    ((copyThat = (actor)) ? (instance)->method : NULL)
```


Same as \$(instance, method) but the value of 'that' in the execution of the method is replaced with 'actor' instead of 'instance'. Used to implement genericity.

```
#define safeFScanf(stream, format, ...) \
do { \
    int copyRet = fscanf(stream, format, __VA_ARGS__); \
    if(copyRet < 0) raiseExc(CapyExc_StreamReadError); \
} while(false)
```

Safe fscanf raising CapyExc_StreamReadError if it fails

```
#define safeFPrintf(stream, format, ...) \
do { \
    int copyRet = fprintf(stream, format, __VA_ARGS__); \
    if(copyRet < 0) raiseExc(CapyExc_StreamWriteError); \
} while(false)
```

Safe fprintf raising CapyExc_StreamWriteError if it fails

```
#define safeFRead(stream, nbByte, ptr) \
do { \
    size_t copyRet = fread(ptr, 1, nbByte, stream); \
    if(copyRet != nbByte) raiseExc(CapyExc_StreamReadError); \
} while(false)
```

Safe fread raising CapyExc_StreamReadError if it fails

```
#define safeFWrite(stream, nbByte, ptr) \
do { \
    size_t copyRet = fwrite(ptr, 1, nbByte, stream); \
    if(copyRet != nbByte) raiseExc(CapyExc_StreamWriteError); \
} while(false)
```

Safe fwrite raising CapyExc_StreamWriteError if it fails

```
#define CAPY_VA_NB_ARGS(type, ...) \
    (size_t)(sizeof((type[]){__VA_ARGS__})/sizeof(type))
```

Return the number of arguments of a variadic macro given the type 'type' of these arguments

```
#define equal(a, b) _Generic(a, \
    float: equalf, \
    float const: equalf, \
    double: equald, \
    double const: equald)(a, b)
```

Equality operator for float and double types.

Input argument(s):

- a,b: the values to compare

Output and side effect(s):

- Return true if the values are considered
- equal using the ULP method described here:
- [https://randomascii.wordpress.com/2012/02/25/](https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/)
- [comparing-floating-point-numbers-2012-edition/](https://randomascii.wordpress.com/2012/02/25/comparing-floating-point-numbers-2012-edition/)

```
#define min(arr, size) _Generic(arr, \
    char*: min_char, \
    char const*: min_char, \
    int8_t*: min_int8_t, \
    int8_t const*: min_int8_t, \
    uint8_t*: min_int8_t, \
    uint8_t const*: min_int8_t, \
    int32_t*: min_int32_t, \
    int32_t const*: min_int32_t, \
    uint32_t*: min_int32_t, \
    uint32_t const*: min_int32_t, \
    int64_t*: min_int64_t, \
    int64_t const*: min_int64_t, \
    uint64_t*: min_int64_t, \
    uint64_t const*: min_int64_t, \
    float*: min_float, \
    float const*: min_float, \
    double*: min_double, \
    double const*: min_double)(arr, size)
```

Get the minimum value in a array of base type values

Input argument(s):

- arr: the array
- size: the size of the array

Output and side effect(s):

- Return the minimum value

```
#define CopyMinDec(type) type min_ ## type(type const* const arr, size_t \
    size);
```



```
#define max(arr, size) _Generic(arr, \
    char*: max_char, \
    char const*: max_char, \
    int8_t*: max_int8_t, \
    int8_t const*: max_int8_t, \
    uint8_t*: max_int8_t, \
    uint8_t const*: max_int8_t, \
    int32_t*: max_int32_t, \
    int32_t const*: max_int32_t, \
    uint32_t*: max_int32_t, \
    uint32_t const*: max_int32_t, \
    int64_t*: max_int64_t, \
    int64_t const*: max_int64_t, \
    uint64_t*: max_int64_t, \
    uint64_t const*: max_int64_t, \
    float*: max_float, \
    float const*: max_float, \
    double*: max_double, \
    double const*: max_double)(arr, size)
```

Get the maximum value in a array of base type values

Input argument(s):

- arr: the array
- size: the size of the array

Output and side effect(s):

- Return the maximum value

```
#define CopyMaxDec(type) type max_ ## type(type const* const arr, size_t \
    size);
```

```
#define iMin(arr, size) _Generic(arr, \
    char*: iMin_char, \
    char const*: iMin_char, \
    int8_t*: iMin_int8_t, \
    int8_t const*: iMin_int8_t, \
    uint8_t*: iMin_int8_t, \
    uint8_t const*: iMin_int8_t, \
    int32_t*: iMin_int32_t, \
    int32_t const*: iMin_int32_t, \
    uint32_t*: iMin_int32_t, \
    uint32_t const*: iMin_int32_t, \
    int64_t*: iMin_int64_t, \
    int64_t const*: iMin_int64_t, \
    uint64_t*: iMin_int64_t, \
    uint64_t const*: iMin_int64_t, \
    float*: iMin_float, \
    float const*: iMin_float, \
    double*: iMin_double, \
    double const*: iMin_double)(arr, size)
```


Get the index of the minimum value in a array of base type values

Input argument(s):

- arr: the array
- size: the size of the array

Output and side effect(s):

- Return the minimum value

```
#define CopyIMinDec(type) \
    size_t iMin_ ## type(type const* const arr, size_t size);
```

```
#define iMax(arr, size) _Generic(arr, \
    char*: iMax_char, \
    char const*: iMax_char, \
    int8_t*: iMax_int8_t, \
    int8_t const*: iMax_int8_t, \
    uint8_t*: iMax_int8_t, \
    uint8_t const*: iMax_int8_t, \
    int32_t*: iMax_int32_t, \
    int32_t const*: iMax_int32_t, \
    uint32_t*: iMax_int32_t, \
    uint32_t const*: iMax_int32_t, \
    int64_t*: iMax_int64_t, \
    int64_t const*: iMax_int64_t, \
    uint64_t*: iMax_int64_t, \
    uint64_t const*: iMax_int64_t, \
    float*: iMax_float, \
    float const*: iMax_float, \
    double*: iMax_double, \
    double const*: iMax_double)(arr, size)
```

Get the index of the maximum value in a array of base type values

Input argument(s):

- arr: the array
- size: the size of the array

Output and side effect(s):

- Return the maximum value

```
#define CopyiMaxDec(type) \
    size_t iMax_ ## type(type const* const arr, size_t size);
```



```

#define foreach(elem, iterator)
\
for(
\
  __typeof__((iterator).datatype) *(elem ## Ptr) =
\
    $(&(iterator), reset>(), elem =
\
      (elem ## Ptr ? *elem ## Ptr : (&(iterator))->datatype);
\
    $(&(iterator), isActive());
\
  elem ## Ptr = $(&(iterator), next>(),
\
    elem = (elem ## Ptr ? *elem ## Ptr : elem))

```

for loop using the 'iterator' to iterates on each 'elem' of the iterator's associated container. A copy of the value in the container can be accessed with 'elem', and the adress of the value with 'elem'Ptr.

```

#define CopyInherits(Instance, Parent, Args) \
  Parent copyParent = Parent ## Create Args; \
  memcpy(&(Instance), &copyParent, sizeof(Parent)); \
  (Instance).destruct ## Parent = (Instance).destruct

```

Inheritance operator. Use it in the 'Create' function of the inheriting class to initialise the 'Parent' properties and methods in 'Instance'. 'Args' are arguments given to the parent 'Create' function (inside parenthesis)

```

#define issigned(t) (((t)(-1)) < ((t) 0))

```

Get teh maximum possible value for an integer type

```

#define umaxof(t) (((0x1ULL << ((sizeof(t) * 8ULL) - 1ULL)) - 1ULL) | \
  (0xFULL << ((sizeof(t) * 8ULL) - 4ULL)))

```

```

#define smaxof(t) (((0x1ULL << ((sizeof(t) * 8ULL) - 1ULL)) - 1ULL) | \
  (0x7ULL << ((sizeof(t) * 8ULL) - 4ULL)))

```

```

#define maxof(t) ((unsigned long long) (issigned(t) ? smaxof(t) : umaxof(t))
)

```

```

#define containerOf(ptr, containerType, fieldName) \
  (containerType*)((char*)(ptr) - offsetof(containerType, fieldName))

```

Get a pointer to the instance of the structure containing 'ptr' given its a structure of type 'containerType' and 'ptr' points to the 'fieldName' field in that structure.

13.2 Enumerations

None.

13.3 Typedefs

```
typedef struct sigaction Sigaction;
```

Typedef for struct sigaction to comply with CBo

13.4 Functions

```
FILE* safeFOpen(  
    char const* const pathname,  
    char const* const mode);
```

Safe fopen raising CopyExc_StreamOpenError if it fails

```
void safeSprintf(  
    char** const str,  
    char const* const fmt,  
    ...);
```

Safe sprintf allocating memory as necessary for the result string and raising CopyExc_MallocFailed if the memory allocation failed

Input argument(s):

- str: pointer to the result string
- fmt: format as in sprintf
- ...: arguments as in sprintf

```
bool equalf(  
    float const a,  
    float const b);
```

```
bool equald(  
    double const a,  
    double const b);
```

```
char* strCreate(  
    char const* fmt,  
    ...);
```


Clone of asprintf

Input argument(s):

- fmt: format as in sprintf
- ...: arguments as in sprintf

Output and side effect(s):

- Return a newly allocated string

Exception(s):

- May raise CpyExc_MallocFailed

```
int CpySleepMs(int32_t const delayMs);
```

Sleep for a given amount of time in milliseconds

Input argument(s):

- delayMs: delay in milliseconds

Output and side effect(s):

- Return -1 if the sleep has been interrupted by an interruption, else 0

```
bool CpyIsAccessibleMem(  
    void const* const ptr,  
    size_t const nbByte);
```

Check if an address is inside the currently accessible address space

Input argument(s):

- ptr: the address to check
- nbByte: the number of bytes checked from that adress, if equal to 0 uses
uses
- 1 byte instead

Output and side effect(s):

- Return true if the 'nbByte' bytes from 'ptr' are in the accessible
address space, else false


```
long CapyGetMemUsed(void);
```

Get the quantity of memory currently used by the process calling this function.

Output and side effect(s):

- Return the quantity of memory in bytes. May return 0 if the quantity
- of used memory couldn't be measured.

```
void CapyChildProcEndsWithoutWait(void);
```

Avoid child process to become zombies and wait until their parent's wait() call. This applies to *all* child processes.

```
bool CapyIsBigEndian(void);
```

Check if the architecture is big endian

Output and side effect(s):

- Return true if the architecture is big endian, else false. If using
- gcc one can also use:
- `__BYTE_ORDER__ == __ORDER_BIG_ENDIAN__`

14 Unit chrono.h

Class to make time measurement.

14.1 Macros

```
#define CAPY_CHRONO_H
```


14.2 Enumerations

```
typedef enum CapyChronoUnit {  
    capyChrono_day,  
    capyChrono_hour,  
    capyChrono_minute,  
    capyChrono_second,  
    capyChrono_millisecond,  
    capyChrono_nbUnit,  
} CapyChronoUnit;
```

Enumeration for the units of the elapsed time

14.3 Typedefs

```
typedef int64_t CapyChronoTime_t;
```

Type for the time values of a CapyChrono

14.4 Struct CapyChrono

14.4.1 Struct CapyChrono's properties

```
CapyChronoTime_t elapsedTime[capyChrono_nbUnit];
```

Elapsed time divided into each unit component. For example 1.2s is [0,0,0,1,200]

```
CapyPad(CapyChronoTime_t, elapsedTime);
```

```
struct timespec startTime;
```

Start and stop times

```
struct timespec stopTime;
```

14.4.2 Struct CapyChrono's methods

```
void (*destruct)(void);
```

Destructor


```
void (*start)(void);
```

Start the timer

```
void (*stop)(void);
```

Stop the timer

```
double (*getElapsedTime)(CapyChronoUnit unit);
```

Get the elapsed time in a given unit between

Input argument(s):

- unit: the unit of the returned value

Output and side effect(s):

- Return the elapsed time between the last call to start() and the last
- call to stop()

```
double (*getElapsedTimeBestUnit)(CapyChronoUnit* const unit);
```

Get the elapsed time in the most convenient unit

Input argument(s):

- unit: a pointer to the chosen unit

Output and side effect(s):

- Return the elapsed time between the last call to start() and the last
- call to stop() into the largest unit such as the returned time is greater
- than 1.0

```
void (*getTimeStamp)(char* const buffer);
```

Get the current date and time in format yyymmddhhiiss

Input argument(s):

- buffer: buffer updated with the result

Output and side effect(s):

- buffer is updated.

14.5 Functions

```
CapyChrono CapyChronoCreate(void);
```

Create a CapyChrono

Output and side effect(s):

- Return a CapyChrono

```
CapyChrono* CapyChronoAlloc(void);
```

Allocate memory for a new CapyChrono and create it

Output and side effect(s):

- Return a CapyChrono

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyChronoFree(CapyChrono** const that);
```

Free the memory used by a CapyChrono* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyChrono to free

15 Unit collisionDetection.h

Collision detection between geometries.

15.1 Macros

```
#define CAPY_COLLISIONDETECTION_H
```

15.2 Enumerations

None.

15.3 Typedefs

None.

15.4 Struct CappyCollisionDetection

15.4.1 Struct CappyCollisionDetection's properties

None.

15.4.2 Struct CappyCollisionDetection's methods

```
void (*destruct)(void);
```

Destructor

```
bool (*isPointCollidingCircle)(  
    CappyPoint2D const* const point,  
    CappyCircle const* const circle);
```

Check if a point collides with a circle.

Input argument(s):

- point: the point
- circle: the circle

Output and side effect(s):

- Return true if there is collision, else false.

```
bool (*isCircleCollidingCircle)(  
    CappyCircle const* const circleA,  
    CappyCircle const* const circleB);
```

Check if a circle collides with a circle.

Input argument(s):

- circleA: the first circle
- circleB: the second circle

Output and side effect(s):

- Return true if there is collision, else false.


```
bool (*isPointCollidingRectangle)(
    CappyPoint2D const* const point,
    CappyRectangle const* const rect);
```

Check if a point collides with a rectangle.

Input argument(s):

- point: the point
- rect: the rectangle

Output and side effect(s):

- Return true if there is collision, else false.

```
bool (*isRectangleCollidingRectangle)(
    CappyRectangle const* const rectA,
    CappyRectangle const* const rectB);
```

Check if a rectangle collides with a rectangle.

Input argument(s):

- rectA: the first rectangle
- rectB: the second rectangle

Output and side effect(s):

- Return true if there is collision, else false.

15.5 Functions

```
CappyCollisionDetection CappyCollisionDetectionCreate(void);
```

Create a CappyCollisionDetection

Output and side effect(s):

- Return a CappyCollisionDetection

```
CappyCollisionDetection* CappyCollisionDetectionAlloc(void);
```

Allocate memory for a new CappyCollisionDetection and create it

Output and side effect(s):

- Return a CpyCollisionDetection

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyCollisionDetectionFree(CpyCollisionDetection** const that);
```

Free the memory used by a CpyCollisionDetection* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyCollisionDetection to free

16 Unit colorChart.h

Class to manipulate color charts.

16.1 Macros

```
#define CAPY_COLORCHART_H
```

```
#define CpyColorChartDimDef union { \
    size_t dims[2]; \
    struct __attribute__((packed)) { \
        size_t nbRow; \
        size_t nbCol; \
    }; \
}
```

Definition of ColorChartDim class

16.2 Enumerations

```
typedef enum CpyColorChartType {
    capyColorChart_QP203,
    capyColorChart_XRiteClassic,
} CpyColorChartType;
```

Types of color chart

16.3 Typedefs

```
typedef CapyColorChartDimDef CapyColorChartDim;
```

ColorChartDim class

```
typedef struct CapyColorChart CapyColorChart;
```

Predeclaration of ColorChart class

16.4 Struct CapyColorChartDetectOpt

16.4.1 Struct CapyColorChartDetectOpt's properties

```
double stdDev;
```

Standard deviation and size of the Gauss kernel

```
CapyImgDims_t gaussKernelSize;
```

```
CapyPad(CapyImgDims_t, gaussKernelSize);
```

```
bool flagScoreGradient;
```

Use the gradient of colors in the scoring function

```
CapyPad(bool, flagScoreGradient);
```

```
double thresholdAspectRatio;
```

Threshold for the aspect ratio of patches

```
double thresholdCoverage;
```

Threshold for the coverage of patches

```
double thresholdParallelism;
```

Threshold for the parallelism of patches

```
double thresholdPairing;
```


Threshold for the pairing of patches

```
size_t nbStepDiffusion;
```

Number of step for score diffusion

```
double hysteresis[2];
```

Thresholds for the hysteresis step in Canny edge detection

```
bool flagKeepPairing;
```

Flag to memorise if we keep pairing after an island of patches has been completed

```
CapyPad(bool, 2);
```

```
bool flagKmeansClustering;
```

Flag to run k-means clustering prior to canny edge detection

```
CapyPad(bool, 3);
```

16.4.2 Struct CapyColorChartDetectOpt's methods

None.

16.5 Struct CapyColorChart

16.5.1 Struct CapyColorChart's properties

```
CapyColorChartType type;
```

Type of the chart

```
CapyPad(CapyColorChartType, type);
```

```
CapyColorChartDimDef;
```

Dimensions of the chart


```
CapyColorData* colors;
```

Colors of the chart (ordered as $[iRow * nbCol + iCol]$)

```
bool* isInsideSRBGGamut;
```

Array of flags to memorise which colors are inside the sRGB gamut.

```
size_t kernelSize;
```

Kernel size in pixel to blur the image when extracting the color values (default: 3)

```
CapyColorSpace colorSpace;
```

Color space of the chart

```
CapyColorIlluminant illuminant;
```

Illuminant to interpret the color chart values (default D65)

16.5.2 Struct CapyColorChart's methods

```
void (*destruct)(void);
```

Destructor

```
CapyColorData (*get)(  
    size_t const iRow,  
    size_t const iCol);
```

Get the color at a given position in the chart

```
CapyQuadrilateral (*locate)(  
    CapyImg const* const img,  
    CapyColorChartDetectOpt const* const opt);
```

Locate a color chart in an image

Input argument(s):

- img: the image containing the chart
- opt: the detection options

Output and side effect(s):

- Return the location of the color chart, clockwise from the
- top-left corner.
- Exceptions:
- May raise `CapyExc_NoColorChartInImage` if no color chart were located.

```
bool (*extract)(
    CapyImg const* const img,
    CapyQuadrilateral const* chartCoord);
```

Update the color values of the chart with those extracted from the color chart contained in an image

Input argument(s):

- `img`: the image containing the color chart
- `coords`: the four coordinates of the corners of the QP203 in the
- image, as returned by `locate()`. If `NULL`, an automatic
- localisation of the color chart in the image is performed
- (to be implemented).

Output and side effect(s):

- Return true if the colors could be extracted, else false.

```
void (*convertToColorSpace)(CapyColorSpace const colorSpace);
```

Convert the color chart to another color space

Input argument(s):

- `colorSpace`: the target color space

```
void (*matchBrightness)(CapyColorChart const* const refChart);
```

Correct the color chart to match the brightness of a given color chart

Input argument(s):

- `refChart`: the target color chart


```
void (*saveToPath)(char const* const path);
```

Save the color chart to a path. Format: type, nbRow, nbCol, colors[0].RGBA[0], colors[0].RGBA[1], colors[0].RGBA[2], colors[0].RGBA[3], colors[1].RGBA[0],

...

Input argument(s):

- path: the path
- Exceptions:
- May raise CpyExc_StreamOpenError, CpyExc_StreamWriteError

```
void (*saveToStream)(CapyStreamIo* const stream);
```

Save the color chart to a stream (in binary mode). Format: type, nbRow, nbCol, colors[0].RGBA[0], colors[0].RGBA[1], colors[0].RGBA[2], colors[0].RGBA[3], colors[1].RGBA[0], ...

Input argument(s):

- stream: the stream
- Exceptions:
- May raise CpyExc_StreamWriteError

```
void (*loadFromPath)(char const* const path);
```

Load the color chart from a path. Format: type, nbRow, nbCol, colors[0].RGBA[0], colors[0].RGBA[1], colors[0].RGBA[2], colors[0].RGBA[3], colors[1].RGBA[0],

...

Input argument(s):

- path: the path
- Exceptions:
- May raise CpyExc_StreamOpenError, CpyExc_StreamReadError

```
void (*loadFromStream)(CapyStreamIo* const stream);
```


Load the color chart from a stream (in binary mode). Format: type, nbRow, nbCol, colors[0].RGBA[0], colors[0].RGBA[1], colors[0].RGBA[2], colors[0].RGBA[3], colors[1].RGBA[0], ...

Input argument(s):

- stream: the stream
- Exceptions:
- May raise CpyExc_StreamWriteError, CpyExc_MallocFailed

```
double (*getSimilarity)(CapyColorChart const* const chart);
```

Get the degree of similarity of the color chart relative to another chart.

Input argument(s):

- chart: the chart to compare to

Output and side effect(s):

- Return 1.0 if the two charts are identical. Return 0.0 if the
- two charts are of different types. Return 1.0 minus the average
- of Euclidean distances between two same patches in the charts,
- hence the lower the returned value (possibly negative) the more
- charts' color differ. If the two charts are in different color
- space, a copy of 'chart' is converted to the color space of
- 'that' and used instead.

```
CapyImgPos (*getPatchPos)(  
    CapyQuadrilateral const* const chartCoord,  
    size_t const iCol,  
    size_t const iRow);
```

Get the position of the center of a patch in an image given the position of the corners of the color chart.

Input argument(s):

- coords: the four coordinates of the corners of the QP203 in the
- image, as returned by locate().
- iCol, iRow: the coordinates of the patch in the chart

Output and side effect(s):

- Return the position in the image of the center of the patch

16.6 Functions

```
CapyColorChart CapyColorChartCreate(  
    CapyColorChartType const type,  
    CapyColorSpace const colorSpace);
```

Create a CapyColorChart

Input argument(s):

- type: the type of the chart
- colorSpace: the color space of the chart

Output and side effect(s):

- Return a CapyColorChart

Exception(s):

- May raise CapyExc_MallocFailed, CapyExc_UndefinedExecution.

```
CapyColorChart* CapyColorChartAlloc(  
    CapyColorChartType const type,  
    CapyColorSpace const colorSpace);
```

Allocate memory for a new CapyColorChart and create it

Input argument(s):

- type: the type of the chart
- colorSpace: the color space of the chart

Output and side effect(s):

- Return a CapyColorChart

Exception(s):

- May raise CapyExc_MallocFailed, CapyExc_UndefinedExecution.

```
CapyColorChart* CapyColorChartClone(CapyColorChart const* const chart);
```

Clone a CapyColorChart

Input argument(s):

- chart: the color chart to clone

Output and side effect(s):

- Return a CpyColorChart

Exception(s):

- May raise CpyExc_MallocFailed, CpyExc_UndefinedExecution.

```
void CpyColorChartFree(CpyColorChart** const that);
```

Free the memory used by a CpyColorChart* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyColorChart to free

17 Unit colorCorrectionBezier.h

Color correction using bezier surface approximation.

17.1 Macros

```
#define CAPY_COLORCORRBEZIER_H
```

17.2 Enumerations

None.

17.3 Typedefs

None.

17.4 Struct CpyColorCorrBezier

17.4.1 Struct CpyColorCorrBezier's properties

```
CpyColorCorrDef;
```


Inherits CpyColorCorr

```
CpyBezier* bezier;
```

Bezier surface used to perform the correction

```
CpyBezierOrder_t order;
```

Order of the bezier surface, if it's equal to 0 when match() is called the best order is automatically calculated (default value: 0)

```
CpyPad(CpyBezierOrder_t, order);
```

```
size_t nbAnchorPerAxis;
```

Number of anchor per axis to avoid divergence (default: 3)

17.4.2 Struct CpyColorCorrBezier's methods

```
void (*destructCpyColorCorr)(void);
```

Destructor

17.5 Functions

```
CpyColorCorrBezier CpyColorCorrBezierCreate(void);
```

Create a CpyColorCorrBezier
Output and side effect(s):

- Return a CpyColorCorrBezier

```
CpyColorCorrBezier* CpyColorCorrBezierAlloc(void);
```

Allocate memory for a new CpyColorCorrBezier and create it
Output and side effect(s):

- Return a CpyColorCorrBezier

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyColorCorrBezierFree(CpyColorCorrBezier** const that);
```

Free the memory used by a CpyColorCorrBezier* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpyColorCorrBezier to free

18 Unit colorCorrection.h

Virtual parent class for color correction classes.

18.1 Macros

```
#define CAPY_COLORCORR_H
```

```
#define CpyColorCorrDef struct {
    double initialFitness, finalFitness;
    CpyColorSpace colorSpace;
    CpyPad(CpyColorSpace, 0);
    void (*destruct)(void);
    void (*match)(
        CpyColorChart const* const chart,
        CpyColorChart const* const refChart);
    void (*apply)(CpyImg* const img);
    void (*saveToPath)(char const* const path);
    void (*saveToStream)(CpyStreamIo* const stream);
    void (*loadFromPath)(char const* const path);
    void (*loadFromStream)(CpyStreamIo* const stream);
}
```

CpyColorCorr object definition macro. Initial and final fitness to evaluate the performance of the correction. Updated by the match() method. double initialFitness, finalFitness; The color space in which the color correction is operating (default: sRGB) CpyColorSpace colorSpace; Find the correction to convert from a given color chart to a reference color chart

Input argument(s):

- chart: the original color chart
- refChart: the reference color chart

Output and side effect(s):

- Update the initialFitness and finalFitness properties.
- void (*match)(
- CpyColorChart const* const chart,
- CpyColorChart const* const refChart);
- Apply the color correction to an image.

Input argument(s):

- img: the image to be corrected

Output and side effect(s):

- The image is corrected.
- void (*apply)(CpyImg* const img);
- Save the corrector to a path.

Input argument(s):

- path: the path
- Exceptions:
- May raise CpyExc_StreamOpenError, CpyExc_StreamWriteError
- void (*saveToPath)(char const* const path);
- Save the corrector to a stream (in binary mode).

Input argument(s):

- stream: the stream
- Exceptions:
- May raise CpyExc_StreamWriteError
- void (*saveToStream)(CpyStreamIo* const stream);
- Load the corrector from a path.

Input argument(s):

- path: the path
- Exceptions:
- May raise CpyExc_StreamOpenError, CpyExc_StreamReadError
- void (*loadFromPath)(char const* const path);
- Load the corrector from a stream (in binary mode).

Input argument(s):

- stream: the stream
- Exceptions:
- May raise CpyExc_StreamWriteError, CpyExc_MallocFailed
- oid (*loadFromStream)(CpyStreamIo* const stream);

18.2 Enumerations

None.

18.3 Typedefs

```
typedef CpyColorCorrDef CpyColorCorr;
```

CpyColorCorr object

18.4 Functions

```
CpyColorCorr CpyColorCorrCreate(void);
```

Create a CpyColorCorr

Output and side effect(s):

- Return a CpyColorCorr

19 Unit colorCorrectionMatrix.h

Color correction matrix algorithm.

19.1 Macros

```
#define CAPY_COLORCORRMAT_H
```

19.2 Enumerations

None.

19.3 Typedefs

None.

19.4 Struct CapyColorCorrMat

19.4.1 Struct CapyColorCorrMat's properties

```
CapyColorCorrDef;
```

Inherits CapyColorCorr

```
double mat[9];
```

Correction matrix (values ordered as [iRow*3+iCol])

```
double thresholdFitness;
```

Thresholds for the optimiser, default values are -0.001 and 100000

```
size_t thresholdIter;
```

19.4.2 Struct CapyColorCorrMat's methods

```
void (*destructCapyColorCorr)(void);
```

Destructor

19.5 Functions

```
CapyColorCorrMat CapyColorCorrMatCreate(void);
```

Create a CapyColorCorrMat

Output and side effect(s):

- Return a CapyColorCorrMat

```
CapyColorCorrMat* CapyColorCorrMatAlloc(void);
```

Allocate memory for a new CapyColorCorrMat and create it

Output and side effect(s):

- Return a CapyColorCorrMat

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyColorCorrMatFree(CapyColorCorrMat** const that);
```

Free the memory used by a CapyColorCorrMat* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyColorCorrMat to free

20 Unit color.h

Color class.

20.1 Macros

```
#define CAPY_COLOR_H
```


20.2 Enumerations

```
typedef enum {  
    capyColorSpace_sRGB,  
    capyColorSpace_HSV,  
    capyColorSpace_XYZ,  
    capyColorSpace_LAB,  
    capyColorSpace_rgb,  
    capyColorSpace_l1l2l3,  
    capyColorSpace_c1c2c3,  
    capyColorSpace_Oklab,  
    capyColorSpace_nb,  
} CapyColorSpace;
```

Color spaces

```
typedef enum {  
    capyColorIlluminant_D50, // 2 degrees  
    capyColorIlluminant_D65, // 2 degrees  
} CapyColorIlluminant;
```

Color illuminant

20.3 Typedefs

```
typedef double CapyColorValue_t;
```

Type for the color values

20.4 Struct CapyColor

20.4.1 Struct CapyColor's properties

```
CapyColorData vals;
```

Color values

```
CapyColorIlluminant illuminant;
```

Color illuminant for conversion (by default D65)

```
CapyPad(CapyColorIlluminant, 0);
```


20.4.2 Struct CpyColor's methods

```
void (*destruct)(void);
```

Destructor

```
CpyColorData (*RGB2HSV)(void);
```

Convert the color from RGB to HSV

Output and side effect(s):

- Return the converted color, values in $[0,1]$
- Can be used with $$$(..., \text{CpyColorData}^*, ...)$

```
CpyColorData (*HSV2RGB)(void);
```

Convert the color from HSV to RGB

Output and side effect(s):

- Return the converted color, values in $[0,1]$
- Can be used with $$$(..., \text{CpyColorData}^*, ...)$

```
CpyColorData (*RGB2XYZ)(void);
```

Convert the color from RGB (in $[0,1]$) to XYZ (in $[0,1]$)

Output and side effect(s):

- Return the converted color
- Can be used with $$$(..., \text{CpyColorData}^*, ...)$

```
CpyColorData (*XYZ2RGB)(void);
```

Convert the color from XYZ (in $[0,1]$) to RGB (in $[0,1]$)

Output and side effect(s):

- Return the converted color, values in $[0,1]$
- Can be used with $$$(..., \text{CpyColorData}^*, ...)$


```
CapyColorData (*LAB2XYZ)(void);
```

Convert the color from LAB to XYZ

Output and side effect(s):

- Return the converted color
- Can be used with `$$(..., CapyColorData*, ...)`

```
CapyColorData (*Oklab2XYZ)(void);
```

Convert the color from Oklab to XYZ (D65 white point)

Output and side effect(s):

- Return the converted color
- Can be used with `$$(..., CapyColorData*, ...)`

```
CapyColorData (*XYZ2LAB)(void);
```

Convert the color from XYZ to LAB

Output and side effect(s):

- Return the converted color, values in $[0,1]$
- Can be used with `$$(..., CapyColorData*, ...)`

```
CapyColorData (*XYZ2Oklab)(void);
```

Convert the color from XYZ (D65 white point) to Oklab

Output and side effect(s):

- Return the converted color
- Can be used with `$$(..., CapyColorData*, ...)`

```
CapyColorData (*RGB2rgb)(void);
```

Convert the color from RGB to rgb

Output and side effect(s):

- Return the converted color
- Can be used with `$$(..., CpyColorData*, ...)`

```
CpyColorData (*RGB2l1l2l3)(void);
```

Convert the color from RGB to l1l2l3

Output and side effect(s):

- Return the converted color
- Can be used with `$$(..., CpyColorData*, ...)`

```
CpyColorData (*RGB2c1c2c3)(void);
```

Convert the color from RGB to c1c2c3

Output and side effect(s):

- Return the converted color
- Can be used with `$$(..., CpyColorData*, ...)`

```
bool (*equalsRGB)(CpyColorData const* const color);
```

Check if that is equal to 'color'

Input argument(s):

- color: the color to be compared with

Output and side effect(s):

- Return true if the colors are identical, else false
- Can be used with `$$(..., CpyColorData*, ...)`

```
bool (*equalsRGBA)(CpyColorData const* const color);
```

```
bool (*equalsHSV)(CpyColorData const* const color);
```

```
bool (*equalsXYZ)(CpyColorData const* const color);
```



```
bool (*equalsLAB)(CapyColorData const* const color);
```

```
bool (*equalsrgb)(CapyColorData const* const color);
```

```
bool (*equalsl1l2l3)(CapyColorData const* const color);
```

```
bool (*equalsc1c2c3)(CapyColorData const* const color);
```

20.5 Struct CapyColorPalette

20.5.1 Struct CapyColorPalette's properties

```
size_t size;
```

Number of colors in the palette

```
CapyColorData* colors;
```

Colors in the palette

20.5.2 Struct CapyColorPalette's methods

```
void (*destruct)(void);
```

Destructor

```
void (*set)(  
    size_t const iColor,  
    CapyColorData const color);
```

Set a color in the palette

Input argument(s):

- iColor: the index of the color
- color: the color to set

Output and side effect(s):

- The color is set.


```
CapyColorData (*get)(size_t const iColor);
```

Get a color in the palette

Input argument(s):

- iColor: the index of the color

Output and side effect(s):

- Return the color.

```
CapyColorData (*getInterpolated)(double const x);
```

Get a color interpolated from the colors of the palette

Input argument(s):

- x: interpolation value in [0,that->size-1]

Output and side effect(s):

- Return the color.

```
void (*setRandomRGB)(CapyRandom* const rng);
```

Set random colors in the palette

Input argument(s):

- rng: the RNG to be used

Output and side effect(s):

- The RGB colors are set.

```
void (*setRandomPastelRGB)(CapyRandom* const rng);
```

Set random pastel colors in the palette

Input argument(s):

- rng: the RNG to be used

Output and side effect(s):

- The RGB colors are set.

20.6 Functions

```
CapyColor CapyColorCreate(CapyColorData const color);
```

Create a CapyColor

Input argument(s):

- color: the color values

Output and side effect(s):

- Return a CapyColor

```
CapyColor* CapyColorAlloc(CapyColorData const color);
```

Allocate memory for a new CapyColor and create it

Input argument(s):

- color: the color values

Output and side effect(s):

- Return a CapyColor

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyColorFree(CapyColor** const that);
```

Free the memory used by a CapyColor* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyColor to free

```
CapyColorPalette CapyColorPaletteCreate(size_t const size);
```

Create a CapyColorPalette

Input argument(s):

- size: the number of colors in the palette

Output and side effect(s):

- Return a CpyColorPalette, all colors initialised to capyColorRGBABlack

```
CpyColorPalette* CpyColorPaletteAlloc(size_t const size);
```

Allocate memory for a new CpyColorPalette and create it

Input argument(s):

- size: the number of colors in the palette

Output and side effect(s):

- Return a CpyColorPalette, all colors initialised to capyColorRGBABlack

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyColorPaletteFree(CpyColorPalette** const that);
```

Free the memory used by a CpyColorPalette* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyColorPalette to free

21 Unit colorHisto.h

Class to manipulate image's color histogram.

21.1 Macros

```
#define CAPY_COLORHISTO_H
```

```
#define CAPY_COLORHISTO_NBBUCKETS 256
```

Number of buckets in the histogram

21.2 Enumerations

```
typedef enum CapyColorHistoType {  
    capyColorHistoType_red = 0,  
    capyColorHistoType_green = 1,  
    capyColorHistoType_blue = 2,  
    capyColorHistoType_intensity,  
    capyColorHistoType_last  
} CapyColorHistoType;
```

Histogram types

21.3 Typedefs

None.

21.4 Struct CapyColorHisto

21.4.1 Struct CapyColorHisto's properties

```
double vals[capyColorHistoType_last][CAPY_COLORHISTO_NBBUCKETS];
```

Histogram values

```
double cumulVals[capyColorHistoType_last][CAPY_COLORHISTO_NBBUCKETS];
```

Histogram cumulative values

21.4.2 Struct CapyColorHisto's methods

```
void (*destruct)(void);
```

Destructor

```
void (*extract)(CapyImg const* const img);
```

Extract the color histograms from an image

Input argument(s):

- img: the image

```
void (*apply)(  
    CapyImg* const img,  
    CapyColorHistoType const type);
```


Correct an image to match the histogram for a given type

Input argument(s):

- img: the image
- type: the type of histogram to be matched

21.5 Functions

```
CapyColorHisto CapyColorHistoCreate(void);
```

Create a CapyColorHisto

Output and side effect(s):

- Return a CapyColorHisto

```
CapyColorHisto* CapyColorHistoAlloc(void);
```

Allocate memory for a new CapyColorHisto and create it

Output and side effect(s):

- Return a CapyColorHisto

Exception(s):

- May raise CapyExc_MallocFailed.

```
CapyColorHisto CapyColorHistoCreateUniform(void);
```

Create a CapyColorHisto with uniform distribution

Output and side effect(s):

- Return a CapyColorHisto

```
CapyColorHisto* CapyColorHistoAllocUniform(void);
```

Allocate memory for a new CapyColorHisto with uniform distribution and create it

Output and side effect(s):

- Return a CapyColorHisto

Exception(s):

- May raise `CapyExc_MallocFailed`.

```
void CapyColorHistoFree(CapyColorHisto** const that);
```

Free the memory used by a `CapyColorHisto*` and reset `**that` to `NULL`

Input argument(s):

- `that`: a pointer to the `CapyColorHisto` to free

22 Unit `colorPatch.h`

Class to manipulate patches of color.

22.1 Macros

```
#define CAPY_COLOR_PATCH_H
```

22.2 Enumerations

None.

22.3 Typedefs

None.

22.4 Struct `CapyColorPatch`

22.4.1 Struct `CapyColorPatch`'s properties

```
CapyImgPixels pixels;
```

Set of pixels in the patch

```
size_t idx;
```

Index of the patch (default: 0)


```
CapyColorData avgColor;
```

Average color of the patch, to be updated by the user with `getAvgColor`

```
CapyImgPos centerOfMass;
```

Center of mass of the patch, to be updated by the user with `getPosCenterOfMass`

```
CapyQuadrilateral approxQuad;
```

Approximating quadrilateral

22.4.2 Struct `CapyColorPatch`'s methods

```
void (*destruct)(void);
```

Destructor

```
void (*updateApproxQuadrilateral)(void);
```

Update the rasterised quadrilateral approximating the color patch. The corners of the resulting quadrilateral are guaranteed to be in clockwise order. The first corner is guaranteed to be the one nearest to the origin.

```
void (*updatePosCenterOfMass)(void);
```

Update the center of mass of the color patch.

```
void (*updateAvgColor)(void);
```

Update the average color of the color patch

22.5 Struct `CapyColorPatches`

22.5.1 Struct `CapyColorPatches`'s properties

```
CapyListColorPatch* list;
```

The list of patches

22.5.2 Struct CpyColorPatches's methods

```
void (*destruct)(void);
```

Destructor

```
CpyColorPatch (*get)(size_t const idx);
```

Get the color patch at a given index

Input argument(s):

- idx: the index

Output and side effect(s):

- Return the CpyColorPatch

```
size_t (*getNbPatch)(void);
```

Get the number of patches

Output and side effect(s):

- Return the number of patches

```
void (*add)(CpyColorPatch* const patch);
```

Add a CpyColorPatch to the patches

Input argument(s):

- patch: the patch to add

22.6 Functions

```
CpyColorPatch CpyColorPatchCreate(void);
```

Create a CpyColorPatch of given dimensions and mode

Output and side effect(s):

- Return a CpyColorPatch

Exception(s):

- May raise CpyExc_MallocFailed.

```
CpyColorPatch* CpyColorPatchAlloc(void);
```

Allocate memory for a new CpyColorPatch and create it
Output and side effect(s):

- Return a CpyColorPatch

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyColorPatchFree(CpyColorPatch** const that);
```

Free the memory used by a CpyColorPatch* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpyColorPatch to free

```
CpyColorPatches CpyColorPatchesCreate(void);
```

Create a CpyColorPatches of given dimensions and mode
Output and side effect(s):

- Return a CpyColorPatches

Exception(s):

- May raise CpyExc_MallocFailed.

```
CpyColorPatches* CpyColorPatchesAlloc(void);
```

Allocate memory for a new CpyColorPatches and create it
Output and side effect(s):

- Return a CpyColorPatches

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyColorPatchesFree(CpyColorPatches** const that);
```

Free the memory used by a CpyColorPatches* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpyColorPatches to free

23 Unit comparator.h

Comparator functions.

23.1 Macros

```
#define CAPY_COMPARATOR_H
```

```
#define CopyComparatorDef struct { \
    void (*destruct)(void);        \
    CopyCmpFun eval;               \
}
```

Definition of a CopyComparator class

```
#define CopyComparatorBasicTypeDef(name) \
extern CopyComparator copyComparator ## name;
```

Comparators for basic types

23.2 Enumerations

None.

23.3 Typedefs

```
typedef CopyComparatorDef CopyComparator;
```

CopyComparator class

23.4 Functions

```
CopyComparator CopyComparatorCreate(void);
```

Create a CopyComparator

Output and side effect(s):

- Return a CopyComparator


```
int CappyCmpStrInc(
    char const* a,
    char const* b);
```

Comparator for string (comparator for numerical type are in capymath.h), increasing version

```
int CappyCmpStrDec(
    char const* a,
    char const* b);
```

Comparator for string (comparator for numerical type are in capymath.h), decreasing version

24 Unit compressor.h

Compressor class.

24.1 Macros

```
#define CAPY_COMPRESSOR_H
```

```
#define CappyCompressorDef {
    void (*destruct)(void);
    CappyCompressorData (*compress)(CappyCompressorData const data);
    CappyCompressorData (*decompress)(CappyCompressorData const data);
}
```

CappyCompressor object definition macro

Compress the data

Input argument(s):

- data: the data to compress

Output and side effect(s):

- Return the compressed data
- CappyCompressorData (*compress)(CappyCompressorData const data);
- Decompress the data

Input argument(s):

- data: the data to decompress

Output and side effect(s):

- Return the decompressed data
- CpyCompressorData (*decompress)(CpyCompressorData const data);

24.2 Enumerations

None.

24.3 Typedefs

```
typedef struct CpyCompressor CpyCompressorDef CpyCompressor;
```

CpyCompressor object

24.4 Struct CpyCompressorData

24.4.1 Struct CpyCompressorData's properties

```
uint8_t* bytes;
```

Bytes of data

```
size_t size;
```

Size in byte

24.4.2 Struct CpyCompressorData's methods

None.

24.5 Struct CpyRLECompressor

24.5.1 Struct CpyRLECompressor's properties

```
struct CpyCompressorDef;
```

Inherit CpyCompressor

24.5.2 Struct CpyRLECompressor's methods

```
void (*destructCpyCompressor)(void);
```

Destructor

24.6 Struct CpyBWTRLECompressor

24.6.1 Struct CpyBWTRLECompressor's properties

```
struct CpyCompressorDef;
```

Inherit CpyCompressor

24.6.2 Struct CpyBWTRLECompressor's methods

```
void (*destructCpyCompressor)(void);
```

Destructor

24.7 Struct CpyHuffmanCompressor

24.7.1 Struct CpyHuffmanCompressor's properties

```
struct CpyCompressorDef;
```

Inherit CpyCompressor

24.7.2 Struct CpyHuffmanCompressor's methods

```
void (*destructCpyCompressor)(void);
```

Destructor

24.8 Functions

```
CpyCompressor CpyCompressorCreate(void);
```

Create a CpyCompressor

Output and side effect(s):

- Return a CpyCompressor

```
CpyCompressor* CpyCompressorAlloc(void);
```

Allocate memory for a new CpyCompressor and create it
Output and side effect(s):

- Return a CpyCompressor

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyCompressorFree(CpyCompressor** const that);
```

Free the memory used by a CpyCompressor* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpyCompressor to free

```
CpyRLECompressor CpyRLECompressorCreate(void);
```

Create a CpyRLECompressor
Output and side effect(s):

- Return a CpyRLECompressor

```
CpyRLECompressor* CpyRLECompressorAlloc(void);
```

Allocate memory for a new CpyRLECompressor and create it
Output and side effect(s):

- Return a CpyRLECompressor

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyRLECompressorFree(CpyRLECompressor** const that);
```


Free the memory used by a CpyCompressor* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpyCompressor to free

```
CpyBWTRLECompressor CpyBWTRLECompressorCreate(void);
```

Create a CpyBWTRLECompressor
Output and side effect(s):

- Return a CpyBWTRLECompressor

```
CpyBWTRLECompressor* CpyBWTRLECompressorAlloc(void);
```

Allocate memory for a new CpyBWTRLECompressor and create it
Output and side effect(s):

- Return a CpyBWTRLECompressor

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyBWTRLECompressorFree(CpyBWTRLECompressor** const that);
```

Free the memory used by a CpyCompressor* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpyCompressor to free

```
CpyHuffmanCompressor CpyHuffmanCompressorCreate(void);
```

Create a CpyHuffmanCompressor
Output and side effect(s):

- Return a CpyHuffmanCompressor

```
CpyHuffmanCompressor* CpyHuffmanCompressorAlloc(void);
```

Allocate memory for a new CpyHuffmanCompressor and create it
Output and side effect(s):

- Return a CpyHuffmanCompressor

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyHuffmanCompressorFree(CpyHuffmanCompressor** const that);
```

Free the memory used by a CpyCompressor* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpyCompressor to free

25 Unit dataset.h

Dataset class.

25.1 Macros

```
#define CAPY_DATASET_H
```

25.2 Enumerations

```
typedef enum CpyDatasetFieldType {
    capyDatasetFieldType_num,
    capyDatasetFieldType_cat,
    capyDatasetFieldType_datetime1,
    capyDatasetFieldType_datetime2,
} CpyDatasetFieldType;
```

Type of field. datetime1: datetime dd-mm-yyyy hh:ii datetime2: datetime hh:ii

```
typedef enum CpyDatasetFieldInterface {
    capyDatasetFieldInterface_in,
    capyDatasetFieldInterface_out,
} CpyDatasetFieldInterface;
```

Type of column.

25.3 Typedefs

None.

25.4 Struct CpyDatasetRow

25.4.1 Struct CpyDatasetRow's properties

```
char* str;
```

Original string of the row with commas replaced with 0.

```
char** fields;
```

Pointers to each field in the row.

```
size_t idx;
```

Index of the row in the dataset, starting at 0 not counting the header lines.

```
bool flagNullValue;
```

Flag to memorise if the row contains null/unknown values

```
CpyPad(bool, flagNullValue);
```

```
size_t nbFieldWithNullValue;
```

Number of fields with a null/unknown value in that row.

25.4.2 Struct CpyDatasetRow's methods

```
void (*destruct)(void);
```

Destructor

25.5 Struct CpyDatasetFieldDesc

25.5.1 Struct CpyDatasetFieldDesc's properties

```
char* label;
```

Pointer to the field label.

```
CpyDatasetFieldType type;
```


Field type.

```
CapyDatasetFieldInterface interface;
```

Field interface.

```
size_t idx;
```

Index in the row.

```
size_t nbCategoryVal;
```

For fields of categorical types, number of value in the category, for field of numerical types, number of row in the dataset

```
char** categoryVals;
```

For fields of categorical types, array of pointer to the category's values

```
CapyRangeDouble range;
```

Range of values (converted to numerical if the field is not numerical)

```
double avgVal;
```

Average value (converted to numerical if the field is not numerical)

```
double medianVal;
```

Median value (converted to numerical if the field is not numerical)

```
bool flagNullValue;
```

Flag to memorise if the field contains null/unknown values

```
CapyPad(bool, flagNullValue);
```

```
size_t nbRowWithNullValue;
```

Number of rows with a null/unknown value in that field.

25.5.2 Struct CapyDatasetFieldDesc's methods

```
void (*destruct)(void);
```

Destructor

25.6 Struct CopyDataset

25.6.1 Struct CopyDataset's properties

```
size_t nbRow;
```

Number of rows.

```
size_t nbRowWithNullValue;
```

Number of rows with at least one null/unknown value.

```
size_t nbField;
```

Number of fields in each row.

```
size_t nbFieldWithNullValue;
```

Number of fields with at least one null/unknown value.

```
char* fieldInterfaceStr;
```

Fields interface row with comma replaced with 0.

```
char* fieldTypeStr;
```

Fields type row with comma replaced with 0.

```
char* fieldLabelStr;
```

Fields label row with comma replaced with 0.

```
CopyDatasetFieldDesc* fields;
```

Fields description.

```
CopyDatasetRow* rows;
```

Array of rows.

```
size_t nbThread;
```

Number of threads for multithreaded operation (default: 10)

25.6.2 Struct CapyDataset's methods

```
void (*destruct)(void);
```

Destructor

```
void (*loadFromPath)(char const* const path);
```

Load the dataset from a file at a given path

Input argument(s):

- path: path to the dataset file

Exception(s):

- May raise CapyExc_MallocFailed, CapyExc_StreamReadError,
- CapyExc_InvalidStream.

```
void (*print)(FILE* const stream);
```

Print the dataset description of the dataset on a given stream.

Input argument(s):

- stream: the stream to print onto

```
void (*printData)(  
    FILE* const stream,  
    size_t const nbRow);
```

Print the dataset data of the dataset on a given stream.

Input argument(s):

- stream: the stream to print onto
- nbRow: if not 0 print the first nbRow rows only

```
size_t (*getNbInput)(void);
```

Get the number of input fields

Output and side effect(s):

- Return the number of input fields


```
size_t (*getNbOutput)(void);
```

Get the number of output fields

Output and side effect(s):

- Return the number of output fields

```
size_t (*getNbFieldOfType)(CapyDatasetFieldType const type);
```

Get the number of fields of a given type

Input argument(s):

- type: the type of field to be counted

Output and side effect(s):

- Return the number of fields

```
size_t (*getIdxInputField)(size_t const iInput);
```

Get the field index of the i-th input

Input argument(s):

- iInput: index of the input

Output and side effect(s):

- Return the index

Exception(s):

- May raise `CapyExc_InvalidElemIdx`.

```
size_t (*getIdxOutputField)(size_t const iOutput);
```

Get the field index of the i-th output

Input argument(s):

- iOutput: index of the output

Output and side effect(s):

- Return the index

Exception(s):

- May raise `CapyExc_InvalidElemIdx`.

```
double (*getValAsNum)(
    size_t const iRow,
    size_t const iField);
```

Get a value as a numeral. Inputs: `iRow`: index of the row `iField`: index of the field

Output and side effect(s):

- For numeral fields return the value as it is, for categorical fields
- return the index of the value in the list of possible values
- (`fieldDesc.categoryVals`)

Exception(s):

- May raise `CapyExc_UndefinedExecution`, `CapyExc_InvalidParameters`.

```
uint64_t (*getValAsUInt64Date)(
    size_t const iRow,
    size_t const iField);
```

Get a value as a uint64 encoded date. Inputs: `iRow`: index of the row `iField`: index of the field

Output and side effect(s):

- Return the uint64 encoded date (see `CapyDate`)

Exception(s):

- May raise `CapyExc_UndefinedExecution`, `CapyExc_InvalidParameters`.

```
double (*getValAsNormalisedNum)(
    size_t const iRow,
    size_t const iField);
```

Get a value as a normalised numeral. Inputs: `iRow`: index of the row `iField`: index of the field

Output and side effect(s):

- Return the value, converted to numerical if the field is categorical,
- after normalisation according to the 'range' property of the field
- description.

Exception(s):

- May raise `CapyExc_UndefinedExecution`.

```
CapyMat (*cvtToMatForSingleCatPredictor)(size_t const iOutput);
```

Convert a dataset to a matrix to be used by a single category predictor

Input argument(s):

- `iOutput`: index of the output

Output and side effect(s):

- Return a matrix with as many rows as there are rows in the dataset, and
- as many columns as there are inputs in the dataset plus one. The output
- must be of type `capyDatasetFieldType_cat`. Input values are converted
- using `getValAsNum`. The output value is assigned to the last
- column in the matrix, and equal to the category index.

Exception(s):

- May raise `CapyExc_UnsupportedFormat`.

```
CapyMat (*cvtToMatForNumericalPredictor)(size_t const iOutput);
```

Convert a dataset to a matrix to be used by a numerical predictor

Input argument(s):

- `iOutput`: index of the output

Output and side effect(s):

- Return a matrix with as many rows as there are rows in the dataset, and
- as many columns as there are inputs in the dataset plus one. The output
- must be of type `copyDatasetFieldType_num`. Input and output values are
- converted using `getValAsNum`. The output value is assigned to the last
- column in the matrix.

Exception(s):

- May raise `CapyExc_UnsupportedFormat`.

```
size_t (*getNbValOutputField)(size_t const iOutput);
```

Get the number of different values for a given output Inputs: `iField`: index of the output

Output and side effect(s):

- Return the number of different values for a categorical output field
- or the number of rows for a numerical output field

```
CapyMat (*cvtToMatForOneHotPredictor)(size_t const iOutput);
```

Convert a dataset to a matrix to be used by a one hot predictor

Input argument(s):

- `iOutput`: index of the output

Output and side effect(s):

- Return a matrix with as many rows as there are rows in the dataset, and
- as many columns as there are inputs in the dataset plus as many values
- the given output takes. The output must be of type
- `copyDatasetFieldType_cat`. Input values are converted using

- getValAsNum. The one hot encoding of the output value is
- assigned to the last columns in the matrix, and take values 0 for 'is
- not this category' and 1 for 'is this category'.

Exception(s):

- May raise CopyExc_UnsupportedFormat.

```
size_t (*getIdxFldFromName)(char const* const name);
```

Get the index of a field from its name

Input argument(s):

- name: the name

Output and side effect(s):

- Return the index of the field, or raise the exception
- CopyExc_InvalidParameters if it couldn't be found.

```
CopyArrSize* (*getDistAsBins)(
    size_t const iField,
    size_t const nbBin);
```

Get the distribution of a field values as an array of bins.

Input argument(s):

- iField: the index of the field
- nbBin: the number of bins

Output and side effect(s):

- Return a CopyArrSize of size 'nbBin'.

```
CopyArrSize* (*getDistAsBinsGivenCatValue)(
    size_t const iField,
    size_t const nbBin,
    size_t const iCatField,
    char const* const valCatField);
```


Get the distribution of a field values as an array of bins for a given value of a given categorical field

Input argument(s):

- iField: the index of the field
- nbBin: the number of bins
- iCatField: the index of the categorical field
- valCatField: the value of the categorical field

Output and side effect(s):

- Return a CpyArrSize of size 'nbBin'.

```
size_t (*getNbRowContainingVal)(  
    size_t const iField,  
    char const* const valField);
```

Get the number of rows containing a particular value for a given field.

Input argument(s):

- iField: the filtering field
- valField: the filtering value

Output and side effect(s):

- Return the number of rows.

```
void (*getValuesFromTwoFieldsAsVectors)(  
    size_t const iField,  
    size_t const jField,  
    CpyVec* const u,  
    CpyVec* const v);
```

Get the pair of values from two fields as two vectors.

Input argument(s):

- iField: the first field
- jField: the second field
- u: the vector receiving values from the first field
- v: the vector receiving values from the second field

Output and side effect(s):

- 'u' and 'v' are destructured, created afresh and populated with values.
- Rows with null value in one or the other field are ignored.

```
bool (*isNullValue)(char const* const val);
```

Check if a value is a null/unknown value

Input argument(s):

- val: the value to check

Output and side effect(s):

- Return true if the value is considered to be null/unknown. A null/unknown
- value is the empty string or a string equal to "nan" (case insensitive).

```
CapyPointCloud* (*toPointCloud)(void);
```

Convert the dataset into a point cloud

Output and side effect(s):

- Return a CapyPointCloud of dimension equal to the number of fields
- and number of point equal to the number of sample. All values are
- converted to numerical values.

25.7 Functions

```
CapyDatasetRow CapyDatasetRowCreate(void);
```

Create a CapyDatasetRow.

Output and side effect(s):

- Return a CapyDatasetRow.

```
CapyDatasetRow* CapyDatasetRowAlloc(void);
```


Allocate memory for a new CpyDatasetRow and create it.

Output and side effect(s):

- Return a CpyDatasetRow.

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyDatasetRowFree(CpyDatasetRow** const that);
```

Free the memory used by a CpyDatasetRow* and reset '*that' to NULL.

Input argument(s):

- that: a pointer to the CpyDatasetRow to free

```
CpyDatasetFieldDesc CpyDatasetFieldDescCreate(void);
```

Create a CpyDatasetFieldDesc.

Output and side effect(s):

- Return a CpyDatasetFieldDesc.

```
CpyDatasetFieldDesc* CpyDatasetFieldDescAlloc(void);
```

Allocate memory for a new CpyDatasetFieldDesc and create it.

Output and side effect(s):

- Return a CpyDatasetFieldDesc.

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyDatasetFieldDescFree(CpyDatasetFieldDesc** const that);
```

Free the memory used by a CpyDatasetFieldDesc* and reset '*that' to NULL.

Input argument(s):

- that: a pointer to the CpyDatasetFieldDesc to free


```
CapyDataset CapyDatasetCreate(void);
```

Create a CapyDataset.

Output and side effect(s):

- Return a CapyDataset.

```
CapyDataset* CapyDatasetAlloc(void);
```

Allocate memory for a new CapyDataset and create it.

Output and side effect(s):

- Return a CapyDataset.

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyDatasetFree(CapyDataset** const that);
```

Free the memory used by a CapyDataset* and reset '*that' to NULL.

Input argument(s):

- that: a pointer to the CapyDataset to free.

26 Unit date.h

Date class.

26.1 Macros

```
#define CAPY_DATE_H
```


26.2 Enumerations

```
typedef enum CapyDateUnit {  
    capyDateUnit_year ,  
    capyDateUnit_month ,  
    capyDateUnit_day ,  
    capyDateUnit_hour ,  
    capyDateUnit_minute ,  
    capyDateUnit_second ,  
    capyDateUnit_nb ,  
} CapyDateUnit;
```

Date units

```
typedef enum CapyWeekDay {  
    capyWeekDay_sunday ,  
    capyWeekDay_monday ,  
    capyWeekDay_tuesday ,  
    capyWeekDay_wednesday ,  
    capyWeekDay_thursday ,  
    capyWeekDay_friday ,  
    capyWeekDay_saturday ,  
    capyWeekDay_nb ,  
} CapyWeekDay;
```

Day of week

26.3 Typedefs

```
typedef int16_t CapyDate_t;
```

Type for the values of a CapyDate

```
typedef struct CapyDate CapyDate;
```

Date object

26.4 Struct CapyDateCosSin

26.4.1 Struct CapyDateCosSin's properties

```
double cos;
```

```
double sin;
```


26.4.2 Struct CpyDateCosSin's methods

None.

26.5 Functions

```
CpyDate CpyDateCreate(void);
```

Create a CpyDate

Output and side effect(s):

- Return a CpyDate

```
CpyDate* CpyDateAlloc(void);
```

Allocate memory for a new CpyDate and create it

Output and side effect(s):

- Return a CpyDate

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyDateFree(CpyDate** const that);
```

Free the memory used by a CpyDate* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyDate to free

27 Unit dict.h

Generic dictionary (implemented as a hash table).

27.1 Macros

```
#define CAPY_DICT_H
```

```
#define CopyDecDictIterator(name, type) \
typedef struct name name; \
typedef struct name ## Entry name ## Entry; \
struct name ## Entry { \
    char* key; \
    type val; \
    CapyPad(type, 0); \
    name ## Entry* next; \
}; \
typedef struct name ## Iterator { \
    size_t idx; \
    name ## Entry* entry; \
    size_t idxHash; \
    name* dict; \
    name ## Entry datatype; \
    void (*destruct)(void); \
    name ## Entry* (*reset)(void); \
    name ## Entry* (*next)(void); \
    bool (*isActive)(void); \
    name ## Entry* (*get)(void); \
} name ## Iterator; \
name ## Iterator name ## Iterator ## Create(name* const dict); \
name ## Iterator* name ## Iterator ## Alloc(name* const dict); \
void name ## Iterator ## Destruct(void); \
void name ## Iterator ## Free(name ## Iterator** const that); \
name ## Entry* name ## Iterator ## Reset(void); \
name ## Entry* name ## Iterator ## Next(void); \
bool name ## Iterator ## IsActive(void); \
name ## Entry* name ## Iterator ## Get(void);
```

Iterator for generic dict structure. Declaration macro for an iterator structure named 'name' ## Iterator, associated to a dict named 'name'

```
#define CopyDefDictIteratorCreate(name) \
name ## Iterator name ## Iterator ## Create(name* const dict) { \
    name ## Iterator that = { \
        .idx = 0, \
        .entry = NULL, \
        .idxHash = 0, \
        .dict = dict, \
        .destruct = name ## Iterator ## Destruct, \
        .reset = name ## Iterator ## Reset, \
        .next = name ## Iterator ## Next, \
        .isActive = name ## Iterator ## IsActive, \
        .get = name ## Iterator ## Get, \
    }; \
    $(&that, reset)(); \
    return that; \
}
```

Create an iterator on a generic dictionary

Input argument(s):

- dict: the generic dict on which to iterate

Output and side effect(s):

- Return the iterator

```
#define CopyDefDictIteratorAlloc(name) \
name ## Iterator* name ## Iterator ## Alloc(name* const dict) { \
    name ## Iterator* that = NULL; \
    safeMalloc(that, sizeof(name ## Iterator)); \
    if(!that) return NULL; \
    *that = name ## Iterator ## Create(dict); \
    return that; \
}
```

Allocate memory and create an iterator on a generic dict

Input argument(s):

- dict: the dictionary on which to iterate

Output and side effect(s):

- Return the iterator

```
#define CopyDefDictIteratorDestruct(name) \
void name ## Iterator ## Destruct(void) { \
    return; \
}
```

Free the memory used by an iterator.

Input argument(s):

- that: the iterator to free

```
#define CopyDefDictIteratorFree(name) \
void name ## Iterator ## Free(name ## Iterator** const that) { \
    if(*that == NULL) return; \
    $(*that, destruct)(); \
    free(*that); \
    *that = NULL; \
}
```

Free the memory used by a pointer to an iterator and reset '*that' to NULL

Input argument(s):

- that: a pointer to the iterator to free


```

#define CopyDefDictIteratorReset(name) \
name ## Entry* name ## Iterator ## Reset(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    that->idx = 0; \
    that->idxHash = 0; \
    that->entry = NULL; \
    while( \
        that->idxHash < that->dict->sizeHash && \
        that->dict->data[that->idxHash] == NULL \
    ) { \
        ++(that->idxHash); \
    } \
    if(that->idxHash < that->dict->sizeHash) \
        that->entry = that->dict->data[that->idxHash]; \
    if(name ## Iterator ## IsActive()) \
        return name ## Iterator ## Get(); \
    else \
        return NULL; \
}

```

Reset the iterator

Output and side effect(s):

- Return the first element of the iteration

```

#define CopyDefDictIteratorNext(name) \
name ## Entry* name ## Iterator ## Next(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    if(that->entry == NULL) return NULL; \
    if(that->entry->next) { \
        that->entry = that->entry->next; \
        ++(that->idx); \
    } else { \
        that->entry = NULL; \
        ++(that->idxHash); \
        while( \
            that->idxHash < that->dict->sizeHash && \
            that->dict->data[that->idxHash] == NULL \
        ) { \
            ++(that->idxHash); \
        } \
        if(that->idxHash < that->dict->sizeHash) { \
            that->entry = that->dict->data[that->idxHash]; \
            ++(that->idx); \
        } \
    } \
    return that->entry; \
}

```

Move the iterator to the next element

Output and side effect(s):

- Return the next element of the iteration


```

#define CopyDefDictIteratorIsActive(name) \
bool name ## Iterator ## IsActive(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    return (that->entry != NULL); \
}

```

Check if the iterator is on a valid element

Output and side effect(s):

- Return true if the iterator is on a valid element, else false

```

#define CopyDefDictIteratorGet(name) \
name ## Entry* name ## Iterator ## Get(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    return that->entry; \
}

```

Get the current element of the iteration

Output and side effect(s):

- Return a pointer to the current element

```

#define CopyDefDictIterator(name) \
    CopyDefDictIteratorCreate(name) \
    CopyDefDictIteratorAlloc(name) \
    CopyDefDictIteratorDestruct(name) \
    CopyDefDictIteratorFree(name) \
    CopyDefDictIteratorReset(name) \
    CopyDefDictIteratorNext(name) \
    CopyDefDictIteratorIsActive(name) \
    CopyDefDictIteratorGet(name)

```

Definition macro calling all the submacros at once for an iterator on an dictionary structure named 'name'

```

#define CopyDecDict(name, type) \
CopyDecDictIterator(name, type) \
typedef struct name { \
    name ## Entry** data; \
    type defaultVal; \
    CopyPad(type, 0); \
    size_t sizeHash; \
    size_t nbEntry; \
    CopyHashFun* hash; \
    name ## Iterator iter; \
    void (*destruct)(void); \
    size_t (*getNbEntry)(void); \
    size_t (*getSizeHash)(void); \
    void (*set)(char const* const key, type const val); \
    type (*get)(char const* const key); \
    void (*remove)(char const* const key); \
}

```



```

type* (*getPtr)(char const* const key); \
void (*initIterator)(void); \
name* (*clone)(void); \
void (*setDefaultVal)(type const val); \
bool (*hasKey)(char const* const key); \
void (*setHash)(CapyHashFun* const hash); \
} name; \
name name ## Create(size_t const size, type const defaultVal); \
name* name ## Alloc(size_t const size, type const defaultVal); \
name* name ## Clone(void); \
void name ## Destruct(void); \
void name ## Free(name** const that); \
size_t name ## GetNbEntry(void); \
size_t name ## GetSizeHash(void); \
void name ## Set(char const* const key, type const val); \
type name ## Get(char const* const key); \
void name ## Remove(char const* const key); \
type* name ## GetPtr(char const* const key); \
void name ## InitIterator(void); \
void name ## SetDefaultVal(type const val); \
void name ## SetHash(CapyHashFun* const hash); \
bool name ## HasKey(char const* const key);

```

Generic dict structure. Declaration macro for a dict structure named 'name', containing entries with keys of type 'char*' and values of type 'type'

```

#define CapyDefDictCreate(name, type) \
name name ## Create( \
    size_t const sizeHash, \
    type const defaultVal) { \
    name that = { \
        .defaultVal = defaultVal, \
        .sizeHash = sizeHash, \
        .nbEntry = 0, \
        .hash = (CapyHashFun*)CapyFNV1aHashFunAlloc(), \
        .destruct = name ## Destruct, \
        .getNbEntry = name ## GetNbEntry, \
        .getSizeHash = name ## GetSizeHash, \
        .set = name ## Set, \
        .remove = name ## Remove, \
        .get = name ## Get, \
        .getPtr = name ## GetPtr, \
        .initIterator = name ## InitIterator, \
        .clone = name ## Clone, \
        .setDefaultVal = name ## SetDefaultVal, \
        .hasKey = name ## HasKey, \
        .setHash = name ## SetHash, \
    }; \
    safeMalloc(that.data, sizeHash); \
    assert(that.data != NULL); \
    loop(iHash, sizeHash) that.data[iHash] = NULL; \
    return that; \
}

```

Generic dict structure. Definition macro for a dict structure named 'name', containing entries with keys of type 'char*' and values of type 'type' Create a dictionary

Input argument(s):

- sizeHash: size of the hash table
- defaultVal: default value returned by 'get' for an unknown key

Output and side effect(s):

- Return an empty dictionary containing elements of type 'type'

```
#define CopyDefDictAlloc(name, type) \
name* name ## Alloc( \
    size_t const sizeHash, \
    type const defaultVal) { \
    name* that = NULL; \
    safeMalloc(that, 1); \
    if(!that) return NULL; \
    *that = name ## Create(sizeHash, defaultVal); \
    that->iter = name ## IteratorCreate(that); \
    return that; \
}
```

Allocate memory for a new dictionary and create it

Input argument(s):

- sizeHash: size of the hash table
- defaultVal: default value returned by 'get' for an unknown key

Output and side effect(s):

- Return a dictionary containing elements of type 'type'

Exception(s):

- May raise CopyExc_MallocFailed.

```
#define CopyDefDictClone(name, type) \
name* name ## Clone(void) { \
    name* that = (name*)copyThat; \
    name* clone = name ## Alloc(that->sizeHash, that->defaultVal); \
    foreach(entry, that->iter) { \
        $(clone, set)(entry.key, entry.val); \
    } \
    return clone; \
}
```

Clone a dictionary

Output and side effect(s):

- Return a clone of the dictionary

Exception(s):

- May raise CpyExc_MallocFailed.

```
#define CpyDefDictDestruct(name, type) \
void name ## Destruct(void) { \
    name* that = (name*)cpyThat; \
    loop(iHash, that->sizeHash) { \
        name ## Entry* elem = that->data[iHash]; \
        while(elem) { \
            name ## Entry* nextElem = elem->next; \
            free(elem->key); \
            free(elem); \
            elem = nextElem; \
        } \
    } \
    free(that->data); \
    CpyHashFunFree(&(that->hash)); \
    *that = (name){0}; \
}
```

Free the memory used by a dictionary. The memory eventually used by the value of the entries must be freed by the user. The key of the entries are automatically freed.

Input argument(s):

- that: the dictionary to free

```
#define CpyDefDictFree(name, type) \
void name ## Free(name** const that) { \
    if(that == NULL || *that == NULL) return; \
    $(*that, destruct)(); \
    free(*that); \
    *that = NULL; \
}
```

Free the memory used by a pointer to a dictionary and reset '*that' to NULL. The memory eventually used by the value of the entries must be freed by the user. The key of the entries are automatically freed.

Input argument(s):

- that: a pointer to the dictionary to free

```
#define CpyDefDictGetNbEntry(name, type) \
size_t name ## GetNbEntry(void) { \
    return ((name*)cpyThat)->nbEntry; \
}
```

Get the number of entries in the dictionary

Output and side effect(s):

- Return the number of entries

```
#define CopyDefDictGetSizeHash(name, type) \
size_t name ## GetSizeHash(void) { \
    return ((name*)copyThat)->sizeHash; \
}
```

Get the size of the hash table of the dictionary

Output and side effect(s):

- Return the size of the hash

```
#define CopyDefDictGet(name, type) \
type name ## Get(char const* const key) { \
    return *(name ## GetPtr(key)); \
}
```

Get a value of the dictionary

Input argument(s):

- key: the key of the element to get

Output and side effect(s):

- Return the value

Exception(s):

- May raise CopyExc_MallocFailed.

```
#define CopyDefDictGetPtr(name, type) \
type* name ## GetPtr(char const* const key) { \
    name* that = (name*)copyThat; \
    size_t iHash = $(that->hash, eval)(key, strlen(key)) % that->sizeHash; \
    name ## Entry* entry = that->data[iHash]; \
    while(entry && strcmp(entry->key, key) != 0) entry = entry->next; \
    if(entry == NULL) { \
        safeMalloc(entry, 1); \
        if(entry == NULL) return NULL; \
        *entry = (name ## Entry){0}; \
        safeSprintf(&(entry->key), "%s", key); \
        entry->val = that->defaultVal; \
        entry->next = that->data[iHash]; \
        that->data[iHash] = entry; \
        ++(that->nbEntry); \
    } \
    return &(entry->val); \
}
```


Get a pointer to a value of the dictionary

Input argument(s):

- key: the key of the element to get, if there is no element for this
- key, create a new one initialised with the default value

Output and side effect(s):

- Return the pointer to the element

```
#define CopyDefDictRemove(name, type) \
void name ## Remove(char const* const key) { \
    name* that = (name*)copyThat; \
    size_t iHash = $(that->hash, eval)(key, strlen(key)) % that->sizeHash; \
    name ## Entry* entry = that->data[iHash]; \
    name ## Entry* prevEntry = NULL; \
    while(entry && strcmp(entry->key, key) != 0) { \
        prevEntry = entry; \
        entry = entry->next; \
    } \
    if(entry != NULL) { \
        if(prevEntry != NULL) prevEntry = entry->next; \
        else that->data[iHash] = entry->next; \
        free(entry); \
        --(that->nbEntry); \
    } \
}
```

Remove an entry from the dictionary

Input argument(s):

- key: the key of the element to remove

Output and side effect(s):

- The entry is removed if it exists, else nothing is done. The user is
- responsible for freeing the memory used by the entry value if any.

```
#define CopyDefDictSet(name, type) \
void name ## Set(char const* const key, type const val) { \
    name* that = (name*)copyThat; \
    type* entryVal = $(that, getPtr)(key); \
    *entryVal = val; \
}
```

Set a value of the dictionary

Input argument(s):

- key: the key of the entry to set
- val: the value to set

Exception(s):

- May raise `CapyExc_MallocFailed`.

```
#define CapyDefDictInitIterator(name, type) \
void name ## InitIterator(void) { \
    name* that = (name*)capyThat; \
    that->iter = name ## IteratorCreate(that); \
}
```

Initialise the iterator of the dictionary, must be called after the creation of the dictionary when it is created with `Create()`, `Alloc()` automatically initialise the iterator.

```
#define CapyDefDictSetDefaultVal(name, type) \
void name ## SetDefaultVal(type const val) { \
    name* that = (name*)capyThat; \
    that->defaultVal = val; \
}
```

Set the default value used when an entry is automatically added

Input argument(s):

- val: the default value

```
#define CapyDefDictHasKey(name, type) \
bool name ## HasKey(char const* const key) { \
    name* that = (name*)capyThat; \
    size_t iHash = $(that->hash, eval)(key, strlen(key)) % that->sizeHash; \
    name ## Entry* entry = that->data[iHash]; \
    while(entry && strcmp(entry->key, key) != 0) entry = entry->next; \
    return (entry != NULL); \
}
```

Check if a key exists in the dictionary

Input argument(s):

- key: the key to check

Output and side effect(s):

- Return true if the key exists, else false


```

#define CopyDefDictSetHash(name, type) \
void name ## SetHash(CapyHashFun* const hash) { \
    name* that = (name*)copyThat; \
    CapyHashFunFree(&(that->hash)); \
    that->hash = hash; \
}

```

Set the hash function

Input argument(s):

- hash: the hash function

```

#define CopyDefDict(name, type) \
    CopyDefDictCreate(name, type) \
    CopyDefDictAlloc(name, type) \
    CopyDefDictClone(name, type) \
    CopyDefDictDestruct(name, type) \
    CopyDefDictFree(name, type) \
    CopyDefDictGetNbEntry(name, type) \
    CopyDefDictGetSizeHash(name, type) \
    CopyDefDictSet(name, type) \
    CopyDefDictGet(name, type) \
    CopyDefDictGetPtr(name, type) \
    CopyDefDictInitIterator(name, type) \
    CopyDefDictSetDefaultVal(name, type) \
    CopyDefDictSetHash(name, type) \
    CopyDefDictHasKey(name, type) \
    CopyDefDictRemove(name, type) \
    CopyDefDictIterator(name)

```

Definition macro calling all the submacros at once for a dictionary structure named 'name', containing elements of type 'type'

27.2 Enumerations

None.

27.3 Typedefs

None.

27.4 Functions

None.

28 Unit diffevo.h

Differential evolution algorithm.

28.1 Macros

```
#define CAPY_DIFFEVO_H
```

28.2 Enumerations

```
typedef enum CapyDiffEvoInitMode {  
    capyDiffEvoInitMode_random,  
    capyDiffEvoInitMode_randomisedSeed,  
} CapyDiffEvoInitMode;
```

Initialisation modes

```
typedef enum CapyDiffEvoMode {  
    capyDiffEvoMode_std,  
    capyDiffEvoMode_sequential  
} CapyDiffEvoMode;
```

Evolution modes

28.3 Typedefs

None.

28.4 Struct CapyDiffEvo

28.4.1 Struct CapyDiffEvo's properties

```
size_t nbIter;
```

Current number of iteration

```
size_t sizeDna;
```

Size of the dna

```
double initialFitness;
```

Initial fitness (first fitness evaluated after calling run())

```
double bestFitness;
```

Current best fitness


```
bool flagStop;
```

Flag to stop the optimisation

```
CapyPad(bool, 0);
```

```
bool running;
```

Flag to memorise if the child is running

```
CapyPad(bool, 1);
```

```
double* bestDna;
```

Current best agent DNA

```
double bestConstraint;
```

Constraint value of the current best dna

```
CapyRandomSeed_t seed;
```

Seed for the RNG By default, the current time. Can be modified before calling run()

```
double probMut;
```

Differential evolution algorithm coefficients By default, probMut = 0.9 and ampMut = 0.8 probMut in [0, 1], ampMut in [0, 2]. Can be modified before calling run(). probMut has no effect if mode==capyDiffEvoMode_sequential

```
double ampMut;
```

```
size_t nbAgent;
```

Population size By default, nbAgent = 10 * dim. Can be modified before calling run()

```
pthread_t thread;
```

The child thread


```
CapyException_t threadExc;
```

Exception value returned by the rendering thread

```
CapyPad(CapyException_t, threadExc);
```

```
sem_t semaphore;
```

Semaphore to manage shared data access

```
double const* seedDna;
```

Seed used to initialise the first DNA of the first agent if not null

```
size_t nbIterReset;
```

Maximum number of iteration without improvement before randomising the agents (never reset if equals to 0, default: 0)

```
CapyDiffEvoInitMode initMode;
```

Initialisation mode (default: capyDiffEvoInitMode_random)

```
CapyPad(CapyDiffEvoInitMode, initMode);
```

```
double coeffRandomisationSeed;
```

Coeff randomisation seed for initMode 'randomisedSeed' (in [0,1], default: 0.1). The seed is randomised as $(1 - c) * \text{seed} + c * r$, where c is coeffRandomisationSeed and r is a random value in the domain.

```
CapyDiffEvoMode mode;
```

Evolution mode (default: capyDiffEvoMode_std)

```
CapyPad(CapyDiffEvoInitMode, mode);
```

```
size_t idxSeq;
```

Index and stride used when mode==capyDiffEvoMode_sequential. strideEq=1 by default, the stride is the number of gene evolving per iteration


```
size_t strideSeq;
```

```
FILE* verboseStream;
```

Verbose stream (default: NULL). If not null print info during search.

28.4.2 Struct CpyDiffEvo's methods

```
void (*destruct)(void);
```

Destructor

```
void (*run)(  
    CpyMathFun* const fitness,  
    CpyMathFun* const constraint);
```

Run the differential evolution algorithm in its own thread

Input argument(s):

- fitness: Object performing the fitness calculation, takes the
- agent's DNA as input and return the fitness value as
- output. The agent's DNA is an array of double values
- in fitness' domain. The higher the fitness the better the agent.
- constraint: function of dimensions (fitness.dimIn, 1) encoding the
- constraints on the inputs as follow: if the output value
- is positive or null the constraint is satisfied, else
- the constraint is not satisfied. The greater the value, the
- better the constraint is satisfied. A null argument indicates
- no constraint.

Output and side effect(s):

- that->bestFitness, bestDna and bestConstraint are updated with the best
- agent. The best agent is either the one respecting the constraint and

- having highest fitness, or if no agent respecting constraint could
- be found it is the one with highest fitness regardless of the constraint.

```
void (*getBestDna)(
    double* const dna,
    double* const fitness,
    double* const constraint);
```

Get the current best DNA (no effect if no optimisation currently running)

Input argument(s):

- dna: array of double updated with the best dna
- fitness: double updated with the fitness value of the dna
- constraint: double updated with the constraint value of the dna

```
void (*stop)(void);
```

Stop the current optimisation (no effect if there is no optimisation currently running)

28.5 Struct CpyDiffEvoThreadData

28.5.1 Struct CpyDiffEvoThreadData's properties

```
CpyDiffEvo* that;
```

the CpyDiffEvo

```
CpyMathFun* fitness;
```

Object performing the fitness calculation, takes the agent's DNA as input and return the fitness value as output. The agent's DNA is an array of double values in fitness' domain. The higher the fitness the better the agent.

```
CpyMathFun* constraint;
```

Constraint on the agent DNA values. If not null, takes the agent's DNA as input and return the constraint satisfaction as output (positive or null means satisfied).

28.5.2 Struct CpyDiffEvoThreadData's methods

None.

28.6 Functions

```
CpyDiffEvo CpyDiffEvoCreate(void);
```

Create a CpyDiffEvo

Output and side effect(s):

- Return a CpyDiffEvo

```
CpyDiffEvo* CpyDiffEvoAlloc(void);
```

Allocate memory for a new CpyDiffEvo and create it

Output and side effect(s):

- Return a CpyDiffEvo

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyDiffEvoFree(CpyDiffEvo** const that);
```

Free the memory used by a CpyDiffEvo* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyDiffEvo to free

29 Unit display.h

Class to display graphics on screen, using GTK.

29.1 Macros

```
#define CAPY_DISPLAY_H
```

```
#define CAPY_DISPLAY_MAX_LENGTH_TITLE 1024
```

Max length of the window's title

```
#define CAPY_DISPLAY_EVT_BUFFER_SIZE 256
```

Size of the buffer for events

29.2 Enumerations

None.

29.3 Typedefs

```
typedef uint64_t CapyDisplayTimeMs_t;
```

Type for the time values of a CapyDisplay

29.4 Struct CapyDisplayKeyEvt

29.4.1 Struct CapyDisplayKeyEvt's properties

```
guint keyval;
```

The key value. See the link below for a complete list of key values. <https://gitlab.gnome.org/GNOME/gnome-shell>

```
CapyPad(guint, keyval);
```

```
CapyDisplayTimeMs_t timeMs;
```

Time of the event

29.4.2 Struct CapyDisplayKeyEvt's methods

None.

29.5 Struct CapyDisplayMouseEvent

29.5.1 Struct CapyDisplayMouseEvent's properties

```
CapyImgPos pos;
```

The coordinates in the display of the mouse at the time of the event

```
CapyImgPos posImg;
```

The coordinates in the image of the mouse at the time of the event

```
GdkEventType type;
```

The type of event (GDK_BUTTON_PRESS or GDK_BUTTON_RELEASE)

```
CapyPad(GdkEventType, type);
```

```
CapyDisplayTimeMs_t timeMs;
```

Time of the event

29.5.2 Struct CapyDisplayMouseEvent's methods

None.

29.6 Struct CapyDisplayCom

29.6.1 Struct CapyDisplayCom's properties

```
gboolean flagClose;
```

Flag to memorise the request from the client to close the CapyDisplay

```
gboolean isDisplayed;
```

Flag to memorise if the windows is displayed

```
CapyDisplayKeyEvt keyPressed[CAPY_DISPLAY_EVT_BUFFER_SIZE];
```

Buffer to memorise the key pressed waiting to be processed by the user process


```
size_t idxLastKeyWrite;
```

```
size_t idxLastKeyRead;
```

```
CapyDisplayMouseEvent mouseEvt[CAPY_DISPLAY_EVT_BUFFER_SIZE];
```

Buffer to memorise the mouse event waiting to be processed by the user process

```
size_t idxLastMouseWrite;
```

```
size_t idxLastMouseRead;
```

```
char title[CAPY_DISPLAY_MAX_LENGTH_TITLE];
```

Window's title

```
CapyImgPos mousePos;
```

Current mouse position in pixel coordinates of the display. If the mouse moves out of the display, it is the last recorded position.

```
CapyDisplayMagnifier magnifier;
```

Display magnifier

29.6.2 Struct CapyDisplayCom's methods

None.

29.7 Struct CapyDisplay

29.7.1 Struct CapyDisplay's properties

```
CapyImgDims_;
```

Dimensions of the CapyDisplay

```
GtkApplication* gtkApp;
```


GTK variables to manage the window

```
GApplication* gApp;
```

```
GtkWidget* window;
```

```
GdkPixbuf* pixbuf;
```

```
CapyImgDims_t rowstride;
```

```
int n_channels;
```

```
GtkImage* image;
```

```
unsigned int timerId;
```

```
int sharedMemId;
```

Variables to manage the two shared segments (one for the pixels value and for the communication between the CapyDisplay and the user process).

```
int comId;
```

```
CapyPad(int, comId);
```

```
guchar* sharedPixels;
```

```
CapyDisplayCom* com;
```

```
CapyChrono chrono;
```

Chronometer to measure time since the Display is running

```
void* onShowArg;
```


Handler called in the child process when the display is shown By default NULL, onShowArg is the argument passed to onShow

```
void* onCloseArg;
```

Handler called in the child process when the display is closed By default NULL, onCloseArg is the argument passed to onClose

29.7.2 Struct CapyDisplay's methods

```
void (*destruct)(void);
```

Destructor

```
void (*onShow)(void*);
```

```
void (*onClose)(void*);
```

```
void (*show)(void);
```

Show a CapyDisplay on the screen. The display is centered on the screen.
Exception(s):

- May raise CapyExc_ForkFailed

```
void (*close)(void);
```

Close the window of the CapyDisplay

```
CapyDisplayKeyEvt (*getKeyPressed)(void);
```

Return the last key pressed since the last call to getKeyPressed

Output and side effect(s):

- Return the key event, or CapyDisplayKeyEvt.keyval=0 if there wasn't
- any key pressed. See the link below for a complete list of key values.
- <https://gitlab.gnome.org/GNOME/gtk/blob/master/gdk/gdkkeysyms.h>


```
CapyDisplayMouseEvent (*getMouseEvent)(void);
```

Return the last mouse event since the last call to getMouseEvent

Output and side effect(s):

- Return the mouse event, or CapyDisplayMouseEvent.type=0 if there
- wasn't any event.

```
void (*setPixel)(  
    CapyImgPos const* const pixel,  
    CapyColorData const* const color);
```

Set the RGB value of a pixel in the CapyDisplay

Input argument(s):

- pixel: the position of the pixel (from left to right and top to bottom)
- color: the new color of the pixel (the alpha channel is forced to 1.0)

```
void (*copyImg)(CapyImg const* const img);
```

Copy a CapyImage into a CapyDisplay

Input argument(s):

- img: the CapyImage to copy (the alpha channel is forced to 1.0)

```
void (*magnifyImg)(CapyImg const* const img);
```

Copy a CapyImage into a CapyDisplay after scaling and translating it with the CapyDisplayMagnifier

Input argument(s):

- img: the CapyImage to copy (the alpha channel is forced to 1.0)

```
gboolean (*isDisplayed)(void);
```

Check if a CapyDisplay is currently displayed

Output and side effect(s):

- Return true if the CapyDisplay is displayed


```
void (*setTitle)(char const* const title);
```

Set the window's title

Input argument(s):

- title: the new title

```
CapyDisplayTimeMs_t (*getTimeMs)(void);
```

Get the time spent in millisecond since the CapyDisplay is displayed.

```
void (*magnifyToFitIn)(CapyImg const* const img);
```

Set the magnifier scale to fit entirely the image in argument in the display.
(The position is left unmodified)

Input argument(s):

- img: the image to fit

```
CapyImgPos (*getMousePos)(void);
```

Get the current mouse position in display coordinates (from left to right and top to bottom).

Output and side effect(s):

- Return the position

```
CapyImgPos (*getMousePosImg)(void);
```

Get the current mouse position converted to image coordinates.

Output and side effect(s):

- Return the position

29.8 Functions

```
CapyDisplay CapyDisplayCreate(  
    CapyImgDims const* const dim,  
    char const* const title);
```

Create a CapyDisplay

Input argument(s):

- dim: the dimension of the display
- title: title displayed in the title bar of the window of the CapyDisplay

Output and side effect(s):

- Return a CapyDisplay

Exception(s):

- May raise CapyExc_MallocFailed

```
CapyDisplay* CapyDisplayAlloc(
    CapyImgDims const* const dim,
    char const* const title);
```

Allocate memory for new CapyDisplay

Input argument(s):

- dim: the dimension of the display
- title: title displayed in the title bar of the window of the CapyDisplay

Output and side effect(s):

- Return a newly allocated CapyDisplay

Exception(s):

- May raise CapyExc_MallocFailed

```
void CapyDisplayFree(CapyDisplay** display);
```

Free the memory used by a CapyDisplay* and reset '*that' to NULL

Input argument(s):

- display: the CapyDisplay to free

30 Unit displayMagnifier.h

Image magnifier class.

30.1 Macros

```
#define CAPY_DISPLAY_MAGNIFIER_H
```

30.2 Enumerations

None.

30.3 Typedefs

None.

30.4 Struct CapyDisplayMagnifier

30.4.1 Struct CapyDisplayMagnifier's properties

```
CapyVecDef(2, x, y) pos;
```

Position

```
double invScale;
```

Scale, memorise the inverse of the scale to avoid using division in the conversion method (supposedly more often used than the scale update methods)

30.4.2 Struct CapyDisplayMagnifier's methods

```
void (*destruct)(void);
```

Destructor

```
void (* reset)(void);
```

Reset the magnifier (position to origin, scale to 1.0)

```
void (* moveTo)(CapyVec const* const pos);
```

Move the magnifier to a position

Input argument(s):

- pos: The position where to move the magnifier


```
void (* translate)(CapyVec const* const v);
```

Translate the magnifier by a vector

Input argument(s):

- v: The vector to add to the current position of the magnifier

```
void (*scaleUp)(  
    double const coeff,  
    CapyVec const* const fixedPos);
```

Scale up the magnification at a given speed

Input argument(s):

- coeff: the speed by which the scale is scaled up (equivalent to
- $\text{scale} *= 1.0 + \text{coeff}$)
- fixedPos: if not null, the magnifier is translated such as the fixed
- position appears at the same location on the display before
- and after scaling. (fixedPos in display coordinates)

```
void (*scaleDown)(  
    double const coeff,  
    CapyVec const* const fixedPos);
```

Scale down the magnification at a given speed

Input argument(s):

- coeff: the speed by which the scale is scaled down (equivalent to
- $\text{scale} *= 1.0 - \text{coeff}$)
- fixedPos: if not null, the magnifier is translated such as the fixed
- position appears at the same location on the display before
- and after scaling. (fixedPos in display coordinates)

```
CapyImgPos (*demagnifyCoord)(CapyImgPos const* const coord);
```

Convert coordinates from magnified to demagnified

Input argument(s):

- coord: the coordinates to convert

Output and side effect(s):

- Return the converted coordinates. Don't care about overflow if the
- converted coordinates become negative, use appropriately. The conversion
- formula is
- $\text{coordImg} = \text{coordDisplay} / \text{scale} - \text{translate}$

30.5 Functions

```
CapyDisplayMagnifier CapyDisplayMagnifierCreate(void);
```

Create a CapyDisplayMagnifier

Output and side effect(s):

- Return a CapyDisplayMagnifier

```
CapyDisplayMagnifier* CapyDisplayMagnifierAlloc(void);
```

Allocate memory for new CapyDisplayMagnifier

Output and side effect(s):

- Return a newly allocated CapyDisplayMagnifier

Exception(s):

- May raise CapyExc_MallocFailed

```
void CapyDisplayMagnifierFree(CapyDisplayMagnifier** that);
```

Free the memory used by a CapyDisplayMagnifier* and reset '*that' to NULL

Input argument(s):

- display: the CapyDisplayMagnifier to free

31 Unit distribution.h

Statistic distribution classes.

31.1 Macros

```
#define CAPY_DISTRIBUTION_H
```

```
#define CopyDistEvtDef struct { \
    CopyVec vec;                \
    void* ptr;                  \
    size_t id;                  \
}
```

Definition of an event for a distribution (to abstract from continuous and discrete) Vector to identify a continuous event CopyVec vals;

Pointer toward a user data structure to identify a discrete event void* ptr;

Id to identify a discrete event size_t id;

```
#define CopyDistDef struct { \
    CopyDistType type;        \
    CopyPad(CopyDistType, 0); \
    void (*destruct)(void);    \
    double (*getProbability)(CopyDistEvt const* const evt); \
    double (*getSurprise)(CopyDistEvt const* const evt); \
    double (*getEntropy)(void); \
    bool (*isEvtInMostProbable)( \
        CopyDistEvt const* const evt, \
        double const threshold); \
}
```

Definition of a CopyDist class (virtual parent class for all the distribution)

Type of the distribution CopyDistType type;

Get the probability of a given event.

Input argument(s):

- evt: the event

Output and side effect(s):

- Return the probability of the event
- double (*getProbability)(CopyDistEvt const* const evt);
- Get the surprise of a given event.

Input argument(s):

- evt: the event

Output and side effect(s):

- Return the surprise of the event ($h(e) = \log(1/p(e))$). The higher the
- surprise the less probable is the event.
- `double (*getSurprise)(CapyDistEvt const* const evt);`
- Get the entropy of the distribution.

Output and side effect(s):

- Return the entropy (average of the surprise). The higher the entropy the
- more uncertain the outcome of drawing a random sample.
- `double (*getEntropy)(void);`
- Check if an event is within the most probable up to a given threshold

Input argument(s):

- `evt`: the event to check
- `threshold`: the threshold

Output and side effect(s):

- Return true if the event is in the most probable events up to the
- threshold.
- `bool (*isEvtInMostProbable)(`
- `CapyDistEvt const* const evt,`
- `double const threshold);`

```
#define CapyDistContinuousDef struct { \
    CapyDistDef; \
    void (*destructCapyDist)(void); \
    size_t dimEvt; \
    CapyRangeDouble* range; \
}
```

Definition of a `CapyDistContinuous` class (probability distribution for continuous random variables)

`dimEvt`: the number of dimension of an event range: the range of possible values for each dimension of an event

31.2 Enumerations

```
typedef enum CapyDistType {  
    capyDistributionType_continuous,  
    capyDistributionType_normal,  
    capyDistributionType_discrete,  
} CapyDistType;
```

Type of distributions

31.3 Typedefs

```
typedef CapyDistEvtDef CapyDistEvt;
```

Event for a distribution

```
typedef CapyDistDef CapyDist;
```

CapyDist class

```
typedef CapyDistContinuousDef CapyDistContinuous;
```

CapyDistContinuous class

31.4 Struct CapyDistNormal

31.4.1 Struct CapyDistNormal's properties

```
CapyDistContinuousDef;
```

```
double* mean;
```

Mean and standard deviation (aka sigma) for each dimension The event variable is considered isotropic (dimensions are uncorrelated)

```
double* stdDev;
```

31.4.2 Struct CapyDistNormal's methods

```
void (*destructCapyDistContinuous)(void);
```


Destructor

```
double (*getProbabilityDerivative)(  
    CpyDistEvt const* const evt,  
    size_t const iAxis);
```

Get the derivative of the probability of a given event along a given axis.

Input argument(s):

- evt: the event
- iAxis: the derivative axis

Output and side effect(s):

- Return the derivative of the probability of the event

31.5 Struct CpyDistDiscreteOccurence

31.5.1 Struct CpyDistDiscreteOccurence's properties

```
double prob;
```

Probability of the event

```
CpyDistEvt evt;
```

Event definition

31.5.2 Struct CpyDistDiscreteOccurence's methods

None.

31.6 Struct CpyDistDiscrete

31.6.1 Struct CpyDistDiscrete's properties

```
CpyDistDef;
```

Parent class

```
CpyDistDiscreteOccs* occ;
```

Occurrences defining the distribution

31.6.2 Struct CopyDistDiscrete's methods

```
void (*destructCopyDist)(void);
```

Destructor

31.7 Functions

```
CopyDistEvt CopyDistEvtCreate(void);
```

Create a CopyDistEvt

Output and side effect(s):

- Return a CopyDistEvt.

```
CopyDist CopyDistCreate(CopyDistType const type);
```

Create a CopyDist

Input argument(s):

- type: type of ditribution

Output and side effect(s):

- Return a CopyDist

```
CopyDistContinuous CopyDistContinuousCreate(size_t const dimEvt);
```

Create a CopyDistContinuous

Input argument(s):

- dimEvt: the dimension of the random variable describing an event

Output and side effect(s):

- Return a CopyDistContinuous

```
CopyDistNormal CopyDistNormalCreate(  
    size_t const dimEvent,  
    double const* const mean,  
    double const* const stdDev);
```


Create a CpyDistNormal

Input argument(s):

- dimEvent: the dimension of the random variable describing an event
- mean: the means in each dimension of the random variable
- stdDev: the standard deviations in each dimension of the random
- variable

Output and side effect(s):

- Return a CpyDistNormal

```
CpyDistNormal* CpyDistNormalAlloc(  
    size_t const dimEvent,  
    double const* const mean,  
    double const* const stdDev);
```

Allocate memory for a new CpyDistNormal and create it

Input argument(s):

- dimEvent: the dimension of the random variable describing an event
- mean: the means in each dimension of the random variable
- stdDev: the standard deviations in each dimension of the random
- variable

Output and side effect(s):

- Return a CpyDistNormal

```
void CpyDistNormalFree(CpyDistNormal** const that);
```

Free the memory used by a CpyDistNormal

Input argument(s):

- that: the CpyDistNormal to free

```
CpyDistDiscrete CpyDistDiscreteCreate(size_t const nbEvt);
```


Create a CpyDistDiscrete

Input argument(s):

- nbEvt: the number of events in the distribution

Output and side effect(s):

- Return a CpyDistDiscrete

```
CpyDistDiscrete* CpyDistDiscreteAlloc(size_t const nbEvt);
```

Allocate memory for a CpyDistDiscrete and create a CpyDistDiscrete

Input argument(s):

- nbEvt: the number of events in the distribution

Output and side effect(s):

- Return a CpyDistDiscrete

```
void CpyDistDiscreteFree(CpyDistDiscrete** const that);
```

Free the memory used by a CpyDistDiscrete

Input argument(s):

- that: the CpyDistDiscrete to free

```
double CpyDistDiscreteGetCrossEntropy(  
    CpyDistDiscrete const* const distA,  
    CpyDistDiscrete const* const distB);
```

Get the cross entropy of two discrete distributions

Input argument(s):

- distA: the first distribution
- distB: the second distribution

Output and side effect(s):

- Return the cross entropy of distA relative to distB. It is higher or equal to the entropy of distA, and increase with the discrepancy between

- the probabilities of the two distributions.

```
double CapyDistDiscreteGetKLDivergence(
    CapyDistDiscrete const* const distA,
    CapyDistDiscrete const* const distB);
```

Get the KL divergence of two discrete distributions

Input argument(s):

- distA: the first distribution
- distB: the second distribution

Output and side effect(s):

- Return the KL divergence of distA relative to distB (equals to cross
- entropy of (distA, distB) minus entropy of distA. If the distribution are
- the same it returns 0.0. The more they diverge the higher the returned
- value.

32 Unit elo.h

ELO ranking class.

32.1 Macros

```
#define CAPY_ELO_H
```

```
#define CAPY_ELO_K 16.0
```

Default K coefficient

32.2 Enumerations

```
typedef enum CapyEloResult {
    capyElo_loose = 0,
    capyElo_tie = 1,
    capyElo_win = 2,
} CapyEloResult;
```

Type of results

32.3 Typedefs

None.

32.4 Struct CapyElo

32.4.1 Struct CapyElo's properties

```
double k;
```

K coefficient

32.4.2 Struct CapyElo's methods

```
void (*destruct)(void);
```

Destructor

```
double (*get)(
    double const eloA,
    CapyEloResult const result,
    double const eloB);
```

Get the Elo delta given two Elos and a result. This delta is to be added to the first Elo and subtracted to the second.

Input argument(s):

- eloA: first Elo
- result: result of eloA against eloB (e.g., if result is 'win' it means that eloA has won over eloB)
- eloB: second Elo

Output and side effect(s):

- Return the Elo delta to be added to eloA and subtracted to eloB

32.5 Functions

```
CapyElo CapyEloCreate(void);
```

Create a CapyElo

Output and side effect(s):

- Return a CapyElo

```
CapyElo* CapyEloAlloc(void);
```

Allocate memory for a new CapyElo and create it

Output and side effect(s):

- Return a CapyElo

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyEloFree(CapyElo** const that);
```

Free the memory used by a CapyElo* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyElo to free

33 Unit externalHeaders.h

Include all the standard librairies used in LibCapy.

33.1 Macros

None.

33.2 Enumerations

None.

33.3 Typedefs

```
typedef struct timeval Timeval;
```

Typedef-ing struct timeval

```
typedef struct timespec Timespec;
```

Typedef-ing struct timespec

```
typedef struct stat FileStat;
```

Typedef-ing struct stat

33.4 Functions

None.

34 Unit feistelCipher.h

Feistel ciphering class.

34.1 Macros

```
#define CAPY_FEISTEL_CIPHER_H
```

```
#define CopyFeistelRoundFunDef struct { \
    void (*destruct)(void);           \
    void (* run)(                      \
        CipherData_t const* const key, \
        CipherData_t const* const block, \
        CipherData_t* const output); \
}
```

Definition of the round function parent class. Round functions passed to CopyFeistelCipher methods must inherits this class. It has a dummy 'run' method for test purpose (which simply xor its input with the key). The inheriting class should replace it with its own round function in its constructor. The 'run' method is called by CopyFeistelCipher at each round with arguments: key: a null terminated string, the key to be used by the round function block: the block of data to which apply the round function sizeData: the size in byte of the block of data output: the adress in memory where to

write the output of the round function (must be already allocated and have same size as 'block')

34.2 Enumerations

```
typedef enum CapyFeistelCipherMode {  
    capyFeistelCipher_ECB,  
    capyFeistelCipher_CBC,  
    capyFeistelCipher_CTR,  
} CapyFeistelCipherMode;
```

Operation modes of the Feistel network

34.3 Typedefs

```
typedef CapyArrChar CipherData_t;
```

Typedef for the keys

```
typedef CapyListArrChar CipherKeys_t;
```

Typedef for the list of keys

```
typedef CapyFeistelRoundFunDef CapyFeistelRoundFun;
```

Round function parent class

```
typedef uint64_t CipherCounter_t;
```

Type of the counter for CTR mode

34.4 Struct CapyFeistelCipher

34.4.1 Struct CapyFeistelCipher's properties

```
CapyFeistelCipherMode opMode;
```

Operation mode

```
CapyPad(CapyFeistelCipherMode, opMode);
```

```
CipherData_t* initVector;
```


Initialisation vector

```
CipherCounter_t counter;
```

Counter for CTR mode

34.4.2 Struct CpyFeistelCipher's methods

```
void (*destruct)(void);
```

Destructor

```
CipherData_t* (*cipher)(  
    CpyFeistelRoundFun* const roundFun,  
    CipherKeys_t const* const keys,  
    CipherData_t const* const data);
```

Cipher function

Input argument(s):

- roundFun: the round function used to cipher
- keys: the keys used to cipher, a list of null terminated strings
- data: the data to cipher

Output and side effect(s):

- Return a newly allocated array of 'sizeData' bytes containing the
- ciphered data.
- Exceptions:
- May raise CpyExc_MallocFailed

```
CipherData_t* (*decipher)(  
    CpyFeistelRoundFun* const roundFun,  
    CipherKeys_t const* const keys,  
    CipherData_t const* const data);
```

Decipher function

Input argument(s):

- roundFun: the round function used to decipher

- keys: the keys used to decipher, a list of null terminated strings
- data: the data to decipher

Output and side effect(s):

- Return a newly allocated array of 'sizeData' bytes containing the deciphered data.
- Exceptions:
- May raise CpyExc_MallocFailed

```
void (*setInitVector)(CipherData_t const* const initVector);
```

Set the initialisation vector for CBC and CTR mode

Input argument(s):

- initVector, the initialisation vector
- Exceptions:
- May raise CpyExc_MallocFailed

34.5 Functions

```
CapyFeistelRoundFun CapyFeistelRoundFunCreate(void);
```

Create a CapyFeistelRoundFun

Output and side effect(s):

- Return a CapyFeistelRoundFun

```
CapyFeistelCipher CapyFeistelCipherCreate(CapyFeistelCipherMode const opMode);
```

Create a CapyFeistelCipher

Input argument(s):

- opMode: operation mode

Output and side effect(s):

- Return a CpyFeistelCipher

Exception(s):

- May raise CpyExc_MallocFailed, CpyExc_UndefinedExecution.

```
CpyFeistelCipher* CpyFeistelCipherAlloc(CpyFeistelCipherMode const opMode);
```

Allocate memory for a new CpyFeistelCipher and create it

Input argument(s):

- opMode: operation mode

Output and side effect(s):

- Return a CpyFeistelCipher

Exception(s):

- May raise CpyExc_MallocFailed, CpyExc_UndefinedExecution.

```
void CpyFeistelCipherFree(CpyFeistelCipher** const that);
```

Free the memory used by a CpyFeistelCipher* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyFeistelCipher to free

35 Unit fft.h

Class and structure implementing DFT and FFT.

35.1 Macros

```
#define CAPY_FFT_H
```

35.2 Enumerations

None.

35.3 Typedefs

None.

35.4 Struct CapyDFTCoeffs

35.4.1 Struct CapyDFTCoeffs's properties

```
struct CapyMathFunDef;
```

Inherits CapyMathFun

```
size_t nb;
```

Number of coefficients

```
CapyRangeDouble range;
```

Original input range of the transformed function (default: [0,1])

```
double complex* vals;
```

Values of the Fourier series coefficients, these are the amplitudes of the cosine (real part) and sine (imaginary part) for each "frequency bin". The k-th frequency bin corresponds to the sine and cosine of frequency k. The original function is then approximated by $f(t) = a_0/2 + \sum_{k=0}^{(N-1)} (a_k \cos(2\pi kt) + b_k \sin(2\pi kt))$

35.4.2 Struct CapyDFTCoeffs's methods

```
void (*destructCapyMathFun)(void);
```

Destructor

```
CapyVec (*getAmpFreqBins)(bool const fold);
```

Get the vector of amplitude (magnitude of the complex number) per frequency bin (aka spectrum plot). Inputs: fold: if true, fold the symmetric frequency and return only half of the bins.

Output and side effect(s):

- Return the amplitude per frequency bins as a vector. It is symmetric
- relative to that->nb/2. One can get the single sided Fourier coefficients
- by taking only the first half values multiplied by two.

35.5 Struct CpyDFT

35.5.1 Struct CpyDFT's properties

None.

35.5.2 Struct CpyDFT's methods

```
void (*destruct)(void);
```

Destructor

```
CpyDFTCoeffs (*fftPolyFromCoeffToVal)(CpyPolynomial1D const* const poly)
;
```

Transform a 1D polynomial from coefficients representation to values representation. The polynomial must have a power-of-2 number of coefficients.

Input argument(s):

- poly: the polynomial

Output and side effect(s):

- Return the value representation of the polynomial as a CpyDFTCoeffs

```
CpyPolynomial1D* (*fftPolyFromValToCoeff)(CpyDFTCoeffs const* const
values);
```

Transform a 1D polynomial from values representation to coefficients representation. The number of values must be a power-of-2.

Input argument(s):

- values: the values representation of the polynomial

Output and side effect(s):

- Return the polynomial corresponding to the value representation

```
CpyDFTCoeffs (*fftFun)(
    CpyMathFun* const fun,
    CpyRangeDouble const* const range,
    size_t const nbSample);
```

Calculate the Discrete Fourier Transform for an input range of a given function

Input argument(s):

- fun: the function (must have one input and one output)
- range: range of the input of the function for which the DFT is calculated
- nbSample: the number of samples taken from the function, equally spaced
- in 'range' (bounds included)

Output and side effect(s):

- Return the DFT coefficients as a CpyDFTCoeffs

```
CpyDFTCoeffs (*fftSamples)(CpyVec const* const samples);
```

Calculate the Discrete Fourier Transform for a given set of samples

Input argument(s):

- samples: the samples value

Output and side effect(s):

- Return the DFT coefficients as a CpyDFTCoeffs

35.6 Struct CpyDFT2DCoeffs

35.6.1 Struct CpyDFT2DCoeffs's properties

```
struct CpyMathFunDef;
```

Inherits CpyMathFun

```
size_t nb[2];
```

Number of coefficients

```
CpyRangeDouble range[2];
```

Original input range of the transformed function (default: [0,1])

```
double complex* vals;
```

Values of the Fourier series coefficients, stored by rows

35.6.2 Struct CpyDFT2DCoeffs's methods

```
void (*destructCpyMathFun)(void);
```

Destructor

```
CpyImg* (*toAmplitudeImg)(  
    bool const center,  
    bool const logScale);
```

Convert the coefficients into a visualisation of the amplitude Inputs: center: if true, center the null frequency logScale: if true, apply logarithmic scaling

Output and side effect(s):

- Return a normalised greyscale image.

```
CpyImg* (*toPhaseImg)(bool const center);
```

Convert the coefficients into a visualisation of the phase Inputs: center: if true, center the null frequency

Output and side effect(s):

- Return a normalised greyscale image.

```
CpyImg* (*toPeriodogramImg)(  
    bool const center,  
    bool const logScale);
```

Convert the coefficients into a periodogram Inputs: center: if true, center the null frequency logScale: if true, apply logarithmic scaling

Output and side effect(s):

- Return a normalised greyscale image.

35.7 Struct CpyDFT2D

35.7.1 Struct CpyDFT2D's properties

None.

35.7.2 Struct CpyDFT2D's methods

```
void (*destruct)(void);
```

Destructor

```
CpyDFT2DCoeffs (*fftImage)(CpyImg const* const img);
```

Calculate the 2D Discrete Fourier Transform for a given image

Input argument(s):

- img: the img to transform

Output and side effect(s):

- Return the DFT coefficients as a CpyDFT2DCoeffs

35.8 Functions

```
CpyDFTCoeffs CpyDFTCoeffsCreate(size_t const nb);
```

Create a CpyDFTCoeffs

Input argument(s):

- nb: the number of coefficients

Output and side effect(s):

- Return a CpyDFTCoeffs

```
CpyDFTCoeffs* CpyDFTCoeffsAlloc(size_t const nb);
```

Allocate memory for a new CpyDFTCoeffs and create it

Input argument(s):

- nb: the number of coefficients

Output and side effect(s):

- Return a CpyDFTCoeffs

```
void CpyDFTCoeffsFree(CpyDFTCoeffs** const that);
```


Free the memory used by a CpyDFTCoeffs* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyDFTCoeffs to free

```
CpyDFT CpyDFTCreate(void);
```

Create a CpyDFT

Output and side effect(s):

- Return a CpyDFT

```
CpyDFT* CpyDFTAlloc(void);
```

Allocate memory for a new CpyDFT and create it

Output and side effect(s):

- Return a CpyDFT

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyDFTFree(CpyDFT** const that);
```

Free the memory used by a CpyDFT* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyDFT to free

```
CpyDFT2DCoeffs CpyDFT2DCoeffsCreate(size_t const nb[2]);
```

Create a CpyDFT2DCoeffs

Input argument(s):

- nb: the number of coefficients

Output and side effect(s):

- Return a CpyDFT2DCoeffs


```
CapyDFT2DCoeffs* CapyDFT2DCoeffsAlloc(size_t const nb[2]);
```

Allocate memory for a new CapyDFT2DCoeffs and create it

Input argument(s):

- nb: the number of coefficients

Output and side effect(s):

- Return a CapyDFT2DCoeffs

```
void CapyDFT2DCoeffsFree(CapyDFT2DCoeffs** const that);
```

Free the memory used by a CapyDFT2DCoeffs* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyDFT2DCoeffs to free

```
CapyDFT2D CapyDFT2DCreate(void);
```

Create a CapyDFT2D

Output and side effect(s):

- Return a CapyDFT2D

```
CapyDFT2D* CapyDFT2DAlloc(void);
```

Allocate memory for a new CapyDFT2D and create it

Output and side effect(s):

- Return a CapyDFT2D

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyDFT2DFree(CapyDFT2D** const that);
```

Free the memory used by a CapyDFT2D* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyDFT2D to free

36 Unit fileFormat.h

Virtual parent class for all file format support.

36.1 Macros

```
#define CAPY_FILEFORMAT_H
```

```
#define CapyFileFormatDef struct {
    CapyFileFormatType type;
    CapyPad(CapyFileFormatType, 0);
    char const* lbl;
    void (*destruct)(void);
    CapyImg* (*loadImg)(CapyStreamIo* const file);
    void (*saveImg)(
        CapyImg const* const img,
        CapyStreamIo* const file);
    CapyPointCloud* (*loadPointCloud)(CapyStreamIo* const file);
    void (*savePointCloud)(
        CapyPointCloud const* const pointCloud,
        CapyStreamIo* const file);
}
```

CapyFileFormat class definition macro. Type of the format CapyFileFormatType type; Human readable type char const* lbl; Given an object of type <T>, Load an instance of the object from a file

Input argument(s):

- file: the file (as an opened CapyStreamIo)

Output and side effect(s):

- Return the instance of the object.
- Exceptions:
- May raise CapyExc_StreamReadError, CapyExc_UnsupportedFormat,
- CapyExc_InvalidStream
- <T>* (*load<T>)(CapyStreamIo* const file);
- Save an instance of the object to a file

Input argument(s):

- obj: the instance to save

- file: the file (as an opened CpyStreamIo)
- Exceptions:
- May raise CpyExc_StreamWriteError, CpyExc_UnsupportedFormat
- void (*save<T>)(
- <T> const* const obj,
- CpyStreamIo* const file);

36.2 Enumerations

```
typedef enum CpyFileFormatType {
    cpyFileFormat_png,
    cpyFileFormat_ply,
    cpyFileFormat_nb,
} CpyFileFormatType;
```

Supported file formats

36.3 Typedefs

```
typedef CpyFileFormatDef CpyFileFormat;
```

CpyFileFormat object

36.4 Functions

```
CpyFileFormat CpyFileFormatCreate(CpyFileFormatType const type);
```

Create a CpyFileFormat

Input argument(s):

- type: the type of the format

Output and side effect(s):

- Return a CpyFileFormat

37 Unit floodFill.h

Flood fill algorithm.

37.1 Macros

```
#define CAPY_FLOODFILL_H
```

37.2 Enumerations

None.

37.3 Typedefs

None.

37.4 Struct CapyFloodFill

37.4.1 Struct CapyFloodFill's properties

None.

37.4.2 Struct CapyFloodFill's methods

```
void (*destruct)(void);
```

Destructor

```
CapyColorPatches* (*segmentImg)(
    CapyImg const* const imgSrc,
    CapyImg const* const imgCond,
    CapyFloodConditionImgFun floodCondition);
```

Segment an image by flood filling it.

Input argument(s):

- imgSrc: the image to segment
- imgCond: the image one which the flooding condition is checked
- floodCondition: the flood condition function to control flooding

Output and side effect(s):

- Return a CapyColorPatches, one CapyColorPatch per segment in the image
- after flood filling.

37.5 Functions

```
CapyFloodFill CapyFloodFillCreate(void);
```

Create a CapyFloodFill

Output and side effect(s):

- Return a CapyFloodFill

```
CapyFloodFill* CapyFloodFillAlloc(void);
```

Allocate memory for a new CapyFloodFill and create it

Output and side effect(s):

- Return a CapyFloodFill

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyFloodFillFree(CapyFloodFill** const that);
```

Free the memory used by a CapyFloodFill* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyFloodFill to free

38 Unit font.h

Font class.

38.1 Macros

```
#define CAPY_FONT_H
```

38.2 Enumerations

None.

38.3 Typedefs

None.

38.4 Struct CapyFont

38.4.1 Struct CapyFont's properties

```
CapyBezierSpline** dict;
```

Font dictionary (array of 128 Bezier splines for the 128 ASCII characters, null for undefined characters, ordered accordingly to ascii code)

```
CapyVec scale;
```

Scale (default (1,1))

```
CapyVec forward;
```

Forward vector of the text (default (1,0))

```
CapyVec spacing;
```

Spacing of letter (default (1,1))

38.4.2 Struct CapyFont's methods

```
void (*destruct)(void);
```

Destructor

```
CapyListBezierSpline* (*textToBezierSpline)(char const* const text);
```

Convert a string to a CapyListBezierSpline

Input argument(s):

- text: the text to convert

Output and side effect(s):

- Return a list of Bezier spline.

38.5 Functions

```
CapyFont CapyFontCreate(void);
```

Create a CapyFont

Output and side effect(s):

- Return a CapyFont

```
CapyFont* CapyFontAlloc(void);
```

Allocate memory for a new CapyFont and create it

Output and side effect(s):

- Return a CapyFont

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyFontFree(CapyFont** const that);
```

Free the memory used by a CapyFont* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyFont to free

39 Unit frequentistHypothesisTesting.h

Class implementing the frequentist hypothesis testing method.

39.1 Macros

```
#define CAPY_FREQUENTIST_HYPOTHESIS_METHOD_H
```

39.2 Enumerations

None.

39.3 Typedefs

None.

39.4 Struct CpyFHT

39.4.1 Struct CpyFHT's properties

```
double statisticalPower;
```

Confidence that an event matches the null hypothesis (in $[0,1]$)

```
double pValue;
```

Confidence that an event doesn't match the alternate hypothesis (in $[0,1]$)

```
CpyDist const* nullHypo;
```

Distribution of the null hypothesis

```
CpyDist const* altHypo;
```

Distribution of the alternate hypothesis

39.4.2 Struct CpyFHT's methods

```
void (*destruct)(void);
```

Destructor

```
bool (*isEvtMatchingNullHypo)(CpyDistEvt const* const evt);
```

Check if an event matches the null hypothesis

Input argument(s):

- evt: the event

Output and side effect(s):

- Return true if the event matches, else false.

39.5 Functions

```
CapyFHT CapyFHTCreate(  
    double const statisticalPower,  
    double const pValue,  
    CapyDist const* const nullHypo,  
    CapyDist const* const altHypo);
```

Create a CapyFHT

Input argument(s):

- statisticalPower: confidence that an event doesn't match the alternate
- hypothesis (in [0,1])
- pValue: confidence that an event matches the null hypothesis
- (in [0,1])
- nullHypo: distribution of the null hypothesis
- altHyp: distribution of the alternate hypothesis

Output and side effect(s):

- Return a CapyFHT

```
CapyFHT* CapyFHTAlloc(  
    double const statisticalPower,  
    double const pValue,  
    CapyDist const* const nullHypo,  
    CapyDist const* const altHypo);
```

Allocate memory for a new CapyFHT and create it.

Input argument(s):

- statisticalPower: confidence that an event doesn't match the alternate
- hypothesis (in [0,1])
- pValue: confidence that an event matches the null hypothesis
- (in [0,1])
- nullHypo: distribution of the null hypothesis
- altHyp: distribution of the alternate hypothesis

Output and side effect(s):

- Return a CpyFHT

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyFHTFree(CpyFHT** const that);
```

Free the memory used by a CpyFHT* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyFHT to free

40 Unit galeshapleypairing.h

GaleShapleyPairing class.

40.1 Macros

```
#define CAPY_GALESHAPLEYPAIRING_H
```

40.2 Enumerations

None.

40.3 Typedefs

None.

40.4 Struct CpyGaleShapleyPairing

40.4.1 Struct CpyGaleShapleyPairing's properties

None.

40.4.2 Struct CappyGaleShapleyPairing's methods

```
void (*destruct)(void);
```

Destructor

```
size_t* (*run)(  
    CappyMat const* const weightA,  
    CappyMat const* const weightB);
```

Find the best pairing for elements in two sets A and B of same size according to the Gale-Shapley algorithm.

Input argument(s):

- weightA: matrix representing the pairing preferences of elements in
- set A. The value of the i-th row and j-th column is equal to
- the preference of the i-th element in set A to be paired with
- the j-th element in set B. The higher the stronger the
- preference.
- weightB: matrix representing the pairing preferences of elements in
- set B. The value of the i-th row and j-th column is equal to
- the preference of the i-th element in set B to be paired with
- the j-th element in set A. The higher the stronger the
- preference.

Output and side effect(s):

- Return an array of indices describing the pairing. The i-th element j of
- the array indicates the i-th element in the set A is paired with the
- the j-th element in the set B. Note that the pairing is biased toward
- preferences of set A.

40.5 Functions

```
CapyGaleShapleyPairing CapyGaleShapleyPairingCreate(void);
```

Create a CapyGaleShapleyPairing

Output and side effect(s):

- Return a CapyGaleShapleyPairing

```
CapyGaleShapleyPairing* CapyGaleShapleyPairingAlloc(void);
```

Allocate memory for a new CapyGaleShapleyPairing and create it

Output and side effect(s):

- Return a CapyGaleShapleyPairing

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyGaleShapleyPairingFree(CapyGaleShapleyPairing** const that);
```

Free the memory used by a CapyGaleShapleyPairing* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyGaleShapleyPairing to free

41 Unit geomap.h

GeoMap class.

41.1 Macros

```
#define CAPY_GEOMAP_H
```

```
#define CAPY_EARTH_CIRCUMFERENCE 40075017.0
```

Approximate circumference of earth at the equator in meter

```
#define CAPY_EARTH_ONE_DEGREE_DISTANCE 111319.49166666667
```

Approximate distance in meter corresponding to one degree on the surface of earth at the equator

41.2 Enumerations

None.

41.3 Typedefs

```
typedef double CpyLongLat_t;
```

Type for latitude (north-south position in degree from the equator) and longitude (east-west position in degree from a reference meridian)

41.4 Struct CpyGeoMapCoord

41.4.1 Struct CpyGeoMapCoord's properties

```
union {
```

```
    CpyLongLat_t vals[2];
```

```
    struct __attribute__((packed)) {CpyLongLat_t longitude, latitude;};
```

```
};
```

41.4.2 Struct CpyGeoMapCoord's methods

None.

41.5 Struct CpyGeoMap

41.5.1 Struct CpyGeoMap's properties

```
CpyImg* img;
```

Image of the map (default: none). Assuming North at the top, South at the bottom, West at the left, East at the right, Mercator projection.

```
union {
```


Range of latitude and longitude (default: [0, 0])

```
CapyRangeLongLat rangeLongLat[2];
```

```
struct __attribute__((packed)) {
```

```
CapyRangeLongLat rangeLongitude, rangeLatitude;
```

```
};
```

```
};
```

41.5.2 Struct CapyGeoMap's methods

```
void (*destruct)(void);
```

Destructor

```
void (*loadImgFromPath)(char const* const path);
```

Load the image of the map from a path.

Input argument(s):

- path: the path to the image

Output and side effect(s):

- The image is loaded. If a previous image existed it is freed.

```
void (*setLatitudeRange)(CapyRangeLongLat const range);
```

Set the latitude range.

Input argument(s):

- range: the range

Output and side effect(s):

- The latitude range is updated.


```
void (*setLongitudeRange)(CapyRangeLongLat const range);
```

Set the longitude range.

Input argument(s):

- range: the range

Output and side effect(s):

- The longitude range is updated.

```
CapyImgPos (*getImgPos)(CapyGeoMapCoord const coord);
```

Get the position in the image for a given latitude and longitude

Input argument(s):

- coord: the latitude and longitude

Output and side effect(s):

- Return a CapyImgPos.

```
CapyGeoMapCoord (*getImgCoord)(CapyImgPos const pos);
```

Get the latitude and longitude for a given position in the image

Input argument(s):

- pos: the position in the image

Output and side effect(s):

- Return a CapyGeoMapCoord.

```
double (*convertMeterToPixel)(double const distMeter);
```

Convert a distance in meter to a distance in pixel in the image

Input argument(s):

- distMeter: the distance in meter

Output and side effect(s):

- Return the distance in pixel, assuming the same scale along latitude and

- longitude.

```
double (*convertMeterToDegree)(double const distMeter);
```

Convert a distance in meter to a distance in degree

Input argument(s):

- distMeter: the distance in meter

Output and side effect(s):

- Return the distance in degree, assuming the earth is a perfect sphere.

41.6 Functions

```
CapyGeoMap CapyGeoMapCreate(void);
```

Create a CapyGeoMap

Output and side effect(s):

- Return a CapyGeoMap

```
CapyGeoMap* CapyGeoMapAlloc(void);
```

Allocate memory for a new CapyGeoMap and create it

Output and side effect(s):

- Return a CapyGeoMap

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyGeoMapFree(CapyGeoMap** const that);
```

Free the memory used by a CapyGeoMap* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyGeoMap to free

42 Unit geometricShape.h

Basic geometric shape classes.

42.1 Macros

```
#define CAPY_GEOMETRICSHAPE_H
```

```
#define CapyGeometry2D_ struct {
    void (*destruct)(void);
    double (*getArea)(void);
    double (*getPerimeter)(void);
    CapyBezierSpline (*getBezierSpline)(void);
}
```

Definition of the parent class for 2D geometric shapes Destructor void (*destruct)(void);

Get the area of the geometry

Output and side effect(s):

- Return the area
- double (*getArea)(void);
- Get the perimeter of the geometry

Output and side effect(s):

- Return the perimeter
- double (*getPerimeter)(void);
- Convert the geometry into a Bezier spline

Output and side effect(s):

- Return the Bezier spline approximating the geometry.
- CapyBezierSpline (*getBezierSpline)(void);

```
#define CAPY_NB_MAX_INTER_POINT_GEOMETRY 2
```

Maximum number of intersecting points in a CapyGeometry2DIntersection

42.2 Enumerations

None.

42.3 Typedefs

```
typedef CapyGeometry2D_ CapyGeometry2D;
```

Parent class for 2D geometric shapes

42.4 Struct CapyPoint2D

42.4.1 Struct CapyPoint2D's properties

```
union {
```

Location of the corner

```
double coords[2];
```

```
struct __attribute__((packed)) {double x; double y;};
```

```
};
```

42.4.2 Struct CapyPoint2D's methods

None.

42.5 Struct CapySegment

42.5.1 Struct CapySegment's properties

```
CapyGeometry2D_;
```

Inherits CapyGeometry2D

```
CapyPoint2D points[2];
```

Array of two positions, the locations of the segment extremities.

42.5.2 Struct CpySegment's methods

```
void (*destructCpyGeometry2D)(void);
```

Destructor

42.6 Struct CpyTriangle

42.6.1 Struct CpyTriangle's properties

```
CpyGeometry2D_;
```

Inherits CpyGeometry2D

```
CpyPoint2D corners[3];
```

Array of three positions, the locations of the triangle's corners.

42.6.2 Struct CpyTriangle's methods

```
void (*destructCpyGeometry2D)(void);
```

Destructor

```
void (*getBarycentricCoord)(  
    CpyPoint2D const* const pos,  
    double* const coord);
```

Get the barycentric coordinates of a position relative to the triangle

Input argument(s):

- pos: the position
- coord: the barycentric coordinates

Output and side effect(s):

- 'coord' is updated. If at least one of the three barycentric coordinates
- is negative, the position is outside the triangle. Barycentric
- coordinates in $[-inf, 1]$.

42.7 Struct CpyQuadrilateral

42.7.1 Struct CpyQuadrilateral's properties

```
CpyGeometry2D_;
```

Inherits CpyGeometry2D

```
CpyPoint2D corners[4];
```

Array of four positions, the location of the quadrilateral's corners, in clockwise order.

42.7.2 Struct CpyQuadrilateral's methods

```
void (*destructCpyGeometry2D)(void);
```

Destructor

```
void (*getBilinearCoords)(  
    double const* const pos,  
    double* const coords);
```

Get the bilinear coordinates of a point in the quadrilateral from its coordinates in world coordinate system

Input argument(s):

- pos: the position in world coordinate system
- coords: double[2] updated with the bilinear coordinates

42.8 Struct CpyRectangle

42.8.1 Struct CpyRectangle's properties

```
CpyGeometry2D_;
```

Inherits CpyGeometry2D

```
CpyPoint2D corners[2];
```

Array of two positions, the location of the quadrilateral's corners (such as corners[0].x < corners[1].x and corners[0].y < corners[1].y).

42.8.2 Struct CapyRectangle's methods

```
void (*destructCapyGeometry2D)(void);
```

Destructor

42.9 Struct CapyCircle

42.9.1 Struct CapyCircle's properties

```
CapyGeometry2D_;
```

Inherits CapyGeometry2D

```
CapyPoint2D center;
```

Center

```
double radius;
```

Radius

42.9.2 Struct CapyCircle's methods

```
void (*destructCapyGeometry2D)(void);
```

Destructor

```
CapyBezierSpline (*getArcAsBezierSpline)(  
    CapyPoint2D const* const pointA,  
    CapyPoint2D const* const pointB);
```

Get an arc of the circle as a Bezier spline

Input argument(s):

- pointA: the beginning of the arc
- pointB: the end of the arc

Output and side effect(s):

- Return a Bezier spine approximating the arc from 'pointA' to 'pointB'
- counter-clockwise. If the points are not on the circle, their projection
- on the circle is used instead.

42.10 Struct CpyGeometry2DIntersection

42.10.1 Struct CpyGeometry2DIntersection's properties

```
size_t nbPoint;
```

Number of intersection point

```
CapyPoint2D points[CAPY_NB_MAX_INTER_POINT_GEOMETRY];
```

Intersection points

42.10.2 Struct CpyGeometry2DIntersection's methods

```
void (*destruct)(void);
```

Destructor

```
void (*getIntersectSegments)(  
    CapySegment const* const segmentA,  
    CapySegment const* const segmentB,  
    bool const flagStrict);
```

Get the intersecting point between two segments Inputs: segmentA: first segment segmentB: second segment flagStrict: flag to restrict to the intersection inside the segments

Output and side effect(s):

- Update the intersection points of 'segmentA' and 'segmentB'

```
void (*getIntersectCircles)(  
    CapyCircle const* const circleA,  
    CapyCircle const* const circleB);
```

Get the intersecting point(s) between two circles Inputs: circleA: first circle circleB: second circle

Output and side effect(s):

Output and side effect(s):

- Update the intersection points of 'circleA' and 'circleB'. Points are
- ordered counter-clockwise on 'circleA' from the intersection of the

- segment connecting the center the circles

```
void (*getIntersectSegmentCircle)(
    CapySegment const* const seg,
    CapyCircle const* const circle,
    bool const flagStrict);
```

Get the intersecting point(s) between a segment and a circle Inputs: seg: the segment circle: the circle flagStrict: flag to restrict to the intersection inside the segment

Output and side effect(s):

- Update the intersection points of 'seg' and 'circle'. Points are
- ordered according to the parameter 't' of the segment.

42.11 Functions

```
CapyGeometry2D CapyGeometry2DCreate(void);
```

Create a CapyGeometry2D

Output and side effect(s):

- Return a CapyGeometry2D

```
CapySegment CapySegmentCreate(void);
```

Create a CapySegment

Output and side effect(s):

- Return a CapySegment, points initialised to ((0, 0), (0, 1))

```
CapySegment* CapySegmentAlloc(void);
```

Allocate memory for a new CapySegment and create it

Output and side effect(s):

- Return a CapySegment, points initialised to ((0, 0), (0, 1))

Exception(s):

- May raise CapyExc_MallocFailed.


```
void CpySegmentFree(CpySegment** const that);
```

Free the memory used by a CpySegment* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpySegment to free

```
CpyTriangle CpyTriangleCreate(void);
```

Create a CpyTriangle
Output and side effect(s):

- Return a CpyTriangle, corners initialised to ((0, 0), (0, 1), (1, 0))

```
CpyTriangle* CpyTriangleAlloc(void);
```

Allocate memory for a new CpyTriangle and create it
Output and side effect(s):

- Return a CpyTriangle, corners initialised to ((0, 0), (0, 1), (1, 0))

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyTriangleFree(CpyTriangle** const that);
```

Free the memory used by a CpyTriangle* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpyTriangle to free

```
CpyQuadrilateral CpyQuadrilateralCreate(void);
```

Create a CpyQuadrilateral
Output and side effect(s):

- Return a CpyQuadrilateral, corners initialised to
- ((0, 0), (0, 1), (1, 1), (1, 0))


```
CapyQuadrilateral* CapyQuadrilateralAlloc(void);
```

Allocate memory for a new CapyQuadrilateral and create it
Output and side effect(s):

- Return a CapyQuadrilateral, corners initialised to
- ((0, 0), (0, 1), (1, 1), (1, 0))

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyQuadrilateralFree(CapyQuadrilateral** const that);
```

Free the memory used by a CapyQuadrilateral* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CapyQuadrilateral to free

```
CapyRectangle CapyRectangleCreate(void);
```

Create a CapyRectangle
Output and side effect(s):

- Return a CapyRectangle, corners initialised to
- ((0, 0), (0, 1), (1, 1), (1, 0))

```
CapyRectangle* CapyRectangleAlloc(void);
```

Allocate memory for a new CapyRectangle and create it
Output and side effect(s):

- Return a CapyRectangle, corners initialised to
- ((0, 0), (0, 1), (1, 1), (1, 0))

Exception(s):

- May raise CapyExc_MallocFailed.


```
void CapyRectangleFree(CapyRectangle** const that);
```

Free the memory used by a CapyRectangle* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CapyRectangle to free

```
CapyCircle CapyCircleCreate(void);
```

Create a CapyCircle

Output and side effect(s):

- Return a CapyCircle, center initialised to (0, 0) and radius
- initialised to 1.

```
CapyCircle* CapyCircleAlloc(void);
```

Allocate memory for a new CapyCircle and create it

Output and side effect(s):

- Return a CapyCircle, center initialised to (0, 0) and radius
- initialised to 1.

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyCircleFree(CapyCircle** const that);
```

Free the memory used by a CapyCircle* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyCircle to free

```
CapyGeometry2DIntersection CapyGeometry2DIntersectionCreate(void);
```

Create a CapyGeometry2DIntersection

Output and side effect(s):

- Return a CapyGeometry2DIntersection initialised to 0

43 Unit gradientDescent.h

Class implementing the gradient descent method.

43.1 Macros

```
#define CAPY_GRADIENT_DESCENT_H
```

43.2 Enumerations

```
typedef enum CapyGradientDescentType {  
    capyGradientDescent_standard,  
    capyGradientDescent_momentum,  
    capyGradientDescent_adam,  
} CapyGradientDescentType;
```

Type of gradient descent

43.3 Typedefs

None.

43.4 Struct CapyGradientDescent

43.4.1 Struct CapyGradientDescent's properties

```
size_t iStep;
```

Current step

```
CapyVec in;
```

Current position in the input space

```
double learnRate;
```

Learn rate (default: 0.1)

```
double decayRates[2];
```


Decay rates for adam (default: [0.9, 0.999]) The higher decayRates[0], the more the recent moment is taken into account The higher decayRates[1], the more stable the estimate of the variance decayRates[1] should be as near to 1 as possible for stable convergence

```
double epsilon;
```

Epsilon for adam (default: 1e-8)

```
double momentum;
```

Momentum (default: 0)

```
CapyVec m;
```

Current first moments

```
CapyVec v;
```

Current second moments

```
CapyVec gradient;
```

Current gradient

```
CapyMathFun* objFun;
```

Reference to the objective function

43.4.2 Struct CapyGradientDescent's methods

```
void (*destruct)(void);
```

Destructor

```
void (*step)(void);
```

Perform one step of the gradient descent method

Output and side effect(s):

- that.in and that.gradient are updated

```
void (*setType)(CapyGradientDescentType const type);
```


Set the type of the gradient descent

Input argument(s):

- type: the type of gradient descent

Output and side effect(s):

- Set the type of gradient descent

43.5 Functions

```
CapyGradientDescent CapyGradientDescentCreate(  
    CapyMathFun* const objFun,  
    double const* const initIn);
```

Create a CapyGradientDescent

Input argument(s):

- objFun: objective function (must have dimOut = 1)
- initIn: initial position in input space (must be of dimension objFun.dimIn)

Output and side effect(s):

- Return a CapyGradientDescent (of standard type)

```
CapyGradientDescent* CapyGradientDescentAlloc(  
    CapyMathFun* const objFun,  
    double const* const initIn);
```

Allocate memory for a new CapyGradientDescent and create it

Input argument(s):

- objFun: objective function (must have dimOut = 1)
- initIn: initial position in input space (must be of dimension objFun.dimIn)

Output and side effect(s):

- Return a CapyGradientDescent (of standard type)

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyGradientDescentFree(CapyGradientDescent** const that);
```

Free the memory used by a CapyGradientDescent* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyGradientDescent to free

44 Unit graph.h

Graph class.

44.1 Macros

```
#define CAPY_GRAPH_H
```

44.2 Enumerations

None.

44.3 Typedefs

None.

44.4 Struct CapyGraphNode

44.4.1 Struct CapyGraphNode's properties

```
size_t id;
```

ID of the node

```
double pageRank;
```

Page rank value (default: 0.0, updated by CapyGraph.pageRank)

```
void* data;
```

User's data

44.4.2 Struct CapyGraphNode's methods

```
void (*destruct)(void);
```

Destructor

44.5 Struct CpyGraphLink

44.5.1 Struct CpyGraphLink's properties

```
CpyGraphNode* nodes[2];
```

Pointer to the two nodes of the link

```
void* data;
```

User's data

44.5.2 Struct CpyGraphLink's methods

None.

44.6 Struct CpyGraph

44.6.1 Struct CpyGraph's properties

```
CpyListGraphNode* nodes;
```

List of nodes

```
CpyListGraphLink* links;
```

List of links

```
bool directed;
```

Flag to memorise if the graph is directed (default: false)

```
CpyPad(bool, directed);
```

```
bool* connectivityMat;
```

Connectivity matrix (updated with updateConnectivityMat()) connectivityMat[i*nbNode+j] is true if there is a link from i to j

```
size_t* linkDistanceMat;
```

Link distance matrix (updated with updateLinkDistanceMat()) linkDistanceMat[i*nbNode+j] is the smallest number of steps (in term of links) from i to j, 0 means there is no path from i to j

44.6.2 Struct CapyGraph's methods

```
void (*destruct)(void);
```

Destructor

```
void (*addNode)(CapyGraphNode const node);
```

Add a node to the graph.

Input argument(s):

- node: the node to add

Exception(s):

- May return CapyExc_MallocFailed

```
void (*addLink)(CapyGraphLink const link);
```

Add a link to the graph.

Input argument(s):

- link: the link to add

Exception(s):

- May return CapyExc_MallocFailed

```
void (*linkNodes)(
    size_t const idA,
    size_t const idB);
```

Link two nodes in the graph given their id.

Input argument(s):

- idA: ID of the first node
- idB: ID of the second node

Exception(s):

- May return CapyExc_MallocFailed, CapyExc_InvalidNodeIdx


```
void (*updateConnectivityMat)(void);
```

Update the connectivity matrix of the graph

Exception(s):

- May return CpyExc_MallocFailed, CpyExc_InvalidNodeIdx

```
size_t (*getIdxOfNode)(CapyGraphNode const* const node);
```

Get the index of a node in the list of nodes of a graph

Input argument(s):

- node: the node

Output and side effect(s):

- Return the index of the node, or raise CpyExc_InvalidNodeIdx
- if the nodes wasn't found

```
void (*updateLinkDistanceMat)(void);
```

Update the link distance matrix of the graph based on the current connectivity matrix (update it with updateConnectivityMat if necessary before using this method)

Exception(s):

- May return CpyExc_MallocFailed

```
void (*pageRank)(size_t const nbIter);
```

Run the PageRank algorithm to evaluate nodes in the graph

Input argument(s):

- nbIter: number of iteration of the algorithm

Output and side effect(s):

- The pageRank properties of nodes is updated

```
void (*removeLinksFromNode)(size_t const id);
```


Remove all links whose first node is equal to the given one

Input argument(s):

- id: id of the first node of the links to be removed

Output and side effect(s):

- All links whose first node is equal to the given one are removed

```
CapyGraphNode* (*getNodeById)(size_t const id);
```

Get a node by its id

Input argument(s):

- id: the node id

Output and side effect(s):

- Return the node, or null if the given id couldn't be found

```
CapyGraphLink* (*getLinkBetweenNodes)(  
    size_t const idA,  
    size_t const idB);
```

Find a link between two nodes

Input argument(s):

- idA: the first node id
- idB: the second node id

Output and side effect(s):

- Return the link, or null if there is no link between the two nodes.
- The order of ids is irrelevant.

44.7 Functions

```
CapyGraphNode CapyGraphNodeCreate(size_t id);
```

Create a CapyGraphNode

Input argument(s):

- id: id of the node in the graph

Output and side effect(s):

- Return the node.

```
CapyGraph CapyGraphCreate(void);
```

Create a CapyGraph

Output and side effect(s):

- Return a CapyGraph

```
CapyGraph* CapyGraphAlloc(void);
```

Allocate memory for a new CapyGraph and create it

Output and side effect(s):

- Return a CapyGraph

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyGraphFree(CapyGraph** const that);
```

Free the memory used by a CapyGraph* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyGraph to free

45 Unit graphplotter.h

GraphPlotter class.

45.1 Macros

```
#define CAPY_GRAPHPLOTTER_H
```

45.2 Enumerations

```
typedef enum CapyGraphPlotterTypeCorrelation {  
    capyGraphPlotterTypeCorrelation_pearson,  
    capyGraphPlotterTypeCorrelation_distance,  
    capyGraphPlotterTypeCorrelation_nb,  
} CapyGraphPlotterTypeCorrelation;
```

Type of correlation used in plot

45.3 Typedefs

None.

45.4 Struct CapyGraphPlotterLegendData

45.4.1 Struct CapyGraphPlotterLegendData's properties

```
char const* titleAbciss;
```

Title for the abciss

```
char const* titleOrdinate;
```

Title for the ordinate

45.4.2 Struct CapyGraphPlotterLegendData's methods

None.

45.5 Struct CapyGraphPlotter

45.5.1 Struct CapyGraphPlotter's properties

```
CapyImgPos_t margin;
```

Margin (in pixel, default 10)


```
CapyImgPos_t marginLegend;
```

Margin for the legend (in pixel, default 20). Space used beside the axii for axis name and values

```
CapyPen penData;
```

Pen used to plot the data (default: default pen and blue color)

```
CapyPen penLegend;
```

Pen used to draw axii and legend (default: default pen and black)

```
union {
```

Flags for log base 10 scale

```
bool logScales[2];
```

```
struct __attribute__((packed)) {bool logScaleX, logScaleY;};
```

```
};
```

```
CapyPad(bool[2], logScales);
```

45.5.2 Struct CapyGraphPlotter's methods

```
void (*destruct)(void);
```

Destructor

```
CapyRectangle* (*getBoundingBoxDataArea)(CapyImg const* const img);
```

Get the bounding box of the area containing data (without margin, axii, etc, ...) for a given image.

Input argument(s):

- img: the image

Output and side effect(s):

- Return the coordinates (in pixel) as a CpyRectangle.

```
void (*drawLegend)(
    CpyImg* const img,
    CpyGraphPlotterLegendData const* const legend);
```

Draw the legend of a graph

Input argument(s):

- img: the image on which to draw
- legend: data about the legend

Output and side effect(s):

- The image is updated with the legend.

```
CpyImg* (*plotDensityDistributions)(
    CpyDataset const* const dataset,
    size_t const iField,
    CpyImgDims const* const dims,
    size_t const nbBin);
```

Plot the density distributions for one field from a CpyDataset.

Input argument(s):

- dataset: the CpyDataset
- iField: the index of the plotted field
- dims: the dimension of the graph
- nbBin: number of bins for the density distribution

Output and side effect(s):

- Return a CpyImg. The ordinate corresponds to the bins from the min to the
- max value of the plotted field and the absciss corresponds to the number
- of records in the dataset for that bin.


```

CapyImg* (*plotDensityDistributionsGivenCatValue)(
    CapyDataset const* const dataset,
        size_t const iCatField,
        char const* const valCatField,
        size_t const iField,
    CapyImgDims const* const dims,
        size_t const nbBin);

```

Plot the density distributions for one categorical and one other field from a CapyDataset.

Input argument(s):

- dataset: the CapyDataset
- iCatField: the index of the categorical field for filtering
- valCatField: the value of the categorical field for filtering
- iField: the index of the plotted field
- dims: the dimension of the graph
- nbBin: number of bins for the density distribution

Output and side effect(s):

- Return a CapyImg. The ordinate corresponds to the bins from the min to the
- max value of the plotted field and the absciss corresponds to the number
- of records in the dataset for that bin. Records are filtered on the value
- of the categorical field.

```

CapyImg* (*plotAllDensityDistributionsGivenCatField)(
    CapyDataset const* const dataset,
        size_t const iCatField,
    CapyImgDims const* const dims,
        size_t const nbBin);

```

Plot the density distributions of all fields in a CapyDataset filtered per value of a given categorical field.

Input argument(s):

- dataset: the CapyDataset
- iCatField: the index of the categorical field

- dims: the dimension of one graph
- nbBin: number of bins for the density distribution

Output and side effect(s):

- Return a CpyImg containing the result graphs. Each graph represent
- the combination of one value of the categorical field and one other
- field. Categorical field values are aligned top-down and other fields
- values are aligned left-right. In one plot, the ordinate corresponds to
- the bins from the min to the max value of the numerical value and the
- absciss corresponds to the number of record in the dataset for that bin.

```
CpyImg* (*plotValuesForGivenPairOfFields)(
    CpyDataset const* const dataset,
        size_t const iField,
        size_t const jField,
    CpyImgDims const* const dims);
```

Plot the records of a dataset on a 2D graph using the values of two given fields.

Input argument(s):

- dataset: the CpyDataset
- iField: the index of the field in absciss
- jField: the index of the field in ordinate
- dims: the dimension the graph

Output and side effect(s):

- Return a CpyImg containing the result graph.

```
CpyImg* (*plotAllCorrelationGraph)(
    CpyDataset const* const dataset,
    CpyImgDims const* const dims,
    CpyGraphPlotterTypeCorrelation const typeCorrelation);
```

Plot the correlation graph between all pair of variables.

Input argument(s):

- dataset: the CapyDataset
- dims: the dimension of one graph
- typeCorrelation: type of correlation

Output and side effect(s):

- Return a CapyImg containing the result graphs. Each graph represent
- a pair of fields. Bottom left graphs contains a plot of records in
- the dataset.

```
CapyImg* (*plotHistogram)(
    CapyArrDouble const* const vals,
    CapyImgDims const* const dims,
    CapyGraphPlotterLegendData const* const legend);
```

Plot an histogram.

Input argument(s):

- vals: values to plot
- dims: dimension of the result image
- legend: data for the legend

Output and side effect(s):

- Return an image of the plot of the histogram

45.6 Functions

```
CapyGraphPlotter CapyGraphPlotterCreate(void);
```

Create a CapyGraphPlotter

Output and side effect(s):

- Return a CapyGraphPlotter

```
CapyGraphPlotter* CapyGraphPlotterAlloc(void);
```

Allocate memory for a new CapyGraphPlotter and create it

Output and side effect(s):

- Return a CapyGraphPlotter

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyGraphPlotterFree(CapyGraphPlotter** const that);
```

Free the memory used by a CapyGraphPlotter* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyGraphPlotter to free

46 Unit greedy.h

Class implementing the greedy algoirhtm.

46.1 Macros

```
#define CAPY_GREEDY_H
```

46.2 Enumerations

None.

46.3 Typedefs

None.

46.4 Struct CapyGreedyObject

46.4.1 Struct CapyGreedyObject's properties

```
double gain;
```

Gain per unit

```
double cost;
```


Cost per unit

```
size_t qty;
```

Quantity

```
union {void const* ptr; size_t id;};
```

Pointer or ID to identify the object

46.4.2 Struct CapyGreedyObject's methods

None.

46.5 Struct CapyGreedyResult

46.5.1 Struct CapyGreedyResult's properties

```
double gain;
```

Gain

```
double cost;
```

Cost

46.5.2 Struct CapyGreedyResult's methods

None.

46.6 Struct CapyGreedy

46.6.1 Struct CapyGreedy's properties

```
CapyGreedyObjects objects;
```

Array of objects

46.6.2 Struct CapyGreedy's methods

```
void (*destruct)(void);
```


Destructor

```
CapyGreedyResult (*select)(double const budget);
```

Run the Greedy algorithm to select the best set of objects amongst that->objects.

Input argument(s):

- budget: the budget allowed to select objects

Output and side effect(s):

- Return the total gain and cost , and that->objects is modified to
- represent the selected objects and their quantity.

46.7 Functions

```
CapyGreedy CapyGreedyCreate(void);
```

Create a CapyGreedy

Output and side effect(s):

- Return a CapyGreedy

```
CapyGreedy* CapyGreedyAlloc(void);
```

Allocate memory for a new CapyGreedy and create it

Output and side effect(s):

- Return a CapyGreedy

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyGreedyFree(CapyGreedy** const that);
```

Free the memory used by a CapyGreedy* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyGreedy to free

47 Unit hashFun.h

Hash function classes.

47.1 Macros

```
#define CAPY_HASHFUN_H
```

```
#define CopyHashFunDef struct { \
    void (*destruct)(void);      \
    CopyHashFunValue_t (*eval)( \
        char const* const data, \
        size_t const sizeData); \
}
```

47.2 Enumerations

None.

47.3 Typedefs

```
typedef uint64_t CopyHashFunValue_t;
```

CopyHashFun object definition macro

```
typedef CopyHashFunDef CopyHashFun;
```

CopyHashFun object

47.4 Struct CopyFNV1aHashFun

47.4.1 Struct CopyFNV1aHashFun's properties

```
CopyHashFunDef;
```

Inherit CopyHashFun

```
CopyHashFunValue_t prime;
```

Prime


```
CapyHashFunValue_t offset;
```

Offset

47.4.2 Struct CapyFNV1aHashFun's methods

```
void (*destructCapyHashFun)(void);
```

Destructor

47.5 Functions

```
CapyHashFun CapyHashFunCreate(void);
```

Create a CapyHashFun

Output and side effect(s):

- Return a CapyHashFun

```
CapyHashFun* CapyHashFunAlloc(void);
```

Allocate memory for a new CapyHashFun and create it

Output and side effect(s):

- Return a CapyHashFun

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyHashFunFree(CapyHashFun** const that);
```

Free the memory used by a CapyHashFun* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyHashFun to free

```
CapyFNV1aHashFun CapyFNV1aHashFunCreate(void);
```

Create a CapyFNV1aHashFun

Output and side effect(s):

- Return a CpyFNV1aHashFun

```
CpyFNV1aHashFun* CpyFNV1aHashFunAlloc(void);
```

Allocate memory for a new CpyFNV1aHashFun and create it
Output and side effect(s):

- Return a CpyFNV1aHashFun

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyFNV1aHashFunFree(CpyFNV1aHashFun** const that);
```

Free the memory used by a CpyFNV1aHashFun* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpyFNV1aHashFun to free

48 Unit hierarchicalclustering.h

HierarchicalClustering class.

48.1 Macros

```
#define CAPY_HIERARCHICALCLUSTERING_H
```

48.2 Enumerations

None.

48.3 Typedefs

None.

48.4 Struct CpyHierarchicalClustering

48.4.1 Struct CpyHierarchicalClustering's properties

```
FILE* verboseStream;
```

Verbose stream (default: NULL)

48.4.2 Struct CpyHierarchicalClustering's methods

```
void (*destruct)(void);
```

Destructor

```
CpyDatasetRowCluster* (*dianaClustering)(CpyDataset const* const dataset  
);
```

Cluster a dataset using the DIANA algorithm

Input argument(s):

- dataset: the dataset to cluster

Output and side effect(s):

- Return the clustering as a tree of dataset's rows. The clustering
- occurs based on input fields converted to numerical values. The dataset
- can have output fields.

```
CpyDatasetRowCluster* (*fastDianaClustering)(  
CpyDataset const* const dataset);
```

Cluster a dataset using a slightly faster version of the DIANA algorithm

Input argument(s):

- dataset: the dataset to cluster

Output and side effect(s):

- Return the clustering as a tree of dataset's rows. The clustering
- occurs based on input fields converted to numerical values. The dataset
- can have output fields.

48.5 Functions

```
CapyHierarchicalClustering CapyHierarchicalClusteringCreate(void);
```

Create a CapyHierarchicalClustering

Output and side effect(s):

- Return a CapyHierarchicalClustering

```
CapyHierarchicalClustering* CapyHierarchicalClusteringAlloc(void);
```

Allocate memory for a new CapyHierarchicalClustering and create it

Output and side effect(s):

- Return a CapyHierarchicalClustering

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyHierarchicalClusteringFree(CapyHierarchicalClustering** const that)
;
```

Free the memory used by a CapyHierarchicalClustering* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyHierarchicalClustering to free

49 Unit idxCombination.h

Iterator on combination of indices.

49.1 Macros

```
#define CAPY_COMBINATOR_H
```

49.2 Enumerations

None.

49.3 Typedefs

None.

49.4 Struct CapyIdxCombination

49.4.1 Struct CapyIdxCombination's properties

```
size_t dim;
```

Dimension of the combination

```
size_t* indices;
```

Current indices

```
CapyRangeSize* ranges;
```

Ranges for each indices (default: [0,1], inclusive)

```
size_t* datatype;
```

Fields for the forEach macro

```
size_t idx;
```

Index in iteration

49.4.2 Struct CapyIdxCombination's methods

```
size_t** (*reset)(void);
```

Methods for the forEach macro

```
bool (*isActive)(void);
```

```
size_t** (*next)(void);
```

```
void (*destruct)(void);
```

Destructor

49.5 Functions

```
CapyIdxCombination CapyIdxCombinationCreate(size_t const dim);
```

Create a CapyIdxCombination

Input argument(s):

- dim: dimension of the combination

Output and side effect(s):

- Return a CapyIdxCombination

```
CapyIdxCombination* CapyIdxCombinationAlloc(size_t const dim);
```

Allocate memory for a new CapyIdxCombination and create it

Input argument(s):

- dim: dimension of the combination

Output and side effect(s):

- Return a CapyIdxCombination

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyIdxCombinationFree(CapyIdxCombination** const that);
```

Free the memory used by a CapyIdxCombination* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyIdxCombination to free

50 Unit image.h

2D image class.

50.1 Macros

```
#define CAPY_IMAGE_H
```

```
#define CopyImgDims_ \
    union { \
        CopyImgDims_t vals[2]; \
        struct __attribute__((packed)) { \
            CopyImgDims_t width; CopyImgDims_t height;}; \
    }
```

Definition of the dimensions of an image

```
#define CopyImgPos_ \
    union { \
        CopyImgPos_t coords[2]; \
        struct __attribute__((packed)) { \
            CopyImgPos_t x; CopyImgPos_t y;}; \
    }
```

Definition of a position in the image (from left to right and top to bottom)

```
#define CopyImgCreate_Greyscale_800x600() \
    CopyImgCreate(capyImgMode_greyscale, capyImgDims_800x600)
```

Shortcuts for CopyImgCreate()

```
#define CopyImgCreate_Greyscale_600x800() \
    CopyImgCreate(capyImgMode_scale, capyImgDims_600x800)
```

```
#define CopyImgCreate_RGB_800x600() \
    CopyImgCreate(capyImgMode_rgb, capyImgDims_800x600)
```

```
#define CopyImgCreate_RGB_600x800() \
    CopyImgCreate(capyImgMode_rgb, capyImgDims_600x800)
```

```
#define CopyImgCreate_RGBA_800x600() \
    CopyImgCreate(capyImgMode_rgba, capyImgDims_800x600)
```

```
#define CopyImgCreate_RGBA_600x800() \
    CopyImgCreate(capyImgMode_rgba, capyImgDims_600x800)
```

```
#define CopyImgCreate_HLS_800x600() \
    CopyImgCreate(capyImgMode_hls, capyImgDims_800x600)
```



```
#define CopyImgCreate_HLS_600x800() \
    CopyImgCreate(capyImgMode_hls, capyImgDims_600x800)
```

```
#define CopyImgAlloc_Greyscale_800x600() \
    CopyImgAlloc(capyImgMode_greyscale, capyImgDims_800x600)
```

Shortcuts for CopyImgAlloc()

```
#define CopyImgAlloc_Greyscale_600x800() \
    CopyImgAlloc(capyImgMode_scale, capyImgDims_600x800)
```

```
#define CopyImgAlloc_RGB_800x600() \
    CopyImgAlloc(capyImgMode_rgb, capyImgDims_800x600)
```

```
#define CopyImgAlloc_RGB_600x800() \
    CopyImgAlloc(capyImgMode_rgb, capyImgDims_600x800)
```

```
#define CopyImgAlloc_RGBA_800x600() \
    CopyImgAlloc(capyImgMode_rgba, capyImgDims_800x600)
```

```
#define CopyImgAlloc_RGBA_600x800() \
    CopyImgAlloc(capyImgMode_rgba, capyImgDims_600x800)
```

```
#define CopyImgAlloc_HLS_800x600() \
    CopyImgAlloc(capyImgMode_hls, capyImgDims_800x600)
```

```
#define CopyImgAlloc_HLS_600x800() \
    CopyImgAlloc(capyImgMode_hls, capyImgDims_600x800)
```

50.2 Enumerations

```
typedef enum CopyImgMode {
    capyImgMode_greyscale,
    capyImgMode_rgb,
    capyImgMode_rgba,
    capyImgMode_hsv
} CopyImgMode;
```

Enumeration of image modes


```
typedef enum CopyImgNeighbour {
    copyImgNeighbour_top,
    copyImgNeighbour_right,
    copyImgNeighbour_bottom,
    copyImgNeighbour_left,
    copyImgNeighbour_topLeft,
    copyImgNeighbour_topRight,
    copyImgNeighbour_bottomRight,
    copyImgNeighbour_bottomLeft,
    copyImgNeighbour_last,
} CopyImgNeighbour;
```

Enumeration to identify the neighbours of a pixel

```
typedef enum CopyImgIteratorType {
    copyImgIteratorType_LeftToRight_TopToBottom,
} CopyImgIteratorType;
```

Enumeration for the types of iterator on a CopyImg

50.3 Typedefs

```
typedef uint32_t CopyImgDims_t;
```

Type for the size of an image (chosen to match png_uint_32)

```
typedef CopyImgDims_ CopyImgDims;
```

Dimensions of an image

```
typedef int32_t CopyImgPos_t;
```

Type to store a pixel coordinate (should be the signed version of CopyImgDims_t)

```
typedef CopyImgPos_ CopyImgPos;
```

Position in the image (from left to right and top to bottom)

```
typedef struct CopyImg CopyImg;
```

CopyImg object predeclaration

```
typedef struct CopyImg CopyImg;
```

CopyImg object

50.4 Struct CpyImgPixel

50.4.1 Struct CpyImgPixel's properties

```
CpyImgPos pos;
```

Position

```
size_t idx;
```

Index in the array of pixels

```
CpyColorData* color;
```

Color data

50.4.2 Struct CpyImgPixel's methods

None.

50.5 Struct CpyImgIterator

50.5.1 Struct CpyImgIterator's properties

```
size_t idx;
```

Index of the current step in the iteration

```
CpyImgPixel datatype;
```

Returned data type

```
CpyImgPixel pixel;
```

Current pixel of the iteration

```
CpyImgIteratorType type;
```

Type of iteration

```
CpyPad(CpyImgIteratorType, type);
```

```
CpyImg* img;
```

Image associated to the iteration

50.5.2 Struct CpyImgIterator's methods

```
void (*destruct)(void);
```

Destructor

```
CpyImgPixel* (*reset)(void);
```

Reset the iterator

Output and side effect(s):

- Return the first pixel of the iteration

```
CpyImgPixel* (*prev)(void);
```

Move the iterator to the previous pixel

Output and side effect(s):

- Return the previous pixel of the iteration

```
CpyImgPixel* (*next)(void);
```

Move the iterator to the next pixel

Output and side effect(s):

- Return the next pixel of the iteration

```
bool (*isActive)(void);
```

Check if the iterator is on a valid pixel

Output and side effect(s):

- Return true if the iterator is on a valid pixel, else false

```
CpyImgPixel* (*get)(void);
```

Get the current pixel of the iteration

Output and side effect(s):

- Return a pointer to the current pixel

```
void (*setType)(CpyImgIteratorType const type);
```

Set the type of the iterator and reset it

Input argument(s):

- type: the new type of the iterator

50.6 Functions

```
CapyImgIterator CapyImgIteratorCreate(  
    CapyImg* const img,  
    CapyImgIteratorType const type);
```

Create an iterator on a CapyImg

Input argument(s):

- img: the image on which to iterate
- type: the type of iterator

Output and side effect(s):

- Return the iterator

```
CapyImgIterator* CapyImgIteratorAlloc(  
    CapyImg* const img,  
    CapyImgIteratorType const type);
```

Allocate memory and create an iterator on a CapyImg

Input argument(s):

- img: the image on which to iterate
- type: the type of iterator

Output and side effect(s):

- Return the iterator

```
void CapyImgIteratorFree(CapyImgIterator** const that);
```

Free the memory used by a pointer to an iterator and reset '*that' to NULL

Input argument(s):

- that: a pointer to the iterator to free

```
CapyImg CapyImgCreate(  
    CapyImgMode const mode,  
    CapyImgDims const dims);
```

Create a CapyImg of given dimensions and mode

Input argument(s):

- mode: mode of the image
- dims: dimensions of the image

Output and side effect(s):

- Return a CpyImg initialised to rgba opaque white

Exception(s):

- May raise CpyExc_MallocFailed.

```
CpyImg* CpyImgAlloc(
    CpyImgMode const mode,
    CpyImgDims const dims);
```

Allocate memory for a new CpyImg and create it

Input argument(s):

- mode: mode of the image
- dims: dimensions of the image

Output and side effect(s):

- Return a CpyImg initialised to rgba opaque white

Exception(s):

- May raise CpyExc_MallocFailed.

```
CpyImg* CpyImgLoadFromPath(char const* const path);
```

Load an image to a given path

Input argument(s):

- path: the path to the image

Output and side effect(s):

- Return a new image.

Exception(s):

- May raise CpyExc_MallocFailed, CpyExc_UnsupportedFormat.


```
CapyImg* CapyImgClone(CapyImg const* const img);
```

Allocate memory and create a clone of an image

Input argument(s):

- img: the image to clone

Output and side effect(s):

- Return a clone of the image in argument

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyImgFree(CapyImg** const that);
```

Free the memory used by a CapyImg* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyImg to free

```
double CapyCm2Pixel(  
    double const length,  
    double const dpi);
```

Convert from cm to pixels given a dpi

Input argument(s):

- length: the length in centimeter to convert
- dpi: the dpi used to convert

Output and side effect(s):

- Return the converted length.

```
double CapyPixel2Cm(  
    double const length,  
    double const dpi);
```

Convert from pixels to cm given a dpi

Input argument(s):

- length: the length in pixels to convert
- dpi: the dpi used to convert

Output and side effect(s):

- Return the converted length.

51 Unit imgKernel.h

Kernel class to perform operations on 2D images.

51.1 Macros

```
#define CAPY_IMGKERNEL_H
```

51.2 Enumerations

None.

51.3 Typedefs

None.

51.4 Struct CpyImgKernel

51.4.1 Struct CpyImgKernel's properties

```
CpyImgDims_t size;
```

Size of the kernel

```
CpyPad(CpyImgDims_t, size);
```

```
double* mat;
```

Kernel matrix (the kernel matrix is a $(1+2*\text{size}) \times (1+2*\text{size})$ matrix). Values stored by rows.

```
size_t sizeMat;
```

Size of the kernel matrix

```
CpyRangeImgPos range;
```

Range to loop on the kernel values

51.4.2 Struct CpyImgKernel's methods

```
void (*destruct)(void);
```

Destructor

```
CpyImg* (*apply)(CpyImg const* const img);
```

Convolve the kernel on a given image

Input argument(s):

- img: the image to which the kernel is applied

Output and side effect(s):

- Return the result of the convolution as a newly allocated image

Exception(s):

- May raise CpyExc_MallocFailed.

```
void (*setToIdentity)(void);
```

Set the kernel to the identity

```
void (*setToGaussianBlur)(double const stdDev);
```

Set the kernel to Gaussian blur (the kernel is normalised)

Input argument(s):

- stdDev: the standard deviation (in pixels) of the blur

Exception(s):

- May raise CpyExc_MallocFailed.

```
void (*setToDerivativeGaussianBlur)(  
    double const stdDev,  
    size_t const axis);
```

Set the kernel to derivative Gaussian blur along a given axis

Input argument(s):

- stdDev: the standard deviation (in pixels) of the blur

- axis: axis of derivation (0:x, 1:y)

Exception(s):

- May raise CpyExc_MallocFailed.

```
void (*setToSharpen)(void);
```

Set the kernel to the sharpen kernel (only center 3x3 is used)

Exception(s):

- May raise CpyExc_MallocFailed.

```
void (*setToEdgeDetection)(void);
```

Set the kernel to the edge detection kernel (only center 3x3 is used)

Exception(s):

- May raise CpyExc_MallocFailed.

51.5 Functions

```
CpyImgKernel CpyImgKernelCreate(CpyImgDims_t const size);
```

Create a CpyImgKernel

Input argument(s):

- size: the size of the kernel (the kernel matrix is a
- $(1+2*\text{size}) \times (1+2*\text{size})$ matrix)

Output and side effect(s):

- Return a CpyImgKernel with identity matrix (in kernel sense)

Exception(s):

- May raise CpyExc_MallocFailed.

```
CpyImgKernel* CpyImgKernelAlloc(CpyImgDims_t const size);
```


Allocate memory for a new `CapyImgKernel` and create it

Input argument(s):

- size: the size of the kernel (the kernel matrix is a
- $(1+2*\text{size}) \times (1+2*\text{size})$ matrix)

Output and side effect(s):

- Return a `CapyImgKernel` with identity matrix (in kernel sense)

Exception(s):

- May raise `CapyExc_MallocFailed`.

```
void CapyImgKernelFree(CapyImgKernel** const that);
```

Free the memory used by a `CapyImgKernel*` and reset `*that` to NULL

Input argument(s):

- that: a pointer to the `CapyImgKernel` to free

52 Unit `kfoldCrossValid.h`

k-fold cross validation class.

52.1 Macros

```
#define CAPY_KFOLD_CROSS_VALID_H
```

52.2 Enumerations

None.

52.3 Typedefs

None.

52.4 Struct CpyKfoldCrossValidResPredictor

52.4.1 Struct CpyKfoldCrossValidResPredictor's properties

```
size_t k;
```

Number of folds

```
union {
```

Result of evaluation training/validation

```
CpyPredictorEvaluation** eval[2];
```

```
struct __attribute__((packed)) {
```

```
CpyPredictorEvaluation **evalTraining, **evalValidation;
```

```
};
```

```
};
```

```
union {
```

Min/avg/max accuracy on training/validation (in [0,1], higher is better)

```
double accuracy[2][3];
```

```
struct __attribute__((packed)) {
```

```
struct {double min, avg, max;} accTraining;
```

```
struct {double min, avg, max;} accValidation;
```

```
};
```

```
};
```


52.4.2 Struct CapyKfoldCrossValidResPredictor's methods

```
void (*destruct)(void);
```

Destructor

52.5 Struct CapyKfoldCrossValid

52.5.1 Struct CapyKfoldCrossValid's properties

```
size_t k;
```

Number of fold

```
bool verbose;
```

Flag to memorise the verbose mode (default: false)

```
CapyPad(bool, verbose);
```

```
FILE* stream;
```

Stream for the output in verbose mode (default: stdout)

```
CapyRandomSeed_t seed;
```

Seed for the pseudo-random generator (default: 0)

```
FILE* streamSplit;
```

Stream for the splits definition, if not null it is used to create the split during evaluation, else random splits are created (default: NULL). The stream is expected to be in same format as the splits file imported using openmlImport.

```
CapyPredictorAccuracyMeasure accType;
```

Accuracy measurement type used (default: mae for numerical predictors and accuracy for categorical predictors)

```
CapyPad(CapyPredictorAccuracyMeasure, accType);
```


52.5.2 Struct CpyKfoldCrossValid's methods

```
void (*destruct)(void);
```

Destructor

```
CpyKfoldCrossValidResPredictor (*evalPredictor)(  
    CpyPredictor* const predictor,  
    CpyDataset const* const dataset);
```

Run the k-fold cross validation for a predictor and a dataset

Input argument(s):

- predictor: the predictor
- dataset: the dataset

Output and side effect(s):

- The dataset is split into k folds (the original dataset is not modified),
- the predictor is trained on all combination of (k-1) fold and evaluated
- on the remaining fold.

52.6 Functions

```
CpyKfoldCrossValidResPredictor CpyKfoldCrossValidResPredictorCreate(  
    size_t const k);
```

Create a CpyKfoldCrossValidResPredictor

Input argument(s):

- k: the number of fold

Output and side effect(s):

- Return a CpyKfoldCrossValidResPredictor

```
CpyKfoldCrossValid CpyKfoldCrossValidCreate(size_t const k);
```

Create a CpyKfoldCrossValid

Input argument(s):

- k: the number of fold

Output and side effect(s):

- Return a CpyKfoldCrossValid

```
CpyKfoldCrossValid* CpyKfoldCrossValidAlloc(size_t const k);
```

Allocate memory for a new CpyKfoldCrossValid and create it

Input argument(s):

- k: the number of fold

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyKfoldCrossValidFree(CpyKfoldCrossValid** const that);
```

Free the memory used by a CpyKfoldCrossValid* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyKfoldCrossValid to free

53 Unit kmeans.h

Class implementing the k-means algorithm.

53.1 Macros

```
#define CAPY_KMEANS_H
```

53.2 Enumerations

```
typedef enum CpyKMeansInit {
    cpyKMeanInit_rand,
    cpyKMeanInit_forgy,
    cpyKMeanInit_plusplus,
} CpyKMeansInit;
```

Type of initialisation for the k-mean algorithm

53.3 Typedefs

None.

53.4 Struct CpyKMeansClusterOfPoint

53.4.1 Struct CpyKMeansClusterOfPoint's properties

```
size_t id;
```

Id of the cluster of the point

```
double dist;
```

Distance from the point to the cluster center

53.4.2 Struct CpyKMeansClusterOfPoint's methods

None.

53.5 Struct CpyKMeans

53.5.1 Struct CpyKMeans's properties

```
CpyKMeansInit typeInit;
```

Type of initialization (default: plusplus)

```
CpyPad(CpyKMeansInit, 0);
```

```
CpyPointCloud* clusters;
```

Clusters center

53.5.2 Struct CpyKMeans's methods

```
void (*destruct)(void);
```

Destructor


```
void (*run)(
    CappyPointCloud const* const pointCloud,
    size_t const k);
```

Cluster a point cloud.

Input argument(s):

- pointCloud: the point cloud to be clustered
- k: the number of clusters

Output and side effect(s):

- Update that->clusters.

Exception(s):

- May raise CappyExc_MallocFailed.

```
CappyKMeansClusterOfPoint (*getClusterOfPoint)(CappyVec const* const v);
```

Get the cluster of a given point

Input argument(s):

- v: the point

Output and side effect(s):

- Return the cluster id for the given point

```
void (*applyToImgColor)(CappyImg* const img);
```

Apply the clustering to an image color

Input argument(s):

- img: the image

Output and side effect(s):

- The image colors are replaced with the color of their cluster)

53.6 Functions

```
CapyKMeans CapyKMeansCreate(void);
```

Create a CapyKMeans

Output and side effect(s):

- Return a CapyKMeans

```
CapyKMeans* CapyKMeansAlloc(void);
```

Allocate memory for a new CapyKMeans and create it

Output and side effect(s):

- Return a CapyKMeans

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyKMeansFree(CapyKMeans** const that);
```

Free the memory used by a CapyKMeans* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyKMeans to free

54 Unit lightray.h

LightRay class.

54.1 Macros

```
#define CAPY_LIGHTRAY_H
```

```
#define CAPY_DEFAULT_WAVELENGTH 589.29
```

Default wavelength

54.2 Enumerations

```
typedef enum CpyMaterialIdx {  
    copyMaterialIdx_vacuum,  
    copyMaterialIdx_air,  
    copyMaterialIdx_glass,  
    copyMaterialIdx_water,  
    copyMaterialIdx_silver,  
    copyMaterialIdx_gold,  
    copyMaterialIdx_copper,  
    copyMaterialIdx_diamond,  
    copyMaterialIdx_nb,  
} CpyMaterialIdx;
```

Predefined materials index

54.3 Typedefs

```
typedef struct CpyLightRay CpyLightRay;
```

LightRay object

54.4 Struct CpyRefractiveCoeff

54.4.1 Struct CpyRefractiveCoeff's properties

```
double vals[2];
```

Cauchy coefficients (for lambda in nanometers)

54.4.2 Struct CpyRefractiveCoeff's methods

None.

54.5 Struct CpyRefractionValue

54.5.1 Struct CpyRefractionValue's properties

```
union {
```

```
double vals[3];
```



```
struct __attribute__((packed)) { double reflective, transmissive, theta;
};
```

```
};
```

54.5.2 Struct CapyRefractionValue's methods

None.

54.6 Functions

```
double CapyGetRefractiveIndex(
    CapyRefractiveCoeff const coeff,
    double const lambda);
```

Get the refractive index for given Cauchy coefficients and wavelength Inputs:

coeff: the Cauchy coefficients lambda: the wavelength in nanometer

Output and side effect(s):

- Return the refractive index

```
CapyRefractiveCoeff CapyEstimateRefractiveCoeffFromReflectiveIndex(
    double const reflective);
```

Estimate the Cauchy coefficients from the reflectivity Inputs: reflectivity:
the reflectivity in [0,1]

Output and side effect(s):

- Return the estimated Cauchy coefficients (B set to 0.0, relative to vacuum)

```
CapyRefractionValue CapyGetRefraction(
    CapyRefractiveCoeff const coeffFrom,
    CapyRefractiveCoeff const coeffTo,
    double const theta);
```

Get the refraction values for given refractive coefficients, angle of incidence and a default wavelength (CAPY_DEFAULT_WAVELENGTH) Inputs: coeffFrom: the Cauchy coefficients of the incoming material coeffTo: the Cauchy coefficients of the outgoing material theta: angle of incidence in radians (angle between the incoming ray of light and the normal of the interface of materials)

Output and side effect(s):

- Return the refraction values when moving from a material with Cauchy
- coefficients 'coeffFrom' to a material with Cauchy coefficients 'coeffTo'.
- In other words, how much light traveling in the 'from' material goes
- back to the 'from' material after encountering the 'to' material.

```
CapyRefractionValue CapyGetRefractionForWavelength(
    CapyRefractiveCoeff const coeffFrom,
    CapyRefractiveCoeff const coeffTo,
    double const theta,
    double const lambda);
```

Get the refraction values for given refractive coefficients, angle of incidence and given wavelength Inputs: coeffFrom: the Cauchy coefficients of the incoming material coeffTo: the Cauchy coefficients of the outgoing material theta: angle of incidence in radians (angle between the incoming ray of light and the normal of the interface of materials) lambda: the wavelength (in nanometer)

Output and side effect(s):

- Return the refractive coefficients for the given wavelength when moving
- from a material with Cauchy coefficients 'coeffFrom' to a material with
- Cauchy coefficients 'coeffTo'
- In other words, how much light traveling in the 'from' material goes
- back to the 'from' material after encountering the 'to' material.

```
double CapyGetCriticalRefractiveAngle(
    CapyRefractiveCoeff const coeffFrom,
    CapyRefractiveCoeff const coeffTo,
    double const lambda);
```

Get the critical refractive angle given refractive coefficients and wavelength Inputs: coeffFrom: the Cauchy coefficients of the incoming material coeffTo: the Cauchy coefficients of the outgoing material lambda: the wavelength (in nanometer)

Output and side effect(s):

- Return the critical angle in radians.


```
CapyLightRay CapyLightRayCreate(void);
```

Create a CapyLightRay

Output and side effect(s):

- Return a CapyLightRay

```
CapyLightRay* CapyLightRayAlloc(void);
```

Allocate memory for a new CapyLightRay and create it

Output and side effect(s):

- Return a CapyLightRay

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyLightRayFree(CapyLightRay** const that);
```

Free the memory used by a CapyLightRay* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyLightRay to free

55 Unit list.h

Generic list class.

55.1 Macros

```
#define CAPY_LIST_H
```

```
#define CapyDecListIterator(name, type_) \
typedef struct name name; \
typedef struct name ## Iterator { \
    size_t idx; \
    name ## Elem* curElem; \
    name* list; \
    type_ datatype; \
    CapyPad(type_, 0); \
    CapyListIteratorType type; \
}
```



```

CapyPad(CapyListIteratorType, 1); \
void (*destruct)(void); \
type_* (*reset)(void); \
type_* (*prev)(void); \
type_* (*next)(void); \
bool (*isActive)(void); \
type_* (*get)(void); \
void (*setType)(CapyListIteratorType type); \
bool (*isFirst)(void); \
bool (*isLast)(void); \
void (*toLast)(void); \
} name ## Iterator; \
name ## Iterator name ## Iterator ## Create( \
    name* const arr, CapyListIteratorType const type); \
name ## Iterator* name ## Iterator ## Alloc( \
    name* const arr, CapyListIteratorType const type); \
void name ## Iterator ## Destruct(void); \
void name ## Iterator ## Free(name ## Iterator** const that); \
type_* name ## Iterator ## Reset(void); \
type_* name ## Iterator ## Prev(void); \
type_* name ## Iterator ## Next(void); \
bool name ## Iterator ## IsActive(void); \
type_* name ## Iterator ## Get(void); \
bool name ## Iterator ## IsFirst(void); \
bool name ## Iterator ## IsLast(void); \
void name ## Iterator ## ToLast(void); \
void name ## Iterator ## SetType(CapyListIteratorType type);

```

Iterator for generic list structure. Declaration macro for an iterator structure named 'name' ## Iterator, associated to a list named 'name' containing elements of type 'type'

```

#define CapyDefListIteratorCreate(name, type_) \
name ## Iterator name ## Iterator ## Create( \
    name* const list, \
    CapyListIteratorType const type) { \
    name ## Iterator that = { \
        .idx = 0, \
        .curElem = NULL, \
        .list = list, \
        .type = type, \
        .destruct = name ## Iterator ## Destruct, \
        .reset = name ## Iterator ## Reset, \
        .prev = name ## Iterator ## Prev, \
        .next = name ## Iterator ## Next, \
        .isActive = name ## Iterator ## IsActive, \
        .get = name ## Iterator ## Get, \
        .setType = name ## Iterator ## SetType, \
        .isFirst = name ## Iterator ## IsFirst, \
        .isLast = name ## Iterator ## IsLast, \
        .toLast = name ## Iterator ## ToLast, \
    }; \
    $(&that, reset)(); \
    return that; \
}

```

Create an iterator on a generic list

Input argument(s):

- list: the generic list on which to iterate
- type: the type of iterator

Output and side effect(s):

- Return the iterator

```
#define CopyDefListIteratorAlloc(name, type_) \
name ## Iterator* name ## Iterator ## Alloc( \
    name* const list, \
    CopyListIteratorType const type) { \
    name ## Iterator* that = NULL; \
    safeMalloc(that, 1); \
    if(!that) return NULL; \
    *that = name ## Iterator ## Create(list, type); \
    return that; \
}
```

Allocate memory and create an iterator on a generic list

Input argument(s):

- list: the generic list on which to iterate
- type: the type of iterator

Output and side effect(s):

- Return the iterator

```
#define CopyDefListIteratorDestruct(name, type) \
void name ## Iterator ## Destruct(void) { \
    return; \
}
```

Free the memory used by an iterator.

Input argument(s):

- that: the iterator to free

```
#define CopyDefListIteratorFree(name, type) \
void name ## Iterator ## Free(name ## Iterator** const that) { \
    if(*that == NULL) return; \
    $(*that, destruct)(); \
    free(*that); \
    *that = NULL; \
}
```


Free the memory used by a pointer to an iterator and reset '*that' to NULL
Input argument(s):

- that: a pointer to the iterator to free

```
#define CopyDefListIteratorReset(name, type_) \
type_* name ## Iterator ## Reset(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    assert(that->list && "List iterator uninitialised"); \
    that->idx = 0; \
    switch(that->type) { \
        case copyListIteratorType_forward: \
            that->curElem = that->list->head; \
            break; \
        case copyListIteratorType_backward: \
            that->curElem = that->list->tail; \
            break; \
    } \
    if(name ## Iterator ## IsActive()) \
        return name ## Iterator ## Get(); \
    else \
        return NULL; \
}
```

Reset the iterator

Output and side effect(s):

- Return the first element of the iteration

```
#define CopyDefListIteratorPrev(name, type_) \
type_* name ## Iterator ## Prev(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    --(that->idx); \
    switch(that->type) { \
        case copyListIteratorType_forward: \
            if(name ## Iterator ## IsActive()) \
                that->curElem = that->curElem->prev; \
            break; \
        case copyListIteratorType_backward: \
            if(name ## Iterator ## IsActive()) \
                that->curElem = that->curElem->next; \
            break; \
    } \
    if(name ## Iterator ## IsActive()) \
        return name ## Iterator ## Get(); \
    else \
        return NULL; \
}
```

Move the iterator to the previous element

Output and side effect(s):

- Return the previous element of the iteration


```

#define CopyDefListIteratorNext(name, type_) \
type_* name ## Iterator ## Next(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    ++(that->idx); \
    switch(that->type) { \
        case copyListIteratorType_forward: \
            if(name ## Iterator ## IsActive()) \
                that->curElem = that->curElem->next; \
            break; \
        case copyListIteratorType_backward: \
            if(name ## Iterator ## IsActive()) \
                that->curElem = that->curElem->prev; \
            break; \
    } \
    if(name ## Iterator ## IsActive()) \
        return name ## Iterator ## Get(); \
    else \
        return NULL; \
}

```

Move the iterator to the next element

Output and side effect(s):

- Return the next element of the iteration

```

#define CopyDefListIteratorIsActive(name, type_) \
bool name ## Iterator ## IsActive(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    return (that->curElem != NULL); \
}

```

Check if the iterator is on a valid element

Output and side effect(s):

- Return true if the iterator is on a valid element, else false

```

#define CopyDefListIteratorGet(name, type_) \
type_* name ## Iterator ## Get(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    return &(amp;that->curElem->data); \
}

```

Get the current element of the iteration

Output and side effect(s):

- Return a pointer to the current element


```

#define CopyDefListIteratorSetType(name, type_) \
void name ## Iterator ## SetType(CopyListIteratorType type) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    that->type = type; \
    name ## Iterator ## Reset(); \
}

```

Set the type of the iterator and reset it

Input argument(s):

- type: the new type of the iterator

```

#define CopyDefListIteratorIsFirst(name, type_) \
bool name ## Iterator ## IsFirst(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    bool ret = false; \
    switch(that->type) { \
        case copyListIteratorType_forward: \
            ret = (that->curElem != NULL && that->curElem->prev == NULL); \
            break; \
        case copyListIteratorType_backward: \
            ret = (that->curElem != NULL && that->curElem->next == NULL); \
            break; \
    } \
    return ret; \
}

```

Return true if the iterator is on the first element of the iteration

Output and side effect(s):

- Return true or false

```

#define CopyDefListIteratorIsLast(name, type_) \
bool name ## Iterator ## IsLast(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    bool ret = false; \
    switch(that->type) { \
        case copyListIteratorType_forward: \
            ret = (that->curElem != NULL && that->curElem->next == NULL); \
            break; \
        case copyListIteratorType_backward: \
            ret = (that->curElem != NULL && that->curElem->prev == NULL); \
            break; \
    } \
    return ret; \
}

```

Return true if the iterator is on the last element of the iteration

Output and side effect(s):

- Return true or false


```

#define CopyDefListIteratorToLast(name, type_) \
void name ## Iterator ## ToLast(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    switch(that->type) { \
        case copyListIteratorType_forward: \
            that->curElem = that->list->tail; \
            break; \
        case copyListIteratorType_backward: \
            that->curElem = that->list->head; \
            break; \
    } \
}

```

Move the iterator to the last element of the iteration

```

#define CopyDefListIterator(name, type) \
    CopyDefListIteratorCreate(name, type) \
    CopyDefListIteratorAlloc(name, type) \
    CopyDefListIteratorDestruct(name, type) \
    CopyDefListIteratorFree(name, type) \
    CopyDefListIteratorReset(name, type) \
    CopyDefListIteratorPrev(name, type) \
    CopyDefListIteratorNext(name, type) \
    CopyDefListIteratorIsActive(name, type) \
    CopyDefListIteratorGet(name, type) \
    CopyDefListIteratorIsFirst(name, type) \
    CopyDefListIteratorIsLast(name, type) \
    CopyDefListIteratorToLast(name, type) \
    CopyDefListIteratorSetType(name, type)

```

Definition macro calling all the submacros at once for an iterator on an array structure named 'name', containing elements of type 'type'

```

#define CopyDecList(name, type) \
typedef struct name ## Elem name ## Elem; \
typedef struct name ## Elem { \
    name ## Elem* prev; \
    name ## Elem* next; \
    type data; \
    CopyPad(type, 0); \
} name ## Elem; \
CopyDecListIterator(name, type) \
typedef struct name { \
    name ## Elem* head; \
    name ## Elem* tail; \
    name ## Iterator iter; \
    void (*destruct)(void); \
    size_t (*getSize)(void); \
    void (*push)(type const val); \
    void (*add)(type const val); \
    type (*pop)(void); \
    type (*drop)(void); \
    type (*get)(size_t const idx); \
    type* (*getPtr)(size_t const idx); \
    void (*flush)(void); \
    type (*getHead)(void); \
    type (*getTail)(void); \
    type* (*getPtrHead)(void); \
}

```



```

type* (*getPtrTail)(void); \
bool (*isEmpty)(void); \
void (*initIterator)(void); \
void (*remove)( \
    CapyComparator* cmp, \
    type const* const elem); \
type* (*find)( \
    CapyComparator* cmp, \
    type const* const elem); \
void (*insertSort)( \
    CapyComparator* cmp, \
    type const val); \
} name; \
name name ## Create(void); \
name* name ## Alloc(void); \
void name ## Destruct(void); \
void name ## Free(name** const that); \
size_t name ## GetSize(void); \
void name ## Push(type const val); \
void name ## Add(type const val); \
type name ## Pop(void); \
type name ## Drop(void); \
type name ## Get(size_t const idx); \
type* name ## GetPtr(size_t const idx); \
void name ## Flush(void); \
type name ## GetHead(void); \
type name ## GetTail(void); \
type* name ## GetPtrHead(void); \
type* name ## GetPtrTail(void); \
bool name ## IsEmpty(void); \
void name ## Remove( \
    CapyComparator* cmp, \
    type const* const elem); \
type* name ## Find( \
    CapyComparator* cmp, \
    type const* const elem); \
void name ## InsertSort( \
    CapyComparator* cmp, \
    type const val); \
void name ## InitIterator(void);

```

Generic double linked list structure. Declaration macro for a list structure named 'name', containing elements of type 'type'

```

#define CapyDefListCreate(name, type) \
name name ## Create(void) { \
    return (name){ \
        .head = NULL, \
        .tail = NULL, \
        .destruct = name ## Destruct, \
        .getSize = name ## GetSize, \
        .push = name ## Push, \
        .add = name ## Add, \
        .pop = name ## Pop, \
        .drop = name ## Drop, \
        .get = name ## Get, \
        .getPtr = name ## GetPtr, \
        .flush = name ## Flush, \
        .getHead = name ## GetHead, \
        .getTail = name ## GetTail, \
    } \
}

```



```

    .getPtrHead = name ## GetPtrHead,      \
    .getPtrTail = name ## GetPtrTail,      \
    .initIterator = name ## InitIterator,  \
    .isEmpty = name ## IsEmpty,            \
    .remove = name ## Remove,              \
    .find = name ## Find,                  \
    .insertSort = name ## InsertSort,      \
};
}

```

Definition macro and submacros for a list named 'name', containing elements of type 'type' Create a list containing elements of type 'type'

Output and side effect(s):

- Return a list

```

#define CopyDefListAlloc(name, type)      \
name* name ## Alloc(void) {               \
    name* that = NULL;                    \
    safeMalloc(that, 1);                   \
    if(!that) return NULL;                 \
    *that = name ## Create();              \
    that->iter = name ## IteratorCreate(   \
        that, copyListIteratorType_forward); \
    return that;                           \
}

```

Allocate memory for a new array and create it

Output and side effect(s):

- Return an array containing 'size_' elements of type 'type'

Exception(s):

- May raise CopyExc_MallocFailed.

```

#define CopyDefListDestruct(name, type)   \
void name ## Destruct(void) {             \
    name* that = (name*)copyThat;         \
    $(that, flush)();                     \
}

```

Free the memory used by a list.

Input argument(s):

- that: the list to free


```

#define CopyDefListFree(name, type) \
void name ## Free(name** const that) { \
    if(*that == NULL) return; \
    $(*that, destruct)(); \
    free(*that); \
    *that = NULL; \
}

```

Free the memory used by a pointer to a list and reset '*that' to NULL

Input argument(s):

- that: a pointer to the list to free

```

#define CopyDefListGetSize(name, type) \
size_t name ## GetSize(void) { \
    name* that = (name*)copyThat; \
    name ## Elem* elem = that->head; \
    size_t size = 0; \
    while(elem != NULL) { \
        ++size; \
        elem = elem->next; \
    } \
    return size; \
}

```

Get the size of a list Output Return the size of the list

```

#define CopyDefListPush(name, type) \
void name ## Push(type val) { \
    name* that = (name*)copyThat; \
    name ## Elem* elem = NULL; \
    safeMalloc(elem, 1); \
    if(!elem) return; \
    elem->next = that->head; \
    elem->prev = NULL; \
    elem->data = val; \
    if(that->head != NULL) \
        that->head->prev = elem; \
    that->head = elem; \
    if(that->tail == NULL) \
        that->tail = elem; \
}

```

Push an element to the head of a list

Input argument(s):

- val: the element's value

Exception(s):

- May raise CpyExc_MallocFailed.


```

#define CopyDefListAdd(name, type) \
void name ## Add(type val) { \
    name* that = (name*)copyThat; \
    name ## Elem* elem = NULL; \
    safeMalloc(elem, 1); \
    if(!elem) return; \
    elem->prev = that->tail; \
    elem->next = NULL; \
    elem->data = val; \
    if(that->tail != NULL) \
        that->tail->next = elem; \
    that->tail = elem; \
    if(that->head == NULL) \
        that->head = elem; \
}

```

Add an element to the tail of a list

Input argument(s):

- val: the element's value

Exception(s):

- May raise CpyExc_MallocFailed.

```

#define CopyDefListPop(name, type) \
type name ## Pop(void) { \
    name* that = (name*)copyThat; \
    name ## Elem* elem = that->head; \
    that->head = elem->next; \
    if(elem->next != NULL) \
        elem->next->prev = NULL; \
    if(that->tail == elem) \
        that->tail = NULL; \
    type val = elem->data; \
    free(elem); \
    return val; \
}

```

Remove an element from the head of a list

Output and side effect(s):

- Return the removed element's value

```

#define CopyDefListDrop(name, type) \
type name ## Drop(void) { \
    name* that = (name*)copyThat; \
    name ## Elem* elem = that->tail; \
    that->tail = elem->prev; \
    if(elem->prev != NULL) \
        elem->prev->next = NULL; \
    if(that->head == elem) \
}

```



```

    that->head = NULL;           \
    type val = elem->data;       \
    free(elem);                 \
    return val;                 \
}

```

Remove an element from the tail of a list

Output and side effect(s):

- Return the removed element's value

```

#define CopyDefListGet(name, type) \
type name ## Get(size_t const idx) { \
    name* that = (name*)copyThat; \
    name ## Elem* elem = that->head; \
    for(size_t i = 0; i < idx && elem != NULL; ++i) \
        elem = elem->next; \
    if(elem == NULL) raiseExc(CopyExc_InvalidElemIdx); \
    return elem->data; \
}

```

Get the value of the element in the list at the given position

Output and side effect(s):

- Return the element's value

```

#define CopyDefListGetPtr(name, type) \
type* name ## GetPtr(size_t const idx) { \
    name* that = (name*)copyThat; \
    name ## Elem* elem = that->head; \
    for(size_t i = 0; i < idx && elem != NULL; ++i) \
        elem = elem->next; \
    if(elem == NULL) raiseExc(CopyExc_InvalidElemIdx); \
    return &(elem->data); \
}

```

Get a pointer to the value of the element in the list at the given position

Output and side effect(s):

- Return the pointer to the element's value

```

#define CopyDefListFlush(name, type) \
void name ## Flush(void) { \
    name* that = (name*)copyThat; \
    while(that->head != NULL) $(that, pop)(); \
}

```

Empty the list by removing all its elements


```

#define CopyDefListGetHead(name, type) \
type name ## GetHead(void) {          \
    name* that = (name*)copyThat;      \
    return that->head->data;             \
}

```

Get the value of the element at the head of the list

Output and side effect(s):

- Return the element's value

```

#define CopyDefListGetTail(name, type) \
type name ## GetTail(void) {          \
    name* that = (name*)copyThat;      \
    return that->tail->data;             \
}

```

Get the value of the element at the tail of the list

Output and side effect(s):

- Return the element's value

```

#define CopyDefListGetPtrHead(name, type) \
type* name ## GetPtrHead(void) {         \
    name* that = (name*)copyThat;        \
    return &(amp;that->head->data);         \
}

```

Get a pointer to the value of the element at the head of the list

Output and side effect(s):

- Return the pointer to the element's value

```

#define CopyDefListGetPtrTail(name, type) \
type* name ## GetPtrTail(void) {         \
    name* that = (name*)copyThat;        \
    return &(amp;that->tail->data);         \
}

```

Get a pointer to the value of the element at the tail of the list

Output and side effect(s):

- Return the pointer to the element's value


```

#define CopyDefListIsEmpty(name, type)      \
bool name ## IsEmpty(void) {               \
    return ((name*)copyThat)->head == NULL; \
}

```

Check if a list is empty

Output and side effect(s):

- Return true if the list is empty, else false

```

#define CopyDefListRemove(name, type)      \
void name ## Remove(                       \
    CopyComparator* cmp,                   \
    type const* const elem) {              \
    name* that = (name*)copyThat;          \
    name ## Elem* ptr = that->head;         \
    while(ptr != NULL) {                   \
        int resCmp = $(cmp, eval)(&(ptr->data), elem); \
        if(resCmp == 0) {                  \
            name ## Elem* next = ptr->next; \
            if(ptr->prev != NULL) ptr->prev->next = ptr->next; \
            if(ptr->next != NULL) ptr->next->prev = ptr->prev; \
            if(that->head == ptr) that->head = ptr->next; \
            if(that->tail == ptr) that->tail = ptr->prev; \
            free(ptr);                      \
            ptr = next;                     \
        } else ptr = ptr->next;             \
    }                                       \
    $(&(that->iter), reset)();              \
}

```

Remove elements from the list

Input argument(s):

- cmp: comparator to find the element
- elem: the element to be removed

Output and side effect(s):

- All the elements in the list equal (according to the given comparator)
- to the given element are removed. The iterator is reset.

```

#define CopyDefListFind(name, type)      \
type* name ## Find(                       \
    CopyComparator* cmp,                   \
    type const* const elem) {              \
    name* that = (name*)copyThat;          \
    name ## Elem* ptr = that->head;         \
    while(ptr != NULL) {                   \

```



```

    int resCmp = $(cmp, eval)(&(ptr->data), elem);      \
    if(resCmp == 0) return &(ptr->data);                \
    ptr = ptr->next;                                     \
}                                                         \
return NULL;                                           \
}

```

Find an element in the list

Input argument(s):

- cmp: comparator to find the element
- elem: the element to be found

Output and side effect(s):

- The first element in the list equal (according to the given comparator)
- to the given element is returned. If none could be found, return NULL.

```

#define CopyDefListInsertSort(name, type)              \
void name ## InsertSort(                               \
    CopyComparator* cmp,                               \
    type const val) {                                  \
    name* that = (name*)copyThat;                      \
    name ## Elem* ptr = that->head;                    \
    while(ptr != NULL) {                               \
        int resCmp = $(cmp, eval)(&(ptr->data), &val); \
        if(resCmp >= 0) {                              \
            if(ptr->prev == NULL) {                    \
                $(that, push)(val);                   \
            } else {                                    \
                name ## Elem* elem = NULL;            \
                safeMalloc(elem, 1);                   \
                if(!elem) return;                     \
                elem->prev = ptr->prev;                 \
                elem->prev->next = elem;               \
                ptr->prev = elem;                     \
                elem->next = ptr;                     \
                elem->data = val;                     \
            }                                           \
        }                                           \
        return;                                       \
    }                                           \
    ptr = ptr->next;                                   \
}                                           \
$(that, add)(val);                                   \
}

```

Insert an element in the list at a position defined by a comparator

Input argument(s):

- cmp: comparator to find the element position
- elem: the element to be added

Output and side effect(s):

- Insert the element before the first element in the list equal or after
- the added element (according to the given comparator).

```
#define CopyDefListInitIterator(name, type) \
void name ## InitIterator(void) {          \
    name* that = (name*)copyThat;         \
    that->iter = name ## IteratorCreate(   \
        that, copyListIteratorType_forward); \
}
```

Initialise the iterator of the list, must be called after the creation of the list when it is created with Create(), Alloc() automatically initialise the iterator.

```
#define CopyDefList(name, type) \
    CopyDefListCreate(name, type) \
    CopyDefListAlloc(name, type) \
    CopyDefListDestruct(name, type) \
    CopyDefListFree(name, type) \
    CopyDefListGetSize(name, type) \
    CopyDefListPush(name, type) \
    CopyDefListAdd(name, type) \
    CopyDefListPop(name, type) \
    CopyDefListDrop(name, type) \
    CopyDefListGet(name, type) \
    CopyDefListGetPtr(name, type) \
    CopyDefListFlush(name, type) \
    CopyDefListGetHead(name, type) \
    CopyDefListGetTail(name, type) \
    CopyDefListGetPtrHead(name, type) \
    CopyDefListGetPtrTail(name, type) \
    CopyDefListIsEmpty(name, type) \
    CopyDefListRemove(name, type) \
    CopyDefListFind(name, type) \
    CopyDefListInsertSort(name, type) \
    CopyDefListInitIterator(name, type) \
    CopyDefListIterator(name, type)
```

Definition macro calling all the submacros at once for a list structure named 'name', containing elements of type 'type'

55.2 Enumerations

```
typedef enum CopyListIteratorType {
    copyListIteratorType_forward,
    copyListIteratorType_backward,
} CopyListIteratorType;
```

Enumeration for the types of iterator on generic list

55.3 Typedefs

None.

55.4 Functions

None.

56 Unit lsystem.h

LSystem class.

56.1 Macros

```
#define CAPY_LSYSTEM_H
```

```
#define CopyDecLSystem(N, T) \
typedef struct N { \
    T* state; \
    size_t len; \
    void* params; \
    void (*destruct)(void); \
    void (*step)(void); \
    T* (*rule)( \
        size_t const idx, \
        size_t* const lenOut); \
} N; \
void N ## Destruct(void); \
void N ## Step(void); \
N N ## Create( \
    T* const initState, \
    size_t const len); \
N* N ## Alloc( \
    T* const initState, \
    size_t const len); \
void N ## Free(N** const that);
```

Declaration macro for a LSystem object with name N and data type T (must have a destruct method, but maybe null).

```
#define CopyDefLSystem(N, T, R) \
void N ## Destruct(void) { \
    N* that = (N*)copyThat; \
    loop(i, that->len) if(that->state[i].destruct) { \
        $(that->state + i, destruct)(); \
    } \
    free(that->state); \
    that->state = NULL; \
    that->len = 0; \
}
```



```

}
void N ## Step(void) {
    N* that = (N*)copyThat;
    T* res = NULL;
    size_t lenRes = 0;
    loop(idx, that->len) {
        size_t lenOut = 0;
        T* out = $(that, rule)(idx, &lenOut);
        if(out != NULL) {
            safeRealloc(res, lenRes + lenOut);
            assert(res != NULL);
            loop(i, lenOut) res[lenRes + i] = out[i];
            free(out);
            lenRes += lenOut;
        }
    }
    loop(i, that->len) if(that->state[i].destruct) {
        $(that->state + i, destruct)();
    }
    free(that->state);
    that->state = res;
    that->len = lenRes;
}
N N ## Create(
    T* const initState,
    size_t const len) {
    N that = {
        .state = NULL,
        .len = 0,
        .params = NULL,
        .destruct = N ## Destruct,
        .step = N ## Step,
        .rule = R,
    };
    if(len > 0) {
        safeMalloc(that.state, len);
        assert(that.state);
        loop(i, len) that.state[i] = initState[i];
        that.len = len;
    }
    return that;
}
N* N ## Alloc(
    T* const initState,
    size_t const len) {
    N* that = NULL;
    safeMalloc(that, 1);
    if(!that) return NULL;
    *that = N ## Create(initState, len);
    return that;
}
void N ## Free(N** const that) {
    if(that == NULL || *that == NULL) return;
    $(*that, destruct)();
    free(*that);
    *that = NULL;
}

```

Definition macro for a LSystem object with name N, data type T (must have a destruct method), and rule R. The rule take the index of the current symbol

in the current state of the L-System and return a null terminated array of symbols (eventually empty). When calling 'step', the rule is applied to each symbol of the current state, and the current state is replaced by the result state.

56.2 Enumerations

None.

56.3 Typedefs

None.

56.4 Struct CopyLSysLindenMayerAlgaeState

56.4.1 Struct CopyLSysLindenMayerAlgaeState's properties

```
char val;
```

```
CopyPad(char, val);
```

56.4.2 Struct CopyLSysLindenMayerAlgaeState's methods

```
void (*destruct)(void);
```

56.5 Struct CopyLSysKochSnowflakeState

56.5.1 Struct CopyLSysKochSnowflakeState's properties

```
char val;
```

```
CopyPad(char, val);
```

56.5.2 Struct CopyLSysKochSnowflakeState's methods

```
void (*destruct)(void);
```


56.6 Struct CapyLSysFractalBinaryTreeState

56.6.1 Struct CapyLSysFractalBinaryTreeState's properties

```
char val;
```

```
CapyPad(char, val);
```

56.6.2 Struct CapyLSysFractalBinaryTreeState's methods

```
void (*destruct)(void);
```

56.7 Struct CapyLSysHilbertCurveState

56.7.1 Struct CapyLSysHilbertCurveState's properties

```
char val;
```

```
CapyPad(char, val);
```

56.7.2 Struct CapyLSysHilbertCurveState's methods

```
void (*destruct)(void);
```

56.8 Struct CapyLSysHeighwayDragonCurveState

56.8.1 Struct CapyLSysHeighwayDragonCurveState's properties

```
char val;
```

```
CapyPad(char, val);
```


56.8.2 Struct CapyLSysHeighwayDragonCurveState's methods

```
void (*destruct)(void);
```

56.9 Struct CapyLSysGosperCurveState

56.9.1 Struct CapyLSysGosperCurveState's properties

```
char val;
```

```
CapyPad(char, val);
```

56.9.2 Struct CapyLSysGosperCurveState's methods

```
void (*destruct)(void);
```

56.10 Functions

None.

57 Unit markovdecisionprocess.h

MarkovDecisionProcess class.

57.1 Macros

```
#define CAPY_MARKOVDECISIONPROCESS_H
```

```
#define CapyMDPPolicyDef {
    size_t nbState;
    double* values;
    size_t* actions;
    void (*destruct)(void);
    size_t (*getAction)(size_t const state);
    double (*getProbAction)(
        size_t const state,
        size_t const action);
}
```


MarkovDecisionProcess policy definition Number of states size_t nbState;

States value double* values;

States optimal action size_t* actions;

Destructor void (*destruct)(void);

Get the action for a given state

Input argument(s):

- state: the state

Output and side effect(s):

- Return the action
- size_t (*getAction)(size_t const state);
- Get the probability that a given action is selected given a state

Input argument(s):

- state: the state
- action: the action

Output and side effect(s):

- Return the probability in [0,1]
- double (*getProbAction)(
- size_t const state,
- size_t const action);

```
#define CopyMDPEnvironmentDef { \
    void (*destruct)(void); \
    size_t (*step)( \
        size_t const fromState, \
        size_t const action); \
}
```

MarkovDecisionProcess environment definition

Destructor void (*destruct)(void);

Get the result action for a given state

Input argument(s):

- fromState: the 'from' state

- action: the applied action

Output and side effect(s):

- Return the result state
- size_t (*step)(
- size_t const fromState,
- size_t const action);

57.2 Enumerations

None.

57.3 Typedefs

```
typedef struct CapyMDPPolicy CapyMDPPolicyDef CapyMDPPolicy;
```

CapyMDPPolicy object

```
typedef struct CapyMDPEnvironment CapyMDPEnvironmentDef CapyMDPEnvironment;
```

CapyMDPEnvironment object

57.4 Struct CapyMDPPolicyEpsilonSoft

57.4.1 Struct CapyMDPPolicyEpsilonSoft's properties

```
struct CapyMDPPolicyDef;
```

Inherits CapyMDPPolicy

```
CapyRandom rng;
```

Random number generator

```
double epsilon;
```

Epsilon constant for action selection

```
size_t nbAction;
```

Number of action

57.4.2 Struct CapyMDPPolicyEpsilonSoft's methods

```
void (*destructCapyMDPPolicy)(void);
```

Destructor for the parent class

57.5 Struct CapyMDPTransition

57.5.1 Struct CapyMDPTransition's properties

```
size_t fromState;
```

Index of the origin state of the transition

```
size_t action;
```

Index of the action of the transition

```
size_t toState;
```

Index of the termination state of the transition

```
double prob;
```

Probability of transition

```
double reward;
```

Reward for transitioning through that transition

```
double value;
```

Action value of the transition

```
size_t nbOccurence;
```

Number of time this transition has been visited

```
size_t nbVisit;
```

Number of time a transition with same fromState and action has been visited

57.5.2 Struct CapyMDPTransition's methods

None.

57.6 Struct CpyMDPTransitionRecorder

57.6.1 Struct CpyMDPTransitionRecorder's properties

```
size_t nbTransition;
```

Number of transition

```
size_t nbMaxTransition;
```

Size of the recorder memory (in transition number)

```
CpyMDPTransition* transitions;
```

Recorded transitions

57.6.2 Struct CpyMDPTransitionRecorder's methods

```
void (*destruct)(void);
```

Destructor

```
void (*reset)(void);
```

Reset the recorder

Output and side effect(s):

- 'nbTransition' is reset to 0.

```
void (*addTransition)(CpyMDPTransition const* const transition);
```

Record one transition

Input argument(s):

- transition: the transition to be recorded

Output and side effect(s):

- A copy of the transition is added to the end of 'transitions' which
- is reallocated if necessary, 'nbTransition' and 'nbMaxTransition' are
- updated as necessary.

57.7 Struct CapyMarkovDecisionProcess

57.7.1 Struct CapyMarkovDecisionProcess's properties

```
size_t nbState;
```

Number of state

```
size_t nbAction;
```

Number of action

```
size_t nbTransition;
```

Number of transition ($\text{nbState} * \text{nbAction} * \text{nbState}$)

```
CapyMDPTransition* transitions;
```

Transition definition

```
bool* flagStartStates;
```

Start state flags

```
bool* flagEndStates;
```

End state flags

```
CapyMDPPolicy optimalPolicy;
```

Optimal policy

```
size_t curState;
```

Index of the current state (default: 0)

```
size_t nbStep;
```

Number of step executed (reset by `setCurState()` and incremented by `step()`)

```
size_t nbMaxStep;
```

Maximum number of step to avoid infinite stepping (default: 1e9)


```
CapyRandom rng;
```

Pseudo random generator to step the process (initialised with current time)

```
double discount;
```

Discount factor (default 0.9, in $[0,1]$, the lower the more influent the near future rewards compare to far future ones)

```
double epsilon;
```

Epsilon value for convergence during search for optimal policy (default: 1e-6)

```
bool flagEveryVisit;
```

Flag to select between "first visit" and "each visit" during montecarlo search (default: false)

```
CapyPad(bool, flagEveryVisit);
```

```
CapyMDPEnvironment* environment;
```

Reference to the environment modelised by the MDP (default: NULL)

57.7.2 Struct CapyMarkovDecisionProcess's methods

```
void (*destruct)(void);
```

Destructor

```
CapyMDPTransition* (*getTransition)(  
    size_t const fromState,  
    size_t const action,  
    size_t const toState);
```

Get a transition

Input argument(s):

- fromState: index of the origin state
- action: index of the action
- toState: index of the termination state

Output and side effect(s):

- Return a reference to the transition

```
void (*setCurState)(size_t const state);
```

Set the current state

Input argument(s):

- state: index of the current state

Output and side effect(s):

- The current state is set and the number of step is reset

```
size_t (*getCurState)(void);
```

Get the current state

Output and side effect(s):

- Return the index of the current state

```
size_t (*getNbStep)(void);
```

Get the number of step

Output and side effect(s):

- Return the number of step

```
CapyMDPTransition* (*step)(void);
```

Step the MDP according to its transitions definition

Output and side effect(s):

- The current state and the number of step are updated. Return the
- transition.

```
CapyMDPTransition* (*stepPolicy)(  
    CapyMDPPolicy const* const policy);
```


Step the MDP according to a given policy

Output and side effect(s):

- The current state and the number of step are updated. Return the
- transition. If the MDP's environment is known it is used to get the
- result state.

```
void (*resetRng)(CapyRandomSeed_t const seed);
```

Initialise the pseudo random generator

Input argument(s):

- seed: the seed

Output and side effect(s):

- The pseudo random generator is reset.

```
void (*searchOptimalPolicy)(void);
```

Search the optimal policy (given that the MDP's transitions are all set with the correct transitions probabilities and rewards)

Output and side effect(s):

- Calculate the optimal policy, update 'optimalPolicy' which is also
- used as the initial policy for the search

```
double (*getExpReward)(size_t const nbRun);
```

Get the expected sum of reward

Input argument(s):

- nbRun: number of run used to calculate the expected reward

Output and side effect(s):

- Return the expected sum of reward, or 0.0 and raise
- CapyExc_UndefinedExecution if the MDP can't reach an end state within

- that->nbMaxIter. The start state is selected at random. Randomly
- select the transitions according to their probabilities.

```
double (*getExpRewardFromState)(
    size_t const fromState,
    size_t const nbRun);
```

Get the expected sum of reward from a given start state

Input argument(s):

- fromState: the start state
- nbRun: number of run used to calculate the expected reward

Output and side effect(s):

- Return the expected sum of reward, or 0.0 and raise
- CopyExc_UndefinedExecution if the MDP can't reach an end state within
- that->nbMaxIter. Randomly select the transitions according to their
- probabilities.

```
double (*getExpRewardForPolicy)(
    size_t const nbRun,
    CopyMDPPolicy const* const policy);
```

Get the expected sum of reward using a given policy

Input argument(s):

- nbRun: number of run used to calculate the expected reward
- policy: the policy

Output and side effect(s):

- Return the expected sum of reward, or 0.0 and raise
- CopyExc_UndefinedExecution if the MDP can't reach an end state within
- that->nbMaxIter. The start state is selected at random. Select the

- transitions according to the given policy.

```
double (*getExpRewardFromStateForPolicy)(
    size_t const fromState,
    size_t const nbRun,
    CappyMDPPolicy const* const policy);
```

Get the expected sum of reward from a given start state using a given policy

Input argument(s):

- fromState: the start state
- nbRun: number of run used to calculate the expected reward
- policy: the policy

Output and side effect(s):

- Return the expected sum of reward, or 0.0 and raise
- CappyExc_UndefinedExecution if the MDP can't reach an end state within
- that->nbMaxIter. Select the transitions according to the given policy.

```
void (*recordTrajectory)(
    CappyMDPTransitionRecorder* const recorder,
    size_t const startState,
    CappyMDPPolicy const* const policy);
```

Record a trajectory through the MDP given an initial state and a policy

Input argument(s):

- recorder: the recorder
- startState: the initial state of the trajectory
- policy: the policy used to select transitions

Output and side effect(s):

- The recorder is reset and updated with the trajectory. The trajectory
- stops when encountering an end state, or when it reaches
- 'that->nbMaxStep'. The current state of the MDP is modified.


```
void (*qLearning)(
    double const epsilon,
    double const alpha,
    size_t const nbEpisode);
```

Search the optimal policy using Q-Learning (converges to the optimal policy by exploring the environment instead of using transitions probabilities, only needs the transition rewards; uses an epsilon-soft policy to explore the transitions)

Input argument(s):

- epsilon: exploration coefficient (in $]0, 1]$)
- alpha: learning rate (in $]0, 1]$)
- nbEpisode: number of training episodes

Output and side effect(s):

- Calculate the optimal policy, update 'optimalPolicy' which is also
- used as the initial policy for the search.

```
size_t (*getRndStartState)(void);
```

Get a random start state

Output and side effect(s):

- Return one of the start states. If there are no start states, return 0 by
- default.

57.8 Functions

```
CapyMDPPolicy CapyMDPPolicyCreate(size_t const nbState);
```

Create a CapyMDPPolicy

Input argument(s):

- nbState: the number of state

Output and side effect(s):

- Return a CapyMDPPolicy


```
CapyMDPPolicy* CapyMDPPolicyAlloc(size_t const nbState);
```

Allocate memory for a new CapyMDPPolicy and create it

Input argument(s):

- nbState: the number of state

Output and side effect(s):

- Return a CapyMDPPolicy

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyMDPPolicyFree(CapyMDPPolicy** const that);
```

Free the memory used by a CapyMDPPolicy* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyMDPPolicy to free

```
CapyMDPPolicyEpsilonSoft CapyMDPPolicyEpsilonSoftCreate(  
    size_t const nbState,  
    size_t const nbAction,  
    double const epsilon);
```

Create a new CapyMDPPolicyEpsilonSoft

Input argument(s):

- nbState: the number of state
- nbAction: the number of action
- epsilon: the epsilon constant for the action selection

Output and side effect(s):

- Return a CapyMDPPolicyEpsilonSoft

```
CapyMDPPolicyEpsilonSoft* CapyMDPPolicyEpsilonSoftAlloc(  
    size_t const nbState,  
    size_t const nbAction,  
    double const epsilon);
```


Allocate memory for a new `CapyMDPPolicyEpsilonSoft` and create it

Input argument(s):

- `nbState`: the number of state
- `nbAction`: the number of action
- `epsilon`: the epsilon constant for the action selection

Output and side effect(s):

- Return a `CapyMDPPolicyEpsilonSoft`

Exception(s):

- May raise `CapyExc_MallocFailed`.

```
void CapyMDPPolicyEpsilonSoftFree(CapyMDPPolicyEpsilonSoft** const that);
```

Free the memory used by a `CapyMDPPolicyEpsilonSoft*` and reset `*that` to NULL

Input argument(s):

- `that`: a pointer to the `CapyMDPPolicyEpsilonSoft` to free

```
CapyMDPTransitionRecorder CapyMDPTransitionRecorderCreate(void);
```

Create a `CapyMDPTransitionRecorder`

Output and side effect(s):

- Return a `CapyMDPTransitionRecorder`

```
CapyMDPEnvironment CapyMDPEnvironmentCreate(void);
```

Create a `CapyMDPEnvironment`

Output and side effect(s):

- Return a `CapyMDPEnvironment`

```
CapyMDPEnvironment* CapyMDPEnvironmentAlloc(void);
```

Allocate memory for a new `CapyMDPEnvironment` and create it

Output and side effect(s):

- Return a CpyMDPEnvironment

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyMDPEnvironmentFree(CpyMDPEnvironment** const that);
```

Free the memory used by a CpyMDPEnvironment* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyMDPEnvironment to free

```
CpyMarkovDecisionProcess CpyMarkovDecisionProcessCreate(
    size_t const nbState,
    size_t const nbAction);
```

Create a CpyMarkovDecisionProcess

Input argument(s):

- nbState: the number of state
- nbAction: the number of action

Output and side effect(s):

- Return a CpyMarkovDecisionProcess

```
CpyMarkovDecisionProcess* CpyMarkovDecisionProcessAlloc(
    size_t const nbState,
    size_t const nbAction);
```

Allocate memory for a new CpyMarkovDecisionProcess and create it

Input argument(s):

- nbState: the number of state
- nbAction: the number of action

Output and side effect(s):

- Return a CpyMarkovDecisionProcess

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyMarkovDecisionProcessFree(CpyMarkovDecisionProcess** const that);
```

Free the memory used by a CpyMarkovDecisionProcess* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyMarkovDecisionProcess to free

58 Unit mathfun.h

Class to manipulate mathematical functions.

58.1 Macros

```
#define CAPY_MATHFUN_H
```

```
#define CpyMathFunDef {
    union {
        size_t dims[2];
        struct __attribute__((packed)) {
            size_t dimIn, dimOut;
        };
    };
    size_t nbIterSolveNewtonMethod;
    CpyRangeDouble* domains;
    double* outBuffer;
    void (*destruct)(void);
    void (*eval)(
        double const* const in,
        double* const out);
    void (*evalJacobian)(
        double* const in,
        double* const jacobian);
    void (*evalDerivative)(
        double* const in,
        size_t const iDim,
        double* const out);
    void (*evalIntegral)(
        CpyRangeDouble const* const domains,
        double* const out);
    double (*evalDivergence)(
        double* const in);
    void (*solveByNewtonMethod)(
        double* const in,
        double const* const out);
}
```

CpyMathFun object definition macro

58.2 Enumerations

None.

58.3 Typedefs

```
typedef struct CapyMathFun CapyMathFunDef CapyMathFun;
```

CapyMathFun object

```
typedef struct CapyDataset CapyDataset;
```

Predeclaration

58.4 Struct CapyHyperplane

58.4.1 Struct CapyHyperplane's properties

```
struct CapyMathFunDef;
```

Inherit CapyMathFun

```
CapyVec u;
```

Hyperplane parameters, the hyperplane is defined as the points X such as $H(X)=u.[X-1]=0$

58.4.2 Struct CapyHyperplane's methods

```
void (*destructCapyMathFun)(void);
```

Destructor

58.5 Struct CapyLinCombFun

58.5.1 Struct CapyLinCombFun's properties

```
struct CapyMathFunDef;
```

Inherit CapyMathFun


```
size_t nbComb;
```

Number of linearly combined functions

```
CapyVec coeff;
```

Vector of coefficients for the combination

```
CapyVec bias;
```

Bias coefficient

```
CapyVec out;
```

Temporary vector for calculation

```
CapyMathFun** combFuns;
```

References to the combined functions

58.5.2 Struct CapyLinCombFun's methods

```
void (*destructCapyMathFun)(void);
```

Destructor

```
void (*linearRegression)(  
    CapyDataset const* const dataset,  
    size_t const iOutput);
```

Calculate the combination coefficients that best fit a dataset

Input argument(s):

- dataset: the dataset
- iOutput: the index of the output

Output and side effect(s):

- Calculate the best fit coefficients using linear regression. The
- number of input fields in the dataset must match the input dimension
- of combined functions. Can be used for dimOut=1 only.

58.6 Functions

```
CapyMathFun CapyMathFunCreate(  
    size_t const dimIn,  
    size_t const dimOut);
```

Create a CapyMathFun

Input argument(s):

- dimIn: input dimension
- dimOut: output dimension

Output and side effect(s):

- Return a CapyMathFun. domains set by default to [0, 1]

```
CapyHyperplane CapyHyperplaneCreate(size_t const dimIn);
```

Create a CapyHyperplane

Input argument(s):

- dimIn: input dimension

Output and side effect(s):

- Return a CapyHyperplane.

```
CapyHyperplane* CapyHyperplaneAlloc(size_t const dimIn);
```

Allocate memory and create a CapyHyperplane

Input argument(s):

- dimIn: input dimension

Output and side effect(s):

- Return a CapyHyperplane.

```
void CapyHyperplaneFree(CapyHyperplane** const that);
```

Free the memory used by a CapyHyperplane* and reset '*that' to NULL

Input argument(s):

- that: the CappyHyperplane to free

```
CapyLinCombFun CapyLinCombFunCreate(
    size_t const nbComb,
    size_t const dimIn,
    size_t const dimOut);
```

Create a CappyLinCombFun

Input argument(s):

- nbComb: number of linearly combined functions
- dimIn: input dimension
- dimOut: output dimension

Output and side effect(s):

- Return a CappyLinCombFun.

```
CapyLinCombFun* CapyLinCombFunAlloc(
    size_t const nbComb,
    size_t const dimIn,
    size_t const dimOut);
```

Allocate memory and create a CappyLinCombFun

Input argument(s):

- nbComb: number of linearly combined functions
- dimIn: input dimension
- dimOut: output dimension

Output and side effect(s):

- Return a CappyLinCombFun.

```
void CapyLinCombFunFree(CapyLinCombFun** const that);
```

Free the memory used by a CappyLinCombFun* and reset '*that' to NULL

Input argument(s):

- that: the CappyLinCombFun to free

59 Unit maze.h

Maze class.

59.1 Macros

```
#define CAPY_MAZE_H
```

59.2 Enumerations

None.

59.3 Typedefs

None.

59.4 Struct CapyMaze2D

59.4.1 Struct CapyMaze2D's properties

```
size_t width;
```

Width of the maze

```
size_t height;
```

Height of the maze

```
CapyGraph graph;
```

Graph representing the maze, nodes are rooms and links are connections between rooms. Node id of room at position (x,y) is equal to (x+y*width)

59.4.2 Struct CapyMaze2D's methods

```
void (*destruct)(void);
```

Destructor

```
void (*generateByOriginShift)(CapyRandom* const rng);
```


Generate the maze using the origin shift algorithm

Input argument(s):

- rng: the pseudo random generator

Output and side effect(s):

- Empty the graph if it is not empty, then create a graph equivalent
- to a 2D maze where rooms are represented with nodes laying on a 2D
- grid of maze dimensions, and corridors between rooms are represented
- with links. The maze is guaranteed to have one single path between each
- pair of rooms. All room are 1x1 in dimension, and all room in the 2D
- grid exist. The maze contains no loop and no island.

```
void (*draw)(
    CapiImg* const img,
    CapiPen const* const pen,
    CapiImgDims const dims,
    CapiImgPos const pos);
```

Draw the maze

Input argument(s):

- img: the image on which to draw
- pen: the pen to draw the maze
- dim: the dimension of the maze in the image
- pos: the position of the top-left corner of maze in the image

Output and side effect(s):

- Draw the maze on the image by drawing vertical and horizontal lines
- between unconnected rooms.

59.5 Functions

```
CapyMaze2D CapyMaze2DCreate(  
    size_t const width,  
    size_t const height);
```

Create a CapyMaze2D

Input argument(s):

- width: the width of the maze
- height: the height of the maze

Output and side effect(s):

- Return a CapyMaze2D

```
CapyMaze2D* CapyMaze2DAlloc(  
    size_t const width,  
    size_t const height);
```

Allocate memory for a new CapyMaze2D and create it width: the width of the maze height: the height of the maze

Output and side effect(s):

- Return a CapyMaze2D

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyMaze2DFree(CapyMaze2D** const that);
```

Free the memory used by a CapyMaze2D* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyMaze2D to free

60 Unit memoryPool.h

Memory pool generic class.

60.1 Macros

```
#define CAPY_MEMORY_POOL_H
```

```
#define CapyDecMemPool(N, T) \
typedef struct N { \
    T* head; \
    T* tail; \
    size_t sizeMax; \
    size_t size; \
    size_t sizeElem; \
    size_t nbUsed; \
    void (*destruct)(void); \
    T* (*alloc)(void const* const initElem); \
    void (*free)(T** const elem); \
    void (*flush)(void); \
} N; \
N N ## Create(size_t const sizeElem); \
N* N ## Alloc(size_t const sizeElem); \
void N ## Destruct(void); \
void N ## Free(N** const that); \
T* N ## AllocElem(void const* const initElem); \
void N ## FreeElem(T** const elem); \
void N ## Flush(void);
```

Generic memory pool class. Declaration macro for a memory pool of a given type. The definition of the type must include CapyMemPoolFields (see below).

```
#define CapyMemPoolFields(T) \
struct { \
    T* prev; \
    T* next; \
    bool isFree; \
    CapyPad(bool, MemPoolFieldIsFree); \
} mempool
```

Macro to add necessary fields to a structure to make it usable with CapyMemPool

```
#define CapyDefMemPoolCreate(N, T) \
N N ## Create(size_t const sizeElem) { \
    return (N){ \
        .head = NULL, \
        .tail = NULL, \
        .sizeMax = 0, \
        .size = 0, \
        .sizeElem = sizeElem, \
        .nbUsed = 0, \
        .destruct = N ## Destruct, \
        .alloc = N ## AllocElem, \
        .free = N ## FreeElem, \
        .flush = N ## Flush, \
    }; \
}
```



```
}

```

Create a memory pool

Output and side effect(s):

- Return an empty memory pool

```
#define CopyDefMemPoolAlloc(N, T) \
N* N ## Alloc(size_t const sizeElem) { \
    N* that = NULL; \
    safeMalloc(that, 1); \
    assert(that != NULL); \
    *that = N ## Create(sizeElem); \
    return that; \
}

```

Allocate a memory pool

Output and side effect(s):

- Return an empty memory pool

```
#define CopyDefMemPoolDestruct(N, T) \
void N ## Destruct(void) { \
    N* that = (N*)copyThat; \
    $(that, flush)(); \
}

```

Flush and destruct a memory pool

Input argument(s):

- that: the memory pool to destruct

```
#define CopyDefMemPoolFree(N, T) \
void N ## Free(N** const that) { \
    if(that == NULL || *that == NULL) return; \
    $(*that, destruct)(); \
    free(*that); \
    *that = NULL; \
}

```

Flush and free a memory pool

Input argument(s):

- that: the memory pool to free


```

#define CopyDefMemPoolAllocElem(N, T) \
T* N ## AllocElem(void const* const initElem) { \
    N* that = (N*)copyThat; \
    T* ptr = that->head; \
    if(ptr && ptr->mempool.isFree) { \
        if(ptr->mempool.next) { \
            that->head = ptr->mempool.next; \
            that->head->mempool.prev = NULL; \
        } \
        if(initElem) memcpy(ptr, initElem, that->sizeElem); \
        if(ptr != that->tail) { \
            ptr->mempool.next = NULL; \
            ptr->mempool.prev = that->tail; \
            that->tail->mempool.next = ptr; \
            that->tail = ptr; \
        } \
    } else if(that->sizeMax == 0 || that->size < that->sizeMax) { \
        char* mem = NULL; \
        safeMalloc(mem, that->sizeElem); \
        if(mem == NULL) return NULL; \
        if(initElem) memcpy(mem, initElem, that->sizeElem); \
        else memset(mem, 0, that->sizeElem); \
        ptr = (T*)mem; \
        ++(that->size); \
        ptr->mempool.prev = that->tail; \
        ptr->mempool.next = NULL; \
        if(that->tail == NULL) that->head = that->tail = ptr; \
        else { \
            that->tail->mempool.next = ptr; \
            that->tail = ptr; \
        } \
    } else return NULL; \
    ptr->mempool.isFree = false; \
    ++(that->nbUsed); \
    return ptr; \
}

```

Alloc memory using the memory pool

Input argument(s):

- initElem: if not null the allocated memory is initialized with initElem,
- must be of size sizeElem bytes

Output and side effect(s):

- Return a 'new' element from the pool. If there was a free one, it is
- reused, else a new element is allocated and added to the pool before
- returning it.


```

#define CopyDefMemPoolFreeElem(N, T) \
void N ## FreeElem(T** const elem) { \
    if(elem == NULL || *elem == NULL) return; \
    N* that = (N*)copyThat; \
    if((*elem)->destruct) $(*elem, destruct)(); \
    (*elem)->mempool.isFree = true; \
    --(that->nbUsed); \
    if((*elem) != that->head) { \
        (*elem)->mempool.prev->mempool.next = (*elem)->mempool.next; \
        if((*elem)->mempool.next) { \
            (*elem)->mempool.next->mempool.prev = (*elem)->mempool.prev; \
        } \
        if(that->tail == (*elem)) that->tail = (*elem)->mempool.prev; \
        (*elem)->mempool.prev = NULL; \
        (*elem)->mempool.next = that->head; \
        that->head->mempool.prev = (*elem); \
        that->head = (*elem); \
    } \
    *elem = NULL; \
}

```

Free an element allocated using the pool

Input argument(s):

- elem: pointer to the element to free

Output and side effect(s):

- The element is marked as free to reuse in the memory pool. The destructor
- is applied to the element.

```

#define CopyDefMemPoolFlush(N, T) \
void N ## Flush(void) { \
    N* that = (N*)copyThat; \
    T* ptr = that->head; \
    while(ptr) { \
        T* next = ptr->mempool.next; \
        if(ptr->mempool.isFree == false && ptr->destruct) \
            $(ptr, destruct)(); \
        free(ptr); \
        ptr = next; \
    } \
    that->nbUsed = 0; \
    that->size = 0; \
    that->head = that->tail = NULL; \
}

```

Flush a memory pool

Output and side effect(s):

- All the elements in the pool are freed (their memory is released using the

- standard 'free' function). The destructor is applied to all remaining
- element in the pool

```
#define CapyDefMemPool(N, T) \
    CapyDefMemPoolCreate(N, T) \
    CapyDefMemPoolAlloc(N, T) \
    CapyDefMemPoolDestruct(N, T) \
    CapyDefMemPoolFree(N, T) \
    CapyDefMemPoolAllocElem(N, T) \
    CapyDefMemPoolFreeElem(N, T) \
    CapyDefMemPoolFlush(N, T)
```

Definition macro for a memory pool of a given type.

60.2 Enumerations

None.

60.3 Typedefs

None.

60.4 Functions

None.

61 Unit minimax.h

MiniMax class.

61.1 Macros

```
#define CAPY_MINIMAX_H
```

```
#define CAPY_MINIMAX_NB_MAX_ACTOR 4
```

Maximum number of actors in a MiniMaxWorld


```

#define CopyMiniMaxWorldDef { \
    size_t nbActor; \
    size_t sente; \
    uint64_t depth; \
    CopyMiniMaxWorld* parent; \
    CopyMiniMaxWorld* child; \
    CopyMiniMaxWorld* sibling; \
    bool flagAvoid; \
    CopyPad(bool, flagAvoid); \
    double values[COPY_MINIMAX_NB_MAX_ACTOR]; \
    CopyMemPoolFields(CopyMiniMaxWorld); \
    void (*destruct)(void); \
    CopyMiniMaxWorld* (*clone)(CopyMiniMaxWorldMemPool* const mempool); \
    void (*createChilds)(CopyMiniMaxWorldMemPool* const mempool); \
    bool (*isSame)(CopyMiniMaxWorld const* const world); \
}

```

Macro to define the MiniMaxWorld structure

61.2 Enumerations

None.

61.3 Typedefs

```
typedef struct CopyMiniMaxWorld CopyMiniMaxWorld;
```

Predeclaration of the parent structure for MiniMax worlds

```
typedef struct CopyMiniMaxWorldMemPool CopyMiniMaxWorldMemPool;
```

Predeclaration of the memory pool of MiniMaxWorld

61.4 Struct CopyMiniMax

61.4.1 Struct CopyMiniMax's properties

```
pthread_t exploreThread;
```

Exploration thread.

```
pthread_mutex_t exploreMutex;
```

Mutex to control the exploration thread.

```
bool flagStop;
```


Flag to stop the exploration thread

```
CapyPad(bool, flagStop);
```

```
uint64_t maxDepthExplore;
```

Max depth exploration (default: 1)

```
double pruningThreshold;
```

Pruning threshold (default: 1.0)

```
CapyMiniMaxWorld* currentWorld;
```

Current state

```
CapyMiniMaxWorldMemPool mempool;
```

Memory pool for the worlds

```
size_t nbIter;
```

Counter for the number of iteration, incremented each time setCurrentWorld is called

61.4.2 Struct CapyMiniMax's methods

```
void (*destruct)(void);
```

Destructor

```
void (*start)(void);
```

Start the exploration

```
void (*stop)(void);
```

Stop the exploration

```
CapyMiniMaxWorld* (*getNextBest)(void);
```

Get the next best world

Output and side effect(s):

- Return a clone of the next best world.

```
void (*setCurrentWorld)(CopyMiniMaxWorld* const world);
```

Set the new current world.

Input argument(s):

- world: the new current world

Output and side effect(s):

- Free the current history except for the world in argument and its subtree
- if it's in the current history, and set the world in argument as the new
- root of the history.

61.5 Functions

```
CopyMiniMaxWorld CopyMiniMaxWorldCreate(size_t const nbActor);
```

Create a MiniMaxWorld and return it. Inputs: nbActor: the number of actors

Output and side effect(s):

- Return the MiniMaxWorld.

```
void CopyMiniMaxWorldFree(
    CopyMiniMaxWorld** const that,
    CopyMiniMaxWorldMemPool* const mempool);
```

Free a MiniMaxWorld Inputs: that: the world to free mempool: the memory pool this world is belonging to

Output and side effect(s):

- Free the word in the memory pool.

```
CopyMiniMax CopyMiniMaxCreate(size_t const sizeWorld);
```

Create a CopyMiniMax

Input argument(s):

- sizeWorld: size in byte of the world structure

Output and side effect(s):

- Return a CpyMiniMax

```
CpyMiniMax* CpyMiniMaxAlloc(size_t const sizeWorld);
```

Allocate memory for a new CpyMiniMax and create it

Input argument(s):

- sizeWorld: size in byte of the world structure

Output and side effect(s):

- Return a CpyMiniMax

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyMiniMaxFree(CpyMiniMax** const that);
```

Free the memory used by a CpyMiniMax* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyMiniMax to free

62 Unit mirrorballcamera.h

MirrorBallCamera class.

62.1 Macros

```
#define CAPY_MIRRORBALLCAMERA_H
```

62.2 Enumerations

None.

62.3 Typedefs

None.

62.4 Struct CpyMirrorBallCamera

62.4.1 Struct CpyMirrorBallCamera's properties

```
CpyImg* img;
```

Image of the mirror ball (must be square and the mirror ball must fill the image)

```
CpyCamera* camera;
```

Camera to control the view (front=0,0,1 is considered to correspond to looking toward the center of the mirror ball image)

```
double alpha;
```

Distortion correction factor (default: $\pi/2$, in $]0, \pi/2]$)

62.4.2 Struct CpyMirrorBallCamera's methods

```
void (*destruct)(void);
```

Destructor

```
CpyImgPos (*fromViewDirectionToImgPos)(double const* const dir);
```

Calculate the coordinates in the mirror ball image for that direction vector

Input argument(s):

- dir: the view direction (normalised, 0,0,1 is toward the center of the mirror ball image)

Output and side effect(s):

- Return the position in the mirror ball image for the given direction.
- May return invalid position for invisible direction.


```
CapyImg* (*render)(CapyImgDims const dims);
```

Render the view from the mirror ball given the current camera.

Input argument(s):

- dims: dimensions of the result image

Output and side effect(s):

- Return the rendered image.

62.5 Functions

```
CapyMirrorBallCamera CapyMirrorBallCameraCreate(CapyImg* const img);
```

Create a CapyMirrorBallCamera

Input argument(s):

- img: the image of the mirror ball

Output and side effect(s):

- Return a CapyMirrorBallCamera

```
CapyMirrorBallCamera* CapyMirrorBallCameraAlloc(CapyImg* const img);
```

Allocate memory for a new CapyMirrorBallCamera and create it

Input argument(s):

- img: the image of the mirror ball

Output and side effect(s):

- Return a CapyMirrorBallCamera

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyMirrorBallCameraFree(CapyMirrorBallCamera** const that);
```

Free the memory used by a CapyMirrorBallCamera* and reset **that' to NULL

Input argument(s):

- that: a pointer to the CapyMirrorBallCamera to free

63 Unit neuralNetwork.h

Class implementing a neural network.

63.1 Macros

```
#define CAPY_NEURALNETWORK_H
```

```
#define CapyNNActivationFunDef struct { \
    struct CapyMathFunDef; \
    CapyNNActivationFunType type; \
    CapyPad(CapyNNActivationFunType, type); \
    void (*destructCapyMathFun)(void); \
    void (*exportToCFun)( \
        FILE* const stream, \
        char const* const name); \
    void (*exportToJavascript)( \
        FILE* const stream, \
        char const* const name); \
}
```

Activation function class definition macro

```
#define CapyNNLayerDef_ \
    struct { \
        size_t nbNode; \
        CapyNNActivationFun* activation; \
    }
```

Macro of the definition of one layer Number of nodes size_t nbNode; Reference to the activation function for the nodes of this layer CapyMathFun* activation;

63.2 Enumerations

```
typedef enum CapyNNActivationFunType { \
    capyNNActivationFun_none, \
    capyNNActivationFun_linear, \
    capyNNActivationFun_step, \
    capyNNActivationFun_sigmoid, \
    capyNNActivationFun_hypertangent, \
    capyNNActivationFun_relu, \
    capyNNActivationFun_silu, \
    capyNNActivationFun_nb, \
} CapyNNActivationFunType;
```

Types of activation function

63.3 Typedefs

```
typedef CapyNNActivationFunDef CapyNNActivationFun;
```

Activation function class

```
typedef CapyNNLayerDef_ CapyNNLayerDef;
```

Definition of one layer

63.4 Struct CapyNNLink

63.4.1 Struct CapyNNLink's properties

```
size_t iLayer;
```

Index of the input layer

```
size_t iNode;
```

Index of the input node in the layer

```
size_t idxWeight;
```

Index of the weight of the link in the array of params

63.4.2 Struct CapyNNLink's methods

None.

63.5 Struct CapyNNNode

63.5.1 Struct CapyNNNode's properties

```
double valInput;
```

Value of the node before applying the activation function (i.e. $w.x+b$)

```
double val;
```

Value of the node after applying the activation function (i.e. $A(w.x+b)$)


```
double prevVal;
```

Previous value of the node

```
double propagatingDeriv;
```

Variable to memorise and reuse the propagating derivation during training

```
size_t nbLink;
```

Number of input links

```
CapyNNLink* links;
```

Input links

```
size_t idxBias;
```

Index of the bias of the node in the array of params

63.5.2 Struct CapyNNNode's methods

None.

63.6 Struct CapyNNLayer

63.6.1 Struct CapyNNLayer's properties

```
CapyNNLayerDef_;
```

Layer definition

```
size_t depth;
```

Depth of the layer

```
CapyNNNode* nodes;
```

Nodes

63.6.2 Struct CapyNNLayer's methods

None.

63.7 Struct CpyNNModel

63.7.1 Struct CpyNNModel's properties

```
size_t nbLayer;
```

Number of layers

```
CpyNNLayerDef* layers;
```

Number of node and activation function per layer

63.7.2 Struct CpyNNModel's methods

None.

63.8 Struct CpyNeuralNetwork

63.8.1 Struct CpyNeuralNetwork's properties

```
size_t nbInput;
```

Number of input

```
size_t nbOutput;
```

Number of output

```
size_t nbLayer;
```

Number of layers

```
CpyNNLayer* layers;
```

Layers (including input and output layers)

```
size_t nbParam;
```

Total number of params (weights and biases) in the model

```
double* params;
```

Array of all weights and biases, ordered by layer and node


```
bool flagSoftmaxOutput;
```

Flag to automatically apply sigmoid function on output of evaluation (default: false)

```
CapyPad(bool, flagSoftmaxOutput);
```

63.8.2 Struct CapyNeuralNetwork's methods

```
void (*destruct)(void);
```

Destructor

```
void (*eval)(  
    double const* const in,  
    double* const out);
```

Evaluate the neural network

Input argument(s):

- in: the values of the first layer
- out: result values (nbOutput array)

Output and side effect(s):

- 'out' is updated with the value of the last layer after evaluation.

```
void (*save)(FILE* const stream);
```

Save the neural network to a stream Input: stream: the stream on which to save Output: The neural network is saved on the stream

```
void (*evalGradient)(  
    double const* const in,  
    double* const out,  
    size_t const iOutput);
```

Evaluate the gradient of a neural network relative to one of its output

Input argument(s):

- in: the values of the first layer
- out: result values (nbParam array)

- iOutput: output used to calculate the gradient

Output and side effect(s):

- 'out' is updated with the value of the gradient of 'params' relative to
- 'iOutput'.

63.9 Struct CpyNNActivationLinear

63.9.1 Struct CpyNNActivationLinear's properties

<code>CpyNNActivationFunDef;</code>

63.9.2 Struct CpyNNActivationLinear's methods

None.

63.10 Struct CpyNNActivationStep

63.10.1 Struct CpyNNActivationStep's properties

<code>CpyNNActivationFunDef;</code>

63.10.2 Struct CpyNNActivationStep's methods

None.

63.11 Struct CpyNNActivationSigmoid

63.11.1 Struct CpyNNActivationSigmoid's properties

<code>CpyNNActivationFunDef;</code>

63.11.2 Struct CpyNNActivationSigmoid's methods

None.

63.12 Struct CpyNNActivationHyperTangent

63.12.1 Struct CpyNNActivationHyperTangent's properties

```
CpyNNActivationFunDef;
```

63.12.2 Struct CpyNNActivationHyperTangent's methods

None.

63.13 Struct CpyNNActivationReLU

63.13.1 Struct CpyNNActivationReLU's properties

```
CpyNNActivationFunDef;
```

63.13.2 Struct CpyNNActivationReLU's methods

None.

63.14 Struct CpyNNActivationSiLU

63.14.1 Struct CpyNNActivationSiLU's properties

```
CpyNNActivationFunDef;
```

63.14.2 Struct CpyNNActivationSiLU's methods

None.

63.15 Functions

```
CpyNNModel CpyNNModelCopy(CpyNNModel const* const that);
```

Copy a CpyNNModel

Input argument(s):

- that: the model to copy

Output and side effect(s):

- Return a copy.

```
void CappyNNModelDestruct(CappyNNModel* const that);
```

Destruct a CappyNNModel

Input argument(s):

- that: the model to destruct

Output and side effect(s):

- The model is destructed.

```
CapyNeuralNetwork CappyNeuralNetworkCreateFullyConnected(  
    size_t const nbInput,  
    CappyNNModel const* const model,  
    size_t const nbOutput);
```

Create a CappyNeuralNetwork

Input argument(s):

- nbInput: size of the input vector
- model: the layers model
- nbOutput: size of the output vector

Output and side effect(s):

- Return a CappyNeuralNetwork

```
CapyNeuralNetwork* CappyNeuralNetworkAllocFullyConnected(  
    size_t const nbInput,  
    CappyNNModel const* const model,  
    size_t const nbOutput);
```

Allocate memory for a new CappyNeuralNetwork and create it

Input argument(s):

- nbInput: size of the input vector
- model: the layers model

- nbOutput: size of the output vector

Output and side effect(s):

- Return a CapyNeuralNetwork

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyNeuralNetworkFree(CapyNeuralNetwork** const that);
```

Free the memory used by a CapyNeuralNetwork* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CapyNeuralNetwork to free

```
CapyNeuralNetwork* CapyNeuralNetworkLoad(
    FILE* const stream,
    CapyNNModel* const model);
```

Load a CapyNeuralNetwork from a stream

Input argument(s):

- stream: the stream from which the NN is loaded
- model: CapyNNModel updated with the model of the loaded NN

Output and side effect(s):

- Return a CapyNeuralNetwork

```
CapyNNActivationLinear CapyNNActivationLinearCreate(void);
```

Create a step activation function

Output and side effect(s):

- Return a CapyMathFun implementing a step function

```
CapyNNActivationLinear* CapyNNActivationLinearAlloc(void);
```

Allocate memory and create a step activation function

Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
void CpyNNActivationLinearFree(CpyNNActivationLinear** const that);
```

Free the memory used by a
Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
CpyNNActivationStep CpyNNActivationStepCreate(void);
```

Create a step activation function
Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
CpyNNActivationStep* CpyNNActivationStepAlloc(void);
```

Allocate memory and create a step activation function
Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
void CpyNNActivationStepFree(CpyNNActivationStep** const that);
```

Free the memory used by a
Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
CpyNNActivationSigmoid CpyNNActivationSigmoidCreate(void);
```

Create a step activation function
Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
CpyNNActivationSigmoid* CpyNNActivationSigmoidAlloc(void);
```


Allocate memory and create a step activation function

Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
void CpyNNActivationSigmoidFree(CpyNNActivationSigmoid** const that);
```

Free the memory used by a

Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
CpyNNActivationHyperTangent CpyNNActivationHyperTangentCreate(void);
```

Create a step activation function

Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
CpyNNActivationHyperTangent* CpyNNActivationHyperTangentAlloc(void);
```

Allocate memory and create a step activation function

Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
void CpyNNActivationHyperTangentFree(  
    CpyNNActivationHyperTangent** const that);
```

Free the memory used by a

Output and side effect(s):

- Return a CpyMathFun implementing a step function

```
CpyNNActivationReLU CpyNNActivationReLUCreate(void);
```

Create a step activation function

Output and side effect(s):

- Return a CpyMathFun implementing a step function


```
CapyNNActivationReLU* CapyNNActivationReLUAlloc(void);
```

Allocate memory and create a step activation function

Output and side effect(s):

- Return a CapyMathFun implementing a step function

```
void CapyNNActivationReLUFree(CapyNNActivationReLU** const that);
```

Free the memory used by a

Output and side effect(s):

- Return a CapyMathFun implementing a step function

```
CapyNNActivationSiLU CapyNNActivationSiLUCreate(void);
```

Create a step activation function

Output and side effect(s):

- Return a CapyMathFun implementing a step function

```
CapyNNActivationSiLU* CapyNNActivationSiLUAlloc(void);
```

Allocate memory and create a step activation function

Output and side effect(s):

- Return a CapyMathFun implementing a step function

```
void CapyNNActivationSiLUFree(CapyNNActivationSiLU** const that);
```

Free the memory used by a

Output and side effect(s):

- Return a CapyMathFun implementing a step function

64 Unit nnPredictor.h

Class implementing a predictor based on a fully connected neural network.

64.1 Macros

```
#define CAPY_NNPREDICTOR_H
```

64.2 Enumerations

```
typedef enum {  
    capyNNPredictorLossType_mse,  
    capyNNPredictorLossType_mae,  
    capyNNPredictorLossType_huber,  
    capyNNPredictorLossType_cross_entropy,  
    capyNNPredictorLossType_nb,  
} CapyNNPredictorLossType;
```

Type of loss function used for training

64.3 Typedefs

None.

64.4 Struct CapyNNPredictor

64.4.1 Struct CapyNNPredictor's properties

```
CapyPredictorDef;
```

Inherits CapyPredictor

```
CapyNNModel nnModel;
```

Neural network model

```
CapyNeuralNetwork* nn;
```

Neural network

```
bool verbose;
```

Verbose mode (default: false)

```
CapyPad(bool, 1);
```



```
double timeTraining;
```

Time available for training (in second, default: 60, no time limit if 0)

```
size_t nbIterTrainMax;
```

Number of iteration available for training (default: 0, no limit if 0)

```
size_t batchSize;
```

Batch size for training (default: 100)

```
size_t nbIterTrain;
```

Counter for iteration during training

```
double learnRate;
```

Step size for the gradient descent (default: 0.1)

```
double momentum;
```

Momentum for the gradient descent (default: 0.1)

```
CapyRandomSeed_t seed;
```

Seed for the pseudo random generator (default: 0)

```
double bestLoss;
```

Best loss during training

```
double gradientNorm;
```

Current gradient norm during training

```
double gradientNormEpsilon;
```

Threshold to stop the training if the gradient is null (default: 1e-6)

```
double lossEpsilon;
```

Threshold to stop the training if the loss is null (default: 1e-6)

```
CapyGradientDescentType gradientDescentType;
```


Type of gradient descent (default: adam)

```
CapyPad(CapyGradientDescentType, gradientDescentType);
```

```
double huberThreshold;
```

Threshold for Huber loss (default: 1.0)

```
double decayRates[2];
```

Decay rates for adam gradient descent (default: [0.9, 0.999])

```
CapyNNPredictorLossType lossType;
```

Type of loss function (default: capyNNPredictorLossType_mse)

```
CapyPad(CapyNNPredictorLossType, lossType);
```

64.4.2 Struct CapyNNPredictor's methods

```
void (*destructCapyPredictor)(void);
```

Destructor

```
void (*initParams)(double* const params);
```

Initialise the parameters value

Input argument(s):

- params: the array of parameters
- Ouput:
- The array of parameters is updated

```
void (*exportBodyToHtml)(
    FILE* const stream,
    char const* const title,
    CapyDataset const* const dataset,
    double const expectedAccuracy);
```

Function herited from the parent to export the body to HTML.

Input argument(s):

- stream: the stream where to export
- title: the title of the web app
- dataset: the training dataset
- expectedAccuracy: the expected accuracy of the predictor (in [0,1])

Output and side effect(s):

- The <head> and <body> part of a ready to use web app implementing the
- predictor is written on the stream. The web app can be completed by
- calling exportToHtml on the predictor to write the <script> part.

64.5 Functions

```
CapyNNPredictor CapyNNPredictorCreate(
    CapyNNModel const* const model,
    CapyPredictorType const type);
```

Create a CapyNNPredictor

Input argument(s):

- model: the neural network model
- type: type of predictor

Output and side effect(s):

- Return a CapyNNPredictor

```
CapyNNPredictor* CapyNNPredictorAlloc(
    CapyNNModel const* const model,
    CapyPredictorType const type);
```

Allocate memory for a new CapyNNPredictor and create it

Input argument(s):

- model: the neural network model
- type: type of predictor

Output and side effect(s):

- Return a CpyNNPredictor

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyNNPredictorFree(CpyNNPredictor** const that);
```

Free the memory used by a CpyNNPredictor* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyNNPredictor to free

```
CpyNNPredictor* CpyNNPredictorLoad(FILE* const stream);
```

Load a CpyNNPredictor from a stream

Input argument(s):

- stream: the stream from which the predictor is loaded

Output and side effect(s):

- Return a CpyNNPredictor

65 Unit noise.h

Noise generator parent class.

65.1 Macros

```
#define CAPY_NOISE_H
```

```
#define CpyNoiseDef struct {          \
    struct CpyMathFunDef;              \
    CpyRandom rng;                     \
    void (*destructCpyMathFun)(void);   \
    void (*init)(void);                 \
}
```

CpyNoise object definition macro

65.2 Enumerations

None.

65.3 Typedefs

```
typedef CapyNoiseDef CapyNoise;
```

CapyNoise object

65.4 Functions

```
CapyNoise CapyNoiseCreate(  
    CapyRandomSeed_t const seed,  
        size_t const dimIn,  
        size_t const dimOut);
```

Create a CapyNoise

Input argument(s):

- seed: seed of the noise
- dimIn: number of inputs
- dimOut: number of outputs

Output and side effect(s):

- Return a CapyNoise

```
CapyNoise* CapyNoiseAlloc(  
    CapyRandomSeed_t const seed,  
        size_t const dimIn,  
        size_t const dimOut);
```

Allocate memory for a new CapyNoise and create it

Input argument(s):

- seed: seed of the noise
- dimIn: number of inputs
- dimOut: number of outputs

Output and side effect(s):

- Return a CappyNoise

Exception(s):

- May raise CappyExc_MallocFailed.

```
void CappyNoiseFree(CappyNoise** const that);
```

Free the memory used by a CappyNoise* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CappyNoise to free

66 Unit pathfinder.h

PathFinder class.

66.1 Macros

```
#define CAPY_PATHFINDER_H
```

66.2 Enumerations

None.

66.3 Typedefs

None.

66.4 Struct CappyPath

66.4.1 Struct CappyPath's properties

```
size_t nbStep;
```

Number of steps in the path

```
size_t* points;
```


Indices of the points (array of nbStep+1 elements) forming the path from start to end

```
double sumWeight;
```

Sum of traversed link's weight

66.4.2 Struct CapyPath's methods

```
void (*destruct)(void);
```

Destructor

66.5 Struct CapyPathFinder

66.5.1 Struct CapyPathFinder's properties

None.

66.5.2 Struct CapyPathFinder's methods

```
void (*destruct)(void);
```

Destructor

```
CapyPath (*searchPointCloud)(  
    CapyPointCloud const* const pointCloud,  
    size_t const iPointFrom,  
    size_t const iPointTo);
```

Search the path in a CapyPointCloud using the A* algorithm Inputs: point-Cloud: the searched point cloud iPointFrom: the start point of the path iPointTo: the end point of the path

Output and side effect(s):

- Return a CapyPath, eventually with nbStep=0 if no path could be found.
- The returned path minimised the links' weight. Uses
- CapyPointCloud.estimateWeightPath() as a heuristic.

66.6 Functions

```
CapyPath CapyPathCreate(void);
```

Create a CapyPath

Output and side effect(s):

- Return a CapyPath

```
CapyPathFinder CapyPathFinderCreate(void);
```

Create a CapyPathFinder

Output and side effect(s):

- Return a CapyPathFinder

```
CapyPathFinder* CapyPathFinderAlloc(void);
```

Allocate memory for a new CapyPathFinder and create it

Output and side effect(s):

- Return a CapyPathFinder

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyPathFinderFree(CapyPathFinder** const that);
```

Free the memory used by a CapyPathFinder* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyPathFinder to free

67 Unit pen.h

Pen class to draw 2d graphics on an image.

67.1 Macros

```
#define CAPY_PEN_H
```

67.2 Enumerations

```
typedef enum CapyPenHardness {  
    capyPenHardness_9H = 1,  
    capyPenHardness_8H,  
    capyPenHardness_7H,  
    capyPenHardness_6H,  
    capyPenHardness_5H,  
    capyPenHardness_4H,  
    capyPenHardness_3H,  
    capyPenHardness_2H,  
    capyPenHardness_H,  
    capyPenHardness_F,  
    capyPenHardness_HB,  
    capyPenHardness_B,  
    capyPenHardness_2B,  
    capyPenHardness_3B,  
    capyPenHardness_4B,  
    capyPenHardness_5B,  
    capyPenHardness_6B,  
    capyPenHardness_7B,  
    capyPenHardness_8B,  
    capyPenHardness_9B,  
    capyPenHardness_none,  
} CapyPenHardness;
```

Pen hardness

67.3 Typedefs

None.

67.4 Struct CapyPen

67.4.1 Struct CapyPen's properties

```
CapyColorData color;
```

Color

```
double size;
```

Size (radius in pixel)


```
CapyPenHardness hardness;
```

Hardness

```
CapyPad(CapyPenHardness, hardness);
```

67.4.2 Struct CapyPen's methods

```
void (*destruct)(void);
```

Destructor

```
void (*drawPoint)(  
    double const* const pos,  
    CapyImg* const img);
```

Draw a point on a CapyImage.

Input argument(s):

- pos: the coordinate in pixel of the point
- img: the image on which to draw

```
void (*drawLine)(  
    double const* const from,  
    double const* const to,  
    CapyImg* const img);
```

Draw a line on a CapyImage.

Input argument(s):

- from: the coordinate in pixel of the start point
- to: the coordinate in pixel of the end point
- img: the image on which to draw

```
void (*drawBezier)(  
    CapyBezier* const curve,  
    CapyImg* const img);
```

Draw a CapyBezier curve on a CapyImage. The curve is expected to have one input (in $[0, 1]$) and return two outputs (in pixel coordinates)

Input argument(s):

- curve: the curve to draw
- img: the image on which to draw

```
void (*drawBezierSpline)(
    CappyBezierSpline* const spline,
    CappyImg* const img);
```

Draw a CappyBezierSpline curve on a CappyImage. The curve is expected to have one input (in [0, spline.nbSegment]) and return two outputs (in pixel coordinates)

Input argument(s):

- spline: the spline to draw
- img: the image on which to draw

```
void (*drawQuadrilateral)(
    CappyQuadrilateral const* const quad,
    CappyImg* const img);
```

Draw a quadrilateral on a CappyImage.

Input argument(s):

- quad: the quadrilateral to draw
- img: the image on which to draw

```
void (*drawRectangle)(
    CappyRectangle const* const rect,
    CappyImg* const img);
```

Draw a rectangle on a CappyImage.

Input argument(s):

- rect: the rectangle to draw
- img: the image on which to draw

```
void (*drawFilledRectangle)(
    CappyRectangle const* const rect,
    CappyImg* const img);
```

Draw a filled rectangle on a CappyImage.

Input argument(s):

- rect: the rectangle to draw
- img: the image on which to draw

```
void (*drawSegment)(
    CapySegment const* const seg,
    CapyImg* const img);
```

Draw a segment on a CapyImage.

Input argument(s):

- seg: the segment to draw
- img: the image on which to draw

```
void (*drawTriangle)(
    CapyTriangle const* const tri,
    CapyImg* const img);
```

Draw a triangle on a CapyImage.

Input argument(s):

- tri: the triangle to draw
- img: the image on which to draw

```
void (*drawCircle)(
    CapyCircle const* const circle,
    CapyImg* const img);
```

Draw a circle on a CapyImage.

Input argument(s):

- circle: the circle to draw
- img: the image on which to draw

```
void (*drawFilledCircle)(
    CapyCircle const* const circle,
    CapyImg* const img);
```

Draw a filled circle on a CapyImage.

Input argument(s):

- circle: the circle to draw

- img: the image on which to draw

```
void (*drawGeometry2D)(
    CappyGeometry2D const* const geometry,
    CappyImg* const img);
```

Draw a Geometry2D on a CappyImage.

Input argument(s):

- geometry: the geometry to draw
- img: the image on which to draw

```
void (*drawText)(
    double const* const pos,
    char const* const text,
    CappyFont const* const font,
    CappyImg* const img);
```

Draw a text with a given font

Input argument(s):

- pos: the coordinates of the top-left corner of the block of text
- text: the text to draw
- font: the font of the text
- img: the image on which to draw

67.5 Functions

```
CappyPen CappyPenCreate(void);
```

Create a CappyPen

Output and side effect(s):

- Return a CappyPen with default color white, hardness HB and
- radius 1.5

```
CappyPen* CappyPenAlloc(void);
```

Allocate memory for a new CappyPen and create it

Output and side effect(s):

- Return a CappyPen with default color white, hardness HB and
- radius 1.5

Exception(s):

- May raise CappyExc_MallocFailed.

```
void CappyPenFree(CappyPen** const that);
```

Free the memory used by a CappyPen* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CappyPen to free

68 Unit plyFormat.h

Class to manipulate file in PLY format

68.1 Macros

```
#define CAPY_PLYFORMAT_H
```

68.2 Enumerations

None.

68.3 Typedefs

None.

68.4 Struct CappyPlyFormat

68.4.1 Struct CappyPlyFormat's properties

```
CappyFileFormatDef;
```

Parent class


```
bool isBinary;
```

Flag to memorise if the file is in binary format (else it's in ascii format).
Updated when loading, used when saving.

```
CapyPad(bool, 1);
```

68.4.2 Struct CapyPlyFormat's methods

```
void (*destructCapyFileFormat)(void);
```

Destructor

68.5 Functions

```
CapyPlyFormat CapyPlyFormatCreate(void);
```

Create a CapyPlyFormat
Output and side effect(s):

- Return a CapyPlyFormat

```
CapyPlyFormat* CapyPlyFormatAlloc(void);
```

Allocate memory for a new CapyPlyFormat and create it
Output and side effect(s):

- Return a CapyPlyFormat

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyPlyFormatFree(CapyPlyFormat** const that);
```

Free the memory used by a CapyPlyFormat* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CapyPlyFormat to free

69 Unit pngFormat.h

Class to manipulate file in PNG format

69.1 Macros

```
#define CAPY_PNGFORMAT_H
```

69.2 Enumerations

None.

69.3 Typedefs

None.

69.4 Struct CpyPngFormat

69.4.1 Struct CpyPngFormat's properties

```
CpyFileFormatDef;
```

Parent class

69.4.2 Struct CpyPngFormat's methods

```
void (*destructCpyFileFormat)(void);
```

Destructor

69.5 Functions

```
CpyPngFormat CpyPngFormatCreate(void);
```

Create a CpyPngFormat

Output and side effect(s):

- Return a CpyPngFormat


```
CapyPngFormat* CapyPngFormatAlloc(void);
```

Allocate memory for a new CapyPngFormat and create it
Output and side effect(s):

- Return a CapyPngFormat

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyPngFormatDestruct(CapyPngFormat* const that);
```

Free the memory used by a CapyPngFormat
Input argument(s):

- that: the CapyPngFormat to free

```
void CapyPngFormatFree(CapyPngFormat** const that);
```

Free the memory used by a CapyPngFormat* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CapyPngFormat to free

70 Unit pointCloud.h

Point cloud class.

70.1 Macros

```
#define CAPY_POINT_CLOUD_H
```

70.2 Enumerations

None.

70.3 Typedefs

None.

70.4 Struct CpyPointCloudLink

70.4.1 Struct CpyPointCloudLink's properties

```
size_t iPoints[2];
```

Indices of the linked points

```
double weight;
```

Weight of the link

70.4.2 Struct CpyPointCloudLink's methods

None.

70.5 Struct CpyPointCloud

70.5.1 Struct CpyPointCloud's properties

```
size_t dim;
```

Dimension of a point in the cloud.

```
size_t size;
```

Number of points in the cloud.

```
CpyVec* points;
```

Points

```
size_t nbLink;
```

Number of links in the cloud.

```
CpyPointCloudLink* links;
```

Arrays of links between points

```
CpyVec mean;
```

Mean vector


```
CapyVec stdDev;
```

Standard deviation vector

```
CapyMat covariance;
```

Covariance matrix

```
CapyMat pearsonCorrelation;
```

Pearson correlation matrix

```
CapyMat principalComponent;
```

Principal components (one component per column, ordered from the most significant)

```
CapyVec eigenValue;
```

Eigen values of the principal components

```
CapyRangeDouble* range;
```

Range of values in each dimension

70.5.2 Struct CapyPointCloud's methods

```
void (*destruct)(void);
```

Destructor

```
CapyBezier* (*getApproxBezier)(  
    size_t const nbIn,  
    CapyBezierOrder_t const order);
```

Get the Bezier approximating the point cloud.

Input argument(s):

- nbIn: the number of dimension from the first dimension of a point
- treated as inputs of the Bezier (the remaining ones being treated
- as the output).
- order: the Bezier order.

Output and side effect(s):

- Return a new CapiBezier which takes the first 'nbIn' values of a point
- as input and return as output the approximation of the remaining values
- of that point. Return NULL if the Bezier couldn't be created.

Exception(s):

- May raise CapiExc_MallocFailed.

```
void (*updateMean)(void);
```

Update the mean vector

Output and side effect(s):

- that->mean is updated.

```
void (*updateStdDev)(void);
```

Update the stdDev vector

Output and side effect(s):

- that->mean and that->stdDev is updated.

```
void (*updateCovariance)(void);
```

Update the covariance matrix

Output and side effect(s):

- that->covariance and that->mean are updated.

```
void (*updatePearsonCorrelation)(void);
```

Update the pearson correlation matrix

Output and side effect(s):

- that->pearsonCorrelation, that->stdDev, that->covariance and

- that->mean are updated.

```
void (*updatePrincipalComponent)(void);
```

Update the principal components

Output and side effect(s):

- that->principalComponent, that eigenValue, that->covariance and
- that->mean are updated.

```
void (*updateRange)(void);
```

Update the ranges

Output and side effect(s):

- that->range is updated.

```
double (*estimateWeightPath)(
    size_t const iNodeFrom,
    size_t const iNodeTo);
```

Get an estimate of the minimum sum of weights along paths linking two points (cf CapyPathFinder) Inputs: iPointFrom: the start node of the path
iPointTo: the end node of the path

Output and side effect(s):

- Return the estimate, which must be less or equal to the actual minimum
- sum of weights.

70.6 Struct CapyPointCloudNearestNeighbourRes

70.6.1 Struct CapyPointCloudNearestNeighbourRes's properties

```
size_t iPoint;
```

Index of the point in the point cloud

```
double dist;
```

Distance to the query

```
size_t nbTestedPoint;
```

Counter for the number of point tested (for performance evaluation)

70.6.2 Struct CapyPointCloudNearestNeighbourRes's methods

```
void (*destruct)(void);
```

Destructor

70.7 Struct CapyPointCloudNearestNeighbour

70.7.1 Struct CapyPointCloudNearestNeighbour's properties

```
CapyPointCloud const* pointCloud;
```

Reference to the point cloud

```
BTreePointCloudNearestNeighbourRes* btree;
```

BTree representing the point cloud

```
CapyVec origin;
```

Reference point for distance calculation

70.7.2 Struct CapyPointCloudNearestNeighbour's methods

```
void (*destruct)(void);
```

Destructor

```
CapyPointCloudNearestNeighbourRes (*query)(  
    CapyVec const* const query,  
    size_t const iInitPoint);
```

Query the nearest point in the point cloud

Input argument(s):

- query: the query point
- iInitPoint: index of the point in the point cloud used for
- initialisation

Output and side effect(s):

- Return the index of the nearest point in the point cloud and its
- distance to the query.

70.8 Functions

```
CapyPointCloud CapyPointCloudCreate(size_t const dim);
```

Create a CapyPointCloud.

Input argument(s):

- dim: the dimension of a point in the cloud

Output and side effect(s):

- Return a CapyPointCloud

Exception(s):

- May raise CapyExc_MallocFailed.

```
CapyPointCloud* CapyPointCloudAlloc(size_t const dim);
```

Allocate memory for a new CapyPointCloud and create it

Input argument(s):

- dim: the dimension of a point in the cloud

Output and side effect(s):

- Return a CapyPointCloud

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyPointCloudFree(CapyPointCloud** const that);
```

Free the memory used by a CapyPointCloud* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyPointCloud to free

```
CapyPointCloudNearestNeighbour CapyPointCloudNearestNeighbourCreate(  
    CapyPointCloud const* const pointCloud,  
        size_t const nbMaxElem,  
    CapyVec const* const origin);
```


Create a `CapyPointCloudCloudNearestNeighbour`.

Input argument(s):

- `pointCloud`: the point cloud
- `nbMaxElem`: max number of elements in the underlying BTree
- `origin`: reference point for distance calculation

Output and side effect(s):

- Return a `CapyPointCloudCloudNearestNeighbour`

Exception(s):

- May raise `CapyExc_MallocFailed`.

```
CapyPointCloudNearestNeighbour* CapyPointCloudNearestNeighbourAlloc(  
    CapyPointCloud const* const pointCloud,  
        size_t const nbMaxElem,  
    CapyVec const* const origin);
```

Allocate memory for a new `CapyPointCloudCloudNearestNeighbour` and create it

Input argument(s):

- `pointCloud`: the point cloud
- `nbMaxElem`: max number of elements in the underlying BTree
- `origin`: reference point for distance calculation

Output and side effect(s):

- Return a `CapyPointCloudCloudNearestNeighbour`

Exception(s):

- May raise `CapyExc_MallocFailed`.

```
void CapyPointCloudNearestNeighbourFree(  
    CapyPointCloudNearestNeighbour** const that);
```

Free the memory used by a `CapyPointCloudNearestNeighbour*` and reset `*that` to NULL

Input argument(s):

- `that`: a pointer to the `CapyPointCloudNearestNeighbour` to free

71 Unit poissonSampling.h

Poisson disk sampling class.

71.1 Macros

```
#define CAPY_POISSON_DISK_SAMPLING_H
```

71.2 Enumerations

None.

71.3 Typedefs

None.

71.4 Struct CapyPoissonSampling

71.4.1 Struct CapyPoissonSampling's properties

```
CapySamplingDef;
```

Inherits CapySampling

```
double r;
```

Minimum distance between samples (default: 1.0)

```
size_t k;
```

Threshold before rejection (default: 30)

71.4.2 Struct CapyPoissonSampling's methods

```
void (*destructCapySampling)(void);
```

Destructor

71.5 Functions

```
CapyPoissonSampling CapyPoissonSamplingCreate(  
    CapyRandomSeed_t const seed,  
    size_t const dim);
```

Create a CapyPoissonSampling

Input argument(s):

- seed: seed for the random number generator
- dim: dimension of the samples

Output and side effect(s):

- Return a CapyPoissonSampling

```
CapyPoissonSampling* CapyPoissonSamplingAlloc(  
    CapyRandomSeed_t const seed,  
    size_t const dim);
```

Allocate memory for a new CapyPoissonSampling and create it

Input argument(s):

- seed: seed for the random number generator
- dim: dimension of the samples

Output and side effect(s):

- Return a CapyPoissonSampling

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyPoissonSamplingFree(CapyPoissonSampling** const that);
```

Free the memory used by a CapyPoissonSampling* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyPoissonSampling to free

72 Unit policygradient.h

PolicyGradient class.

72.1 Macros

```
#define CAPY_POLICYGRADIENT_H
```

```
#define CapyPEnvironmentDef { \
    CapyRandom rng; \
    size_t nbAction; \
    CapyVec paramAction; \
    CapyVec paramValue; \
    CapyVec curState; \
    CapyVec actionsProb; \
    void (*destruct)(void); \
    void (*setToInitialState)(void); \
    CapyPGTransition (*step)(size_t const action); \
    bool (*isEndState)(void); \
    size_t (*getAction)(CapyVec const* const state); \
    size_t (*getBestAction)(CapyVec const* const state); \
    void (*getActionsProb)( \
        CapyVec const* const state, \
        CapyVec* const actionsProb); \
    double (*getValue)(CapyVec const* const state); \
    void (*getGradientValue)( \
        CapyVec const* const state, \
        CapyVec* const gradValue); \
    void (*getGradientActionsProb)( \
        CapyVec const* const state, \
        size_t const iAction, \
        CapyVec* const gradProb); \
    void (*getGradientActionsLogProb)( \
        CapyVec const* const state, \
        size_t const iAction, \
        CapyVec* const gradProb); \
}
```

PolicyGradient environment definition

Random number generator CapyRandom rng;
Number of possible action size_t nbAction;
Parameters for actions probability evaluation CapyVec paramAction;
Parameters for value evaluation CapyVec paramValue;
Current state features value CapyVec curState;
Output vector for the actions probability evaluation CapyVec actionsProb;
Destructor void (*destruct)(void);
Step the environment Input: action: the applied action
Output and side effect(s):

- Update the current state according to the action, and return the transition
- `CapyPGTransition (*step)(size_t const action);`
- Set the current state to an initial state

Output and side effect(s):

- The current state is set to an initial state
- `void (*setToInitialState)(void);`
- Check if the current state is an end state

Output and side effect(s):

- Return true if the current state is an end state, else false
- `bool (*isEndState)(void);`
- Get an action for a given state according to their probabilities

Input argument(s):

- state: the state to use for evaluation

Output and side effect(s):

- Return the selected action.
- `size_t (*getAction)(CapyVec const* const state);`
- Get the action with highest probability for a given state

Input argument(s):

- state: the state to use for evaluation

Output and side effect(s):

- Return the selected action.
- `size_t (*getBestAction)(CapyVec const* const state);`
- Evaluate the action probabilities

Input argument(s):

- state: the state used for evaluation
- actionsProb: the evaluated actions probability

Output and side effect(s):

- 'actionsProb' is updated.
- void (*getActionsProb)(
- CopyVec const* const state,
- CopyVec* const actionsProb);
- Evaluate the value

Input argument(s):

- state: the state used for evaluation

Output and side effect(s):

- Return the evaluated value
- double (*getValue)(CopyVec const* const state);
- Evaluate the gradient of values

Input argument(s):

- state: the state used for evaluation
- gradValue: the result gradient

Output and side effect(s):

- 'gradValue' is updated.
- void (*getGradientValue)(
- CopyVec const* const state,
- CopyVec* const gradValue);
- Evaluate the gradient of actions probability

Input argument(s):

- state: the state used for evaluation
- iAction: the action to be evaluated
- gradProb: the result gradient

Output and side effect(s):

- 'gradProb' is updated.
- void (*getGradientActionsProb)(
- CpyVec const* const state,
- size_t const iAction,
- CpyVec* const gradProb);
- Evaluate the gradient of actions log probability

Input argument(s):

- state: the state used for evaluation
- iAction: the action to be evaluated
- gradProb: the result gradient

Output and side effect(s):

- 'gradProb' is updated.
- void (*getGradientActionsLogProb)(
- CpyVec const* const state,
- size_t const iAction,
- CpyVec* const gradProb);

72.2 Enumerations

None.

72.3 Typedefs

```
typedef struct CopyPGEnvironment CopyPGEnvironmentDef CopyPGEnvironment;
```

CopyPGEnvironment object

72.4 Struct CopyPGTransition

72.4.1 Struct CopyPGTransition's properties

```
CopyVec fromState;
```

'from' state

```
size_t action;
```

Action

```
CopyVec toState;
```

'to' state

```
double reward;
```

Reward

72.4.2 Struct CopyPGTransition's methods

None.

72.5 Struct CopyPGTransitionRecorder

72.5.1 Struct CopyPGTransitionRecorder's properties

```
size_t nbTransition;
```

Number of transition

```
size_t nbMaxTransition;
```

Size of the recorder memory (in transition number)

```
CopyPGTransition* transitions;
```

Recorded transitions

72.5.2 Struct CpyPGTransitionRecorder's methods

```
void (*destruct)(void);
```

Destructor

```
void (*reset)(void);
```

Reset the recorder

Output and side effect(s):

- 'nbTransition' is reset to 0.

```
void (*addTransition)(CpyPGTransition const* const transition);
```

Record one transition

Input argument(s):

- transition: the transition to be recorded

Output and side effect(s):

- A copy of the transition is added to the end of 'transitions' which
- is reallocated if necessary, 'nbTransition' and 'nbMaxTransition' are
- updated as necessary.

72.6 Struct CpyPolicyGradient

72.6.1 Struct CpyPolicyGradient's properties

```
CpyPGEnvironment* env;
```

The trained environment

```
double learnRateAction;
```

Learning rate for action probabilities (in]0,1], should be small, default: 0.01)

```
double learnRateState;
```


Learning rate for state value (in $]0,1]$, should be small, default: 0.01)

```
double discount;
```

Discount rate (in $]0,1]$, default: 0.9)

```
size_t nbMaxStep;
```

Max number of step when sampling trajectory (default: 1000)

```
double avgReward;
```

Average reward during training

```
double avgFinalReward;
```

Average final reward during training

```
double avgNbStep;
```

Average number of step per episode during training

```
double coeffClipping;
```

Clipping coefficient for PPO (in $]0,+\infty[$, default: 0.2, the lower the more stable but the slower learning)

```
CapyGradientDescent* gdAction;
```

Gradient descent for the action probabilities (adam)

```
CapyGradientDescent* gdValue;
```

Gradient descent for the state values (standard)

72.6.2 Struct CapyPolicyGradient's methods

```
void (*destruct)(void);
```

Destructor

```
void (*reinforce)(size_t const nbEpisode);
```


Learn the weights of action probabilities and state value functions using the reinforce with baseline algorithm Inputs: nbEpisode: number of training episode

Output and side effect(s):

- The environment's action probabilities parameters and state values
- parameters are updated.

```
void (*proximalPolicyOptimisation)(size_t const nbEpisode);
```

Learn the weights of action probabilities and state value functions using the proximal policy optimisation algorithm Inputs: nbEpisode: number of training episode

Output and side effect(s):

- The environment's action probabilities parameters and state values
- parameters are updated.

72.7 Functions

```
CapyPGTransition CapyPGTransitionCreate(size_t const nbFeature);
```

Create a CapyPGTransition

Input argument(s):

- nbFeature: number of features describing an environment

Output and side effect(s):

- Return a CapyPGTransition

```
void CapyPGTransitionDestruct(CapyPGTransition* const that);
```

Destruct a CapyPGTransition

```
CapyPGEnvironment CapyPGEnvironmentCreate(  
    size_t const nbFeature,  
    size_t const nbAction,  
    size_t const nbParamAction,  
    size_t const nbParamValue,  
    CapyRandomSeed_t const seed);
```


Create a CpyPGEEnvironment

Input argument(s):

- nbFeature: number of features describing an environment
- nbAction: number of possible actions
- nbParamAction: number of parameters for actions probability evaluation
- nbParamValue: number of parameters for value evaluation
- seed: seed for the random number generator

Output and side effect(s):

- Return a CpyPGEEnvironment

```
CpyPGEEnvironment* CpyPGEEnvironmentAlloc(  
    size_t const nbFeature,  
    size_t const nbAction,  
    size_t const nbParamAction,  
    size_t const nbParamValue,  
    CpyRandomSeed_t const seed);
```

Allocate memory for a new CpyPGEEnvironment and create it

Input argument(s):

- nbFeature: number of features describing an environment
- nbAction: number of possible actions
- nbParamAction: number of parameters for actions probability evaluation
- nbParamValue: number of parameters for value evaluation
- seed: seed for the random number generator

Output and side effect(s):

- Return a CpyPGEEnvironment

Exception(s):

- May raise CpyExc_MallocFailed.


```
void CapyPGEnvironmentFree(CapyPGEnvironment** const that);
```

Free the memory used by a CapyPGEnvironment* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CapyPGEnvironment to free

```
CapyPGTransitionRecorder CapyPGTransitionRecorderCreate(void);
```

Create a CapyPGTransitionRecorder
Output and side effect(s):

- Return a CapyPGTransitionRecorder

```
CapyPolicyGradient CapyPolicyGradientCreate(CapyPGEnvironment* const env);
```

Create a CapyPolicyGradient Inputs: env: the environment to train
Output and side effect(s):

- Return a CapyPolicyGradient

```
CapyPolicyGradient* CapyPolicyGradientAlloc(CapyPGEnvironment* const env);
```

Allocate memory for a new CapyPolicyGradient and create it Inputs: env:
the environment to train
Output and side effect(s):

- Return a CapyPolicyGradient

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyPolicyGradientFree(CapyPolicyGradient** const that);
```

Free the memory used by a CapyPolicyGradient* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CapyPolicyGradient to free

73 Unit polynomial.h

Polynomial class.

73.1 Macros

```
#define CAPY_POLYNOMIAL_H
```

73.2 Enumerations

None.

73.3 Typedefs

None.

73.4 Struct CapyPolynomial1D

73.4.1 Struct CapyPolynomial1D's properties

```
struct CapyMathFunDef;
```

Inherits CapyMathFun

```
CapyVec coeffs;
```

Coefficients of the polynomial in order x^0 , x^1 , x^2 , ...

73.4.2 Struct CapyPolynomial1D's methods

```
void (*destructCapyMathFun)(void);
```

Destructor

73.5 Functions

```
CapyPolynomial1D CapyPolynomial1DCreate(CapyVec const* const coeffs);
```


Create a CpyPolynomial1D

Input argument(s):

- coeffs: the coefficients of the polynomial in order x^0 , x^1 , x^2 , ...

Output and side effect(s):

- Return a CpyPolynomial1D

```
CpyPolynomial1D* CpyPolynomial1DAlloc(CpyVec const* const coeffs);
```

Allocate memory for a new CpyPolynomial1D and create it

Input argument(s):

- coeffs: the coefficients of the polynomial in order x^0 , x^1 , x^2 , ...

Output and side effect(s):

- Return a CpyPolynomial1D

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyPolynomial1DFree(CpyPolynomial1D** const that);
```

Free the memory used by a CpyPolynomial1D* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyPolynomial1D to free

74 Unit predictor.h

Parent class and structures for the predictor classes.

74.1 Macros

```
#define CAPY_PREDICTOR_H
```

```
#define CapyPredictorEvaluationDef \
struct { \
    double accuracies[CapyPredictorAccuracyMeasure_nb]; \
    size_t* confusionMatrix; \
    void (*destruct)(void); \
    double (*getAccuracy)(CapyPredictorAccuracyMeasure const type); \
}
```

PredictorEvaluation, class to memorise the result of evaluation of a predictor

Overall accuracy For categorical predictor it is the percentage of correct answers (in $[0.0, 1.0]$). For numerical predictor it is the MAE (mean absolute error) double accuracy;

Confusion matrix, such as `confusionMatrix[i * n + j]` is the number of times the i-th category has been predicted as the j-th category, where n is the number of possible categories `size_t* confusionMatrix`;

Return the evaluated accuracy of the predictor

Input argument(s):

- type: type of accuracy measure

Output and side effect(s):

- Return the accuracy.
- double `(*getAccuracy)(CapyPredictorAccuracyMeasure const type);`

```
#define CapyPredictorDef \
struct { \
    size_t iOutput; \
    CapyPredictorType type; \
    CapyPad(CapyPredictorType, type); \
    CapyPredictorFeatureScaling featureScaling; \
    CapyPad(CapyPredictorFeatureScaling, featureScaling); \
    size_t nbInput; \
    CapyRangeDouble* scalingFrom; \
    CapyRangeDouble scalingTo; \
    void (*destruct)(void); \
    void (*train)(CapyDataset const* const dataset); \
    CapyPredictorPrediction (*predict)(CapyVec const* const inp); \
    CapyPredictorEvaluation* (*evaluate)(CapyDataset const* const dataset); \
    CapyMat (*cvtDatasetToMat)(CapyDataset const* const dataset); \
    void* (*clone)(void); \
    void (*exportToCFun)( \
        FILE* const stream, \
        char const* const name, \

```



```

    CapyDataset const* const dataset);
void (*exportToHtml)(
    FILE* const stream,
    char const* const title,
    CapyDataset const* const dataset,
    double const expectedAccuracy);
void (*scaleTrainingInputFeatures)(
    CapyMat* const mat,
    CapyDataset const* const dataset);
void (*scaleInputFeatures)(CapyVec* const inp);
void (*exportScaleInputToCFun)(FILE* const stream);
void (*save)(FILE* const stream);
}

```

Predictor, parent class for all predictors

Index of the predicted output in the dataset (default: 0) size_t iOutput;

Type of predictor (default: categorical) CapyPredictorType type;

Type of feature scaling (default: none) CapyPredictorFeatureScaling featureScaling

Number of inputs size_t nbInput;

Train the predictor on a dataset

Input argument(s):

- dataset: the dataset

Output and side effect(s):

- The predictor is trained.

Exception(s):

- May raise CapyExc_UnsupportedFormat
- void (*train)(CapyDataset const* const dataset);
- Predict an input

Input argument(s):

- inp: the input

Output and side effect(s):

- Return the prediction
- CapyPredictorPrediction (*predict)(CapyVec const* const inp);
- Evaluate the predictor on a dataset

Input argument(s):

- dataset: the dataset

Output and side effect(s):

- Return the evaluation of the predictor.
- CopyPredictorEvaluation* (*evaluate)(CopyDataset const* const dataset);
- Convert a CopyDataset into a CopyMat usable by the predictor

Input argument(s):

- dataset: the dataset to be converted

Output and side effect(s):

- Return a matrix formatted as necessary
- CopyMat (*cvtDatasetToMat)(
- CopyDataset const* const dataset);
- Clone a predictor

Output and side effect(s):

- Return a clone of the predictor.
- void* (*clone)(void);
- Export the predictor as a C function

Input argument(s):

- stream: the stream where to export
- name: the name of the function
- dataset: the training dataset

Output and side effect(s):

- A ready to use C function implementing the predictor is written on the
- stream. See the comment exported with the function to know how to use

- the exported function.
- `void (*exportToCFun)(`
- `FILE* const stream,`
- `char const* const name,`
- `CapyDataset const* const dataset);`
- Export the predictor as a HTML web app

Input argument(s):

- stream: the stream where to export
- title: the title of the web app
- dataset: the training dataset
- expectedAccuracy: the expected accuracy of the predictor (in $[0,1]$)

Output and side effect(s):

- A ready to use web app implementing the predictor is written on the
- stream.
- `void (*exportToHtml)(`
- `FILE* const stream,`
- `char const* const title,`
- `CapyDataset const* const dataset,`
- `double const expectedAccuracy);`
- Preprocess the input features in the training data

Input argument(s):

- mat: the training data
- dataset: the training dataset

Output and side effect(s):

- 'mat' is updated
- void (*scaleTrainingInputFeatures)(
- CpyMat* const mat,
- CpyDataset const* const dataset);
- Preprocess the input features in the input vector

Input argument(s):

- inp: the input vector

Output and side effect(s):

- 'inp' is updated
- void (*scaleInputFeatures)(CpyVec* const inp);
- Export the input feature scaling as C code
- Input:
- stream: the stream on which the code is exported
- Output:
- The scaling code is written to the stream (to be used by the
- exportToCFun method.
- void (*exportScaleInputToCFun)(FILE* const stream);
- Save the predictor to a stream
- Input:
- stream: the stream on which to save
- Output:
- The predictor data are saved on the stream
- void (*save)(FILE* const stream);

74.2 Enumerations

```
typedef enum CapyPredictorType {  
    capyPredictorType_categorical,  
    capyPredictorType_numerical,  
    capyPredictorType_nb,  
} CapyPredictorType;
```

Type of predictor

```
typedef enum CapyPredictorAccuracyMeasure {  
    capyPredictorAccuracyMeasure_mae,  
    capyPredictorAccuracyMeasure_rmse,  
    capyPredictorAccuracyMeasure_rSquared,  
    capyPredictorAccuracyMeasure_accuracy,  
    capyPredictorAccuracyMeasure_nb,  
} CapyPredictorAccuracyMeasure;
```

Accuracy measure types for numerical predictor MAE (mean absolute error), RMSE (root mean square error) and R^2 (coefficient of determination) are calculated for numerical predictor accuracy is calculated for categorical predictors

```
typedef enum CapyPredictorFeatureScaling {  
    capyPredictorFeatureScaling_none,  
    capyPredictorFeatureScaling_minMaxNormalization,  
    capyPredictorFeatureScaling_minMaxNormalizationSym,  
    capyPredictorFeatureScaling_meanNormalization,  
    capyPredictorFeatureScaling_standardization,  
    capyPredictorFeatureScaling_nb,  
} CapyPredictorFeatureScaling;
```

Types of input features scaling minMaxNormalization: $[\min, \max] \rightarrow [0, 1]$ minMaxNormalizationSym: $[\min, \max] \rightarrow [-1, 1]$ meanNormalization: $(x - \text{avg}) / (\max - \min)$ standardization: $(x - \text{avg}) / \text{sigma}$

74.3 Typedefs

```
typedef CapyPredictorEvaluationDef CapyPredictorEvaluation;
```

CapyPredictorEvaluation declaration

```
typedef CapyPredictorDef CapyPredictor;
```

Predictor structure

74.4 Struct CpyPredictorPrediction

74.4.1 Struct CpyPredictorPrediction's properties

```
union {
```

Predicted category/value

```
size_t category;
```

```
double val;
```

```
};
```

```
double confidence;
```

Confidence of the prediction (in $[0, +\infty[$, the higher the more confident)

74.4.2 Struct CpyPredictorPrediction's methods

None.

74.5 Functions

```
CpyPredictorEvaluation CpyPredictorEvaluationCreate(void);
```

Create a CpyPredictorEvaluation

Output and side effect(s):

- Return a CpyPredictorEvaluation

```
CpyPredictorEvaluation* CpyPredictorEvaluationAlloc(void);
```

Allocate memory for a new CpyPredictorEvaluation and create it

Output and side effect(s):

- Return a CpyPredictorEvaluation

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyPredictorEvaluationFree(CpyPredictorEvaluation** const that);
```

Free the memory used by a CpyPredictorEvaluation* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyPredictorEvaluation to free

```
CpyPredictor CpyPredictorCreate(CpyPredictorType const type);
```

Create a CpyPredictor

Input argument(s):

- type: type of predictor

Output and side effect(s):

- Return a CpyPredictor

```
CpyPredictor* CpyPredictorAlloc(CpyPredictorType const type);
```

Allocate memory for a new CpyPredictor and create it

Input argument(s):

- type: type of predictor

Output and side effect(s):

- Return a CpyPredictor

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyPredictorFree(CpyPredictor** const that);
```

Free the memory used by a CpyPredictor* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyPredictor to free

75 Unit quaternion.h

Quaternion class.

75.1 Macros

```
#define CAPY_QUATERNION_H
```

75.2 Enumerations

None.

75.3 Typedefs

```
typedef struct CapyQuaternion CapyQuaternion;
```

Quaternion object

75.4 Functions

```
CapyQuaternion CapyQuaternionCreate(void);
```

Create a CapyQuaternion

Output and side effect(s):

- Return a CapyQuaternion with default null rotation

```
CapyQuaternion* CapyQuaternionAlloc(void);
```

Allocate memory for a new CapyQuaternion and create it

Output and side effect(s):

- Return a CapyQuaternion

Exception(s):

- May raise CapyExc_MallocFailed.


```
CapyQuaternion CapyQuaternionCreateFromRotMat(CapyMat const* const rotMat);
```

Create a new static quaternion from the rotation matrix 'rotMat'

```
CapyQuaternion* CapyQuaternionAllocFromRotMat(CapyMat const* const rotMat);
```

Allocate memory and create a new Quaternion from the rotation matrix 'rotMat'

```
CapyQuaternion CapyQuaternionCreateFromRotAxis(  
    double const* const axis,  
    double const theta);
```

Create a new static quaternion corresponding to the rotation around 'axis' (must be normalized) by 'theta' (in radians)

```
CapyQuaternion* CapyQuaternionAllocFromRotAxis(  
    double const* const axis,  
    double const theta);
```

Allocate memory and create a new Quaternion corresponding to the rotation around 'axis' (must be normalized) by 'theta' (in radians)

```
CapyQuaternion CapyQuaternionCreateRotFromVecToVec(  
    double const* const from,  
    double const* const to);
```

Create a new static quaternion corresponding to the rotation bringing the vector 'from' to the vector 'to'

```
CapyQuaternion* CapyQuaternionAllocRotFromVecToVec(  
    double const* const from,  
    double const* const to);
```

Allocate memory and create a new quaternion corresponding to the rotation bringing the vector 'from' to the vector 'to'

```
void CapyQuaternionFree(CapyQuaternion** const that);
```

Free the memory used by a CapyQuaternion* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CapyQuaternion to free

76 Unit random.h

Pseudo random generation class.

76.1 Macros

```
#define CAPY_RANDOM_H
```

76.2 Enumerations

```
typedef enum CapyRandomPrng {  
    capyRandomPrng_pcg_xsh_rr,  
    capyRandomPrng_xorshift_star,  
} CapyRandomPrng;
```

Type of PRNG algorithms supported by CapyRandom

76.3 Typedefs

```
typedef uint64_t CapyRandomSeed_t;
```

Type for the seed of a CapyRandom

```
typedef uint64_t CapyRandomState_t;
```

Type for the state of a CapyRandom

76.4 Struct CapyRandom

76.4.1 Struct CapyRandom's properties

```
CapyRandomPrng type;
```

Type of algorithm (default: capyRandomPrng_pcg_xsh_rr)

```
CapyPad(CapyRandomPrng, type);
```

```
CapyRandomSeed_t seed;
```

Seed

```
CapyRandomState_t state;
```

State


```
CapyRandomState_t bits;
```

Random bits

```
CapyRandomState_t buffer;
```

Buffer for optimisation

```
uint8_t nbRemainingBits;
```

```
CapyPad(uint8_t, nbRemainingBits);
```

76.4.2 Struct CapyRandom's methods

```
void (*destruct)(void);
```

Destructor

```
void (*step)(void);
```

Step the state

```
void (*setPrng)(CapyRandomPrng const type);
```

Set the type of PRNG

Input argument(s):

- type: type of PRNG

Output and side effect(s):

- The type is updated and the CapyRandom is re-initialised.

```
bool (*getBool)(void);
```

Return the next pseudo random number as a boolean

Output and side effect(s):

- Return the random number as a boolean


```
uint8_t (*getUInt8)(void);
```

Return the next pseudo random number as an integer using the xorshift algorithm

Output and side effect(s):

- Return the random number as an integer

```
uint16_t (*getUInt16)(void);
```

```
uint32_t (*getUInt32)(void);
```

```
uint64_t (*getUInt64)(void);
```

```
size_t (*getSize)(void);
```

```
double (*getDouble)(void);
```

Return the next pseudo random number as a double using the Downey algorithm

Output and side effect(s):

- Return the random number as a double in [0.0, 1.0]

```
double (*getDoubleFast)(void);
```

Return the next pseudo random number as a double using integer division

Output and side effect(s):

- Return the random number as a double in [0.0, 1.0]. It is around
- two times faster than the Downey's algorithm but generates random
- floating-point number less efficiently. See
- <https://baillehachepascal.dev/2021/random.float.php>

```
uint8_t (*getUInt8Range)(CapyRangeUInt8 const* const range);
```


Return the next pseudo random number as an integer mapped into a range

Input argument(s):

- range: the range to map the random number to

Output and side effect(s):

- Return the random number as an integer in the given range, bounds are
- inclusive

```
int8_t (*getInt8Range)(CapyRangeInt8 const* const range);
```

```
uint16_t (*getUInt16Range)(CapyRangeUInt16 const* const range);
```

```
int16_t (*getInt16Range)(CapyRangeInt16 const* const range);
```

```
uint32_t (*getUInt32Range)(CapyRangeUInt32 const* const range);
```

```
int32_t (*getInt32Range)(CapyRangeInt32 const* const range);
```

```
uint64_t (*getUInt64Range)(CapyRangeUInt64 const* const range);
```

```
int64_t (*getInt64Range)(CapyRangeInt64 const* const range);
```

```
size_t (*getSizeRange)(CapyRangeSize const* const range);
```

```
uint32_t (*getUInt32RangeLemire)(CapyRangeUInt32 const* const range);
```

Return the next pseudo random number as an integer mapped into a range using Lemire's algorithm

Input argument(s):

- range: the range to map the random number to

Output and side effect(s):

- Return the random number as an integer in the given range, bounds are
- inclusive

```
double (*getDoubleRange)(CapyRangeDouble const* const range);
```

Return the next pseudo random number as a double mapped into a range

Input argument(s):

- range: the range to map the random number to

Output and side effect(s):

- Return the random number as a double in the given range, bounds are
- inclusive

```
double (*getDoubleRangeFast)(CapyRangeDouble const* const range);
```

```
CapyRatio (*getRatio)(void);
```

Return the next pseudo random number as a CapyRatio

Output and side effect(s):

- Return a random CapyRatio in [0.0, 1.0]

```
uint32_t (*getSquirrel3)(uint32_t const position);
```

Get a pseudo-random uint32_t using the Squirrel3 algorithm.

Input argument(s):

- position: the position in the pseudo-random sequence

Output and side effect(s):

- Return the pseudo-random uint32_t at the given position in the
- sequence. Note that this sequence is different from the one
- generated by the other methods of CapyRandom. Note also that the
- sequence is completely determined by the seed of the CapyRandom.


```
CapyDistEvt (*getDistEvt)(CapyDist const* const dist);
```

Get a random event in a statistical distribution

Input argument(s):

- dist: the distribution

Output and side effect(s):

- Return a distribution event

```
size_t (*getIdxGivenProbVec)(CapyVec const* const prob);
```

Get a random index according to a vector of probability per index

Input argument(s):

- prob: the vector of probabilities

Output and side effect(s):

- Return the index

76.5 Functions

```
CapyRandom CapyRandomCreate(CapyRandomSeed_t const seed);
```

Create a CapyRandom

Input argument(s):

- seed: seed of the generator

Output and side effect(s):

- Return a CapyRandom

```
CapyRandom* CapyRandomAlloc(CapyRandomSeed_t const seed);
```

Allocate memory for a new CapyRandom and create it

Input argument(s):

- seed: seed of the generator

Output and side effect(s):

- Return a CopyRandom

Exception(s):

- May raise CopyExc_MallocFailed.

```
void CopyRandomFree(CopyRandom** const that);
```

Free the memory used by a CopyRandom* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CopyRandom to free

```
uint32_t CopyGetTrulyRandomValue(void);
```

Get a truly random value to seed a random generator

Output and side effect(s):

- Return the truly random value.

77 Unit range.h

Class to implement range of value.

77.1 Macros

```
#define CAPY_RANGE_H
```

```
#define CopyDecRange(name, type) \
typedef struct CopyRange ## name CopyRange ## name; \
struct CopyRange ## name { \
    union { \
        type vals[2]; \
        struct __attribute__((packed)) {type min, max;}; \
    }; \
    CopyPad(type[2], 0); \
    void (*destruct)(void); \
    type (*trim)(type const val); \
    void (*clip)(CopyRange ## name const* const range); \
}; \
CopyRange ## name CopyRange ## name ## Create( \
    type const min, \
    type const max); \
CopyRange ## name* CopyRange ## name ## Alloc(
```



```

    type const min,
    type const max);
void CopyRange ## name ## Destruct(void);
void CopyRange ## name ## Free(CopyRange ## name** const that);
type CopyRange ## name ## Trim(type const val);
void CopyRange ## name ## Clip(CopyRange ## name const* const range);

```

Range object. Declaration macro for a range of values of type 'type' and name 'name'

```

#define CopyDefRangeCreate(name, type)
CopyRange ## name CopyRange ## name ## Create(
    type const min,
    type const max) {
    CopyRange ## name that = {0};
    that.vals[0] = min;
    that.vals[1] = max;
    that.destruct = CopyRange ## name ## Destruct;
    that.trim = CopyRange ## name ## Trim;
    that.clip = CopyRange ## name ## Clip;
    return that;
}

```

Create a new CopyRange

Input argument(s):

- min: the lower value of the range
- max: the upper value of the range

Output and side effect(s):

- Return a new CopyRange

```

#define CopyDefRangeAlloc(name, type)
CopyRange ## name* CopyRange ## name ## Alloc(
    type const min,
    type const max) {
    CopyRange ## name* that = NULL;
    safeMalloc(that, 1);
    if(!that) return NULL;
    *that = CopyRange ## name ## Create(min, max);
    return that;
}

```

Create a newly allocated CopyRange

Input argument(s):

- min: the lower value of the range
- max: the upper value of the range

Output and side effect(s):

- Return a newly allocated CopyRange

Exception(s):

- May raise CopyExc_MallocFailed

```
#define CopyDefRangeDestruct(name, type) \
void CopyRange ## name ## Destruct(void) { \
    return; \
}
```

Free the memory used by a CopyRange

Input argument(s):

- that: the CopyRange to free

```
#define CopyDefRangeFree(name, type) \
void CopyRange ## name ## Free(CopyRange ## name** const that) { \
    if(that == NULL || *that == NULL) return; \
    $(*that, destruct)(); \
    free(*that); \
    *that = NULL; \
}
```

Free the memory used by a pointer to a CopyRange and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CopyRange to free

```
#define CopyDefRangeTrim(name, type) \
type CopyRange ## name ## Trim(type const val) { \
    CopyRange ## name* that = (CopyRange ## name*)copyThat; \
    return (val < that->vals[0] ? that->vals[0] : \
            val > that->vals[1] ? that->vals[1] : val); \
}
```

Trim a value to a CopyRange interval

Input argument(s):

- val: the value to trim

Output and side effect(s):

- Returned the value trimmed to the interval of the range


```

#define CopyDefRangeClip(name, type) \
void CopyRange ## name ## Clip(CopyRange ## name const* const range) { \
    CopyRange ## name* that = (CopyRange ## name*)copyThat; \
    if(range != NULL) { \
        if(that->min < range->min) that->min = range->min; \
        if(that->max > range->max) that->max = range->max; \
    } \
}

```

Clip the range to with another

Input argument(s):

- range: the other range

Output and side effect(s):

- The range is modified to the intersection of the range and the other
- range. max < min means the intersection is empty

```

#define CopyDefRange(name, type) \
    CopyDefRangeCreate(name, type) \
    CopyDefRangeAlloc(name, type) \
    CopyDefRangeDestruct(name, type) \
    CopyDefRangeFree(name, type) \
    CopyDefRangeTrim(name, type) \
    CopyDefRangeClip(name, type)

```

Definition macro calling all the submacros at once for a range object

77.2 Enumerations

None.

77.3 Typedefs

None.

77.4 Functions

None.

78 Unit ratio.h

Rational number class.

78.1 Macros

```
#define CAPY_RATIO_H
```

```
#define PRIratiofmt "%ld+%lu/%lu"
```

Format for print instructions

```
#define PRIratioval(a) (a).base, (a).num, (a).den
```

78.2 Enumerations

None.

78.3 Typedefs

```
typedef int64_t CopyRatioBase_t;
```

Type for the base of a CopyRatio

```
typedef uint64_t CopyRatioFrac_t;
```

Type for the fractional parts of a CopyRatio

78.4 Struct CopyRatio

78.4.1 Struct CopyRatio's properties

```
CopyRatioBase_t base;
```

Components of the ratio

```
CopyRatioFrac_t num;
```

```
CopyRatioFrac_t den;
```

78.4.2 Struct CopyRatio's methods

None.

78.5 Functions

```
CapyRatio CapyRatioCreate(  
    CapyRatioBase_t const base,  
    CapyRatioFrac_t const num,  
    CapyRatioFrac_t const den);
```

Create a CapyRatio equal to $\text{base} + \text{num}/\text{den}$

Input argument(s):

- base: the base
- num: the numerator
- den: the denominator

Output and side effect(s):

- Return a CapyRatio

```
CapyRatio* CapyRatioAlloc(  
    CapyRatioBase_t const base,  
    CapyRatioFrac_t const num,  
    CapyRatioFrac_t const den);
```

Allocate memory for a new CapyRatio and create it

Input argument(s):

- base: the base
- num: the numerator
- den: the denominator

Output and side effect(s):

- Return a CapyRatio

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyRatioDestruct(CapyRatio* const that);
```

Free the memory used by a CapyRatio

Input argument(s):

- that: the CpyRatio to free

```
void CpyRatioFree(CpyRatio** const that);
```

Free the memory used by a CpyRatio* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyRatio to free

```
bool CpyRatioIsNaN(CpyRatio const r);
```

Check if a rational is equal to NaN

Input argument(s):

- r: the CpyRatio to check

Output and side effect(s):

- Return true if the rational is equal to NaN (denominator equals to 0)

```
CpyRatio CpyRatioFromDouble(double const a);
```

Create a CpyRatio from a double

Input argument(s):

- a: the double

Output and side effect(s):

- Return a new CpyRatio representing the nearest possible value to the input double.

```
double CpyRatioToDouble(CpyRatio const that);
```

Convert a CpyRatio to a double

Input argument(s):

- that: the CpyRatio to convert

Output and side effect(s):

- Return a double representing approximating the CpyRatio.

```
CpyRatio CpyRatioReduce(CpyRatio const that);
```

Reduce a CpyRatio

Input argument(s):

- that: the CpyRatio to reduce

Output and side effect(s):

- Return a new CpyRatio equals to the reduced CpyRatio

Exception(s):

- May raise CpyExc_NumericalOverflow

```
CpyRatio CpyRatioAdd(  
    CpyRatio const x,  
    CpyRatio const y);
```

Add two CpyRatios

Input argument(s):

- x: the first CpyRatio (must be in reduced form)
- y: the second CpyRatio (must be in reduced form)

Output and side effect(s):

- Return a new CpyRatio (in reduced form) equal to x+y

Exception(s):

- May raise CpyExc_NumericalOverflow

```
CpyRatio CpyRatioNeg(CpyRatio const x);
```

Get the negative of a CpyRatio

Input argument(s):

- x: the CpyRatio (must be in reduced form)

Output and side effect(s):

- Return a new `CapyRatio` (in reduced form) equal to $-x$

Exception(s):

- May raise `CapyExc_NumericalOverflow`

```
CapyRatio CapyRatioSub(
    CapyRatio const x,
    CapyRatio const y);
```

Subtract two `CapyRatios`

Input argument(s):

- `x`: the first `CapyRatio` (must be in reduced form)
- `y`: the second `CapyRatio` (must be in reduced form)

Output and side effect(s):

- Return a new `CapyRatio` (in reduced form) equal to $x-y$

Exception(s):

- May raise `CapyExc_NumericalOverflow`

```
int8_t CapyRatioCmp(
    CapyRatio const x,
    CapyRatio const y);
```

Compare two `CapyRatios`

Input argument(s):

- `x`: the first `CapyRatio` (must be in reduced form)
- `y`: the second `CapyRatio` (must be in reduced form)

Output and side effect(s):

- Return -1 if $x < y$, else 0 if $x == y$, else 1 if $x > y$

```
CapyRatio CapyRatioMul(
    CapyRatio const x,
    CapyRatio const y);
```


Multiply two CpyRatios

Input argument(s):

- x: the first CpyRatio (must be in reduced form)
- y: the second CpyRatio (must be in reduced form)

Output and side effect(s):

- Return a new CpyRatio (in reduced form) equal to $x*y$

Exception(s):

- May raise CpyExc_NumericalOverflow

```
CpyRatio CpyRatioInv(CpyRatio const x);
```

Get the inverse of a CpyRatio

Input argument(s):

- x: the CpyRatio (must be in reduced form)

Output and side effect(s):

- Return a new CpyRatio (in reduced form) equal to $1/\text{that}$

Exception(s):

- May raise CpyExc_NumericalOverflow

```
CpyRatio CpyRatioDiv(  
    CpyRatio const x,  
    CpyRatio const y);
```

Divide two CpyRatios

Input argument(s):

- x: the first CpyRatio (must be in reduced form)
- y: the second CpyRatio (must be in reduced form)

Output and side effect(s):

- Return a new CpyRatio (in reduced form) equal to x/y

Exception(s):

- May raise `CapyExc_NumericalOverflow`

```
CapyRatio CapyRatioAbs(CapyRatio const x);
```

Get the absolute value of a `CapyRatio`

Input argument(s):

- `x`: the `CapyRatio` (must be in reduced form)

Output and side effect(s):

- Return a new `CapyRatio` (in reduced form) equal to `abs(x)`

Exception(s):

- May raise `CapyExc_NumericalOverflow`

```
CapyRatio CapyRatioPowi(  
    CapyRatio const x,  
    CapyRatioBase_t const n);
```

Raise a `CapyRatio` to an integer power

Input argument(s):

- `x`: the `CapyRatio`
- `n`: the power

Output and side effect(s):

- Return a new `CapyRatio` (in reduced form) equal to x^n

Exception(s):

- May raise `CapyExc_NumericalOverflow`

```
CapyRatio CapyRatioSqrt(CapyRatio const x);
```

Get the square root of a `CapyRatio`

Input argument(s):

- `x`: the `CapyRatio` (must be in reduced form)

Output and side effect(s):

- Return a new `CapyRatio` (in reduced form) equal to `sqrt(x)`

Exception(s):

- May raise `CapyExc_NumericalOverflow`

79 Unit rsacipher.h

RSACipher class.

79.1 Macros

```
#define CAPY_RSACIPHER_H
```

79.2 Enumerations

None.

79.3 Typedefs

None.

79.4 Struct CopyRSACipherKey

79.4.1 Struct CopyRSACipherKey's properties

```
uint64_t n;
```

```
uint64_t e;
```

```
uint64_t d;
```

79.4.2 Struct CopyRSACipherKey's methods

None.

79.5 Struct CopyRSACipherData

79.5.1 Struct CopyRSACipherData's properties

```
size_t size;
```

```
uint8_t* data;
```


79.5.2 Struct CpyRSACipherData's methods

None.

79.6 Struct CpyRSACipher

79.6.1 Struct CpyRSACipher's properties

```
CpyRSACipherKey keys;
```

Keys

79.6.2 Struct CpyRSACipher's methods

```
void (*destruct)(void);
```

Destructor

```
void (*generateKeys)(  
    uint64_t const p,  
    uint64_t const q);
```

Generate keys

Input argument(s):

- p: the first prime number to be used for key generation
- q: the second prime number to be used for key generation

Output and side effect(s):

- 'that->keys' is updated.

```
uint64_t (*cipher)(uint64_t const message);
```

Cipher a message

Input argument(s):

- message: the message to cipher

Output and side effect(s):

- Return the ciphered message.


```
uint64_t (*decipher)(uint64_t const message);
```

Decipher a message

Input argument(s):

- message: the message to decipher

Output and side effect(s):

- Return the deciphered message.

```
CapyRSACipherData (*cipherBlock)(CapyRSACipherData const message);
```

Cipher a message divided into blocks of appropriate sizes

Input argument(s):

- message: the message to cipher

Output and side effect(s):

- Return the ciphered message.

```
CapyRSACipherData (*decipherBlock)(CapyRSACipherData const message);
```

Decipher a message divided into blocks of appropriate sizes

Input argument(s):

- message: the message to decipher

Output and side effect(s):

- Return the deciphered message.

79.7 Functions

```
CapyRSACipher CapyRSACipherCreate(void);
```

Create a CapyRSACipher

Output and side effect(s):

- Return a CapyRSACipher


```
CapyRSACipher* CapyRSACipherAlloc(void);
```

Allocate memory for a new CapyRSACipher and create it
Output and side effect(s):

- Return a CapyRSACipher

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyRSACipherFree(CapyRSACipher** const that);
```

Free the memory used by a CapyRSACipher* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CapyRSACipher to free

80 Unit rulerandcompass.h

RulerAndCompass class.

80.1 Macros

```
#define CAPY_RULERANDCOMPASS_H
```

```
#define CAPY_RULERANDCOMPASS_SIZE_HASH 100
```

Size of the hash table in dictionaries of points, circles, and segments

80.2 Enumerations

None.

80.3 Typedefs

None.

80.4 Struct CapyRulerAndCompass

80.4.1 Struct CapyRulerAndCompass's properties

```
CapyPen pen;
```

Pen for drawing

```
double radiusPoint;
```

Size for drawing points (default: 0.0, points are not drawn if the radius is less than 1.0)

```
CapyRulerAndCompassPoints* points;
```

Dictionary of points

```
CapyRulerAndCompassSegments* segments;
```

Dictionary of segments

```
CapyRulerAndCompassCircles* circles;
```

Dictionary of circles

80.4.2 Struct CapyRulerAndCompass's methods

```
void (*destruct)(void);
```

Destructor

```
void (*addSegment)(  
    char const* const label,  
    char const* const pointA,  
    char const* const pointB);
```

Add a segment to the geometry

Input argument(s):

- label: label of the segment
- pointA: the label of the first extremity of the segment
- pointB: the label of the second extremity of the segment

Output and side effect(s):

- Add the segment to the geometry

```
void (*addCircle)(
    char const* const label,
    char const* const center,
    char const* const point);
```

Add a circle to the geometry given two points

Input argument(s):

- label: label of the circle
- center: the label of the center of the circle
- point: the label of a point on the circle

Output and side effect(s):

- Add the circle to the geometry

```
void (*addCircleRadius)(
    char const* const label,
    char const* const center,
    char const* const pointA,
    char const* const pointB);
```

Add a circle to the geometry given three points

Input argument(s):

- label: label of the circle
- center: the label of the center of the circle
- pointA: the label of the first point defining the radius
- pointB: the label of the second point defining the radius

Output and side effect(s):

- Add the circle to the geometry

```
size_t (*addPoint)(
    char const* const labelA,
    char const* const labelB,
    char const* const geomA,
    char const* const geomB);
```


Add the point(s) of the intersection of two geometries

Input argument(s):

- labelA: label of the first intersection
- labelB: label of the second intersection
- geomA: the label of the first geometry
- geomB: the label of the second geometry

Output and side effect(s):

- The point(s) at the intersection of the geometries are added. If there
- is no intersection nothing happens. If there is only one intersection
- 'labelB' is ignored. Return the number of intersection

```
void (*drawAll)(CapyImg* const img);
```

Draw all the geometries

Input argument(s):

- img: the image on which to draw

Output and side effect(s):

- The geometries are drawn on 'img' using 'that->pen'.

```
void (*drawGeometry)(  
    CapyImg* const img,  
    char const* const lbl);
```

Draw the requested geometry

Input argument(s):

- img: the image on which to draw
- lbl: the label of the geometry

Output and side effect(s):

- The geometry 'lbl' are drawn on 'img' using 'that->pen'. If the label
- is not found nothing happens.


```
void (*drawArc)(
    CapiImg* const img,
    char const* const lblCircle,
    char const* const lblPointA,
    char const* const lblPointB);
```

Draw an arc of a circle between two given points

Input argument(s):

- img: the image on which to draw
- lblCircle: the label of the circle
- lblPointA: the label of the origin point
- lblPointB: the label of the destination point

Output and side effect(s):

- Draw the arc of the circle between the projection of the points on the
- circle, counter-clockwise. If the label are not found nothing happens.

80.5 Functions

```
CapyRulerAndCompass CapyRulerAndCompassCreate(
    CapyPoint2D const* const pointA,
    CapyPoint2D const* const pointB);
```

Create a CapyRulerAndCompass

Input argument(s):

- pointA: the first initial point
- pointB: the second initial point

Output and side effect(s):

- Return a CapyRulerAndCompass with 'pointA' and 'pointB' as initial
- points for the geometry, respectively labeled 'a' and 'b'

```
CapyRulerAndCompass* CapyRulerAndCompassAlloc(
    CapyPoint2D const* const pointA,
    CapyPoint2D const* const pointB);
```


Allocate memory for a new `CapyRulerAndCompass` and create it

Input argument(s):

- `pointA`: the first initial point
- `pointB`: the second initial point

Output and side effect(s):

- Return a `CapyRulerAndCompass` with '`pointA`' and '`pointB`' as initial
- points for the geometry, respectively labeled '`a`' and '`b`'

Exception(s):

- May raise `CapyExc_MallocFailed`.

```
void CapyRulerAndCompassFree(CapyRulerAndCompass** const that);
```

Free the memory used by a `CapyRulerAndCompass*` and reset '*that' to NULL

Input argument(s):

- `that`: a pointer to the `CapyRulerAndCompass` to free

81 Unit `rungekutta.h`

RungeKutta implementation.

81.1 Macros

```
#define CAPY_RUNGEKUTTA_H
```

81.2 Enumerations

None.

81.3 Typedefs

None.

81.4 Struct CpyRungeKutta

81.4.1 Struct CpyRungeKutta's properties

```
CpyMathFun* derivative;
```

Derivative function $F()$. For first order: $F(t, X) = X'$ For second order: $F(t, X, X') = X''$

```
size_t order;
```

Order of the derivative (in 1,2), used to interpret the shape of X in the derivative function

```
size_t dimVar;
```

Number of variables (automatically set based on `derivation.nbInput` and `order`)

```
CpyVec initVal;
```

Initial values $[t_0, X_0]$ or $[t_0, X_0, X'_0]$ (default: all 0)

```
CpyVec vecs[5];
```

Temporary vectors for calculation

```
double deltaT;
```

Discretise step Δt (default: $1e-3$)

81.4.2 Struct CpyRungeKutta's methods

```
void (*eval)(
    double const t,
    double* const vals);
```

Approximate $X(t)$ using RK4

Input argument(s):

- t : value for which we want to approximate $X(t)$
- $vals$: values $[t_0, X(t_0)]$ or $[t_0, X(t_0), X'(t_0)]$

Output and side effect(s):

- Update 'vals' with the approximated values at 't'

```
void (*step)(double* const vals);
```

Run one step

Input argument(s):

- vals: current values $[t_0, X(t_0)]$ or $[t_0, X(t_0), X'(t_0)]$

Output and side effect(s):

- 'vals' is updated with the result of one step of RK4

```
void (*destruct)(void);
```

Destructor

81.5 Functions

```
CapyRungeKutta CapyRungeKuttaCreate(
    CapyMathFun* const derivative,
    size_t const order);
```

Create a CapyRungeKutta

Input argument(s):

- derivative: the derivative function
- order: order of the derivative (in 1,2)

Output and side effect(s):

- Return a CapyRungeKutta

```
CapyRungeKutta* CapyRungeKuttaAlloc(
    CapyMathFun* const derivative,
    size_t const order);
```

Allocate memory for a new CapyRungeKutta and create it

Input argument(s):

- derivative: the derivative function

- order: order of the derivative (in 1,2)

Output and side effect(s):

- Return a CpyRungeKutta

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyRungeKuttaFree(CpyRungeKutta** const that);
```

Free the memory used by a CpyRungeKutta* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyRungeKutta to free

82 Unit sampling.h

Sampling function parent class.

82.1 Macros

```
#define CAPY_SAMPLING_H
```

```
#define CpySamplingDef struct { \
    CpyRandom rng; \
    size_t dim; \
    CpyRangeDouble* domains; \
    void (*destruct)(void); \
    CpySamples* (*getSamples)(void); \
}
```

CpySampling object definition macro

82.2 Enumerations

None.

82.3 Typedefs

```
typedef CpySamplingDef CpySampling;
```

CpySampling object

82.4 Struct CpySample

82.4.1 Struct CpySample's properties

```
double* vals;
```

Values of the sample

82.4.2 Struct CpySample's methods

```
void (*destruct)(void);
```

Destructor

82.5 Functions

```
CpySample CpySampleCreate(size_t const dim);
```

Create a CpySample

Input argument(s):

- dim: dimension of the sample

Output and side effect(s):

- Return a CpySample

```
CpySampling CpySamplingCreate(  
    CpyRandomSeed_t const seed,  
    size_t const dim);
```

Create a CpySampling

Input argument(s):

- seed: seed for the random number generator
- dim: dimension of the samples

Output and side effect(s):

- Return a CpySampling


```
CapySampling* CapySamplingAlloc(  
    CapyRandomSeed_t const seed,  
    size_t const dim);
```

Allocate memory for a new CapySampling and create it

Input argument(s):

- seed: seed for the random number generator
- dim: dimension of the samples

Output and side effect(s):

- Return a CapySampling

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapySamplingFree(CapySampling** const that);
```

Free the memory used by a CapySampling* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapySampling to free

83 Unit sdfcsg.h

Sdf class.

83.1 Macros

```
#define CAPY_SDFCSG_H
```

```
#define CAPY_MAX_NB_TRANSFORM_SDF 10
```

Maximum number of transformation per SdfShape

83.2 Enumerations

```
typedef enum CappySdfTransformType {  
    cappySdfTransformType_none,  
    cappySdfTransformType_translate,  
    cappySdfTransformType_scale,  
    cappySdfTransformType_rotate,  
    cappySdfTransformType_extrusion,  
    cappySdfTransformType_symmetry,  
    cappySdfTransformType_revolution,  
    cappySdfTransformType_twist,  
} CappySdfTransformType;
```

Types of transformation applied to a CappySdfShape or CappySdfCsg

```
typedef enum CappySdfType {  
    cappySdf_box, cappySdf_sphere, cappySdf_torus, cappySdf_capsule,  
    cappySdf_ellipsoid, cappySdf_cone, cappySdf_plane,  
    cappySdf_merge, cappySdf_union, cappySdf_intersection, cappySdf_difference,  
    cappySdf_user,  
} CappySdfType;
```

Type of shape

```
typedef enum CappySdfParam {  
    cappySdfParam_radius = 0, cappySdfParam_halfLength,  
    cappySdfParam_majorRadius = 0, cappySdfParam_minorRadius,  
    cappySdfParam_radiusX = 0, cappySdfParam_radiusY, cappySdfParam_radiusZ,  
    cappySdfParam_halfWidth = 0, cappySdfParam_halfHeight,  
        cappySdfParam_halfDepth,  
    cappySdfParam_baseRadius = 0, cappySdfParam_topRadius, cappySdfParam_height,  
    cappySdfParam_normX = 0, cappySdfParam_normY, cappySdfParam_normZ,  
    cappySdfParam_dist,  
    cappySdfParam_nbParam = 4  
} CappySdfParam;
```

Indices of shape parameters

83.3 Typedefs

```
typedef struct CappySdf CappySdf;
```

Predeclaration of CappySdf

83.4 Struct CappySdfTransform

83.4.1 Struct CappySdfTransform's properties

```
CappySdfTransformType type;
```


Type of the transformation

```
CapyPad(CapySdfTransformType, type);
```

```
double vals[4];
```

Transformation's values. For translate, the 3 first values are used. For scale the first value is used. For rotation, the 4 values are the quaternion values (x,y,z and theta). For extrusion, the first value is used (distance from origin along z axis)

83.4.2 Struct CapySdfTransform's methods

None.

83.5 Struct CapyRayMarchingRes

83.5.1 Struct CapyRayMarchingRes's properties

```
bool flagIntersect;
```

Flag to memorise if the ray intersects the shape

```
CapyPad(bool, flagIntersect);
```

```
bool hasGaveUp;
```

Flag to memorise if the ray marching reached the maximum number of steps

```
CapyPad(bool, hasGaveUp);
```

```
CapySdf* shape;
```

Reference to the shape of the nearest intersection

```
double dist;
```

Distance from the ray origin to the intersection

```
double distMin;
```


Shortest observed signed distance between the ray and a shape during the steps of ray marching

```
double pos[3];
```

Coordinates of the intersection in world coordinate system

```
double posLocal[3];
```

Coordinates of the intersection in the shape local coordinate system

```
double normal[3];
```

Normal at the intersection in world coordinate system

83.5.2 Struct `CapyRayMarchingRes`'s methods

None.

83.6 Functions

```
double CapySdfGetEpsilon(void);
```

Getter and setter for the epsilon value

```
void CapySdfSetEpsilon(double epsilon);
```

```
size_t CapySdfGetRayMarchNbMaxStep(void);
```

Getter and setter for the maximum number of step (to avoid infinite loop) during ray marching.

```
void CapySdfSetRayMarchNbMaxStep(size_t const nb);
```

```
CapySdf CapySdfCreate(void);
```

Create a `CapySdf` (by default a unit box)

Output and side effect(s):

- Return a `CapySdf`


```
CapySdf* CapySdfAlloc(void);
```

Allocate memory for a new CapySdf and create it (by default a unit box)

Output and side effect(s):

- Return a CapySdf

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapySdfFree(CapySdf** const that);
```

Free the memory used by a CapySdf* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapySdf to free

```
CapySdf CapySdfCreateBox(  
    double const width,  
    double const height,  
    double const depth);
```

Create a CapySdf for a box of given dimensions Inputs: width: width of the box (x axis) height: height of the box (y axis) depth: depth of the box (z axis)

Output and side effect(s):

- Return a CapySdf

```
CapySdf* CapySdfAllocBox(  
    double const width,  
    double const height,  
    double const depth);
```

Allocate memory for a new CapySdf for a box of given dimensions and create it Inputs: width: width of the box (x axis) height: height of the box (y axis) depth: depth of the box (z axis)

Output and side effect(s):

- Return a CapySdf

Exception(s):

- May raise CpyExc_MallocFailed.

```
CapySdf CapySdfCreatePlane(
    double const norm[3],
    double const dist);
```

Create a CapySdf for a plane Inputs: norm: normal of the plane (toward the 'out' side of the plane) dist: distance from the origin in the direction of the normal

Output and side effect(s):

- Return a CapySdf

```
CapySdf* CapySdfAllocPlane(
    double const norm[3],
    double const dist);
```

Allocate memory for a new CapySdf for a plane and create it Inputs: norm: normal of the plane (toward the 'out' side of the plane) dist: distance from the origin in the direction of the normal

Output and side effect(s):

- Return a CapySdf

Exception(s):

- May raise CpyExc_MallocFailed.

```
CapySdf CapySdfCreateSphere(double const radius);
```

Create a CapySdf for a sphere of given dimensions Inputs: radius: radius of the sphere

Output and side effect(s):

- Return a CapySdf

```
CapySdf* CapySdfAllocSphere(double const radius);
```

Allocate memory for a new CapySdf for a sphere of given dimensions and create it Inputs: radius: radius of the sphere

Output and side effect(s):

- Return a CappySdf

Exception(s):

- May raise CappyExc_MallocFailed.

```
CappySdf CappySdfCreateCone(
    double const baseRadius,
    double const topRadius,
    double const height);
```

Create a CappySdf for a cone (center at the origin, rotation axis along the y axis, top toward +y) of given dimensions and create it Inputs: radius: radius of the sphere

Output and side effect(s):

- Return a CappySdf

```
CappySdf* CappySdfAllocCone(
    double const baseRadius,
    double const topRadius,
    double const height);
```

Allocate memory for a new CappySdf for a cone (center at the origin, rotation axis along the y axis, top toward +y) of given dimensions and create it Inputs: baseRadius: radius of the cone at

Output and side effect(s):

- Return a CappySdf

Exception(s):

- May raise CappyExc_MallocFailed.

```
CappySdf CappySdfCreateEllipsoid(
    double const radiusX,
    double const radiusY,
    double const radiusZ);
```

Create a CappySdf for an ellipsoid of given dimensions (aligned with x, y,z axis) Inputs: radiusX: radius of the ellipsoid along the x axis radiusY: radius of the ellipsoid along the y axis radiusZ: radius of the ellipsoid along the z axis

Output and side effect(s):

- Return a CappySdf

```
CappySdf* CappySdfAllocEllipsoid(
    double const radiusX,
    double const radiusY,
    double const radiusZ);
```

Allocate memory for a new CappySdf for an ellipsoid of given dimensions (aligned with x, y,z axis) and create it Inputs: radiusX: radius of the ellipsoid along the x axis radiusY: radius of the ellipsoid along the y axis radiusZ: radius of the ellipsoid along the z axis

Output and side effect(s):

- Return a CappySdf

Exception(s):

- May raise CappyExc_MallocFailed.

```
CappySdf CappySdfCreateTorus(
    double const majorRadius,
    double const minorRadius);
```

Create a CappySdf for a torus of given dimensions (in xz plane) Inputs: majorRadius: major radius of the torus minorRadius: minor radius of the torus

Output and side effect(s):

- Return a CappySdf

```
CappySdf CappySdfCreateCapsule(
    double const radius,
    double const length);
```

Create a CappySdf for a capsule (aligned with y axis) of given dimensions Inputs: radius: radius of the capsule length: length of the capsule

Output and side effect(s):

- Return a CappySdf

```
CappySdf* CappySdfAllocCapsule(
    double const radius,
    double const length);
```


Allocate memory for a new CappySdf for a capsule (aligned with y axis) of given dimensions and create it Inputs: radius: radius of the capsule length: length of the capsule

Output and side effect(s):

- Return a CappySdf

Exception(s):

- May raise CappyExc_MallocFailed.

```
CappySdf* CappySdfAllocTorus(  
    double const majorRadius,  
    double const minorRadius);
```

Allocate memory for a new CappySdf for a torus of given dimensions (in xz plane) and create it Inputs: majorRadius: major radius of the torus minorRadius: minor radius of the torus

Output and side effect(s):

- Return a CappySdf

Exception(s):

- May raise CappyExc_MallocFailed.

```
CappySdf CappySdfCreateMerge(  
    CappySdf* const shapeA,  
    CappySdf* const shapeB);
```

Create a CappySdf for a merge of two shapes Inputs: shapeA: the first shape shapeB: the second shape

Output and side effect(s):

- Return a CappySdf. Component shapes can't be shared with between several
- CSG, they must be dynamically allocated, and they are free-d by their
- parent.


```
CapySdf* CapySdfAllocMerge(  
    CapySdf* const shapeA,  
    CapySdf* const shapeB);
```

Allocate memory for a new CapySdf for a merge of two shapes and create it
Inputs: shapeA: the first shape shapeB: the second shape

Output and side effect(s):

- Return a CapySdf. Component shapes can't be shared with between several
- CSG, they must be dynamically allocated, and they are free-d by their
- parent.

Exception(s):

- May raise CapyExc_MallocFailed.

```
CapySdf CapySdfCreateUnion(  
    CapySdf* const shapeA,  
    CapySdf* const shapeB);
```

Create a CapySdf for an union of two shapes Inputs: shapeA: the first shape
shapeB: the second shape

Output and side effect(s):

- Return a CapySdf. Component shapes can't be shared with between several
- CSG, they must be dynamically allocated, and they are free-d by their
- parent.

```
CapySdf* CapySdfAllocUnion(  
    CapySdf* const shapeA,  
    CapySdf* const shapeB);
```

Allocate memory for a new CapySdf for an union of two shapes and create it
Inputs: shapeA: the first shape shapeB: the second shape

Output and side effect(s):

- Return a CapySdf. Component shapes can't be shared with between several

- CSG, they must be dynamically allocated, and they are free-d by their
- parent.

Exception(s):

- May raise CpyExc_MallocFailed.

```
CapySdf CapySdfCreateIntersection(
    CapySdf* const shapeA,
    CapySdf* const shapeB);
```

Create a CapySdf for an intersection of two shapes Inputs: shapeA: the first shape shapeB: the second shape

Output and side effect(s):

- Return a CapySdf. Component shapes can't be shared with between several
- CSG, they must be dynamically allocated, and they are free-d by their
- parent.

```
CapySdf* CapySdfAllocIntersection(
    CapySdf* const shapeA,
    CapySdf* const shapeB);
```

Allocate memory for a new CapySdf for an intersection of two shapes and create it Inputs: shapeA: the first shape shapeB: the second shape

Output and side effect(s):

- Return a CapySdf. Component shapes can't be shared with between several
- CSG, they must be dynamically allocated, and they are free-d by their
- parent.

Exception(s):

- May raise CpyExc_MallocFailed.


```
CapySdf CapySdfCreateDifference(  
    CapySdf* const shapeA,  
    CapySdf* const shapeB);
```

Create a CapySdf for a difference of two shapes Inputs: shapeA: the first shape shapeB: the second shape

Output and side effect(s):

- Return a CapySdf. Component shapes can't be shared with between several
- CSG, they must be dynamically allocated, and they are free-d by their
- parent.

```
CapySdf* CapySdfAllocDifference(  
    CapySdf* const shapeA,  
    CapySdf* const shapeB);
```

Allocate memory for a new CapySdf for a difference of two shapes and create it Inputs: shapeA: the first shape shapeB: the second shape

Output and side effect(s):

- Return a CapySdf. Component shapes can't be shared with between several
- CSG, they must be dynamically allocated, and they are free-d by their
- parent.

Exception(s):

- May raise CapyExc_MallocFailed.

```
CapySdf CapySdfCreateUserDefined(sdfFun const sdf);
```

Create a CapySdf for a user defined distance field Inputs: sdf: the user defined signed distance field (canonical shape untransformed, taking a reference to the shape and a position in local coordinate system)

Output and side effect(s):

- Return a CapySdf.


```
CapySdf* CapySdfAllocUserDefined(sdfFun const sdf);
```

Allocate memory for a new CapySdf for a user defined distance field and create it Inputs: sdf: the user defined signed distance field (canonical shape untransformed, taking a reference to the shape and a position in local coordinate system)

Output and side effect(s):

- Return a CapySdf.

Exception(s):

- May raise CapyExc_MallocFailed.

84 Unit slidingAverage.h

Sliding average class.

84.1 Macros

```
#define CAPY_SLIDING_AVERAGE_H
```

84.2 Enumerations

None.

84.3 Typedefs

None.

84.4 Struct CapySlidingAverage

84.4.1 Struct CapySlidingAverage's properties

```
size_t size;
```

Size of the sliding window

```
size_t counter;
```


Counter to memorise when the window is filled

```
double val;
```

Current average value

84.4.2 Struct `CapySlidingAverage`'s methods

```
void (*destruct)(void);
```

Destructor

```
void (*update)(double const val);
```

Update the average with a new value

Input argument(s):

- val: the new value

Output and side effect(s):

- Update the current average and counter

84.5 Functions

```
CapySlidingAverage CapySlidingAverageCreate(size_t const size);
```

Create a `CapySlidingAverage`

Input argument(s):

- size: the size of the sliding window

Output and side effect(s):

- Return a `CapySlidingAverage`

```
CapySlidingAverage* CapySlidingAverageAlloc(size_t const size);
```

Allocate memory for a new `CapySlidingAverage` and create it

Input argument(s):

- size: the size of the sliding window

Output and side effect(s):

- Return a CpySlidingAverage

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpySlidingAverageFree(CpySlidingAverage** const that);
```

Free the memory used by a CpySlidingAverage* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpySlidingAverage to free

85 Unit sort.h

Sorting algorithms.

85.1 Macros

```
#define CAPY_SORT_H
```

85.2 Enumerations

None.

85.3 Typedefs

None.

85.4 Functions

```
void CpyQuickSort(  
    void* const data,  
    size_t const sizeElem,  
    size_t const nbElem,  
    CpyComparator* const cmp);
```

Quick sort

Input argument(s):

- data: the array of data to sort
- sizeElem: the size in byte of one element in the array
- nbElem: the number of elements in the array
- cmp: the comparator used to sort the array

86 Unit springmass.h

SpringMass class.

86.1 Macros

```
#define CAPY_SPRINGMASS_H
```

86.2 Enumerations

None.

86.3 Typedefs

None.

86.4 Struct CapySpringMassNode

86.4.1 Struct CapySpringMassNode's properties

```
size_t id;
```

Mass id

```
double mass;
```

Mass of the node

```
CapyVec pos;
```

Node position


```
CapyVec speed;
```

Node velocity

```
CapyVec acc;
```

Node acceleration

```
bool flagFixed;
```

Fixed mass (true: position stays constant)

```
CapyPad(bool, flagFixed);
```

```
void* data;
```

User data

86.4.2 Struct CapySpringMassNode's methods

None.

86.5 Struct CapySpringMassLink

86.5.1 Struct CapySpringMassLink's properties

```
double coeffs[2];
```

Spring coefficients ([0]: in compression regime, [1]: in extension regime)

```
double restLength;
```

Length at rest

```
CapySpringMassNode* masses[2];
```

Links to the masses

```
void* data;
```

User data

86.5.2 Struct CpySpringMassLink's methods

None.

86.6 Struct CpySpringMassDeriv

86.6.1 Struct CpySpringMassDeriv's properties

```
struct CpyMathFunDef;
```

Inherit CpyMathFun

```
double* accs;
```

Acceleration force

86.6.2 Struct CpySpringMassDeriv's methods

```
void (*destructCpyMathFun)(void);
```

Destructor

86.7 Struct CpySpringMass

86.7.1 Struct CpySpringMass's properties

```
size_t dim;
```

Space dimension

```
double dampening;
```

Dampening coefficient (default: 1.0, in [0.0, 1.0], the lower the stronger the dampening, 1.0 is no dampening)

```
CpyGraph* graph;
```

Graph of the spring-mass sytem

```
CpySpringMassDeriv* deriv;
```

Derivative function


```
CapyRungeKutta* rk;
```

Runge-Kutta instance

```
size_t nbMaxStep;
```

Nb max step when searching for stability (default: 1000)

```
double epsilon;
```

Epsilon value used to check stability (default: 1e-6)

86.7.2 Struct CapySpringMass's methods

```
void (*destruct)(void);
```

Destructor

```
CapySpringMassNode* (*addMass)(size_t const id);
```

Add one mass

Input argument(s):

- id: id of the mass

Output and side effect(s):

- If no mass with the given id already exists a new one is added. The
- mass id is the id of the node in the graph. Return the mass. If created
- the new mass is initialised with a mass of 1.0 and all other params
- to 0.0.

```
CapySpringMassNode* (*getMass)(size_t const id);
```

Get a mass given its id

Input argument(s):

- id: the id

Output and side effect(s):

- Return the mass, or NULL if there is no mass with the requested id

```
CapySpringMassLink* (*addSpring)(
    size_t const idA,
    size_t const idB);
```

Add one spring

Input argument(s):

- idA: id of the first mass
- idB: id of the second mass

Output and side effect(s):

- Add a new spring, initialised with spring coefficients of 1.0 and
- rest length of 1.0. Return the new spring. If there is no mass with the
- given id they are automatically created.

```
CapySpringMassLink* (*getSpring)(
    size_t const idA,
    size_t const idB);
```

Get a spring given its masses id

Input argument(s):

- idA: the first mass id
- idB: the second mass id

Output and side effect(s):

- Return the spring, or NULL if there is no spring with the requested ids

```
void (*step)(double const deltaT);
```

Step the spring mass system

Input argument(s):

- deltaT: size of the step

Output and side effect(s):

- The mass properties are updated.

```
void (*stepToStableState)(double const deltaT);
```

Step the spring mass system until it stabilizes

Input argument(s):

- deltaT: size of the step

Output and side effect(s):

- The mass properties are updated.

```
double (*getStress)(void);
```

Get the stress of the system

Output and side effect(s):

- Return the total of difference between rest length and actual length
- of all links

```
double (*getSpringStress)(
    size_t const idA,
    size_t const idB);
```

Get the stress of a link

Input argument(s):

- idA: the first mass id
- idB: the second mass id

Output and side effect(s):

- Return the difference between rest length and actual length

```
double (*getSpringLength)(
    size_t const idA,
    size_t const idB);
```

Get the current length of a link

Input argument(s):

- idA: the first mass id
- idB: the second mass id

Output and side effect(s):

- Return the current length of the link

86.8 Functions

```
CapySpringMass CapySpringMassCreate(size_t const dim);
```

Create a CapySpringMass

Input argument(s):

- dim: space dimension

Output and side effect(s):

- Return a CapySpringMass

```
CapySpringMass* CapySpringMassAlloc(size_t const dim);
```

Allocate memory for a new CapySpringMass and create it

Input argument(s):

- dim: space dimension

Output and side effect(s):

- Return a CapySpringMass

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapySpringMassFree(CapySpringMass** const that);
```

Free the memory used by a CapySpringMass* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapySpringMass to free

87 Unit strDecorator.h

Class to embellish strings with ANSI escape code.

87.1 Macros

```
#define CAPY_STRDECORATOR_H
```


87.2 Enumerations

```
typedef enum CapyStrDecoratorOption {  
    capyStrDecoratorBold,  
    capyStrDecoratorUnderline,  
    capyStrDecoratorSlowBlink,  
    capyStrDecoratorFgColor,  
    capyStrDecoratorBgColor,  
    capyStrDecoratorNbOption  
} CapyStrDecoratorOption;
```

Enumeration of available options to decorate a string

87.3 Typedefs

None.

87.4 Struct CapyStrDecorator

87.4.1 Struct CapyStrDecorator's properties

```
CapyColorData fgColor;
```

Foreground and background color (default: white and black)

```
CapyColorData bgColor;
```

```
bool activeOptions[capyStrDecoratorNbOption];
```

Flags to memorise the active options

```
CapyPad(bool[capyStrDecoratorNbOption], 0);
```

87.4.2 Struct CapyStrDecorator's methods

```
void (*destruct)(void);
```

Destructor

```
void (*activate)(CapyStrDecoratorOption option);
```

Activate an option

Input argument(s):

- option: the option to activate

```
void (*deactivate)(CapyStrDecoratorOption option);
```

Deactivate an option

Input argument(s):

- option: the option to deactivate

```
bool (*isActive)(CapyStrDecoratorOption option);
```

Get the status of an option

Input argument(s):

- option: the option to get the status of
- Ouput:
- Returns true if the option is active, else false

```
int (*fprintf)(
    FILE* const stream,
    char const* const fmt,
    ...);
```

Print a decorated string on a stream

Input argument(s):

- stream: the stream on which to print
- fmt: the format string to decorate and print
- ...: the arguments of the format string

Output and side effect(s):

- Return the number of characters printed (excluding the terminating null
- byte). If an output error is encountered, return a negative value.

```
void (*setFgColor)(CapyColorData const* const color);
```

Set the foreground color used by the option capyStrDecoratorFgColor

Input argument(s):

- color: the RGB color of the foreground

```
void (*setBgColor)(CapyColorData const* const color);
```

Set the background color used by the option `copyStrDecoratorBgColor`
Input argument(s):

- color: the RGB color of the foreground

87.5 Functions

```
CapyStrDecorator CapyStrDecoratorCreate(void);
```

Create a `CapyStrDecorator`
Output and side effect(s):

- Return a `CapyStrDecorator` initialised with no active options

```
CapyStrDecorator* CapyStrDecoratorAlloc(void);
```

Allocate memory for a new `CapyStrDecorator` and create it
Output and side effect(s):

- Return a `CapyStrDecorator` initialised with no active options

Exception(s):

- May raise `CapyExc_MallocFailed`.

```
void CapyStrDecoratorFree(CapyStrDecorator** const that);
```

Free the memory used by a `CapyStrDecorator*` and reset `*that` to NULL
Input argument(s):

- that: a pointer to the `CapyStrDecorator` to free

88 Unit streamIo.h

I/O stream class.

88.1 Macros

```
#define CAPY_STREAM_IO_H
```

88.2 Enumerations

None.

88.3 Typedefs

None.

88.4 Struct CapyStreamIo

88.4.1 Struct CapyStreamIo's properties

```
char* pathname;
```

The pathname to the stream (may be null)

```
FILE* stream;
```

The stream

88.4.2 Struct CapyStreamIo's methods

```
void (*destruct)(void);
```

Destructor

```
void (*open)(
    char const* const pathname,
    char const* const mode);
```

Open the stream for a given pathname and given mode. If the StreamIo was already opened it is closed before opening the new one.

Input argument(s):

- `pathname`: the pathname to the file of the stream
- `mode`: the mode as in `fopen()`

- Exceptions:
- May raise `CapyExc_MallocFailed`, `CapyExc_StreamOpenError`

```
void (*close)(void);
```

Close the stream

```
CapyListArrChar* (*readLines)(void);
```

Load the content of a text file line per line

Output and side effect(s):

- Return the content of the file as a newly allocated list of
- array of char
- Exceptions:
- May raise `CapyExc_MallocFailed`, `CapyExc_StreamReadError`

```
void (*writeLines)(CapyListArrChar const* const lines);
```

Save a list of lines to a text file

Input argument(s):

- lines: a list of array of char, the lines to save
- Exceptions:
- May raise `CapyExc_StreamWriteError`

```
CapyArrChar* (*read)(void);
```

Load the raw content of a file

Output and side effect(s):

- Return the content of the file as an array of char
- Exceptions:
- May raise `CapyExc_MallocFailed`, `CapyExc_StreamReadError`


```
void (*write)(CapyArrChar const* const data);
```

Save data to a file

Input argument(s):

- data: an array of char, the data to save
- Exceptions:
- May raise CapyExc_StreamWriteError

```
bool (*isOpenned)(void);
```

Check if a stream is opened

Output and side effect(s):

- Return true if the stream is opened, else false

```
void (*readBytes)(  
    void* const data,  
    size_t const size);
```

Read bytes from a file

Input argument(s):

- data: pointer to where the data are written
- size: the size in bytes of the read data
- Exceptions:
- May raise CapyExc_StreamReadError

```
void (*writeBytes)(  
    void const* const data,  
    size_t const size);
```

Write bytes to a file

Input argument(s):

- data: pointer to the data
- size: the size in bytes of the data
- Exceptions:

- May raise `CapyExc_StreamWriteError`

```
size_t (*getSize)(void);
```

Get the size of the file associated to the stream.

Output and side effect(s):

- Return the size in bytes of the file, or 0 if the stream is not opened
- or any error occurred.

88.5 Functions

```
CapyStreamIo CapyStreamIoCreate(void);
```

Create a `CapyStreamIo`

Output and side effect(s):

- Return a `CapyStreamIo`

```
CapyStreamIo* CapyStreamIoAlloc(void);
```

Allocate memory for a new `CapyStreamIo` and create it

Output and side effect(s):

- Return a `CapyStreamIo`

Exception(s):

- May raise `CapyExc_MallocFailed`.

```
void CapyStreamIoFree(CapyStreamIo** const that);
```

Free the memory used by a `CapyStreamIo*` and reset `*that` to NULL

Input argument(s):

- `that`: a pointer to the `CapyStreamIo` to free

89 Unit supportVectorMachine.h

Support vector machine class.

89.1 Macros

```
#define CAPY_SUPPORT_VECTOR_MACHINE_H
```

```
#define CapySVMKernelDef \
struct { \
    void (*destruct)(void); \
    double (*eval)( \
        CapyVec const* const u, \
        CapyVec const* const v); \
    void (*exportToCFun)( \
        FILE* const stream, \
        char const* const name); \
    void (*exportToJavascript)( \
        FILE* const stream, \
        char const* const name); \
}
```

SVMKernel parent class

89.2 Enumerations

None.

89.3 Typedefs

```
typedef CapySVMKernelDef CapySVMKernel;
```

SVMKernel structure

89.4 Struct CapySVMKernelLinear

89.4.1 Struct CapySVMKernelLinear's properties

```
CapySVMKernelDef;
```

Inherits CapySVMKernel

89.4.2 Struct CapySVMKernelLinear's methods

```
void (*destructCapySVMKernel)(void);
```

Destructor

89.5 Struct CpySVMKernelPolynomial

89.5.1 Struct CpySVMKernelPolynomial's properties

```
CpySVMKernelDef;
```

Inherits CpySVMKernel

```
double power;
```

Power of the polynomial (default: 1.0)

```
double theta;
```

Threshold (default: 0.0)

89.5.2 Struct CpySVMKernelPolynomial's methods

```
void (*destructCpySVMKernel)(void);
```

Destructor

89.6 Struct CpySVMKernelGaussian

89.6.1 Struct CpySVMKernelGaussian's properties

```
CpySVMKernelDef;
```

Inherits CpySVMKernel

```
double gamma;
```

Gamma (default: 1.0)

89.6.2 Struct CpySVMKernelGaussian's methods

```
void (*destructCpySVMKernel)(void);
```

Destructor

89.7 Struct CpySVMEvaluation

89.7.1 Struct CpySVMEvaluation's properties

```
CpyPredictorEvaluationDef;
```

Parent class

```
double reducNbSupportVector;
```

Reduction coefficient of the support vector (=1-nbSupport/nbExample)

89.7.2 Struct CpySVMEvaluation's methods

```
void (*destructCpyPredictorEvaluation)(void);
```

Destructor

89.8 Struct CpySupportVectorMachine

89.8.1 Struct CpySupportVectorMachine's properties

```
CpyPredictorDef;
```

Inherit CpyPredictor

```
CpySVMKernel* kernel;
```

Reference to the used kernel

```
CpyVec lambda;
```

Lagrangian multipliers

```
double bias;
```

Bias

```
CpyMat support;
```

Support vectors (one vector per row, input values (normalised in [0,1]) followed by category in -1,1


```
double tolerance;
```

Tolerance (default: 1.0e-3)

```
double coeffRelax;
```

Margin relaxation coefficient (default: 1e-2, the higher the more sensitive to outliers, must be strictly greater than that.tolerance)

```
CapyRandomSeed_t seed;
```

Seed for the pseudo random generator used to shuffle the rows during training

```
size_t iCat;
```

Index of the predicted category (default: 1)

```
double reducNbSupportVector;
```

Reduction coefficient of the support vector ($=1-\text{nbSupport}/\text{nbExample}$), used during evaluation of the SVM

```
size_t nbMaxIterTraining;
```

Maximum number of iteration during training (default: 0, if equal to 0 the number rows in the dataset is used instead)

```
bool verbose;
```

Flag for verbose mode (default: false)

```
CapyPad(bool, 1);
```

89.8.2 Struct CapySupportVectorMachine's methods

```
void (*destructCapyPredictor)(void);
```

Destructor

```
void (*exportBodyToHtml)(  
    FILE* const stream,  
    char const* const title,  
    CapyDataset const* const dataset,  
    double const expectedAccuracy);
```


Function herited from the parent to export the body to HTML.

Input argument(s):

- stream: the stream where to export
- title: the title of the web app
- dataset: the training dataset
- expectedAccuracy: the expected accuracy of the predictor (in $[0,1]$)

Output and side effect(s):

- The `<head>` and `<body>` part of a ready to use web app implementing the
- predictor is written on the stream. The web app can be completed by
- calling `exportToHtml` on the predictor to write the `<script>` part.

```
void (*setKernel)(CapySVMKernel* const kernel);
```

Set the kernel

Input argument(s):

- kernel: the kernel

Output and side effect(s):

- The reference to the kernel updated.

89.9 Functions

```
CapySVMKernel CapySVMKernelCreate(void);
```

Create a `CapySVMKernel`

Output and side effect(s):

- Return a `CapySVMKernel`

```
CapySVMKernelLinear CapySVMKernelLinearCreate(void);
```

Create a `CapySVMKernelLinear`

Output and side effect(s):

- Return a CpySVMKernelLinear

```
CpySVMKernelLinear* CpySVMKernelLinearAlloc(void);
```

Allocate memory for a new CpySVMKernelLinear and create it
Output and side effect(s):

- Return a CpySVMKernelLinear

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpySVMKernelLinearDestruct(CpySVMKernelLinear* const that);
```

Free the memory used by a CpySVMKernelLinear
Input argument(s):

- that: the CpySVMKernelLinear to free

```
void CpySVMKernelLinearFree(CpySVMKernelLinear** const that);
```

Free the memory used by a CpySVMKernelLinear* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpySVMKernelLinear to free

```
CpySVMKernelPolynomial CpySVMKernelPolynomialCreate(void);
```

Create a CpySVMKernelPolynomial

Output and side effect(s):

- Return a CpySVMKernelPolynomial

```
CpySVMKernelPolynomial* CpySVMKernelPolynomialAlloc(void);
```

Allocate memory for a new CpySVMKernelPolynomial and create it
Output and side effect(s):

- Return a CpySVMKernelPolynomial

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpySVMKernelPolynomialFree(CpySVMKernelPolynomial** const that);
```

Free the memory used by a CpySVMKernelPolynomial* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpySVMKernelPolynomial to free

```
CpySVMKernelGaussian CpySVMKernelGaussianCreate(void);
```

Create a CpySVMKernelGaussian

Output and side effect(s):

- Return a CpySVMKernelGaussian

```
CpySVMKernelGaussian* CpySVMKernelGaussianAlloc(void);
```

Allocate memory for a new CpySVMKernelGaussian and create it

Output and side effect(s):

- Return a CpySVMKernelGaussian

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpySVMKernelGaussianFree(CpySVMKernelGaussian** const that);
```

Free the memory used by a CpySVMKernelGaussian* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpySVMKernelGaussian to free

```
CpySVMEvaluation CpySVMEvaluationCreate(void);
```


Create a CapySVMEvaluation

Output and side effect(s):

- Return a CapySVMEvaluation

```
CapySVMEvaluation* CapySVMEvaluationAlloc(void);
```

Allocate memory for a new CapySVMEvaluation and create it

Output and side effect(s):

- Return a CapySVMEvaluation

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapySVMEvaluationFree(CapySVMEvaluation** const that);
```

Free the memory used by a CapySVMEvaluation* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapySVMEvaluation to free

```
CapySupportVectorMachine CapySupportVectorMachineCreate(void);
```

Create a CapySupportVectorMachine

Output and side effect(s):

- Return a CapySupportVectorMachine

```
CapySupportVectorMachine* CapySupportVectorMachineAlloc(void);
```

Allocate memory for a new CapySupportVectorMachine and create it

Output and side effect(s):

- Return a CapySupportVectorMachine

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapySupportVectorMachineFree(CapySupportVectorMachine** const that);
```

Free the memory used by a CapySupportVectorMachine* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapySupportVectorMachine to free

90 Unit tree.h

Generic tree class.

90.1 Macros

```
#define CAPY_TREE_H
```

```
#define CopyDecTreeIterator(name, type_) \
typedef struct name name; \
typedef struct name ## IteratorStep name ## IteratorStep; \
struct name ## IteratorStep { \
    name* node; \
    name ## IteratorStep* next; \
}; \
typedef struct name ## Iterator { \
    size_t idx; \
    name* tree; \
    type_ datatype; \
    CopyPad(type_, datatype); \
    CopyTreeIteratorType type; \
    CopyPad(CopyTreeIteratorType, 1); \
    name ## IteratorStep* steps; \
    void (*destruct)(void); \
    type_* (*reset)(void); \
    type_* (*next)(void); \
    bool (*isActive)(void); \
    type_* (*get)(void); \
    void (*setType)(CopyTreeIteratorType type); \
} name ## Iterator; \
name ## Iterator name ## IteratorCreate( \
    name* const arr, CopyTreeIteratorType const type); \
name ## Iterator* name ## IteratorAlloc( \
    name* const arr, CopyTreeIteratorType const type); \
void name ## IteratorDestruct(void); \
void name ## IteratorFree(name ## Iterator** const that); \
type_* name ## IteratorReset(void); \
type_* name ## IteratorNextBreadthFirst(void); \
type_* name ## IteratorNextDepthFirst(void); \
bool name ## IteratorIsActive(void); \
void name ## IteratorToLast(void); \
type_* name ## IteratorGet(void); \
void name ## IteratorSetType(CopyTreeIteratorType type);
```

Iterator for generic tree structure. Declaration macro for an iterator structure named 'name' ## Iterator, associated to a tree named 'name' containing elements of type 'type'

```
#define CopyDefTreeIteratorCreate(name, type_) \
name ## Iterator name ## IteratorCreate( \
    name* const tree, \
    CopyTreeIteratorType const type) { \
    name ## Iterator that = { \
```



```

        .idx = 0,
        .tree = tree,
        .type = type,
        .steps = NULL,
        .destruct = name ## IteratorDestruct,
        .reset = name ## IteratorReset,
        .isActive = name ## IteratorIsActive,
        .get = name ## IteratorGet,
        .setType = name ## IteratorSetType,
    };
    if(type == copyTreeIteratorType_breadthFirst)
        that.next = name ## IteratorNextBreadthFirst;
    else if(type == copyTreeIteratorType_depthFirst)
        that.next = name ## IteratorNextDepthFirst;
    $(&that, reset)();
    return that;
}

```

Create an iterator on a generic tree

Input argument(s):

- tree: the generic tree on which to iterate
- type: the type of iterator

Output and side effect(s):

- Return the iterator

```

#define CopyDefTreeIteratorAlloc(name, type_)
name ## Iterator* name ## IteratorAlloc(
    name* const tree,
    CopyTreeIteratorType const type) {
    name ## Iterator* that = NULL;
    safeMalloc(that, 1);
    if(!that) return NULL;
    name ## Iterator i = name ## IteratorCreate(tree, type);
    memcpy(that, &i, sizeof(name ## Iterator));
    return that;
}

```

Allocate memory and create an iterator on a generic tree

Input argument(s):

- tree: the generic tree on which to iterate
- type: the type of iterator

Output and side effect(s):

- Return the iterator


```

#define CopyDefTreeIteratorDestruct(name, type) \
void name ## IteratorFreeSteps(name ## Iterator* const that) { \
    name ## IteratorStep* step = that->steps; \
    while(that->steps != NULL) { \
        step = that->steps->next; \
        free(that->steps); \
        that->steps = step; \
    } \
} \
void name ## IteratorDestruct(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    name ## IteratorFreeSteps(that); \
}

```

Free the memory used by an iterator.

Input argument(s):

- that: the iterator to free

```

#define CopyDefTreeIteratorFree(name, type) \
void name ## IteratorFree(name ## Iterator** const that) { \
    if(that == NULL || *that == NULL) return; \
    $(*that, destruct)(); \
    free(*that); \
    *that = NULL; \
}

```

Free the memory used by a pointer to an iterator and reset '*that' to NULL

Input argument(s):

- that: a pointer to the iterator to free

```

#define CopyDefTreeIteratorReset(name, type_) \
type_* name ## IteratorReset(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    assert(that->tree && "Tree iterator uninitialised"); \
    that->idx = 0; \
    name ## IteratorFreeSteps(that); \
    safeMalloc(that->steps, 1); \
    that->steps->node = that->tree; \
    that->steps->next = NULL; \
    if(name ## IteratorIsActive()) \
        return name ## IteratorGet(); \
    else \
        return NULL; \
}

```

Reset the iterator

Output and side effect(s):

- Return the first element of the iteration


```

#define CopyDefTreeIteratorNextBreadthFirst(name, type_) \
type_* name ## IteratorNextBreadthFirst(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    if(name ## IteratorIsActive()) { \
        ++(that->idx); \
        name* child = that->steps->node->child; \
        name ## IteratorStep* first = that->steps; \
        that->steps = that->steps->next; \
        free(first); \
        name ## IteratorStep* last = that->steps; \
        while(last && last->next) last = last->next; \
        while(child != NULL) { \
            name ## IteratorStep* step = NULL; \
            safeMalloc(step, 1); \
            step->node = child; \
            step->next = NULL; \
            if(last == NULL) { \
                that->steps = step; \
                last = step; \
            } else { \
                last->next = step; \
                last = step; \
            } \
            child = child->brother; \
        } \
        return $(that, get)(); \
    } else return NULL; \
}

```

Move the iterator to the next element (breadth first)
Output and side effect(s):

- Return the next element of the iteration

```

#define CopyDefTreeIteratorNextDepthFirst(name, type_) \
type_* name ## IteratorNextDepthFirst(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    if(name ## IteratorIsActive()) { \
        ++(that->idx); \
        name* child = that->steps->node->child; \
        name ## IteratorStep* first = that->steps; \
        that->steps = that->steps->next; \
        free(first); \
        first = that->steps; \
        that->steps = NULL; \
        name ## IteratorStep* last = NULL; \
        while(child != NULL) { \
            name ## IteratorStep* step = NULL; \
            safeMalloc(step, 1); \
            step->node = child; \
            step->next = NULL; \
            if(last == NULL) { \
                that->steps = step; \
                last = step; \
            } else { \
                last->next = step; \
                last = step; \
            } \
            child = child->brother; \
        } \
        return $(that, get)(); \
    } else return NULL; \
}

```



```

    }
    child = child->brother;
}
if(last != NULL) last->next = first;
else that->steps = first;
return $(that, get)();
} else return NULL;
}
\
\
\
\
\
\

```

Move the iterator to the next element (depth first)

Output and side effect(s):

- Return the next element of the iteration

```

#define CopyDefTreeIteratorIsActive(name, type) \
bool name ## IteratorIsActive(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    return (that->steps != NULL); \
}
\
\

```

Check if the iterator is on a valid element

Output and side effect(s):

- Return true if the iterator is on a valid element, else false

```

#define CopyDefTreeIteratorGet(name, type_) \
type_* name ## IteratorGet(void) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    if(that->steps == NULL) return NULL; \
    return &(amp;that->steps->node->data); \
}
\
\

```

Get the current element of the iteration

Output and side effect(s):

- Return a pointer to the current element

```

#define CopyDefTreeIteratorSetType(name, type_) \
void name ## IteratorSetType(CopyTreeIteratorType type) { \
    name ## Iterator* that = (name ## Iterator*)copyThat; \
    that->type = type; \
    if(type == copyTreeIteratorType_breadthFirst) \
        that->next = name ## IteratorNextBreadthFirst; \
    else if(type == copyTreeIteratorType_depthFirst) \
        that->next = name ## IteratorNextDepthFirst; \
    name ## IteratorReset(); \
}
\
\

```

Set the type of the iterator and reset it

Input argument(s):

- type: the new type of the iterator

```
#define CopyDefTreeIterator(name, type) \
    CopyDefTreeIteratorCreate(name, type) \
    CopyDefTreeIteratorAlloc(name, type) \
    CopyDefTreeIteratorDestruct(name, type) \
    CopyDefTreeIteratorFree(name, type) \
    CopyDefTreeIteratorReset(name, type) \
    CopyDefTreeIteratorNextBreadthFirst(name, type) \
    CopyDefTreeIteratorNextDepthFirst(name, type) \
    CopyDefTreeIteratorIsActive(name, type) \
    CopyDefTreeIteratorGet(name, type) \
    CopyDefTreeIteratorSetType(name, type)
```

Definition macro calling all the submacros at once for an iterator on an array structure named 'name', containing elements of type 'type'

```
#define CopyDecTree(name, type) \
    CopyDecTreeIterator(name, type) \
    struct name { \
        type data; \
        CopyPad(type, data); \
        name ## Iterator iter; \
        name* parent; \
        name* brother; \
        name* child; \
        void (*destruct)(void); \
        void (*setData)(type const* const val); \
        type (*getData)(void); \
        type* (*getDataPtr)(void); \
        name* (*getParent)(void); \
        name* (*getNextBrother)(void); \
        name* (*getFirstChild)(void); \
        name* (*addChild)(type const* const val); \
        name* (*addBrother)(type const* const val); \
        void (*initIterator)(void); \
        size_t (*getNbLeaf)(void); \
        void (*print)( \
            FILE* const stream, \
            CopyTreePrintData const fun, \
            void* const user); \
    }; \
    name name ## Create(type const* const val); \
    name* name ## Alloc(type const* const val); \
    void name ## Destruct(void); \
    void name ## Free(name** const that); \
    void name ## SetData(type const* const val); \
    type name ## GetData(void); \
    type* name ## GetDataPtr(void); \
    name* name ## GetParent(void); \
    name* name ## GetNextBrother(void); \
    name* name ## GetFirstChild(void); \
    name* name ## AddChild(type const* const val); \
    name* name ## AddBrother(type const* const val); \
    size_t name ## GetNbLeaf(void); \
    void name ## Print( \
        FILE* const stream, \
        CopyTreePrintData const fun,
```



```

        void* const user);
void name ## InitIterator(void);

```

Generic tree structure. Declaration macro for a tree structure named 'name', containing data of type 'type'

```

#define CpyDefTreeCreate(name, type) \
name name ## Create(type const* const val) { \
    return (name){ \
        .data = *val, \
        .parent = NULL, \
        .brother = NULL, \
        .child = NULL, \
        .destruct = name ## Destruct, \
        .setData = name ## SetData, \
        .getData = name ## GetData, \
        .initIterator = name ## InitIterator, \
        .getDataPtr = name ## GetDataPtr, \
        .getParent = name ## GetParent, \
        .getNextBrother = name ## GetNextBrother, \
        .getFirstChild = name ## GetFirstChild, \
        .addChild = name ## AddChild, \
        .addBrother = name ## AddBrother, \
        .getNbLeaf = name ## GetNbLeaf, \
        .print = name ## Print, \
        .iter = {.tree = NULL}, \
    }; \
}

```

Generic tree structure. Definition macro and submacros for a tree structure named 'name', containing data of type 'type' Create a tree

Output and side effect(s):

- Return a tree containing data of type 'type'

```

#define CpyDefTreeAlloc(name, type) \
name* name ## Alloc(type const* const val) { \
    name* that = NULL; \
    safeMalloc(that, 1); \
    if(!that) return NULL; \
    *that = name ## Create(val); \
    that->iter = name ## IteratorCreate( \
        that, copyTreeIteratorType_breadthFirst); \
    return that; \
}

```

Allocate memory for a new tree and create it

Output and side effect(s):

- Return a tree containing data of type 'type'

Exception(s):

- May raise `CapyExc_MallocFailed`.

```
#define CapyDefTreeDestruct(name, type) \
void name ## Destruct(void) { \
    name* that = (name*)capyThat; \
    if(that->child != NULL) name ## Free(&(that->child)); \
    if(that->brother != NULL) name ## Free(&(that->brother)); \
    if(that->data.destruct != NULL) $(&(that->data), destruct)(); \
    $(&(that->iter), destruct)(); \
    *that = (name){0}; \
}
```

Free the memory used by a tree.

Input argument(s):

- `that`: the tree to free

```
#define CapyDefTreeFree(name, type) \
void name ## Free(name** const that) { \
    if(that == NULL || *that == NULL) return; \
    $(*that, destruct)(); \
    free(*that); \
    *that = NULL; \
}
```

Free the memory used by a pointer to a tree and reset `*that` to NULL

Input argument(s):

- `that`: a pointer to the tree to free

```
#define CapyDefTreeSetData(name, type) \
void name ## SetData(type const* const val) { \
    assert (val != NULL); \
    name* that = (name*)capyThat; \
    that->data = *val; \
}
```

Set the data of the tree node

Input argument(s):

- `val`: the value to set

```
#define CapyDefTreeGetData(name, type) \
type name ## GetData(void) { \
    name* that = (name*)capyThat; \
    return that->data; \
}
```


Get the data of the tree node

Output and side effect(s):

- Return the data value

```
#define CopyDefTreeGetDataPtr(name, type) \
type* name ## GetDataPtr(void) {        \
    name* that = (name*)copyThat;        \
    return &(amp;that->data);              \
}
```

Get a pointer to the data of the tree node

Output and side effect(s):

- Return the pointer to the data

```
#define CopyDefTreeGetParent(name, type) \
name* name ## GetParent(void) {         \
    name* that = (name*)copyThat;        \
    return that->parent;                  \
}
```

Get a pointer to the parent of the tree node

Output and side effect(s):

- Return the pointer to the parent node

```
#define CopyDefTreeGetFirstChild(name, type) \
name* name ## GetFirstChild(void) {          \
    name* that = (name*)copyThat;            \
    return that->child;                       \
}
```

Get a pointer to the first child of the tree node

Output and side effect(s):

- Return the pointer to the first child node

```
#define CopyDefTreeGetNextBrother(name, type) \
name* name ## GetNextBrother(void) {         \
    name* that = (name*)copyThat;            \
    return that->brother;                     \
}
```

Get a pointer to the next brother of the tree node

Output and side effect(s):

- Return the pointer to the next brother node

```
#define CopyDefTreeInitIterator(name, type) \
void name ## InitIterator(void) { \
    name* that = (name*)copyThat; \
    that->iter = \
        name ## IteratorCreate(that, copyTreeIteratorType_breadthFirst); \
}
```

Initialise the iterator of the tree, must be called after the creation of the tree when it is created with Create(), Alloc() automatically initialise the iterator.

```
#define CopyDefTreeAddChild(name, type) \
name* name ## AddChild(type const* const val) { \
    name* that = (name*)copyThat; \
    if(that->child == NULL) { \
        that->child = name ## Alloc(val); \
        that->child->parent = that; \
        return that->child; \
    } else return $(that->child, addBrother)(val); \
}
```

Add a child to the tree node. The child is added to the brothers of the current child, or become the child if there isn't yet. The new child is returned.

```
#define CopyDefTreeAddBrother(name, type) \
name* name ## AddBrother(type const* const val) { \
    name* that = (name*)copyThat; \
    if(that->brother == NULL) { \
        that->brother = name ## Alloc(val); \
        that->brother->parent = that; \
        return that->brother; \
    } else { \
        name* lastBrother = that->brother; \
        while(lastBrother->brother) lastBrother = lastBrother->brother; \
        lastBrother->brother = name ## Alloc(val); \
        lastBrother->brother->parent = lastBrother->parent; \
        return lastBrother->brother; \
    } \
}
```

Add a brother to the tree node. The brother is added to the end of the list. The new brother is returned.

```
#define CopyDefTreeGetNbLeaf(name, type) \
size_t name ## GetNbLeaf(void) { \
    name* that = (name*)copyThat; \
    size_t nbLeaf = 0; \
    foreach(node, that->iter) { \
        if(that->iter.steps->node->child == NULL) nbLeaf += 1; \
    } \
    return nbLeaf; \
}
```


Get the number of leaf (node with no child) in the tree. The iterator is modified.

```

#define CopyDefTreePrint(name, type) \
void name ## PrintRec( \
    name* const that, \
    size_t const depth, \
    uint32_t* const prefix, \
    FILE* const stream, \
    CopyTreePrintData const fun, \
    void* const user) { \
    if(depth >= 1023) raiseExc(CapyExc_InfiniteLoop); \
    if(that->child == NULL) { \
        size_t last = 0; \
        loop(i, depth) if(prefix[i] == L' ' || prefix[i] == L' ') last = i; \
        \
        loop(i, depth) { \
            if(prefix[i] == L' ' && i < last) fprintf(stream, " "); \
            else if(prefix[i] == L' ' && i < last) fprintf(stream, " "); \
            else if(prefix[i] == L' ' && i < last) fprintf(stream, " "); \
            else fprintf(stream, "%lc", prefix[i]); \
        } \
        fprintf(stream, " "); \
        fun(stream, &(that->data), user); \
        fprintf(stream, "\n"); \
    } else { \
        name* child = that->child; \
        prefix[depth] = L' '; \
        prefix[depth + 1] = '\0'; \
        while(child) { \
            name ## PrintRec(child, depth + 1, prefix, stream, fun, user); \
            child = child->brother; \
            if(child && child->brother) prefix[depth] = L' '; \
            else prefix[depth] = L' '; \
        } \
        prefix[depth] = '\0'; \
    } \
} \
void name ## Print( \
    FILE* const stream, \
    CopyTreePrintData const fun, \
    void* const user) { \
    name* that = (name*)capyThat; \
    uint32_t* prefix = NULL; \
    safeMalloc(prefix, 1024); \
    loop(i, 1024) prefix[i] = 0; \
    name ## PrintRec(that, 0, prefix, stream, fun, user); \
    free(prefix); \
}

```

Print the tree in text mode, one line per leaf with the arborescence in ascii art, and the data printed using the 'fun' function which must print one data on one line without line return The locale must be set such as ascii art is displayable, for example with: char *locale = setlocale(LC_ALL, "");

```

#define CopyDefTree(name, type) \
    CopyDefTreeCreate(name, type) \

```



```

CopyDefTreeAlloc(name, type)      \
CopyDefTreeDestruct(name, type)   \
CopyDefTreeFree(name, type)       \
CopyDefTreeSetData(name, type)    \
CopyDefTreeGetData(name, type)    \
CopyDefTreeGetDataPtr(name, type) \
CopyDefTreeGetParent(name, type)  \
CopyDefTreeGetFirstChild(name, type) \
CopyDefTreeGetNextBrother(name, type) \
CopyDefTreeAddBrother(name, type)  \
CopyDefTreeAddChild(name, type)   \
CopyDefTreeInitIterator(name, type) \
CopyDefTreeGetNbLeaf(name, type)  \
CopyDefTreePrint(name, type)      \
CopyDefTreeIterator(name, type)

```

Definition macro calling all the submacros at once for an array structure named 'name', containing 'size_' elements of type 'type'

90.2 Enumerations

```

typedef enum CopyTreeIteratorType {
    copyTreeIteratorType_breadthFirst,
    copyTreeIteratorType_depthFirst,
} CopyTreeIteratorType;

```

Enumeration for the types of iterator on generic tree

90.3 Typedefs

None.

90.4 Functions

None.

91 Unit trycatch.h

Exception management.

91.1 Macros

```

#define CAPY_TRYCATCH_H

```



```
#define CAPY_MAX_TRYCATCHLVL 256
```

Size of the stack of TryCatch blocks, define how many recursive incursion of TryCatch blocks can be done, overflow is checked at the beginning of each TryCatch blocks with CapyTryCatchGuardOverflow()

```
#define try \
    CapyTryCatchGuardOverflow(); \
    switch(setjmp(*CapyTryCatchGetJmpBufOnStackTop())) { \
        case 0:
```

Head of the TryCatch block, to be used as

```
try /*... code of the TryCatch block here ...*/
```

Comments on the macro: // Guard against recursive incursion overflow
CapyTryCatchGuardOverflow(); // Memorise the jmp_buf on the top of the stack, setjmp returns 0 switch(setjmp(*CapyTryCatchGetJmpBufOnStackTop()))
// Entry point for the code of the TryCatch block case 0:

```
#define catch(e) \
    CapyTryCatchExitCatchBlock(); \
    break; \
    case e: \
        CapyTryCatchEnterCatchBlock();
```

Catch segment in the TryCatch block, to be used as

```
try /*... code of the TryCatch block here ...*/ catch (/*... one of  
CapyException or user-defined exception ...*/) /*... code executed if the  
exception has been raised in the TryCatch block ...*/
```

Comments on the macro: // Exit the previous Catch block CapyTryCatchExitCatchBlock(); // End of the previous case break; // case of the raised exception case e: // Flag the entrance into the Catch block CapyTryCatchEnterCatchBlock();

```
#define catchAlso(e) /* fall through */ \
    case e: \
        CapyTryCatchEnterCatchBlock();
```

Macro to assign several exceptions to one Catch segment in the TryCatch block, to be used as

```
try /*... code of the TryCatch block here ...*/ catch (/*... one of  
CapyException or user-defined exception ...*/) catchAlso (/*... another one  
...*/) /*... as many catchAlso statement as you need ...*/ /*... code executed  
if one of the exception has been raised in the TryCatch block ... (Use  
CapyGetLastExcId() if you need to know which exception as been raised) */
```

Comments on the macro: // Avoid the fall through warning due to the
// CapyTryCatchEnterCatchBlock() at the entrance of the catch case /* fall

through `*/ // case of the raised exception case e: // Flag the entrance into the Catch block CappyTryCatchEnterCatchBlock();`

```
#define catchDefault \
    CappyTryCatchExitCatchBlock(); \
    break; \
default: \
    CappyTryCatchEnterCatchBlock();
```

Macro to declare the default Catch segment in the TryCatch block, must be the last Catch segment in the TryCatch block, to be used as

`try /*... code of the TryCatch block here ...*/ catchDefault /*... code executed if an exception has been raised in the TryCatch block and hasn't been caught by a previous Catch segment... (Use CappyGetLastExcId() if you need to know which exception as been raised) */`

Comments on the macro: `// Exit the previous Catch block CappyTryCatchExitCatchBlock(); // End of the previous case break; // default case default: // Flag the entrance into the Catch block CappyTryCatchEnterCatchBlock();`

```
#define endCatch \
    CappyTryCatchExitCatchBlock(); \
    break; \
} \
CappyTryCatchEnd()
```

Tail of the TryCatch block, to be used as

`try /*... code of the TryCatch block here ...*/ endCatch;`

Comments on the macro: `// Exit the previous Catch block CappyTryCatchExitCatchBlock(); // End of the previous case break; // default case, i.e. any raised exception which hasn't been caught // by a previous Catch is caught here default: // Processing of uncaught exception CappyTryCatchDefault(); // End of the switch statement at the head of the TryCatch block // Post processing of the TryCatchBlock CappyTryCatchEnd()`

```
#define raiseExc(exc) CappyRaise(exc, __FILE__, __LINE__)
```

Macro to raise the CappyException exc, to be used as

`raiseExc(/* ... one of CappyException or user-defined exception ... */);`

```
#define recatch(block) \
do { \
    Try { block; } \
    catchDefault { raiseExc(CappyTryCatchGetLastExc()); } \
endCatch; \
} while(false)
```


Macro to recatch and forward an exception. This is usefull when an exception may be raised by a handler, in which case the trace loose track of where the exception has occured. By recatching the block of code susceptible of triggering the handler, one can ensure the trace will properly indicates this block of code as the source of the exception. To be used as

```
recatch(/* ... block of code eventually raising an exception ... */);
```

91.2 Enumerations

```
typedef enum CapyException {
    CapyExc_TooManyExcToStrFun = 1,
    CapyExc_MallocFailed,
    CapyExc_StreamOpenError,
    CapyExc_StreamReadError,
    CapyExc_StreamWriteError,
    CapyExc_InvalidStream,
    CapyExc_ForkFailed,
    CapyExc_InvalidCLIArg,
    CapyExc_UndefinedExecution,
    CapyExc_NumericalOverflow,
    CapyExc_NoColorChartInImage,
    CapyExc_MatrixInversionFailed,
    CapyExc_InvalidNodeIdx,
    CapyExc_InvalidElemIdx,
    CapyExc_NoTimeAvailable,
    CapyExc_QRDecompositionFailed,
    CapyExc_UnsupportedFormat,
    CapyExc_InvalidParameters,
    CapyExc_FloatingPointImprecision,
    CapyExc_RSAModulusTooSmall,
    CapyExc_InvalidStateIdx,
    CapyExc_InvalidActionIdx,
    CapyExc_InfiniteLoop,
    CapyExc_LastID
} CapyException;
```

List of exceptions ID, must start at 1 (0 is reserved for the setjmp at the beginning of the TryCatch blocks). If the user wants to define and use its own exceptions, their ID must not be in [0, CapyExc_LastID - 1]

91.3 Typedefs

```
typedef int CapyException_t;
```

Type of an exception ID. This must be compatible with an enumeration type.

91.4 Functions


```
void CpyTryCatchGuardOverflow(void);
```

Function called at the beginning of a TryCatch block to guard against overflow of the stack of jump_buf

```
jmp_buf* CpyTryCatchGetJmpBufOnStackTop(void);
```

Function called to get the jmp_buf on the top of the stack when starting a new TryCatch block

Output and side effect(s):

- Remove the jmp_buf on the top of the stack and return it

```
void CpyTryCatchEnterCatchBlock(void);
```

Function called when entering a catch block

```
void CpyTryCatchExitCatchBlock(void);
```

Function called when exiting a catch block

```
void CpyTryCatchEnd(void);
```

Function called at the end of a TryCatch block

```
void CpyRaise(  
    CpyException_t const exc,  
    char const* const filename,  
    int16_t const line);
```

Function called to raise the CpyException 'exc'. An uncaught exception has no effect and simply fall through.

Input argument(s):

- exc: The CpyException to raise. Do not use the type enum
- CpyException to allow the user to extend the list of
- exceptions with user-defined exception outside of enum
- CpyException.
- filename: File where the exception has been raised
- line: Line where the exception has been raised


```
CapyException_t CapyGetLastExcId(void);
```

Function to get the ID of the last raised exception

Output and side effect(s):

- Return the id of the last raised exception

```
char const* CapyExcToStr(CapyException_t const exc);
```

Function to convert an exception ID to char*. It uses the conversion functions provided via CapyAddExcToStrFun() to convert user defined exception. CapyException exceptions are converted automatically.

Input argument(s):

- exc: The exception ID

Output and side effect(s):

- Return the stringified exception

```
void CapyAddExcToStrFun(char const* (*fun)(CapyException_t));
```

Function to add a function used by TryCatch to convert user-defined exception to a string. The function in argument must return NULL if its argument is not an exception ID it is handling, else a pointer to a string. It is highly recommended to provide conversion functions to cover all the user defined exceptions as it also allows TryCatch to detect conflict between exception IDs.

Input argument(s):

- fun: The conversion function to add

```
void CapySetRaiseStream(FILE* const stream);
```

Set the stream on which to print messages when exception are raised, set it to NULL to turn off messages. (By default it's NULL)

Input argument(s):

- stream: The stream to used

```
FILE* CapyGetRaiseStream(void);
```


Get the stream on which the messages are print when exception are raised.
Output and side effect(s):

- Return the used stream

```
void CpyForwardExc(void);
```

Function to forward the current exception, if any, to the eventual TryCatch block including the CpyForwardExc call. Note that even if an exception is forwarded from inside a parallel section a TryCatch block outside the parallel section may not see it.

```
void CpyCancelExc(void);
```

Function to cancel the current exception if any

92 Unit turtlegraphic.h

TurtleGraphic class.

92.1 Macros

```
#define CAPY_TURTLEGRAPHIC_H
```

92.2 Enumerations

None.

92.3 Typedefs

None.

92.4 Struct CpyTurtleGraphicState

92.4.1 Struct CpyTurtleGraphicState's properties

```
CpyVec step;
```


Step vector

```
CapyVec pos;
```

Position

92.4.2 Struct CapyTurtleGraphicState's methods

```
void (*destruct)(void);
```

Destructor

92.5 Struct CapyTurtleGraphic

92.5.1 Struct CapyTurtleGraphic's properties

```
CapyTurtleGraphicState state;
```

Current state (initially, stepLength=1, theta=0);

```
CapyTurtleGraphicStack* stack;
```

List of stacked states

```
CapyImg* img;
```

Image on which the turtle draws (default: null)

```
CapyColorData color;
```

Color of the line (default: white)

92.5.2 Struct CapyTurtleGraphic's methods

```
void (*destruct)(void);
```

Destructor

```
void (*forward)(uint64_t const step);
```

Forward command

Input argument(s):

- step: the turtle moves forward 'step' steps while drawing a line

```
void (*move)(uint64_t const step);
```

Move command

Input argument(s):

- step: the turtle moves forward 'step' steps without drawing a line

```
void (*turn)(double const theta);
```

Turn command

Input argument(s):

- theta: 'theta' is added to the turtle direction

```
void (*resize)(double const scale);
```

Resize command

Input argument(s):

- scale: the turtle step distance is multiplied by 'scale'

```
void (*push)(void);
```

Push command

Output and side effect(s):

- The current state is added on the top of the stack

```
void (*pop)(void);
```

Pop command

Output and side effect(s):

- The state on the top of the stack is removed and becomes the current
- state.

92.6 Functions

```
CapyTurtleGraphic CapyTurtleGraphicCreate(void);
```

Create a CapyTurtleGraphic

Output and side effect(s):

- Return a CapyTurtleGraphic

```
CapyTurtleGraphic* CapyTurtleGraphicAlloc(void);
```

Allocate memory for a new CapyTurtleGraphic and create it

Output and side effect(s):

- Return a CapyTurtleGraphic

Exception(s):

- May raise CapyExc_MallocFailed.

```
void CapyTurtleGraphicFree(CapyTurtleGraphic** const that);
```

Free the memory used by a CapyTurtleGraphic* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CapyTurtleGraphic to free

93 Unit verletintegration.h

VerletIntegration implementation.

93.1 Macros

```
#define CAPY_VERLETINTEGRATION_H
```

93.2 Enumerations

None.

93.3 Typedefs

None.

93.4 Struct CapyVerletIntegration

93.4.1 Struct CapyVerletIntegration's properties

```
CapyMathFun* derivative;
```

Derivative function $F()$. $F(t, X, X') = X''$ The derivative should not rely on X' (only on t and X)

```
size_t dimVar;
```

Number of variables (automatically set based on derivation.nbInput and order)

```
CapyVec initVal;
```

Initial values $[t_0, X_0, X'_0]$ (default: all 0)

```
CapyVec vecs[2];
```

Temporary vectors for calculation

```
double deltaT;
```

Discretise step Δt (default: $1e-3$)

93.4.2 Struct CapyVerletIntegration's methods

```
void (*eval)(
    double const t,
    double* const vals);
```

Approximate $X(t)$ using Verlet integration

Input argument(s):

- t : value for which we want to approximate $X(t)$
- $vals$: values $[t_0, X(t_0), X'(t_0)]$

Output and side effect(s):

- Update 'vals' with the approximated values at 't'

```
void (*step)(double* const vals);
```

Run one step

Input argument(s):

- vals: values [t0, X(t0), X'(t0)]

Output and side effect(s):

- 'vals' is updated with the result of one step of Verlet integration

```
void (*destruct)(void);
```

Destructor

93.5 Functions

```
CapyVerletIntegration CapyVerletIntegrationCreate(
    CapyMathFun* const derivative);
```

Create a CapyVerletIntegration

Input argument(s):

- derivative: the derivative function, should not rely on X' (only on t and X)

Output and side effect(s):

- Return a CapyVerletIntegration

```
CapyVerletIntegration* CapyVerletIntegrationAlloc(
    CapyMathFun* const derivative);
```

Allocate memory for a new CapyVerletIntegration and create it

Input argument(s):

- derivative: the derivative function, should not rely on X' (only on t and X)

- X)

Output and side effect(s):

- Return a CpyVerletIntegration

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyVerletIntegrationFree(CpyVerletIntegration** const that);
```

Free the memory used by a CpyVerletIntegration* and reset '*that' to NULL
Input argument(s):

- that: a pointer to the CpyVerletIntegration to free

94 Unit voting.h

Voting systems implementation.

94.1 Macros

```
#define CAPY_VOTING_H
```

94.2 Enumerations

None.

94.3 Typedefs

None.

94.4 Struct CpyRankedChoiceVote

94.4.1 Struct CpyRankedChoiceVote's properties

```
uint8_t choice;
```


Current choice (default: 0)

```
CapyPad(uint8_t, choice);
```

```
uint8_t* rankings;
```

Ranking of choices (initialised to 0, 1, 2, ...)

94.4.2 Struct CapyRankedChoiceVote's methods

None.

94.5 Struct CapyRankedChoiceVotingResult

94.5.1 Struct CapyRankedChoiceVotingResult's properties

```
size_t nbVotes[256];
```

Number of votes for each choice before being eliminated (so the winner(s) can be determined by decreasing number of votes).

94.5.2 Struct CapyRankedChoiceVotingResult's methods

None.

94.6 Struct CapyRankedChoiceVoting

94.6.1 Struct CapyRankedChoiceVoting's properties

```
uint8_t nbChoice;
```

Number of choice

```
CapyPad(uint8_t, nbChoice);
```

```
size_t nbVoter;
```

Number of voter

```
CapyRankedChoiceVote* votes;
```

Voter's choices

94.6.2 Struct CpyRankedChoiceVoting's methods

```
void (*destruct)(void);
```

Destructor

```
void (*setRanking)(
    size_t const iVoter,
    uint8_t const iRanking,
    uint8_t const iChoice);
```

Set a voter's ranking for a given choice

Input argument(s):

- iVoter: id of the voter
- iRanking: updated ranking
- iChoice: id of the choice for the given voter and ranking

Output and side effect(s):

- The choice for the given rank and voter is updated

```
CpyRankedChoiceVotingResult (*getResult)(void);
```

Get the result of election

Output and side effect(s):

- Check the integrity of the votes and if they are correct return the
- result, else raise CpyExc_InvalidParameters. The voters' choice is
- updated.

94.7 Functions

```
CpyRankedChoiceVoting CpyRankedChoiceVotingCreate(
    uint8_t const nbChoice,
    size_t const nbVoter);
```

Create a CpyRankedChoiceVoting

Input argument(s):

- nbChoice: number of choice

- nbVoter: number of voter

Output and side effect(s):

- Return a CpyRankedChoiceVoting.

```
CpyRankedChoiceVoting* CpyRankedChoiceVotingAlloc(
    uint8_t const nbChoice,
    size_t const nbVoter);
```

Allocate memory for a new CpyRankedChoiceVoting and create it

Input argument(s):

- nbChoice: number of choice
- nbVoter: number of voter

Output and side effect(s):

- Return a CpyRankedChoiceVoting

Exception(s):

- May raise CpyExc_MallocFailed.

```
void CpyRankedChoiceVotingFree(CpyRankedChoiceVoting** const that);
```

Free the memory used by a CpyRankedChoiceVoting* and reset '*that' to NULL

Input argument(s):

- that: a pointer to the CpyRankedChoiceVoting to free

95 Unit x11display.h

Class to display graphics on screen, using X11.

95.1 Macros

```
#define CAPY_X11_DISPLAY
```


95.2 Enumerations

```
typedef enum CapyX11Evt {  
    capyX11Evt_noEvent,  
    capyX11Evt_press,  
    capyX11Evt_release,  
} CapyX11Evt;
```

Types of events

```
typedef enum CapyX11Src {  
    capyX11Evt_key,  
    capyX11Evt_mouse,  
} CapyX11Src;
```

Types of source of events

95.3 Typedefs

```
typedef struct CapyX11Display CapyX11Display;
```

X11Display opaque structure

```
typedef CapyImgPos_t CapyX11Dims_t;
```

Type for the dimensions of a CapyX11Display

```
typedef CapyImgPos CapyX11DisplayPos;
```

CapyX11DisplayPos structure returned by CapyX11DisplayGetPointerPos()

```
typedef uint8_t CapyX11RGB_t;
```

Type of one channel in pixel's RGB data

95.4 Struct CapyX11DisplayEvt

95.4.1 Struct CapyX11DisplayEvt's properties

```
CapyX11Evt type;
```

0: no event, 1: press, 2: release


```
CapyX11Src src;
```

0: key, 1: mouse

```
CapyChronoTime_t time;
```

Time of the event in millisecond (comes from XLib, not sure relative to what)

```
size_t state;
```

key or button mask, 'or' combination of: NoModMask, Button1Mask, Button2Mask, Button3Mask, Button4Mask, Button5Mask, ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, and Mod5Mask

```
size_t val;
```

key/button value, one of: for key: NoSymbol or the XK.... macro (see complete list here: /usr/include/X11/keysymdef.h) which map to ASCII (XK_a == 'a') for button: Button1, Button2, Button3, Button4, Button5

```
CapyX11DisplayPos pos;
```

Coordinates of the pointer at the time of event

95.4.2 Struct CapyX11DisplayEvt's methods

None.

95.5 Struct CapyX11RGB

95.5.1 Struct CapyX11RGB's properties

```
union {
```

Position (0,0 is the top-left corner, x toward right, y toward bottom)

```
CapyX11RGB_t vals[3];
```

```
struct __attribute__((packed)) { CapyX11RGB_t r, g, b;};
```

```
};
```


95.5.2 Struct CopyX11RGB's methods

None.