cnn

Generated by Doxygen 1.9.5

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 digits Struct Reference

data structure holding the convolution filters to detect the digits.

`#include <digits.h>`

Public Attributes

- char * layer1Prefix
- char * layer2Prefix
- Layer * layer1
- Layer * layer2

3.1.1 Detailed Description

data structure holding the convolution filters to detect the digits.

3.1.2 Member Data Documentation

3.1.2.1 layer1

```
Layer* digits::layer1
```

3.1.2.2 layer1Prefix

```
char* digits::layer1Prefix
```

3.1.2.3 layer2

```
Layer* digits::layer2
```

3.1.2.4 layer2Prefix

```
char* digits::layer2Prefix
```

The documentation for this struct was generated from the following file:

- digits.h

3.2 filter Struct Reference

Data structure to store a filter.

```
#include <filter.h>
```

Public Attributes

- Img * img
- long int weight
- long int threshold
- long int maxVal
- int percent
- char * data

3.2.1 Detailed Description

Data structure to store a filter.

3.2.2 Member Data Documentation

3.2.2.1 data

```
char* filter::data
```

data associated with this filter if any.

3.2.2.2 img

```
Img* filter::img
```

image used for the filter

3.2.2.3 maxVal

```
long int filter::maxVal
```

maximum value when the image is used as a filter

3.2.2.4 percent

```
int filter::percent
```

percentage used to compute threshold from maxVal

3.2.2.5 threshold

```
long int filter::threshold
```

threshold to trigger white pixel, used in filters only

3.2.2.6 weight

```
long int filter::weight
```

weight of white pixels (sum of all pixel values)

The documentation for this struct was generated from the following file:

- filter.h

3.3 filterfam Struct Reference

A structure to store a collection (also called familly) of filters.

```
#include <filterfam.h>
```

Public Attributes

- int count
- Filter ** filters

3.3.1 Detailed Description

A structure to store a collection (also called familly) of filters.

3.3.2 Member Data Documentation

3.3.2.1 count

```
int filterfam::count
```

number of images in the familly

3.3.2.2 filters

```
Filter** filterfam::filters
```

filter data

The documentation for this struct was generated from the following file:

- filterfam.h

3.4 img Struct Reference

A structure to store greyscale image data.

```
#include <img.h>
```

Public Attributes

- int width
- int height
- unsigned char * data

3.4.1 Detailed Description

A structure to store greyscale image data.

3.4.2 Member Data Documentation

3.4.2.1 data

```
unsigned char* img::data
```

This is an allocated buffer of all pixels in the picture. Its size should be width times height.

3.4.2.2 height

```
int img::height
```

number of vertical pixels

3.4.2.3 width

```
int img::width
```

number of horizontal pixels

The documentation for this struct was generated from the following file:

- img.h

3.5 imgfam Struct Reference

A structure to store a collection (also called familly) of images.

```
#include <imgfam.h>
```

Public Attributes

- int count
- Img ** imgs

3.5.1 Detailed Description

A structure to store a collection (also called familly) of images.

3.5.2 Member Data Documentation

3.5.2.1 count

```
int imgfam::count
```

number of images in the familly

3.5.2.2 imgs

```
Img** imgfam::imgs
```

image data

The documentation for this struct was generated from the following file:

- imgfam.h

3.6 jpegerrmgr Struct Reference

Public Attributes

- struct jpeg_error_mgr pub
- jmp_buf setjmp_buffer

3.6.1 Detailed Description

Data structure errors when manipulating jpeg.

3.6.2 Member Data Documentation

3.6.2.1 pub

```
struct jpeg_error_mgr jpegerrmgr::pub
```

"public" fields

3.6.2.2 setjmp_buffer

```
jmp_buf jpegerrmgr::setjmp_buffer
```

for return to caller

The documentation for this struct was generated from the following file:

- img.c

3.7 layer Struct Reference

```
#include <layer.h>
```

Public Attributes

- char * name
- char * convFilterLoc
- FilterFam * convFilter
- int downSamplePoolSize
- int downSampleStride

3.7.1 Detailed Description

a layer in a convolution neural network.

3.7.2 Member Data Documentation

3.7.2.1 convFilter

```
FilterFam* layer::convFilter
```

Filters used

3.7.2.2 convFilterLoc

```
char* layer::convFilterLoc
```

prefix of the path name where filters are stored.

3.7.2.3 downSamplePoolSize

```
int layer::downSamplePoolSize
```

value of pool size

3.7.2.4 downSampleStride

```
int layer::downSampleStride
```

stride value

3.7.2.5 name

```
char* layer::name
```

name of the layer

The documentation for this struct was generated from the following file:

- layer.h

Chapter 4

File Documentation

4.1 digits.c File Reference

```
#include <libgen.h>
#include <unistd.h>
#include "imgfam.h"
#include "img.h"
#include "filterfam.h"
#include "digits.h"
#include "layer.h"
```

Functions

- Digits * newDigits (char *layerPrefix)

  allocates a memory for a new set of filters to detect digits
- void deleteDigits (Digits *d)
- int digitsGetDefaultLayerPrefixSize ()

  Size of the buffer to allocate the path to store layer location to parse digits.
- void digitsGetDefaultLayerPrefix (char *layerPrefix)

  Writes the directory to store layers to parse digits.
- void digitsGenerateLayer1 (char *layerPrefix)

  Layer 1 is made of 5x5 filters with a rotating white bar on a black backound.
- void digitsTestLayer1 (char *layerPrefix)
- void digitsGenerateLayer2 (char *layerPrefix)

  Generate layer 2 to recognized digit n.
- void digitsGenerateImageOfFont (char *directory, int numFonts)

  Creates images of characters 1 to 9.
- void generateLayer3 ()
- void testFilterForDigit (char *font)
- void generateTestData ()
- void digitsUsage (FILE *f, char *name)

  Tells how to use this program.
- int digitsMain (int argc, char **argv)

  main program for the digit executable. Parses options from the command line.

4.1.1 Function Documentation

4.1.1.1 deleteDigits()

```
void deleteDigits (
            Digits * d )
```

Deletes an allocated Digit structure previously allocated with newDigits..

Parameters

d	the data structure to free.

See also

newDigits

```
43                          {
44     free(d->layer1Prefix);
45     free(d->layer2Prefix);
46     if (d->layer1)
47         deleteLayer(d->layer1);
48     if (d->layer2)
49         deleteLayer(d->layer2);
50     memset(d,0,sizeof(Digits));
51     free(d);
52 }
```

References deleteLayer(), digits::layer1, digits::layer1Prefix, digits::layer2, and digits::layer2Prefix.

4.1.1.2 digitsGenerateImageOfFont()

```
void digitsGenerateImageOfFont (
            char * directory,
            int numFonts )
```

Creates images of characters 1 to 9.

This function creates a bash script which calls the 'convert' program to: list all fonts on the system and generate 28x28 pictures of each digits 1 to 9 for the first numFonts fonts found.

Parameters

directory	where to create the images
numFonts	number of fonts to create.

```
232                                                     {
233     char * tmpFile = (char*)malloc(strlen(directory)+40);
234     snprintf(tmpFile,strlen(directory)+40,
235              "%s/generate_png_from_font.sh",directory);
236     FILE * f = fopen(tmpFile,"w");
237     HERE(tmpFile);
238     fprintf(f,"#/bin/sh\n");
239     fprintf(f,"# this file has been automatically generated\n");
```

```
240     fprintf(f,"# from c code %s:%d.\n",__FILE__,__LINE__);
241     fprintf(f,"set -e\n");
242     fprintf(f,"cd `dirname $0`\n");
243     fprintf(f,"fonts=(`convert -list font  | grep Font: | grep -vi emoji| sed -e s/Font:// | tr -d '
        ' `)\n");
244     fprintf(f,"fmax=${#fonts[@]}\n");
245     fprintf(f,"fmax=$[fmax-1]\n");
246     fprintf(f,"if [ $fmax -gt %d ]; then\n",numFonts);
247     fprintf(f,"    fmax=%d\n",numFonts);
248     fprintf(f,"fi\n");
249     fprintf(f,"for f in `seq 0 $fmax`; do\n");
250     fprintf(f,"    for i in `seq 1 9`; do\n");
251     fprintf(f,"        convert -background white -fill black -font ${fonts[$f]} -size 28x28 -gravity
        center \"caption:$i\" $[f+1]_$i.png\n");
252     fprintf(f,"    done\n");
253     fprintf(f,"done\n");
254     fclose(f);
255     char cmd[99];
256     snprintf(cmd,99,"bash %s",tmpFile);
257     HERE(cmd);
258     if (system(cmd)) {
259         ERROR("Command failed, maybe 'convert' not found: ",cmd);
260     }
261 }
```

References ERROR, and HERE.

4.1.1.3 digitsGenerateLayer1()

```
void digitsGenerateLayer1 (
              char * layerPrefix )
```

Layer 1 is made of 5x5 filters with a rotating white bar on a black backound.

Parameters

layerPrefix	where data should be written, generally the value returned by digitsGetDefaultLayerPrefix

```
83                                              {
84     // create 6 images 15x15 images with a rotating bar
85     Img * allBars[6];
86     allBars[0]=newImgVerticalBar(15,3);
87     for (int i=1;i<6;++i)
88         allBars[i]=imgRotate(allBars[0],30*i);
89     // make each image 5x5
90     for (int i=0;i<6;++i) {
91         Img * tmp = allBars[i];
92         allBars[i]=imgDownSampleAvg(tmp,3,3);
93         deleteImg(tmp);
94     }
95     // creates corresponding filter familly
96     FilterFam * filterFamilly = newFilterFam(6);
97     for (int i=0;i<6;++i) {
98         filterFamSetFilter(filterFamilly,
99                            i,
100                            newFilter(imgInvert(allBars[i]),80));
101         // make sure they all have the same weight
102         filterSetWeight(filterFamilly->filters[i],1300);
103         HERE("filter weight");
104         HERED((int)filterFamilly->filters[i]->weight);
105         deleteImg(allBars[i]);
106     }
107     // save the familly on the disk
108     char * s = stringAdd(layerPrefix,"_1");
109     filterFamWrite(filterFamilly,s);
110     // release memory
111     free(s);
112     deleteFilterFam(filterFamilly);
113 }
```

References deleteFilterFam(), deleteImg(), filterFamSetFilter(), filterFamWrite(), filterfam::filters, filterSetWeight(), HERE, HERED, imgDownSampleAvg(), imgInvert(), imgRotate(), newFilter(), newFilterFam(), newImgVerticalBar(), stringAdd(), and filter::weight.

4.1.1.4 digitsGenerateLayer2()

```
void digitsGenerateLayer2 (
            char * layerPrefix )
```

Generate layer 2 to recognized digit n.

Parameters

n	digit to recognize

```
196                                                 {
197     Digits * d = newDigits(layerPrefix);
198     if (d->layer1==NULL) {
199         ERROR("Generate layer 1 before generating layer 2.","");
200     }
201     for (int n=1;n<=9;++n) {
202         char s[99];
203         snprintf(s,99,"%s/digits/reference_%d.png",CFG_DATAROOTDIR,n);
204         if (access(s, F_OK) != 0) {
205             ERROR("File not found: ",s);
206         }
207         Img * tmpimg = newImgRead(s);
208         Img * img = imgInvert(tmpimg);
209         deleteImg(tmpimg);
210         ImgFam * firstLayerOutput =
211             layerPassImg(d->layer1,img);
212         snprintf(s,99,"_layer_%d",n);
213         imgFamWrite(firstLayerOutput,s);
214 
215         // free allocated memory
216         deleteImg(img);
217         deleteImgFam(firstLayerOutput);
218     }
219     deleteDigits(d);
220 }
```

References deleteDigits(), deleteImg(), deleteImgFam(), ERROR, imgFamWrite(), imgInvert(), digits::layer1, layerPassImg(), newDigits(), and newImgRead().

4.1.1.5 digitsGetDefaultLayerPrefix()

```
void digitsGetDefaultLayerPrefix (
            char * layerPrefix )
```

Writes the directory to store layers to parse digits.

Parameters

layerPrefix	is a buffer of at least digitsGetDefaultLayerPrefixSize size to store the path where layers are stored.

See also

digitsGetDefaultLayerPrefixSize

```
71                                                 {
72     snprintf(layerPrefix,digitsGetDefaultLayerPrefixSize(),
73             "%s/digits/layer",CFG_DATAROOTDIR);
```

```
74 }
```

References digitsGetDefaultLayerPrefixSize().

4.1.1.6 digitsGetDefaultLayerPrefixSize()

```
int digitsGetDefaultLayerPrefixSize ( )
```

Size of the buffer to allocate the path to store layer location to parse digits.

Returns

size of the buffer to allocate.

See also

digitsGetDefaultLayerPrefix

```
61                                      {
62     return strlen("/digits/layer") + strlen(CFG_DATAROOTDIR)+1;
63 }
```

4.1.1.7 digitsMain()

```
int digitsMain (
              int argc,
              char ** argv )
```

main program for the digit executable. Parses options from the command line.

Parameters

argc	number of arguments given
argv	value of arguments.

Returns

0 if no error occurs.

```
457                                     {
458     int i=1;
459     char layerPrefix [digitsGetDefaultLayerPrefixSize()];
460     digitsGetDefaultLayerPrefix(layerPrefix);
461     while (i<argc) {
462         if (strcmp(argv[i],"--create")==0 ||
463             strcmp(argv[i],"-c")==0 ) {
464             ++i;
465             if (i>=argc) {
466                 digitsUsage(stderr,argv[0]);
467                 ERROR("-c|--create expects the number of layer to generate","");
468             }
469             int l = atoi(argv[i]);
470             if (l<1 || l>3) {
471                 digitsUsage(stderr,argv[0]);
```

```
472                 ERROR("-c|--create 1, 2 or 3 as argument","");
473             }
474             if (l==1) {
475                 digitsGenerateLayer1(layerPrefix);
476             } else if (l==2) {
477                 digitsGenerateLayer2(layerPrefix);
478             } else {
479                 generateLayer3();
480             }
481         } else if (strcmp(argv[i],"--test")==0 ||
482                    strcmp(argv[i],"-t")==0 ) {
483             ++i;
484             if (i>=argc) {
485                 digitsUsage(stderr,argv[0]);
486                 ERROR("-c|--create expects the number of layer to generate","");
487             }
488             int t=atoi(argv[i]);
489             if (t<1 || t>3) {
490                 digitsUsage(stderr,argv[0]);
491                 ERROR("-t|--test 1, 2 or 3 as argument","");
492             }
493             if (t==1) {
494                 digitsTestLayer1(layerPrefix);
495             } else if (t==2) {
496
497             } else if (t==3) {
498                generateTestData();
499             }
500         } else if (strcmp(argv[i],"--help")==0 ||
501                    strcmp(argv[i],"-h")==0 ) {
502             digitsUsage(stdout,argv[0]);
503             exit(0);
504         } else {
505             digitsUsage(stderr,argv[0]);
506             ERROR("unknown option: ",argv[i]);
507         }
508         ++i;
509     }
510     return 0;
511 }
```

References digitsGenerateLayer1(), digitsGenerateLayer2(), digitsGetDefaultLayerPrefix(), digitsGetDefaultLayerPrefixSize(), digitsTestLayer1(), digitsUsage(), ERROR, generateLayer3(), and generateTestData().

4.1.1.8 digitsTestLayer1()

```
void digitsTestLayer1 (
             char * layerPrefix )
115                                          {
116     // create 6 images 30x30 images with a rotating bar
117     Img * testImg[6];
118     Img * base = newImgVerticalBar(30,2);
119     for (int i=0;i<6;++i) {
120         Img * tmp = imgRotate(base,30*i);
121         testImg[i]=imgInvert(tmp);
122         deleteImg(tmp);
123         char s[99];
124         snprintf(s,99,"test_digits_layer_1_%d.png",i);
125         imgWrite(testImg[i],s);
126     }
127     deleteImg(base);
128
129     // read existing filters
130     Digits * d = newDigits(layerPrefix);
131     if (d->layer1==NULL) {
132         ERROR("Layer 1 filters not found, did you generate them?","");
133     }
134
135     for (int i=0;i<6;++i) {
136         // creates a familly of 6 26x26 images
137         ImgFam * firstLayerOutput = layerPassImg(d->layer1,testImg[i]);
138         // verify that the maximum output for a bar with
139         // and angle i*30 is of the i-th image.
140         int maxIndex=-1;
141         int maxVal=-1;
142         for (int j=0;j<6;++j) {
143             int v=imgGetWeight(firstLayerOutput->imgs[j]);
144             if (v>maxVal) {
```

```
145                 maxVal=v;
146                 maxIndex=j;
147             }
148         }
149         if (maxIndex!=i) {
150             ERROR("Wrong index.","");
151         }
152         char s[99];
153         snprintf(s,99,"test_digits_l1out_%d",i);
154         imgFamWrite(firstLayerOutput,s);
155     }
156 
157 
158     // create 6 images 30x30 images with a rotating bar
159     // of 30*i+15 degrees
160     Img * testImgPlus15[6];
161     base = newImgVerticalBar(30,3);
162     for (int i=0;i<6;++i) {
163         Img * tmp = imgRotate(base,15+30*i);
164         testImgPlus15[i]=imgInvert(tmp);
165         deleteImg(tmp);
166         char s[99];
167         snprintf(s,99,"test_digits_layer_1_15_%d.png",i);
168         imgWrite(testImgPlus15[i],s);
169     }
170     deleteImg(base);
171 
172     // pass the 30x30 images in the convolution filter
173     for (int i=0;i<6;++i) {
174         // creates a familly of 6 26x26 images
175         ImgFam * firstLayerOutput =
176             layerPassImg(d->layer1,testImgPlus15[i]);
177         char s[99];
178         snprintf(s,99,"test_digits_l1out_15_%d",i);
179         imgFamWrite(firstLayerOutput,s);
180 
181 
182         printf("%d degrees\n",30*i+15);
183         int v[6];
184         for (int j=0;j<6;++j) {
185             v[j]=imgGetWeight(firstLayerOutput->imgs[j]);
186             printf("%4d ",v[j]);
187         }
188         printf("\n");
189     }
190 }
```

References deleteImg(), ERROR, imgFamWrite(), imgGetWeight(), imgInvert(), imgRotate(), imgfam::imgs, imgWrite(), digits::layer1, layerPassImg(), newDigits(), and newImgVerticalBar().

4.1.1.9 digitsUsage()

```
void digitsUsage (
             FILE * f,
             char * name )
```

Tells how to use this program.

Parameters

f	where to write the info (stdout or stderr).
name	the value of argv[0].

```
435                                 {
436     char * bname=basename(name);
437     fprintf(f,"%s usage:\n", bname);
438     fprintf(f,"    %s [ options ]\n", bname);
439     fprintf(f,"Where option is one of:\n");
440     fprintf(f,"    [-c|--create] <n>:\n");
441     fprintf(f,"         creates the filters to detect digits for layer <n> \n");
442     fprintf(f,"         in directory:\n");
443     fprintf(f,"             %s/digits\n",CFG_DATAROOTDIR);
```

```
444     fprintf(f,"    [-t|--test] <n>:\n");
445     fprintf(f,"         tests the filters for digits.\n");
446     fprintf(f,"    [-h|--help] :\n");
447     fprintf(f,"         display this help message and exits.\n");
448 }
```

4.1.1.10 generateLayer3()

```
void generateLayer3 ( )
263                     {
264     char tmpDirName[60];
265     //memset(tmpDirName,0,60);
266     createTmpDir(tmpDirName,15);
267     int numFonts=30;
268     digitsGenerateImageOfFont(tmpDirName,numFonts);
269     FILE * layer3File = fopen("l3.c","w");
270     for (int f=1;f<=numFonts;++f) {
271         char font[80];
272         snprintf(font,80,"%s/%d",tmpDirName,f);
273 
274         char s[99];
275         snprintf(s,99,"%s/digits/layer_1",CFG_DATAROOTDIR);
276         FilterFam * firstLevelFilters=filterFamRead(s);
277 
278         FilterFam * secondLevelFilters[10];
279         for (int i=1;i<10;++i) {
280             char baseName[99];
281             snprintf(baseName,99,"%s/digits/layer_2_Digit%d",CFG_DATAROOTDIR,i);
282             secondLevelFilters[i] = filterFamRead(baseName);
283         }
284         HERE("+++");
285         HERE(font);
286         unsigned char ** outputFromLevel2 = (unsigned char**)malloc(sizeof(char*)*11);
287 
288         for (int digit=0;digit<10;++digit) {
289             outputFromLevel2[digit]=(unsigned char*)malloc(sizeof(char)*11);
290         }
291         for (int digit=1;digit<10;++digit) {
292             char s[99];
293             snprintf(s,99,"%s_%d.png",font,digit);
294             Img * tmpimg = newImgRead(s);
295             Img * img = imgInvert(tmpimg);
296             deleteImg(tmpimg);
297 
298             ImgFam * firstLayerOutput =
299                 filterFamApplyConvolution(firstLevelFilters,img);
300 
301             ImgFam * firstLayerMaxPoolOutput=imgFamDownSampleMax(firstLayerOutput,2,2);
302             //ImgFam * avgPool=imgFamDownSampleMax(firstLayerMaxPoolOutput,2,2);
303             ImgFam * avgPool = firstLayerMaxPoolOutput;
304             HERE("___");
305             int maxIndex=-1;
306             int maxVal=-1;
307             for (int i=1;i<10;++i) {
308                 Img*singlePixelImg=imgFamScalar(avgPool,secondLevelFilters[i]);
309 
310                 //HERED(singlePixelImg->data[0]);
311                 outputFromLevel2[digit][i]=singlePixelImg->data[0];
312                 printf(" %d\n",singlePixelImg->data[0]);
313                 if (singlePixelImg->data[0]>maxVal) {
314                     maxVal=singlePixelImg->data[0];
315                     maxIndex=i;
316                 }
317             }
318             if (maxIndex!=digit) {
319                 fprintf(stderr,
320                         "\033[31m%s:%d: ||| wrong index %d should be %d\033[m\n",
321                         __FILE__,
322                         __LINE__,
323                         maxIndex,
324                         digit);
325             } else {
326                 HERE("OK");
327             }
328         }
329         for (int digit=1;digit<10;++digit) {
330             fprintf(layer3File,"unsigned char tmp_arr_%d_%d[10]={0,",f,digit);
331             for (int i=1;i<10;++i) {
332                 fprintf(layer3File,"%d%s",outputFromLevel2[digit][i],i==9?"":",");
333             }
```

```
334             fprintf(layer3File,"};\n");
335         }
336     }
337
338     fprintf(layer3File,"unsigned char **digits[%d];\n",numFonts+1);
339     fprintf(layer3File,"unsigned char** init_digits() {\n");
340     fprintf(layer3File,"    unsigned char** answer = (unsigned char**)
      malloc(sizeof(char**)*%d*10);\n",numFonts+1);
341     for (int f=1;f<=numFonts;++f) {
342         fprintf(layer3File,
343                 "    digits[%d]=answer+10*%d;\n",f,f);
344         fprintf(layer3File,
345                 "    digits[%d][0]=NULL;\n",f);
346         for (int digit=1;digit<10;++digit) {
347             fprintf(layer3File,
348                     // digits[a][b][c] char
349                     // digits[a][b]    char*
350                     // digits[a]       char**
351                     // digits          char***
352                     // answer[0]=
353                     "    digits[%d][%d]=tmp_arr_%d_%d;\n",
354                     f,digit,
355                     f,digit);
356         }
357     }
358     fprintf(layer3File,"    return answer;\n");
359     fprintf(layer3File,"}\n");
360     fclose(layer3File);
361     HERE("13.c written.");
362 }
```

References createTmpDir(), img::data, deleteImg(), digitsGenerateImageOfFont(), filterFamApplyConvolution(), filterFamRead(), HERE, imgFamDownSampleMax(), imgFamScalar(), imgInvert(), and newImgRead().

4.1.1.11 generateTestData()

```
void generateTestData ( )
417                         {
418     char tmpDirName[60];
419     //memset(tmpDirName,0,60);
420     createTmpDir(tmpDirName,15);
421     int numFonts=30;
422     digitsGenerateImageOfFont(tmpDirName,numFonts);
423     for (int i=1;i<=numFonts;++i) {
424         char fontpath[99];
425         snprintf(fontpath,99,"%s/%d",tmpDirName,i);
426         testFilterForDigit(fontpath);
427     }
428 }
```

References createTmpDir(), digitsGenerateImageOfFont(), and testFilterForDigit().

4.1.1.12 newDigits()

```
Digits * newDigits (
             char * layerPrefix )
```

allocates a memory for a new set of filters to detect digits

Data structure should be freed using deleteDigits.

Returns

a newly allocated Digit structure.

See also

deleteDigits

```
17                                      {
18     Digits * answer = (Digits*)malloc(sizeof(Digits));
19     char layer1location[strlen(layerPrefix)+10];
20     snprintf(layer1location,strlen(layerPrefix)+10,"%s_1",layerPrefix);
21     answer->layer1Prefix=stringCopy(layer1location);
22     if (layerCount(layer1location)>0) {
23         answer->layer1 = newLayer("layer1",layer1location,2,2);
24     } else {
25         answer->layer1=NULL;
26     }
27     char layer2location[strlen(layerPrefix)+10];
28     snprintf(layer2location,strlen(layerPrefix)+10,"%s_2",layerPrefix);
29     answer->layer2Prefix=stringCopy(layer2location);
30     if (layerCount(layer2location)>0) {
31         answer->layer2 = newLayer("layer2",layer2location,2,2);
32     } else {
33         answer->layer2 = NULL;
34     }
35     return answer;
36 }
```

References digits::layer1, digits::layer1Prefix, digits::layer2, digits::layer2Prefix, layerCount(), newLayer(), and stringCopy().

4.1.1.13 testFilterForDigit()

```
void testFilterForDigit (
             char * font )
364                                     {
365
366     char s[99];
367     snprintf(s,99,"%s/digits/layer_1",CFG_DATAROOTDIR);
368     FilterFam * firstLevelFilters=filterFamRead(s);
369
370     FilterFam * secondLevelFilters[10];
371     for (int i=1;i<10;++i) {
372         char baseName[99];
373         snprintf(baseName,99,"%s/digits/layer_2_Digit%d",CFG_DATAROOTDIR,i);
374         secondLevelFilters[i] = filterFamRead(baseName);
375     }
376     HERE("+++");
377     HERE(font);
378     for (int digit=1;digit<10;++digit) {
379         char s[99];
380         snprintf(s,99,"%s_%d.png",font,digit);
381         Img * tmpimg = newImgRead(s);
382         Img * img = imgInvert(tmpimg);
383         deleteImg(tmpimg);
384
385         ImgFam * firstLayerOutput =
386             filterFamApplyConvolution(firstLevelFilters,img);
387
388         ImgFam * firstLayerMaxPoolOutput=imgFamDownSampleMax(firstLayerOutput,2,2);
389         //ImgFam * avgPool=imgFamDownSampleMax(firstLayerMaxPoolOutput,2,2);
390         ImgFam * avgPool=firstLayerMaxPoolOutput;
391         HERE("___");
392         int maxIndex=-1;
393         int maxVal=-1;
394         for (int i=1;i<10;++i) {
395             Img*singlePixelImg=imgFamScalar(avgPool,secondLevelFilters[i]);
396             //HERED(singlePixelImg->data[0]);
397             printf(" %d\n",singlePixelImg->data[0]);
398             if (singlePixelImg->data[0]>maxVal) {
399                 maxVal=singlePixelImg->data[0];
400                 maxIndex=i;
401             }
402         }
403         if (maxIndex!=digit) {
404             fprintf(stderr,
405                     "\033[31m%s:%d: ||| wrong index %d should be %d\033[m\n",
406                     __FILE__,
407                     __LINE__,
408                     maxIndex,
409                     digit);
```

```
410         } else {
411             HERE("OK");
412         }
413     }
414 }
```

References img::data, deleteImg(), filterFamApplyConvolution(), filterFamRead(), HERE, imgFamDownSampleMax(), imgFamScalar(), imgInvert(), and newImgRead().

4.2 digits.h File Reference

```
#include "util.h"
```

Classes

- struct digits

  data structure holding the convolution filters to detect the digits.

Typedefs

- typedef struct layer Layer
- typedef struct digits Digits

  Short cut for struct digits.

Functions

- Digits * newDigits (char *)

  allocates a memory for a new set of filters to detect digits
- void deleteDigits (Digits *)
- int digitsGetDefaultLayerPrefixSize ()

  Size of the buffer to allocate the path to store layer location to parse digits.
- void digitsGetDefaultLayerPrefix (char *layerPrefix)

  Writes the directory to store layers to parse digits.
- void digitsGenerateImageOfFont (char *directory, int numFonts)

  Creates images of characters 1 to 9.
- void digitsGenerateLayer1 (char *layerPrefix)

  Layer 1 is made of 5x5 filters with a rotating white bar on a black backound.
- void digitsTestLayer1 (char *layerPrefix)
- int digitsMain (int argc, char **argv)

  main program for the digit executable. Parses options from the command line.

4.2.1 Typedef Documentation

4.2.1.1 Digits

```
typedef struct digits Digits
```

Short cut for struct digits.

4.2.1.2 Layer

```
typedef struct layer Layer
```

4.2.2 Function Documentation

4.2.2.1 deleteDigits()

```
void deleteDigits (
            Digits * d )
```

Deletes an allocated Digit structure previously allocated with newDigits..

Parameters

d	the data structure to free.

See also

newDigits

```
43                          {
44     free(d->layer1Prefix);
45     free(d->layer2Prefix);
46     if (d->layer1)
47         deleteLayer(d->layer1);
48     if (d->layer2)
49         deleteLayer(d->layer2);
50     memset(d,0,sizeof(Digits));
51     free(d);
52 }
```

References deleteLayer(), digits::layer1, digits::layer1Prefix, digits::layer2, and digits::layer2Prefix.

4.2.2.2 digitsGenerateImageOfFont()

```
void digitsGenerateImageOfFont (
            char * directory,
            int numFonts )
```

Creates images of characters 1 to 9.

This function creates a bash script which calls the 'convert' program to: list all fonts on the system and generate 28x28 pictures of each digits 1 to 9 for the first numFonts fonts found.

Parameters

directory	where to create the images
numFonts	number of fonts to create.

```
232                                                                 {
233     char * tmpFile = (char*)malloc(strlen(directory)+40);
234     snprintf(tmpFile,strlen(directory)+40,
235              "%s/generate_png_from_font.sh",directory);
236     FILE * f = fopen(tmpFile,"w");
237     HERE(tmpFile);
238     fprintf(f,"#/bin/sh\n");
239     fprintf(f,"# this file has been automatically generated\n");
240     fprintf(f,"# from c code %s:%d.\n",__FILE__,__LINE__);
241     fprintf(f,"set -e\n");
242     fprintf(f,"cd `dirname $0`\n");
243     fprintf(f,"fonts=(`convert -list font  | grep Font: | grep -vi emoji| sed -e s/Font:// | tr -d '
      ' `)\n");
244     fprintf(f,"fmax=${#fonts[@]}\n");
245     fprintf(f,"fmax=$[fmax-1]\n");
246     fprintf(f,"if [ $fmax -gt %d ]; then\n",numFonts);
247     fprintf(f,"    fmax=%d\n",numFonts);
248     fprintf(f,"fi\n");
249     fprintf(f,"for f in `seq 0 $fmax`; do\n");
250     fprintf(f,"    for i in `seq 1 9`; do\n");
251     fprintf(f,"        convert -background white -fill black -font ${fonts[$f]} -size 28x28 -gravity
      center \"caption:$i\" ${f+1}_$i.png\n");
252     fprintf(f,"    done\n");
253     fprintf(f,"done\n");
254     fclose(f);
255     char cmd[99];
256     snprintf(cmd,99,"bash %s",tmpFile);
257     HERE(cmd);
258     if (system(cmd)) {
259         ERROR("Command failed, maybe 'convert' not found: ",cmd);
260     }
261 }
```

References ERROR, and HERE.

4.2.2.3 digitsGenerateLayer1()

```
void digitsGenerateLayer1 (
            char * layerPrefix )
```

Layer 1 is made of 5x5 filters with a rotating white bar on a black backound.

Parameters

layerPrefix	where data should be written, generally the value returned by digitsGetDefaultLayerPrefix

```
83                                                  {
84     // create 6 images 15x15 images with a rotating bar
85     Img * allBars[6];
86     allBars[0]=newImgVerticalBar(15,3);
87     for (int i=1;i<6;++i)
88         allBars[i]=imgRotate(allBars[0],30*i);
89     // make each image 5x5
90     for (int i=0;i<6;++i) {
91         Img * tmp = allBars[i];
92         allBars[i]=imgDownSampleAvg(tmp,3,3);
93         deleteImg(tmp);
94     }
95     // creates corresponding filter familly
96     FilterFam * filterFamilly = newFilterFam(6);
97     for (int i=0;i<6;++i) {
98         filterFamSetFilter(filterFamilly,
99                            i,
100                            newFilter(imgInvert(allBars[i]),80));
101         // make sure they all have the same weight
102         filterSetWeight(filterFamilly->filters[i],1300);
```

```
103         HERE("filter weight");
104         HERED((int)filterFamilly->filters[i]->weight);
105         deleteImg(allBars[i]);
106     }
107     // save the familly on the disk
108     char * s = stringAdd(layerPrefix,"_l");
109     filterFamWrite(filterFamilly,s);
110     // release memory
111     free(s);
112     deleteFilterFam(filterFamilly);
113 }
```

References deleteFilterFam(), deleteImg(), filterFamSetFilter(), filterFamWrite(), filterfam::filters, filterSetWeight(), HERE, HERED, imgDownSampleAvg(), imgInvert(), imgRotate(), newFilter(), newFilterFam(), newImgVerticalBar(), stringAdd(), and filter::weight.

4.2.2.4 digitsGetDefaultLayerPrefix()

```
void digitsGetDefaultLayerPrefix (
             char * layerPrefix )
```

Writes the directory to store layers to parse digits.

Parameters

layerPrefix	is a buffer of at least digitsGetDefaultLayerPrefixSize size to store the path where layers are stored.

See also

digitsGetDefaultLayerPrefixSize

```
71                                                  {
72     snprintf(layerPrefix,digitsGetDefaultLayerPrefixSize(),
73              "%s/digits/layer",CFG_DATAROOTDIR);
74 }
```

References digitsGetDefaultLayerPrefixSize().

4.2.2.5 digitsGetDefaultLayerPrefixSize()

```
int digitsGetDefaultLayerPrefixSize ( )
```

Size of the buffer to allocate the path to store layer location to parse digits.

Returns

size of the buffer to allocate.

See also

digitsGetDefaultLayerPrefix

```
61                                      {
62     return strlen("/digits/layer") + strlen(CFG_DATAROOTDIR)+1;
63 }
```

4.2.2.6 digitsMain()

```
int digitsMain (
            int argc,
            char ** argv )
```

main program for the digit executable. Parses options from the command line.

Parameters

argc	number of arguments given
argv	value of arguments.

Returns

0 if no error occurs.

```
457                                 {
458     int i=1;
459     char layerPrefix [digitsGetDefaultLayerPrefixSize()];
460     digitsGetDefaultLayerPrefix(layerPrefix);
461     while (i<argc) {
462         if (strcmp(argv[i],"--create")==0 ||
463             strcmp(argv[i],"-c")==0 ) {
464             ++i;
465             if (i>=argc) {
466                 digitsUsage(stderr,argv[0]);
467                 ERROR("-c|--create expects the number of layer to generate","");
468             }
469             int l = atoi(argv[i]);
470             if (l<1 || l>3) {
471                 digitsUsage(stderr,argv[0]);
472                 ERROR("-c|--create 1, 2 or 3 as argument","");
473             }
474             if (l==1) {
475                 digitsGenerateLayer1(layerPrefix);
476             } else if (l==2) {
477                 digitsGenerateLayer2(layerPrefix);
478             } else {
479                 generateLayer3();
480             }
481         } else if (strcmp(argv[i],"--test")==0 ||
482                    strcmp(argv[i],"-t")==0 ) {
483             ++i;
484             if (i>=argc) {
485                 digitsUsage(stderr,argv[0]);
486                 ERROR("-c|--create expects the number of layer to generate","");
487             }
488             int t=atoi(argv[i]);
489             if (t<1 || t>3) {
490                 digitsUsage(stderr,argv[0]);
491                 ERROR("-t|--test 1, 2 or 3 as argument","");
492             }
493             if (t==1) {
494                 digitsTestLayer1(layerPrefix);
495             } else if (t==2) {
496
497             } else if (t==3) {
498                generateTestData();
499             }
500         } else if (strcmp(argv[i],"--help")==0 ||
501                    strcmp(argv[i],"-h")==0 ) {
502             digitsUsage(stdout,argv[0]);
503             exit(0);
504         } else {
505             digitsUsage(stderr,argv[0]);
506             ERROR("unknown option: ",argv[i]);
507         }
508         ++i;
509     }
510     return 0;
511 }
```

References digitsGenerateLayer1(), digitsGenerateLayer2(), digitsGetDefaultLayerPrefix(), digitsGetDefaultLayerPrefixSize(), digitsTestLayer1(), digitsUsage(), ERROR, generateLayer3(), and generateTestData().

4.2.2.7 digitsTestLayer1()

```
void digitsTestLayer1 (
             char * layerPrefix )
115                                          {
116     // create 6 images 30x30 images with a rotating bar
117     Img * testImg[6];
118     Img * base = newImgVerticalBar(30,2);
119     for (int i=0;i<6;++i) {
120         Img * tmp = imgRotate(base,30*i);
121         testImg[i]=imgInvert(tmp);
122         deleteImg(tmp);
123         char s[99];
124         snprintf(s,99,"test_digits_layer_1_%d.png",i);
125         imgWrite(testImg[i],s);
126     }
127     deleteImg(base);
128
129     // read existing filters
130     Digits * d = newDigits(layerPrefix);
131     if (d->layer1==NULL) {
132         ERROR("Layer 1 filters not found, did you generate them?","");
133     }
134
135     for (int i=0;i<6;++i) {
136         // creates a familly of 6 26x26 images
137         ImgFam * firstLayerOutput = layerPassImg(d->layer1,testImg[i]);
138         // verify that the maximum output for a bar with
139         // and angle i*30 is of the i-th image.
140         int maxIndex=-1;
141         int maxVal=-1;
142         for (int j=0;j<6;++j) {
143             int v=imgGetWeight(firstLayerOutput->imgs[j]);
144             if (v>maxVal) {
145                 maxVal=v;
146                 maxIndex=j;
147             }
148         }
149         if (maxIndex!=i) {
150             ERROR("Wrong index.","");
151         }
152         char s[99];
153         snprintf(s,99,"test_digits_l1out_%d",i);
154         imgFamWrite(firstLayerOutput,s);
155     }
156
157
158     // create 6 images 30x30 images with a rotating bar
159     // of 30*i+15 degrees
160     Img * testImgPlus15[6];
161     base = newImgVerticalBar(30,3);
162     for (int i=0;i<6;++i) {
163         Img * tmp = imgRotate(base,15+30*i);
164         testImgPlus15[i]=imgInvert(tmp);
165         deleteImg(tmp);
166         char s[99];
167         snprintf(s,99,"test_digits_layer_1_15_%d.png",i);
168         imgWrite(testImgPlus15[i],s);
169     }
170     deleteImg(base);
171
172     // pass the 30x30 images in the convolution filter
173     for (int i=0;i<6;++i) {
174         // creates a familly of 6 26x26 images
175         ImgFam * firstLayerOutput =
176             layerPassImg(d->layer1,testImgPlus15[i]);
177         char s[99];
178         snprintf(s,99,"test_digits_l1out_15_%d",i);
179         imgFamWrite(firstLayerOutput,s);
180
181
182         printf("%d degrees\n",30*i+15);
183         int v[6];
184         for (int j=0;j<6;++j) {
185             v[j]=imgGetWeight(firstLayerOutput->imgs[j]);
186             printf("%4d ",v[j]);
187         }
188         printf("\n");
189     }
190 }
```

References deleteImg(), ERROR, imgFamWrite(), imgGetWeight(), imgInvert(), imgRotate(), imgfam::imgs, imgWrite(), digits::layer1, layerPassImg(), newDigits(), and newImgVerticalBar().

4.2.2.8 newDigits()

```
Digits * newDigits (
            char * layerPrefix )
```

allocates a memory for a new set of filters to detect digits

Data structure should be freed using deleteDigits.

Returns

a newly allocated Digit structure.

See also

deleteDigits

```
17                                      {
18     Digits * answer = (Digits*)malloc(sizeof(Digits));
19     char layer1location[strlen(layerPrefix)+10];
20     snprintf(layer1location,strlen(layerPrefix)+10,"%s_1",layerPrefix);
21     answer->layer1Prefix=stringCopy(layer1location);
22     if (layerCount(layer1location)>0) {
23         answer->layer1 = newLayer("layer1",layer1location,2,2);
24     } else {
25         answer->layer1=NULL;
26     }
27     char layer2location[strlen(layerPrefix)+10];
28     snprintf(layer2location,strlen(layerPrefix)+10,"%s_2",layerPrefix);
29     answer->layer2Prefix=stringCopy(layer2location);
30     if (layerCount(layer2location)>0) {
31         answer->layer2 = newLayer("layer2",layer2location,2,2);
32     } else {
33         answer->layer2 = NULL;
34     }
35     return answer;
36 }
```

References digits::layer1, digits::layer1Prefix, digits::layer2, digits::layer2Prefix, layerCount(), newLayer(), and stringCopy().

4.3 digits.h

Go to the documentation of this file.

```
1 #ifndef DIGITS_H
2 #define DIGITS_H
3
4 #include "util.h"
5
6 // external types
7 typedef struct layer Layer;
8
13 struct digits {
14     char * layer1Prefix;
15     char * layer2Prefix;
16     Layer * layer1;
17     Layer * layer2;
18 };
22 typedef struct digits Digits;
23
24 Digits * newDigits(char*);
25 void deleteDigits(Digits*);
26 int digitsGetDefaultLayerPrefixSize();
27 void digitsGetDefaultLayerPrefix(char* layerPrefix);
28 void digitsGenerateImageOfFont(char*directory,int numFonts);
29 void digitsGenerateLayer1 (char * layerPrefix);
30 void digitsTestLayer1(char * layerPrefix);
31 int digitsMain(int argc,char**argv);
32
33 #endif
```

4.4 filter.c File Reference

```
#include "filter.h"
#include <math.h>
```

Functions

- Filter * newFilter (Img *img, int percent)

  Allocates space for a new filter.
- void deleteFilter (Filter *f)

  Releases memory allocated for a filter.
- void filterSetData (Filter *f, char *s)

  Copies a string in the data field of a filter.
- void filterSetWeight (Filter *f, int w)

  sets the filter image to a given weight
- void filterUpdateValues (Filter *f)

  Updates weight, threshold and maxVal values in a filter given the image and the percentage threshold should have.
- void filterWrite (Filter *f, char *basename)

  Saves as png files the filter.
- Filter * newFilterRead (char *basename)

  Reads a filter from the file system.

4.4.1 Function Documentation

4.4.1.1 deleteFilter()

```
void deleteFilter (
            Filter * f )
```

Releases memory allocated for a filter.

Parameters

f	filter for which memory is going to be freed Nothing happens if f is NULL.

```
29                          {
30     if (f==NULL) return;
31     if (f->img)
32         deleteImg(f->img);
33     f->weight=0;
34     f->threshold=0;
35     f->maxVal=0;
36     if (f->data!=NULL)
37         free(f->data);
38     memset(f,0,sizeof(Filter));
39     free(f);
40 }
```

References filter::data, deleteImg(), filter::img, filter::maxVal, filter::threshold, and filter::weight.

4.4.1.2 filterSetData()

```
void filterSetData (
            Filter * f,
            char * s )
```

Copies a string in the data field of a filter.

Parameters

f	a filter
s	a string to be copied in f->data

```
47                                          {
48     f->data=stringCopy(s);
49 }
```

References filter::data, and stringCopy().

4.4.1.3 filterSetWeight()

```
void filterSetWeight (
            Filter * f,
            int w )
```

sets the filter image to a given weight

Parameters

f	filter which weight is going to be changed
w	the new weight of the filter

```
56                                      {
57     long int ww=0;
58     for(int i = 0; i < f->img->height*f->img->width; i++) {
59         ww+=f->img->data[i];
60     }
61     float ratio = (w+0.)/(ww+0.);
62     for(int i = 0; i < f->img->height*f->img->width; i++) {
63         f->img->data[i]=INBYTE((int)round(ratio*f->img->data[i]));
64     }
65     filterUpdateValues(f);
66 }
```

References img::data, filterUpdateValues(), img::height, filter::img, INBYTE, and img::width.

4.4.1.4 filterUpdateValues()

```
void filterUpdateValues (
            Filter * f )
```

Updates weight, threshold and maxVal values in a filter given the image and the percentage threshold should have.

Parameters

f	filter to update

```
73                                  {
74      f->weight=0;
75      f->threshold=0;
76      f->maxVal=f->img->height*f->img->width*255;
77      if (f->percent>100 || f->percent<-100) {
78          ERROR("percent should be between 0 and 100.","");
79      }
80      for(int i = 0; i < f->img->height*f->img->width; i++) {
81          f->weight+=f->img->data[i]-128;
82      }
83      f->threshold=f->maxVal*f->percent/100;
84  }
```

References img::data, ERROR, img::height, filter::img, filter::maxVal, filter::percent, filter::threshold, filter::weight, and img::width.

4.4.1.5 filterWrite()

```
void filterWrite (
            Filter * f,
            char * basename )
```

Saves as png files the filter.

Parameters

f	filter to save.
basename	the base name for files to save. Files basename+'.png' and basename+'.txt' are going to be created.

See also

newFilterRead

```
94                                      {
95      char s[99];
96      // save the picture part
97      snprintf(s,99,"%s.png",basename);
98      imgWrite(f->img,s);
99      // save the data part
100      snprintf(s,99,"%s.txt",basename);
101      FILE * fi = fopen(s,"w");
102      if (fi==NULL)
103          ERROR("Could not open file ",s);
104      fprintf(fi,"%d\n",f->percent);
105      fprintf(fi,"%s\n",f->data);
106      fclose(fi);
107  }
```

References filter::data, ERROR, filter::img, imgWrite(), and filter::percent.

4.4.1.6 newFilter()

```
Filter * newFilter (
            Img * img,
            int percent )
```

Allocates space for a new filter.

Parameters

img	image on which the filter is based
percent	: given maxVal, the maximum value we can get if image img is used as a filter, threshold will be percentage of maxVal.

```
11                                          {
12     Filter * answer = (Filter*) malloc(sizeof(struct filter));
13     answer->img=img;
14     answer->threshold=0;
15     answer->percent=percent;
16     answer->weight=0;
17     answer->maxVal=0;
18     answer->data=NULL;
19     if (img==NULL) return answer;
20     filterUpdateValues(answer);
21     return answer;
22 }
```

References filter::data, filterUpdateValues(), filter::img, filter::maxVal, filter::percent, filter::threshold, and filter::weight.

4.4.1.7 newFilterRead()

```
Filter * newFilterRead (
              char * basename )
```

Reads a filter from the file system.

Parameters

basename	the base name for files to read. Files basename+'.png' and basename+'.txt' are expected to be found.

Returns

the newly created Filter object from data read.

See also

filterWrite

```
117                                         {
118     Filter * answer = (Filter*) malloc(sizeof(struct filter));
119     char s[99];
120     // read the picture part
121     snprintf(s,99,"%s.png",basename);
122     answer->img=newImgRead(s);
123     // read the data part
124     snprintf(s,99,"%s.txt",basename);
125     FILE * fi = fopen(s,"r");
126     if (fi==NULL)
127         ERROR("Could not open file ",s);
128     fscanf(fi,"%d",&(answer->percent));
129
130     char * line = NULL;
131     size_t len = 0;
132     if (getline(&line, &len, fi)==-1) {
133         WARNING("Error while reading ",s);
134     } else {
135         answer->data=stringCopy(line);
136     }
137     if (line) free(line);
```

```
138     fclose(fi);
139     filterUpdateValues(answer);
140     return answer;
141 }
```

References filter::data, ERROR, filterUpdateValues(), filter::img, newImgRead(), filter::percent, stringCopy(), and WARNING.

4.5 filter.h File Reference

```
#include "img.h"
```

Classes

- struct filter

  Data structure to store a filter.

Typedefs

- typedef struct filter Filter

  Short name for 'struct filter'.

Functions

- Filter * newFilter (Img *, int)

  Allocates space for a new filter.
- Filter * newFilterRead (char *basename)

  Reads a filter from the file system.
- void deleteFilter (Filter *)

  Releases memory allocated for a filter.
- void filterSetWeight (Filter *f, int w)

  sets the filter image to a given weight
- void filterUpdateValues (Filter *)

  Updates weight, threshold and maxVal values in a filter given the image and the percentage threshold should have.
- void filterWrite (Filter *f, char *basename)

  Saves as png files the filter.

4.5.1 Typedef Documentation

4.5.1.1 Filter

```
typedef struct filter Filter
```

Short name for 'struct filter'.

4.5.2 Function Documentation

4.5.2.1 deleteFilter()

```
void deleteFilter (
            Filter * f )
```

Releases memory allocated for a filter.

Parameters

f	filter for which memory is going to be freed Nothing happens if f is NULL.

```
29                              {
30     if (f==NULL) return;
31     if (f->img)
32         deleteImg(f->img);
33     f->weight=0;
34     f->threshold=0;
35     f->maxVal=0;
36     if (f->data!=NULL)
37         free(f->data);
38     memset(f,0,sizeof(Filter));
39     free(f);
40 }
```

References filter::data, deleteImg(), filter::img, filter::maxVal, filter::threshold, and filter::weight.

4.5.2.2 filterSetWeight()

```
void filterSetWeight (
            Filter * f,
            int w )
```

sets the filter image to a given weight

Parameters

f	filter which weight is going to be changed
w	the new weight of the filter

```
56                                      {
57     long int ww=0;
58     for(int i = 0; i < f->img->height*f->img->width; i++) {
59         ww+=f->img->data[i];
60     }
61     float ratio = (w+0.)/(ww+0.);
62     for(int i = 0; i < f->img->height*f->img->width; i++) {
63         f->img->data[i]=INBYTE((int)round(ratio*f->img->data[i]));
64     }
65     filterUpdateValues(f);
66 }
```

References img::data, filterUpdateValues(), img::height, filter::img, INBYTE, and img::width.

4.5.2.3 filterUpdateValues()

```
void filterUpdateValues (
            Filter * f )
```

Updates weight, threshold and maxVal values in a filter given the image and the percentage threshold should have.

Parameters

f	filter to update

```
73                                  {
74     f->weight=0;
75     f->threshold=0;
76     f->maxVal=f->img->height*f->img->width*255;
77     if (f->percent>100 || f->percent<-100) {
78         ERROR("percent should be between 0 and 100.","");
79     }
80     for(int i = 0; i < f->img->height*f->img->width; i++) {
81         f->weight+=f->img->data[i]-128;
82     }
83     f->threshold=f->maxVal*f->percent/100;
84 }
```

References img::data, ERROR, img::height, filter::img, filter::maxVal, filter::percent, filter::threshold, filter::weight, and img::width.

4.5.2.4 filterWrite()

```
void filterWrite (
            Filter * f,
            char * basename )
```

Saves as png files the filter.

Parameters

f	filter to save.
basename	the base name for files to save. Files basename+'.png' and basename+'.txt' are going to be created.

See also

newFilterRead

```
94                                      {
95     char s[99];
96     // save the picture part
97     snprintf(s,99,"%s.png",basename);
98     imgWrite(f->img,s);
99     // save the data part
100     snprintf(s,99,"%s.txt",basename);
101     FILE * fi = fopen(s,"w");
102     if (fi==NULL)
103         ERROR("Could not open file ",s);
104     fprintf(fi,"%d\n",f->percent);
105     fprintf(fi,"%s\n",f->data);
106     fclose(fi);
107 }
```

References filter::data, ERROR, filter::img, imgWrite(), and filter::percent.

4.5.2.5 newFilter()

```
Filter * newFilter (
            Img * img,
            int percent )
```

Allocates space for a new filter.

Parameters

img	image on which the filter is based
percent	: given maxVal, the maximum value we can get if image img is used as a filter, threshold will be percentage of maxVal.

```
11                                          {
12     Filter * answer = (Filter*) malloc(sizeof(struct filter));
13     answer->img=img;
14     answer->threshold=0;
15     answer->percent=percent;
16     answer->weight=0;
17     answer->maxVal=0;
18     answer->data=NULL;
19     if (img==NULL) return answer;
20     filterUpdateValues(answer);
21     return answer;
22 }
```

References filter::data, filterUpdateValues(), filter::img, filter::maxVal, filter::percent, filter::threshold, and filter::weight.

4.5.2.6 newFilterRead()

```
Filter * newFilterRead (
            char * basename )
```

Reads a filter from the file system.

Parameters

basename	the base name for files to read. Files basename+'.png' and basename+'.txt' are expected to be found.

Returns

the newly created Filter object from data read.

See also

filterWrite

```
117                                         {
118     Filter * answer = (Filter*) malloc(sizeof(struct filter));
119     char s[99];
120     // read the picture part
121     snprintf(s,99,"%s.png",basename);
122     answer->img=newImgRead(s);
123     // read the data part
124     snprintf(s,99,"%s.txt",basename);
```

```
125     FILE * fi = fopen(s,"r");
126     if (fi==NULL)
127         ERROR("Could not open file ",s);
128     fscanf(fi,"%d",&(answer->percent));
129 
130     char * line = NULL;
131     size_t len = 0;
132     if (getline(&line, &len, fi)==-1) {
133         WARNING("Error while reading ",s);
134     } else {
135         answer->data=stringCopy(line);
136     }
137     if (line) free(line);
138     fclose(fi);
139     filterUpdateValues(answer);
140     return answer;
141 }
```

References filter::data, ERROR, filterUpdateValues(), filter::img, newImgRead(), filter::percent, stringCopy(), and WARNING.

4.6 filter.h

Go to the documentation of this file.

```
1 #ifndef FILTER_H
2 #define FILTER_H
3 
4 #include "img.h"
5 
9 struct filter {
11     Img *img;
13     long int weight;
15     long int threshold;
17     long int maxVal;
19     int percent;
21     char * data;
22 };
23 
24 
28 typedef struct filter Filter;
29 
30 Filter * newFilter(Img*,int);
31 Filter * newFilterRead(char *basename);
32 void deleteFilter(Filter*);
33 void filterSetWeight(Filter*f,int w);
34 void filterUpdateValues(Filter*);
35 void filterWrite(Filter*f,char*basename);
36 #endif
```

4.7 filterfam.c File Reference

```
#include <unistd.h>
#include "filterfam.h"
#include "imgfam.h"
```

Functions

- FilterFam * newFilterFam (int c)

  creates a a new familly of filters.
- void deleteFilterFam (FilterFam *ff)

  Deletes a familly of filters.
- int filterFamCount (char *convFilterLoc)

  counts the number of filters present in familly with prefix convFilterLoc

- void filterFamSetFilter (FilterFam *filterFam, int i, Filter *filter)

  set the i-th image of a familly of images.
- ImgFam * filterFamApplyConvolution (FilterFam *filters, Img *img)

  Apply all filter of the familly on an image with a positive convolution.
- ImgFam * filterFamApplyConvolutionDiff (FilterFam *filters, Img *img)

  Apply all filter of the familly on an image.
- ImgFam * filterFamApplyConvolutionSameSize (FilterFam *filters, Img *img)

  Apply all filter of the familly on an image an keep original image size.
- ImgFam * filterFamApplyConvolutionSameSizeDiff (FilterFam *filters, Img *img)

  Apply all filter of the familly on an image an keep original image size.
- ImgFam * filterFamApplyConvolutionOnFam (FilterFam *filters, ImgFam *imgFam)

  Apply all filter of the familly on a familly of images.
- ImgFam * filterFamApplyConvolutionSameSizeOnFam (FilterFam *filters, ImgFam *imgFam)

  Apply all filter of the familly on a familly of images, and keep same image size.
- void filterFamWrite (FilterFam *filterFam, char *basename)

  Saves as png files the image familly.
- FilterFam * filterFamRead (char *basename)

  Read a set of png files to return a filter familly.

4.7.1 Function Documentation

4.7.1.1 deleteFilterFam()

```
void deleteFilterFam (
            FilterFam * ff )
```

Deletes a familly of filters.

Parameters

ff	the familly to delete.

```
21                                      {
22     if (ff==NULL) return;
23     for (int i=0;i<ff->count;++i) {
24         deleteFilter(ff->filters[i]);
25     }
26     memset(ff->filters,0,sizeof(Filter*)*ff->count);
27     free(ff->filters);
28     memset(ff,0,sizeof(FilterFam));
29     free(ff);
30 }
```

References filterfam::count, deleteFilter(), and filterfam::filters.

4.7.1.2 filterFamApplyConvolution()

```
ImgFam * filterFamApplyConvolution (
            FilterFam * filters,
            Img * img )
```

Apply all filter of the familly on an image with a positive convolution.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
71                                                                  {
72     ImgFam * answer = newImgFam (filters->count);
73     for (int i=0;i<filters->count;++i) {
74         if (filters->filters[i]==NULL) {
75             ERROR("Not all filters in familly have been initialized.","");
76         }
77         imgFamSetImg(answer,i,imgConvolution(img,filters->filters[i]));
78     }
79     return answer;
80 }
```

References filterfam::count, ERROR, filterfam::filters, imgConvolution(), imgFamSetImg(), and newImgFam().

4.7.1.3 filterFamApplyConvolutionDiff()

```
ImgFam * filterFamApplyConvolutionDiff (
             FilterFam * filters,
             Img * img )
```

Apply all filter of the familly on an image.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
89                                                                  {
90     ImgFam * answer = newImgFam (filters->count);
91     for (int i=0;i<filters->count;++i) {
92         if (filters->filters[i]==NULL) {
93             ERROR("Not all filters in familly have been initialized.","");
94         }
95         imgFamSetImg(answer,i,imgConvolutionDiff(img,filters->filters[i]));
96     }
97     return answer;
98 }
```

References filterfam::count, ERROR, filterfam::filters, imgConvolutionDiff(), imgFamSetImg(), and newImgFam().

4.7.1.4 filterFamApplyConvolutionOnFam()

```
ImgFam * filterFamApplyConvolutionOnFam (
             FilterFam * filters,
             ImgFam * imgFam )
```

Apply all filter of the familly on a familly of images.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
146                                                                          {
147     ImgFam * answer = newImgFam (filters->count*imgFam->count);
148     for (int i=0;i<filters->count;++i) {
149         for (int j=0;j<imgFam->count;++j) {
150             int idx=j+i*imgFam->count;
151             imgFamSetImg(answer,idx,imgConvolution(imgFam->imgs[j],
152                                                    filters->filters[i]));
153     }   }
154     return answer;
155 }
```

References filterfam::count, imgfam::count, filterfam::filters, imgConvolution(), imgFamSetImg(), imgfam::imgs, and newImgFam().

4.7.1.5 filterFamApplyConvolutionSameSize()

```
ImgFam * filterFamApplyConvolutionSameSize (
             FilterFam * filters,
             Img * img )
```

Apply all filter of the familly on an image an keep original image size.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
108                                                                  {
109     ImgFam * answer = newImgFam (filters->count);
110     for (int i=0;i<filters->count;++i) {
111         if (filters->filters[i]==NULL) {
112             ERROR("Not all filters in familly have been initialized.","");
113         }
```

```
114         imgFamSetImg(answer,i,imgConvolutionSameSize(img,filters->filters[i]));
115     }
116     return answer;
117 }
```

References filterfam::count, ERROR, filterfam::filters, imgConvolutionSameSize(), imgFamSetImg(), and newImgFam().

4.7.1.6 filterFamApplyConvolutionSameSizeDiff()

```
ImgFam * filterFamApplyConvolutionSameSizeDiff (
             FilterFam * filters,
             Img * img )
```

Apply all filter of the familly on an image an keep original image size.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
128                                                                         {
129     ImgFam * answer = newImgFam (filters->count);
130     for (int i=0;i<filters->count;++i) {
131         if (filters->filters[i]==NULL) {
132             ERROR("Not all filters in familly have been initialized.","");
133         }
134         imgFamSetImg(answer,i,imgConvolutionSameSizeDiff(img,filters->filters[i]));
135     }
136     return answer;
137 }
```

References filterfam::count, ERROR, filterfam::filters, imgConvolutionSameSizeDiff(), imgFamSetImg(), and newImgFam().

4.7.1.7 filterFamApplyConvolutionSameSizeOnFam()

```
ImgFam * filterFamApplyConvolutionSameSizeOnFam (
             FilterFam * filters,
             ImgFam * imgFam )
```

Apply all filter of the familly on a familly of images, and keep same image size.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
167 {
168     ImgFam * answer = newImgFam (filters->count*imgFam->count);
169     for (int i=0;i<filters->count;++i) {
170         for (int j=0;j<imgFam->count;++j) {
171             int idx=j+i*imgFam->count;
172             imgFamSetImg(answer,idx,imgConvolution(imgFam->imgs[j],
173                                                    filters->filters[i]));
174     }   }
175     return answer;
176 }
```

References filterfam::count, imgfam::count, filterfam::filters, imgConvolution(), imgFamSetImg(), imgfam::imgs, and newImgFam().

4.7.1.8 filterFamCount()

```
int filterFamCount (
             char * convFilterLoc )
```

counts the number of filters present in familly with prefix convFilterLoc

Parameters

convFilterLoc	a string to be used as a base for the name of the picture : convFilterLoc_<num>.png

Returns

number of elements in this filter

```
39                                          {
40     int answer =-1;
41     char fname[strlen(convFilterLoc)+10];
42     do {
43         answer ++;
44         snprintf(fname,strlen(convFilterLoc)+10,
45                  "%s_%d.png",convFilterLoc,answer);
46     } while (access(fname, F_OK) == 0);
47     return answer;
48 }
```

4.7.1.9 filterFamRead()

```
FilterFam * filterFamRead (
             char * basename )
```

Read a set of png files to return a filter familly.

Parameters

basename	the base name for all files read. names read have the form basename_i.png where i is the number of the picture in the familly.

Returns

a newly allocated filter familly.

See also

imgFamWrite

```
202                                             {
203     char s[99];
204     int count=0;
205     int found=1;
206     while (found) {
207         snprintf(s,99,"%s_%d.png",basename,count);
208         FILE * f = fopen(s,"r");
209         if (f!=NULL) {
210             fclose(f);
211             ++count;
212         } else  {
213             found=0;
214         }
215     }
216     FilterFam * answer = newFilterFam(count);
217     for (int i=0;i<answer->count;++i) {
218         snprintf(s,99,"%s_%d",basename,i);
219         filterFamSetFilter(answer,i,newFilterRead(s));
220     }
221     return answer;
222 }
```

References filterfam::count, filterFamSetFilter(), newFilterFam(), and newFilterRead().

4.7.1.10 filterFamSetFilter()

```
void filterFamSetFilter (
            FilterFam * filterFam,
            int i,
            Filter * filter )
```

set the i-th image of a familly of images.

Parameters

filterFam	the familly of images in which we are going to set the image.
i	index of the image to be in the collection.
filter	the image that will be at i-th location in filterFam.

```
57                                                                  {
58     if (i<0 && i>=filterFam->count)
59         ERROR("out of bounds.","");
60     filterFam->filters[i]=filter;
61 }
```

References filterfam::count, ERROR, and filterfam::filters.

4.7.1.11 filterFamWrite()

```
void filterFamWrite (
            FilterFam * filterFam,
            char * basename )
```

Saves as png files the image familly.

Parameters

filterFam	the familly to save.
basename	the base name for all files save. The final name of each file will be basename_i.png where i is the number of the picture in the familly.

See also

filterFamRead

```
186                                                     {
187     char s[99];
188     for (int i=0;i<filterFam->count;++i) {
189         snprintf(s,99,"%s_%d",basename,i);
190         filterWrite(filterFam->filters[i],s);
191     }
192 }
```

References filterfam::count, filterfam::filters, and filterWrite().

4.7.1.12 newFilterFam()

```
FilterFam * newFilterFam (
              int c )
```

creates a a new familly of filters.

Parameters

c	the size of the familly.

```
9                               {
10     FilterFam * answer = (FilterFam*)malloc(sizeof(struct filterfam));
11     answer->count=c;
12     answer->filters=(Filter**)malloc(sizeof(Filter**)*c);
13     memset(answer->filters,0,sizeof(Filter*)*c);
14     return answer;
15 }
```

References filterfam::count, and filterfam::filters.

4.8 filterfam.h File Reference

```
#include "filter.h"
```

Classes

- struct filterfam

  A structure to store a collection (also called familly) of filters.

Typedefs

- typedef struct filterfam FilterFam

  Short name for 'struct filterfam'.
- typedef struct imgfam ImgFam

Functions

- FilterFam * newFilterFam (int)

  creates a a new familly of filters.
- ImgFam * filterFamApplyConvolution (FilterFam *filters, Img *img)

  Apply all filter of the familly on an image with a positive convolution.
- ImgFam * filterFamApplyConvolutionDiff (FilterFam *filters, Img *img)

  Apply all filter of the familly on an image.
- ImgFam * filterFamApplyConvolutionOnFam (FilterFam *filters, ImgFam *imgFam)

  Apply all filter of the familly on a familly of images.
- ImgFam * filterFamApplyConvolutionSameSize (FilterFam *filters, Img *img)

  Apply all filter of the familly on an image an keep original image size.
- ImgFam * filterFamApplyConvolutionSameSizeOnFam (FilterFam *filters, ImgFam *imgFam)

  Apply all filter of the familly on a familly of images, and keep same image size.
- ImgFam * filterFamApplyConvolutionSameSizeDiff (FilterFam *filters, Img *img)

  Apply all filter of the familly on an image an keep original image size.
- void filterFamSetFilter (FilterFam *filterFam, int i, Filter *filter)

  set the i-th image of a familly of images.
- void deleteFilterFam (FilterFam *)

  Deletes a familly of filters.
- FilterFam * filterFamRead (char *basename)

  Read a set of png files to return a filter familly.
- void filterFamWrite (FilterFam *filterFam, char *basename)

  Saves as png files the image familly.
- int filterFamCount (char *convFilterLoc)

  counts the number of filters present in familly with prefix convFilterLoc

4.8.1 Typedef Documentation

4.8.1.1 FilterFam

```
typedef struct filterfam FilterFam
```

Short name for 'struct filterfam'.

4.8.1.2 ImgFam

```
typedef struct imgfam ImgFam
```

4.8.2 Function Documentation

4.8.2.1 deleteFilterFam()

```
void deleteFilterFam (
            FilterFam * ff )
```

Deletes a familly of filters.

Parameters

ff	the familly to delete.

```
21                                      {
22     if (ff==NULL) return;
23     for (int i=0;i<ff->count;++i) {
24         deleteFilter(ff->filters[i]);
25     }
26     memset(ff->filters,0,sizeof(Filter*)*ff->count);
27     free(ff->filters);
28     memset(ff,0,sizeof(FilterFam));
29     free(ff);
30 }
```

References filterfam::count, deleteFilter(), and filterfam::filters.

4.8.2.2 filterFamApplyConvolution()

```
ImgFam * filterFamApplyConvolution (
            FilterFam * filters,
            Img * img )
```

Apply all filter of the familly on an image with a positive convolution.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
71                                                              {
72     ImgFam * answer = newImgFam (filters->count);
73     for (int i=0;i<filters->count;++i) {
74         if (filters->filters[i]==NULL) {
75             ERROR("Not all filters in familly have been initialized.","");
76         }
77         imgFamSetImg(answer,i,imgConvolution(img,filters->filters[i]));
78     }
79     return answer;
80 }
```

References filterfam::count, ERROR, filterfam::filters, imgConvolution(), imgFamSetImg(), and newImgFam().

4.8.2.3 filterFamApplyConvolutionDiff()

```
ImgFam * filterFamApplyConvolutionDiff (
            FilterFam * filters,
            Img * img )
```

Apply all filter of the familly on an image.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
89                                                                  {
90     ImgFam * answer = newImgFam (filters->count);
91     for (int i=0;i<filters->count;++i) {
92         if (filters->filters[i]==NULL) {
93             ERROR("Not all filters in familly have been initialized.","");
94         }
95         imgFamSetImg(answer,i,imgConvolutionDiff(img,filters->filters[i]));
96     }
97     return answer;
98 }
```

References filterfam::count, ERROR, filterfam::filters, imgConvolutionDiff(), imgFamSetImg(), and newImgFam().

4.8.2.4 filterFamApplyConvolutionOnFam()

```
ImgFam * filterFamApplyConvolutionOnFam (
            FilterFam * filters,
            ImgFam * imgFam )
```

Apply all filter of the familly on a familly of images.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
146                                                                         {
147     ImgFam * answer = newImgFam (filters->count*imgFam->count);
148     for (int i=0;i<filters->count;++i) {
149         for (int j=0;j<imgFam->count;++j) {
150             int idx=j+i*imgFam->count;
151             imgFamSetImg(answer,idx,imgConvolution(imgFam->imgs[j],
152                                                    filters->filters[i]));
153     }   }
154     return answer;
155 }
```

References filterfam::count, imgfam::count, filterfam::filters, imgConvolution(), imgFamSetImg(), imgfam::imgs, and newImgFam().

4.8.2.5 filterFamApplyConvolutionSameSize()

```
ImgFam * filterFamApplyConvolutionSameSize (
             FilterFam * filters,
             Img * img )
```

Apply all filter of the familly on an image an keep original image size.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
108                                                                     {
109     ImgFam * answer = newImgFam (filters->count);
110     for (int i=0;i<filters->count;++i) {
111         if (filters->filters[i]==NULL) {
112             ERROR("Not all filters in familly have been initialized.","");
113         }
114         imgFamSetImg(answer,i,imgConvolutionSameSize(img,filters->filters[i]));
115     }
116     return answer;
117 }
```

References filterfam::count, ERROR, filterfam::filters, imgConvolutionSameSize(), imgFamSetImg(), and newImgFam().

4.8.2.6 filterFamApplyConvolutionSameSizeDiff()

```
ImgFam * filterFamApplyConvolutionSameSizeDiff (
             FilterFam * filters,
             Img * img )
```

Apply all filter of the familly on an image an keep original image size.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
128                                                                          {
129     ImgFam * answer = newImgFam (filters->count);
130     for (int i=0;i<filters->count;++i) {
131         if (filters->filters[i]==NULL) {
132             ERROR("Not all filters in familly have been initialized.","");
133         }
134         imgFamSetImg(answer,i,imgConvolutionSameSizeDiff(img,filters->filters[i]));
135     }
136     return answer;
137 }
```

References filterfam::count, ERROR, filterfam::filters, imgConvolutionSameSizeDiff(), imgFamSetImg(), and newImgFam().

4.8.2.7 filterFamApplyConvolutionSameSizeOnFam()

```
ImgFam * filterFamApplyConvolutionSameSizeOnFam (
             FilterFam * filters,
             ImgFam * imgFam )
```

Apply all filter of the familly on a familly of images, and keep same image size.

Parameters

filters	the familly of filters to apply.
img	the image on which to apply these filters
the	filters to apply.

Returns

the newly created familly of resulting images.

```
167 {
168     ImgFam * answer = newImgFam (filters->count*imgFam->count);
169     for (int i=0;i<filters->count;++i) {
170         for (int j=0;j<imgFam->count;++j) {
171             int idx=j+i*imgFam->count;
172             imgFamSetImg(answer,idx,imgConvolution(imgFam->imgs[j],
173                                                    filters->filters[i]));
174     }   }
175     return answer;
176 }
```

References filterfam::count, imgfam::count, filterfam::filters, imgConvolution(), imgFamSetImg(), imgfam::imgs, and newImgFam().

4.8.2.8 filterFamCount()

```
int filterFamCount (
            char * convFilterLoc )
```

counts the number of filters present in familly with prefix convFilterLoc

Parameters

convFilterLoc	a string to be used as a base for the name of the picture : convFilterLoc_<num>.png

Returns

number of elements in this filter

```
39                                          {
40     int answer =-1;
41     char fname[strlen(convFilterLoc)+10];
42     do {
43         answer ++;
44         snprintf(fname,strlen(convFilterLoc)+10,
45                  "%s_%d.png",convFilterLoc,answer);
46     } while (access(fname, F_OK) == 0);
47     return answer;
48 }
```

4.8.2.9 filterFamRead()

```
FilterFam * filterFamRead (
            char * basename )
```

Read a set of png files to return a filter familly.

Parameters

basename	the base name for all files read. names read have the form basename_i.png where i is the number of the picture in the familly.

Returns

a newly allocated filter familly.

See also

imgFamWrite

```
202                                          {
203     char s[99];
204     int count=0;
205     int found=1;
206     while (found) {
207         snprintf(s,99,"%s_%d.png",basename,count);
208         FILE * f = fopen(s,"r");
209         if (f!=NULL) {
210             fclose(f);
211             ++count;
212         } else  {
213             found=0;
```

```
214         }
215     }
216     FilterFam * answer = newFilterFam(count);
217     for (int i=0;i<answer->count;++i) {
218         snprintf(s,99,"%s_%d",basename,i);
219         filterFamSetFilter(answer,i,newFilterRead(s));
220     }
221     return answer;
222 }
```

References filterfam::count, filterFamSetFilter(), newFilterFam(), and newFilterRead().

4.8.2.10 filterFamSetFilter()

```
void filterFamSetFilter (
            FilterFam * filterFam,
            int i,
            Filter * filter )
```

set the i-th image of a familly of images.

Parameters

filterFam	the familly of images in which we are going to set the image.
i	index of the image to be in the collection.
filter	the image that will be at i-th location in filterFam.

```
57                                                          {
58     if (i<0 && i>=filterFam->count)
59         ERROR("out of bounds.","");
60     filterFam->filters[i]=filter;
61 }
```

References filterfam::count, ERROR, and filterfam::filters.

4.8.2.11 filterFamWrite()

```
void filterFamWrite (
            FilterFam * filterFam,
            char * basename )
```

Saves as png files the image familly.

Parameters

filterFam	the familly to save.
basename	the base name for all files save. The final name of each file will be basename_i.png where i is the number of the picture in the familly.

See also

filterFamRead

```
186                                                     {
187     char s[99];
188     for (int i=0;i<filterFam->count;++i) {
189         snprintf(s,99,"%s_%d",basename,i);
190         filterWrite(filterFam->filters[i],s);
191     }
192 }
```

References filterfam::count, filterfam::filters, and filterWrite().

4.8.2.12 newFilterFam()

```
FilterFam * newFilterFam (
              int c )
```

creates a a new familly of filters.

Parameters

c	the size of the familly.

```
9                               {
10     FilterFam * answer = (FilterFam*)malloc(sizeof(struct filterfam));
11     answer->count=c;
12     answer->filters=(Filter**)malloc(sizeof(Filter**)*c);
13     memset(answer->filters,0,sizeof(Filter*)*c);
14     return answer;
15 }
```

References filterfam::count, and filterfam::filters.

4.9 filterfam.h

Go to the documentation of this file.

```
1 #ifndef FILTERFAM_H
2 #define FILTERFAM_H
3
4 #include "filter.h"
5
6
11 struct filterfam {
13     int count;
15     Filter ** filters;
16 };
17
21 typedef struct filterfam FilterFam;
22
23 // external types
24 typedef struct imgfam ImgFam;
25
26 FilterFam * newFilterFam(int);
27 ImgFam * filterFamApplyConvolution(FilterFam* filters,Img* img);
28 ImgFam * filterFamApplyConvolutionDiff(FilterFam* filters,Img* img);
29 ImgFam * filterFamApplyConvolutionOnFam(FilterFam* filters,ImgFam* imgFam);
30 ImgFam * filterFamApplyConvolutionSameSize(FilterFam* filters,Img* img);
31 ImgFam * filterFamApplyConvolutionSameSizeOnFam(FilterFam* filters,
32                                                 ImgFam* imgFam);
33 ImgFam * filterFamApplyConvolutionSameSizeDiff(FilterFam* filters,Img* img);
34 void filterFamSetFilter(FilterFam*filterFam,int i,Filter*filter);
35 void deleteFilterFam(FilterFam *);
36 FilterFam * filterFamRead(char*basename);
37 void filterFamWrite(FilterFam*filterFam,char*basename);
38 int filterFamCount(char * convFilterLoc);
39
40 #endif
```

4.10 fontname.c File Reference

```
#include "fontname.h"
#include <stdlib.h>
```

Variables

- int fontcount =1439
- char * fontname [1439]

4.10.1 Variable Documentation

4.10.1.1 fontcount

```
int fontcount =1439
```

4.10.1.2 fontname

```
char* fontname[1439]
```

4.11 fontname.h File Reference

Variables

- char * fontname [ ]
- int fontcount

4.11.1 Variable Documentation

4.11.1.1 fontcount

```
int fontcount  [extern]
```

4.11.1.2 fontname

```
char* fontname[]  [extern]
```

4.12 fontname.h

Go to the documentation of this file.

```
1 // file automatically generated by gen_fontlist.sh.
2 #ifndef FONTNAME_H
3 #define FONTNAME_H
4 extern char * fontname[];
5 extern int fontcount;
6 #endif
```

4.13 grid.c File Reference

```
#include "filterfam.h"
#include "filter.h"
#include "imgfam.h"
```

Functions

- FilterFam * gridGetLayerHoriVertFilters (int i, int l, int t, int p)

  Generates filters for the convolution layer to detect Sudoku grids.
- void gridVertHoriConvo (Img **outputH, Img **outputV, Img *inputH, Img *inputV, int width, int length, int threshold, int poolsize, int stride)

  Perform vertical and horizontal convolution.
- int gridIdentifyNPoints (int N, Img *img, int startRange, int endRange, int *lowerBound, int *upperBound, int *maxCorrelation)

  identify N points equaly spaced in a n by 1 pixel image.
- int gridLocate (Img *img, int *xmin, int *ymin, int *xmax, int *ymax)

  Locates a sudoku grid in a picture.
- void gridUsage (FILE *f, char *name)

  Tells how to use this program.
- int gridMain (int argc, char **argv)

  main function to execute when the grid executable is called from the command line.

Variables

- int gridDumpDebugInfo =1
- int gridIdentifyNPointsCounter =0

4.13.1 Function Documentation

4.13.1.1 gridGetLayerHoriVertFilters()

```
FilterFam * gridGetLayerHoriVertFilters (
            int i,
            int l,
            int t,
            int p )
```

Generates filters for the convolution layer to detect Sudoku grids.

Parameters

i	: if i==0 this is a vertical filter if i==1 this is an horizontal filter
l	length of the bar
t	thickness of the bar
p	filter threshold percentage

Returns

the newly allocated filter

invertedFilter->maxVal/10;

```
21 {
22     FilterFam * answer=newFilterFam(1);
23
24     Img * filter = newImgVerticalBarInRect(5*t,l,t);
25     Img * rotatedFilter = NULL;
26     if (i==0)
27         rotatedFilter=newImgCopy(filter);
28     else
29         rotatedFilter=imgRotate90(filter);
30     Filter * invertedFilter = newFilter(imgInvert(rotatedFilter),p);
31     if (gridDumpDebugInfo) {
32         HERE("layer2 on filter");
33         HERED((int)invertedFilter->weight);
34         HERE("_________");
35         HERED((int)invertedFilter->maxVal);
36     }
38     deleteImg(filter);
39     deleteImg(rotatedFilter);
40     if (gridDumpDebugInfo) {
41         char filterName[99];
42         snprintf(filterName,99,"%s/grid/grid_filter_%d",CFG_DATAROOTDIR,i);
43         filterWrite(invertedFilter,filterName);
44     }
45     filterFamSetFilter(answer,0,invertedFilter);
46     return answer;
47 }
```

References deleteImg(), filterFamSetFilter(), filterWrite(), gridDumpDebugInfo, HERE, HERED, imgInvert(), imgRotate90(), filter::maxVal, newFilter(), newFilterFam(), newImgCopy(), newImgVerticalBarInRect(), and filter::weight.

4.13.1.2 gridIdentifyNPoints()

```
int gridIdentifyNPoints (
              int N,
              Img * img,
              int startRange,
              int endRange,
              int * lowerBound,
              int * upperBound,
              int * maxCorrelation )
```

identify N points equaly spaced in a n by 1 pixel image.

This function, called from gridLocate, is a simple convolution with some filters comming from newImgNDotsHori.

Parameters

N	number of points equaly spaced we are looking for.
img	an image of n by 1 pixels
startRange	minimum distance in pixel among which the N points are spread.
endRange	maximum distance in pixel among which the N points are spread.

See also

newImgNDotsHori

gridLocate

```
139 {
140     int n = img->width;
141     if (img->height!=1) {
142         ERROR("Expected a height of 1","");
143     }
144     gridIdentifyNPointsCounter++;
145     Img * imgs[n];
146     int convVal[n];
147     int weight[n];
148     int max=0;
149     int maxIdx=0;
150     int lowerBoundOffset[n];
151     int upperBoundOffset[n];
152     for (int i=n/2;i<n-1;++i) {
153         Img * f1 = newImgNDotsHori(N,i);
154         Filter * f =newFilter(imgInvert(f1),99);
155         deleteImg(f1);
156         if (gridDumpDebugInfo) {
157             char s[99];
158             snprintf(s,99,"conv_filter_%d_%d.png",N,i);
159             imgWrite(f->img,s);
160         }
161         Img * c = imgConvolutionDiff(img,f);
162         weight[i]=f->weight;
163         deleteFilter(f);
164         imgs[i]=c;
165         if (gridDumpDebugInfo) {
166             char s[99];
167             snprintf(s,99,"conv_result_%d_%d.png",gridIdentifyNPointsCounter,i);
168             imgWrite(c,s);
169         }
170         int localMax=-1;
171         for (int j=0;j<c->width;++j) {
172             if (c->data[j]>localMax) {
173                 localMax=c->data[j];
174                 lowerBoundOffset[i]=j+i/2/N;
175                 upperBoundOffset[i]=j+i-i/2/N;
176             }
177         }
178         if (localMax>max) {
179             max=localMax;maxIdx=i;
180         }
181         convVal[i]=localMax;
182     }
183     for (int i=n/2;i<n-1;++i) {
184         if (max-max/20<convVal[i]) {
185             if (gridDumpDebugInfo) {
186                 HERE("------------");
187                 HERED(convVal[i]);
188                 HERED(i);
189                 //imgPrint(imgs[i]);
190             }
191         }
192         deleteImg(imgs[i]);
193     }
194     *maxCorrelation=convVal[maxIdx];
195     *lowerBound=lowerBoundOffset[maxIdx];
196     *upperBound=upperBoundOffset[maxIdx];
197     return 0;
198 }
```

References img::data, deleteFilter(), deleteImg(), ERROR, gridDumpDebugInfo, gridIdentifyNPointsCounter, img::height, HERE, HERED, filter::img, imgConvolutionDiff(), imgInvert(), imgWrite(), newFilter(), newImgNDotsHori(), filter::weight, and img::width.

4.13.1.3 gridLocate()

```
int gridLocate (
            Img * img,
            int * xmin,
```

```
            int * ymin,
            int * xmax,
            int * ymax )
```

Locates a sudoku grid in a picture.

This function calls gridVertHoriConvo to perform a vertical and horizontal convolution using filters from gridGet↩ LayerHoriVertFilters. The verification that we get 10 lines horizontally or vertically is performed by calling function gridIdentifyNPoints.

Parameters

img	picture in which we are looking.
xmin	pointer to an integer where the horizontal position of the lower left position of the grid will be written.
ymin	pointer to an integer where the vertical position of the lower left position of the grid will be written.
xmax	pointer to an integer where the horizontal position of the upper right position of the grid will be written.
ymax	pointer to an integer where the vertical position of the upper right position of the grid will be written.

Returns

0 if a grid had been found, a number between 1 and 255 otherwise.

See also

gridGetLayerHoriVertFilters

gridVertHoriConvo

gridIdentifyNPoints

```
226 {
227     Img *layer1HO,*layer1VO;
228     gridVertHoriConvo(&layer1HO,&layer1VO,
229                       img,
230                       img,
231                       1,
232                       50, /* length */
233                       0,  /* threshold */
234                       1,1);
235 
236 
237     if (gridDumpDebugInfo) {
238         imgWrite(layer1HO,"gridLayer1OutputVert.png");
239         imgWrite(layer1VO,"gridLayer1OutputHori.png");
240     }
241     Img *layer5HO,*layer5VO;
242     layer5HO=layer1HO;
243     layer5VO=layer1VO;
244     int sumMinVal=8*img->height*0;
245     int sumMaxVal=64*img->height;
246     Img * flattenedHori = newImgColor(layer5VO->width,1,0);
247     for (int x=0;x<layer5VO->width;++x) {
248         int s=0;
249         for (int y=0;y<layer5VO->height;++y) {
250             s+=layer5VO->data[x+layer5VO->width*y];
251         }
252         if (s<sumMinVal) s=0;
253         else if (s>sumMaxVal) s=255;
254         else s=255*(s-sumMinVal)/(sumMaxVal-sumMinVal);
255         flattenedHori->data[x]=s;
256         printf("%d ",s);
257     }
258 
259     if (gridDumpDebugInfo) {
260         imgWrite(flattenedHori,"flattenedHori.png");
261 
262     }
263     printf("\n");
264     sumMinVal=8*img->width;
265     sumMaxVal=64*img->width;
266     Img * flattenedVert = newImgColor(layer5VO->height,1,0);
```

```
267     for (int y=0;y<layer5HO->height;++y) {
268         int s=0;
269         for (int x=0;x<layer5HO->width;++x) {
270             s+=layer5HO->data[x+layer5VO->width*y];
271         }
272         if (s<sumMinVal) s=0;
273         else if (s>sumMaxVal) s=255;
274         else s=255*(s-sumMinVal)/(sumMaxVal-sumMinVal);
275         flattenedVert->data[y]=s;
276         printf("%d ",s);
277     }
278 
279     if (gridDumpDebugInfo) {
280         imgWrite(flattenedVert,"flattenedVert.png");
281     }
282     printf("\n");
283 
284     // the 10 equidistant point should spread on the minimum
285     // between img->width and img->height and that value
286     // divided by 2.
287     int minLength = img->width;
288     if (img->height<minLength) minLength=img->height;
289     int endRange=minLength;
290     int startRange=endRange/2;
291     int maxCorrelationVer=0;
292     gridIdentifyNPoints(10,flattenedVert,startRange, endRange,ymin,ymax,
293                         &maxCorrelationVer);
294 
295     if (gridDumpDebugInfo) {
296         HERE("+++++++++++++++");
297     }
298     int maxCorrelationHori=0;
299     gridIdentifyNPoints(10,flattenedHori,startRange, endRange,xmin,xmax,
300                         &maxCorrelationHori);
301     HERE("maxCorrelationHori");
302     HERED(maxCorrelationHori);
303     HERE("maxCorrelationVer");
304     HERED(maxCorrelationVer);
305 
306     deleteImg(flattenedVert);
307     deleteImg(flattenedHori);
308 
309     deleteImg(layer1HO);
310     deleteImg(layer1VO);
311     return 0;
312 }
```

References img::data, deleteImg(), gridDumpDebugInfo, gridIdentifyNPoints(), gridVertHoriConvo(), img::height, HERE, HERED, imgWrite(), newImgColor(), and img::width.

4.13.1.4 gridMain()

```
int gridMain (
              int argc,
              char ** argv )
```

main function to execute when the grid executable is called from the command line.

Parameters

argc	number of arguments on the command line.
argv	value of arguments on the command line.

```
336                                 {
337     for (int j=1;j<argc;++j) {
338         if (strcmp(argv[j],"-h")==0 || strcmp(argv[j],"--help")==0) {
339             gridUsage(stdout,argv[0]);
340             exit(0);
341         }
342     }
343     if(argc < 3) {
344         gridUsage(stderr,argv[0]);
```

```
345         ERROR("At least 2 argumens expected, the picture to read, and where tp write the output","");
346     }
347     /* read input image */
348     Img * currentImage=newImgRead(argv[1]);
349     int xmin,ymin,xmax,ymax;
350     // locate the grid
351     if (!gridLocate(currentImage,&xmin,&ymin,&xmax,&ymax)) {
352         HERE("Found sudoku grid :");
353         printf("%d %d %d %d\n",xmin,ymin,xmax,ymax);
354         // draw the grid found
355         imgDrawRect(currentImage,xmin,ymin,xmax,ymax);
356         // write the image
357         imgWrite(currentImage,argv[2]);
358     } else {
359         ERROR("Could not find a grid in picture: ",argv[1]);
360     }
361     return 0;
362 }
```

References ERROR, gridLocate(), gridUsage(), HERE, imgDrawRect(), imgWrite(), and newImgRead().

4.13.1.5 gridUsage()

```
void gridUsage (
             FILE * f,
             char * name )
```

Tells how to use this program.

Parameters

f	where to write the info.
name	the value of argv[0].

```
319                                 {
320     char * bname=basename(name);
321     fprintf(f,"%s usage:\n", bname);
322     fprintf(f,"    %s [options] <input-file> <output-file>\n", bname);
323     fprintf(f,"Locates a sudoku grid in an image.:\n");
324     fprintf(f,"\n");
325     fprintf(f,"Where option is one of:\n");
326     fprintf(f,"    [-h|--help] :\n");
327     fprintf(f,"         Displays this help message and leaves.\n");
328 }
```

4.13.1.6 gridVertHoriConvo()

```
void gridVertHoriConvo (
             Img ** outputH,
             Img ** outputV,
             Img * inputH,
             Img * inputV,
             int width,
             int length,
             int threshold,
             int poolsize,
             int stride )
```

Perform vertical and horizontal convolution.

This function should be called several times until convolution only keeps the sudoku grid lines.

Parameters

outputH	newly allocated image for horizontal convolution.
outputV	newly allocated image for vertical convolution.
inputH	current image for horizontal convolution.
inputV	current image for vertical convolution.
width	width of convolution
length	length of convolution
poolsize	pool size to reduce image (1 for no reduction)
stride	step to jump to decuce image (1 for no reduction)

```
72 {
73     FilterFam * filtersVert=
74         gridGetLayerHoriVertFilters(0,length,width,threshold);
75     FilterFam * filtersHori=
76         gridGetLayerHoriVertFilters(1,length,width,threshold);
77 
78     ImgFam * layerVertOutput =
79         filterFamApplyConvolutionSameSizeDiff(filtersVert,inputV);
80     ImgFam * layerHoriOutput =
81         filterFamApplyConvolutionSameSizeDiff(filtersHori,inputH);
82     deleteFilterFam(filtersVert);
83     deleteFilterFam(filtersHori);
84 
85     ImgFam * layerMaxPoolVertOutput=
86         imgFamDownSampleMax(layerVertOutput,poolsize,stride);
87     ImgFam * layerMaxPoolHoriOutput=
88         imgFamDownSampleMax(layerHoriOutput,poolsize,stride);
89     deleteImgFam(layerVertOutput);
90     deleteImgFam(layerHoriOutput);
91 
92     ImgFam * betterContrastHori2 =
93         imgFamLuminosityScale(layerMaxPoolHoriOutput);
94     ImgFam * betterContrastVert2 =
95         imgFamLuminosityScale(layerMaxPoolVertOutput);
96     deleteImgFam(layerMaxPoolVertOutput);
97     deleteImgFam(layerMaxPoolHoriOutput);
98 
99     if (betterContrastHori2->count!=1 ||
100         betterContrastVert2->count!=1) {
101         ERROR("Wrong size.","");
102     }
103     // write address the filter to return to caller
104     *outputV=betterContrastVert2->imgs[0];
105     *outputH=betterContrastHori2->imgs[0];
106     // pretend there is no filter in the filterFam
107     betterContrastHori2->count=0;
108     betterContrastVert2->count=0;
109     // delete filterFam allocated data
110     deleteImgFam(betterContrastHori2);
111     deleteImgFam(betterContrastVert2);
112 }
```

References imgfam::count, deleteFilterFam(), deleteImgFam(), ERROR, filterFamApplyConvolutionSameSizeDiff(), gridGetLayerHoriVertFilters(), imgFamDownSampleMax(), imgFamLuminosityScale(), and imgfam::imgs.

4.13.2 Variable Documentation

4.13.2.1 gridDumpDebugInfo

```
int gridDumpDebugInfo =1
```

4.13.2.2 gridIdentifyNPointsCounter

```
int gridIdentifyNPointsCounter =0
```

4.14 grid.h File Reference

Typedefs

- typedef struct img Img

Functions

- int gridLocate (Img *img, int *xmin, int *ymin, int *xmax, int *ymax)

  Locates a sudoku grid in a picture.
- int gridMain (int argc, char **argv)

  main function to execute when the grid executable is called from the command line.

4.14.1 Typedef Documentation

4.14.1.1 Img

```
typedef struct img Img
```

4.14.2 Function Documentation

4.14.2.1 gridLocate()

```
int gridLocate (
            Img * img,
            int * xmin,
            int * ymin,
            int * xmax,
            int * ymax )
```

Locates a sudoku grid in a picture.

This function calls gridVertHoriConvo to perform a vertical and horizontal convolution using filters from gridGet↩ LayerHoriVertFilters. The verification that we get 10 lines horizontally or vertically is performed by calling function gridIdentifyNPoints.

Parameters

img	picture in which we are looking.
xmin	pointer to an integer where the horizontal position of the lower left position of the grid will be written.
ymin	pointer to an integer where the vertical position of the lower left position of the grid will be written.
xmax	pointer to an integer where the horizontal position of the upper right position of the grid will be written.
ymax	pointer to an integer where the vertical position of the upper right position of the grid will be written.

Returns

0 if a grid had been found, a number between 1 and 255 otherwise.

See also

gridGetLayerHoriVertFilters

gridVertHoriConvo

gridIdentifyNPoints

```
226 {
227     Img *layer1HO,*layer1VO;
228     gridVertHoriConvo(&layer1HO,&layer1VO,
229                       img,
230                       img,
231                       1,
232                       50, /* length */
233                       0,  /* threshold */
234                       1,1);
235 
236 
237     if (gridDumpDebugInfo) {
238         imgWrite(layer1HO,"gridLayer1OutputVert.png");
239         imgWrite(layer1VO,"gridLayer1OutputHori.png");
240     }
241     Img *layer5HO,*layer5VO;
242     layer5HO=layer1HO;
243     layer5VO=layer1VO;
244     int sumMinVal=8*img->height*0;
245     int sumMaxVal=64*img->height;
246     Img * flattenedHori = newImgColor(layer5VO->width,1,0);
247     for (int x=0;x<layer5VO->width;++x) {
248         int s=0;
249         for (int y=0;y<layer5VO->height;++y) {
250             s+=layer5VO->data[x+layer5VO->width*y];
251         }
252         if (s<sumMinVal) s=0;
253         else if (s>sumMaxVal) s=255;
254         else s=255*(s-sumMinVal)/(sumMaxVal-sumMinVal);
255         flattenedHori->data[x]=s;
256         printf("%d ",s);
257     }
258 
259     if (gridDumpDebugInfo) {
260         imgWrite(flattenedHori,"flattenedHori.png");
261 
262     }
263     printf("\n");
264     sumMinVal=8*img->width;
265     sumMaxVal=64*img->width;
266     Img * flattenedVert = newImgColor(layer5VO->height,1,0);
267     for (int y=0;y<layer5HO->height;++y) {
268         int s=0;
269         for (int x=0;x<layer5HO->width;++x) {
270             s+=layer5HO->data[x+layer5VO->width*y];
271         }
272         if (s<sumMinVal) s=0;
273         else if (s>sumMaxVal) s=255;
274         else s=255*(s-sumMinVal)/(sumMaxVal-sumMinVal);
275         flattenedVert->data[y]=s;
276         printf("%d ",s);
277     }
278 
279     if (gridDumpDebugInfo) {
280         imgWrite(flattenedVert,"flattenedVert.png");
281     }
```

```
282     printf("\n");
283
284     // the 10 equidistant point should spread on the minimum
285     // between img->width and img->height and that value
286     // divided by 2.
287     int minLength = img->width;
288     if (img->height<minLength) minLength=img->height;
289     int endRange=minLength;
290     int startRange=endRange/2;
291     int maxCorrelationVer=0;
292     gridIdentifyNPoints(10,flattenedVert,startRange, endRange,ymin,ymax,
293                         &maxCorrelationVer);
294
295     if (gridDumpDebugInfo) {
296         HERE("+++++++++++++++");
297     }
298     int maxCorrelationHori=0;
299     gridIdentifyNPoints(10,flattenedHori,startRange, endRange,xmin,xmax,
300                         &maxCorrelationHori);
301     HERE("maxCorrelationHori");
302     HERED(maxCorrelationHori);
303     HERE("maxCorrelationVer");
304     HERED(maxCorrelationVer);
305
306     deleteImg(flattenedVert);
307     deleteImg(flattenedHori);
308
309     deleteImg(layer1HO);
310     deleteImg(layer1VO);
311     return 0;
312 }
```

References img::data, deleteImg(), gridDumpDebugInfo, gridIdentifyNPoints(), gridVertHoriConvo(), img::height, HERE, HERED, imgWrite(), newImgColor(), and img::width.

4.14.2.2 gridMain()

```
int gridMain (
              int argc,
              char ** argv )
```

main function to execute when the grid executable is called from the command line.

Parameters

argc	number of arguments on the command line.
argv	value of arguments on the command line.

```
336                                 {
337     for (int j=1;j<argc;++j) {
338         if (strcmp(argv[j],"-h")==0 || strcmp(argv[j],"--help")==0) {
339             gridUsage(stdout,argv[0]);
340             exit(0);
341         }
342     }
343     if(argc < 3) {
344         gridUsage(stderr,argv[0]);
345         ERROR("At least 2 argumens expected, the picture to read, and where tp write the output","");
346     }
347     /* read input image */
348     Img * currentImage=newImgRead(argv[1]);
349     int xmin,ymin,xmax,ymax;
350     // locate the grid
351     if (!gridLocate(currentImage,&xmin,&ymin,&xmax,&ymax)) {
352         HERE("Found sudoku grid :");
353         printf("%d %d %d %d\n",xmin,ymin,xmax,ymax);
354         // draw the grid found
355         imgDrawRect(currentImage,xmin,ymin,xmax,ymax);
356         // write the image
357         imgWrite(currentImage,argv[2]);
358     } else {
359         ERROR("Could not find a grid in picture: ",argv[1]);
```

```
360     }
361     return 0;
362 }
```

References ERROR, gridLocate(), gridUsage(), HERE, imgDrawRect(), imgWrite(), and newImgRead().

4.15 grid.h

Go to the documentation of this file.

```
1 #ifndef GRID_H
2 #define GRID_H
3
4 typedef struct img Img;
5
6 int gridLocate(Img * img,
7                int * xmin, int * ymin,
8                int * xmax, int * ymax);
9 int gridMain(int argc,char**argv);
10
11 #endif
```

4.16 img.c File Reference

```
#include <libpng16/png.h>
#include <math.h>
#include <libgen.h>
#include <jpeglib.h>
#include <setjmp.h>
#include "img.h"
#include "filter.h"
```

Classes

- struct jpegerrmgr

Macros

- #define NOT_ENOUGH

Typedefs

- typedef struct jpegerrmgr * MyErrorPtr

  Pointer to struct jpegerrmgr.

Functions

- Img * newImgFromArray (int w, int h, unsigned char *buffer)

  Creates an image from an array of unsigned char.
- Img * newImgColor (int w, int h, unsigned char c)

  create an image of a given color
- Img * newImgVerticalBar (int s, int w)

  create an image of a vertical black bar on a square white background
- Img * newImgVerticalBarInRect (int sx, int sy, int w)

  create an image of a vertical black bar on a rectangular white background
- Img * newImg33edge ()

  create a 3x3 image to perform edge detection.
- Img * newImgCross (int s, int w, int t)

  create an image with a black cross on a w by w white background.
- Img * newImgSquare (int s, int w, int t)

  create an image with a black square on a w by w white background.
- Img * newImg9By9Dots (int w)

  create an image with 9x9 black dots on a picture of size w times w.
- Img * newImgNDotsHori (int N, int w)

  Creates an image with equaly separated N black dots on a picture of size w by 1 pixel.
- Img * newImgSudoku (int sz)

  create an image of a given color
- void my_error_exit (j_common_ptr cinfo)

  used to manage jpeg errors.
- static Img * newImgReadJpeg (char *filename)

  create an image from a jpeg file
- static Img * newImgReadPng (char *filename)

  create an image from a png file
- Img * newImgRead (char *filename)

  create an image from a png or jpeg file
- Img * newImgCopy (Img *myImg)

  create an image from an exiting Img instance
- void deleteImg (Img *myImg)

  Deletes an existing image.
- unsigned char imgGetVal (Img *p, int x, int y)

  Get the value a pixel for a given color. Zero is returned if the pixel is out of the picture.
- void imgPrint (Img *myImg)

  Displays an image with numerical values.
- void imgDrawRect (Img *myImg, int xmin, int ymin, int xmax, int ymax)

  draws a rectangle on the image
- Img * imgExtract (Img *myImg, int xmin, int ymin, int xmax, int ymax)

  Extract an image from and image.
- void imgWrite (Img *myImg, char *filename)

  Writes a Img struct to a png file.
- int imgGetWeight (Img *in)

  Return the sum of all pixels in the picture.
- Img * imgInvert (Img *in)

  Inverts an image.
- Img * imgFlattenContrast (Img *in)

  Flattens the contrast of an image We perform that by raising to the square the normalized difference with 128.
- Img * imgRaiseContrast (Img *in)

Raises the contrast of an image We perform that by computing the square root of normalized difference with 128.

- Img * imgMake3dEffect (Img *in)
- Img * imgScale (Img *in, int s)

  Scales an image to a larger image.
- Img * imgBlur (Img *in, int radius)

  Blurs a picture.
- Img * imgLuminosityScale (Img *in)

  Spread luminosity in the picture.
- Img * imgConvolution (Img *in, Filter *filter)

  perform a convolution between an image and a filter with unsigned char.
- Img * imgConvolutionDiff (Img *in, Filter *filter)

  perform a convolution between an image and a filter with unsigned char.
- Img * imgConvolutionSameSize (Img *in, Filter *filter)

  perform a convolution between an image and a filter applying a threshold given by intFilter and only positive numbers.
- Img * imgConvolutionSameSizeDiff (Img *in, Filter *filter)

  perform a convolution between an image and a filter applying a threshold given by intFilter.
- Img * imgDownSampleMax (Img *img, int poolsize, int stride)

  Downsample an image using a maxpool strategy.
- Img * imgDivideByTwo (Img *img)

  Divide the size of an image by two.
- Img * imgRotate90 (Img *img)

  Rotates an image by 90 degrees.
- Img * imgRotate (Img *img, int deg)

  Rotates an image.
- Img * imgDownSampleAvg (Img *img, int poolsize, int stride)

  Downsample an image using an average pool strategy.
- unsigned char imgScalar (Img *i1, Img *i2, int xoffset, int yoffset)

  Given a large image i1 and a smaller one i2 compute the scaler product of i2 with the subset of i1 at offset (xoffset,yoffset).
- void imgUsage (FILE *f, char *name)

  Tells how to use this program.
- int imgMain (int argc, char *argv[])

  What to do when directly called from the command line.

4.16.1 Macro Definition Documentation

4.16.1.1 NOT_ENOUGH

```
#define NOT_ENOUGH
```

Value:

```
    if (i>=argc) {                                  \
        imgUsage(stderr,argv[0]);                   \
        ERROR("not enough arguments.","");          \
    }
```

4.16.2 Typedef Documentation

4.16.2.1 MyErrorPtr

```
typedef struct jpegerrmgr* MyErrorPtr
```

Pointer to struct jpegerrmgr.

4.16.3 Function Documentation

4.16.3.1 deleteImg()

```
void deleteImg (
            Img * myImg )
```

Deletes an existing image.

Parameters

myImg	an existing image to delete Nothing happens if myImg is NULL.

```
482                         {
483     if (myImg==NULL) return;
484     free(myImg->data);
485     myImg->width=0;
486     myImg->height=0;
487     myImg->data=NULL;
488     free(myImg);
489 }
```

References img::data, img::height, and img::width.

4.16.3.2 imgBlur()

```
Img * imgBlur (
            Img * in,
            int radius )
```

Blurs a picture.

Parameters

in	the picture to blur
radius	intensity of the blur

Returns

the newly allocated picture.

```
743                                 {
744     Img * answer = newImgCopy(in);
```

```
745     int sq=radius*radius;
746     for(int y = 0; y < in->height; y++) {
747         for(int x = 0; x < in->width; x++) {
748             int v=0;
749             for (int xx=-radius;xx<radius;++xx) {
750                 for (int yy=-radius;yy<radius;++yy) {
751                     if (xx*yy<sq)
752                         v+=imgGetVal(in,x+xx,y+yy);
753                 }
754             }
755             answer->data[x + y * in->width]=INBYTE((int)(v/sq/3.14));
756         }
757     }
758     return answer;
759 }
```

References img::data, img::height, imgGetVal(), INBYTE, newImgCopy(), and img::width.

4.16.3.3 imgConvolution()

```
Img * imgConvolution (
              Img * in,
              Filter * filter )
```

perform a convolution between an image and a filter with unsigned char.

The resulting image size is in->width-filter->img->width+1 by in->height-filter->img->height+1.

Parameters

in	the input picture
filter	the filter to use

Returns

the newly allocated picture with represents the convolution.

```
800                                         {
801     if (in->width<filter->img->width)
802         ERROR("Wrong width","");
803     if (in->height<filter->img->height)
804         ERROR("Wrong height","");
805     int aw=in->width-filter->img->width+1;
806     int ah=in->height-filter->img->height+1;
807     Img * answer = newImgColor(aw,ah,0);
808     for(int y = 0; y < ah; y++) {
809         for(int x = 0; x < aw; x++) {
810             long int v=0;
811             for(int yy = 0; yy < filter->img->height; yy++) {
812                 for(int xx = 0; xx < filter->img->width; xx++) {
813                     v+=imgGetVal(in,x+xx,y+yy)*imgGetVal(filter->img,xx,yy);
814                 }
815             }
816             answer->data[x+aw*y]=
817                 (v<filter->threshold)?0
818                 :(v>filter->maxVal)?255
819                 :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
820         }
821     }
822     return answer;
823 }
```

References img::data, ERROR, img::height, filter::img, imgGetVal(), filter::maxVal, newImgColor(), filter::threshold, and img::width.

4.16.3.4 imgConvolutionDiff()

```
Img * imgConvolutionDiff (
              Img * in,
              Filter * filter )
```

perform a convolution between an image and a filter with unsigned char.

The resulting image size is in->width-filter->img->width+1 by in->height-filter->img->height+1.

Parameters

in	the input picture
filter	the filter to use

Returns

the newly allocated picture with represents the convolution.

```
835                                                 {
836     if (in->width<filter->img->width)
837         ERROR("Wrong width","");
838     if (in->height<filter->img->height)
839         ERROR("Wrong height","");
840     int aw=in->width-filter->img->width+1;
841     int ah=in->height-filter->img->height+1;
842     Img * answer = newImgColor(aw,ah,0);
843     for(int y = 0; y < ah; y++) {
844         for(int x = 0; x < aw; x++) {
845             long int v=0;
846             for(int yy = 0; yy < filter->img->height; yy++) {
847                 for(int xx = 0; xx < filter->img->width; xx++) {
848                     v+=((int)imgGetVal(in,x+xx,y+yy))*
849                         (((int)imgGetVal(filter->img,xx,yy))-128);
850                 }
851             }
852             answer->data[x+aw*y]=
853                 (v<filter->threshold)?0
854                 :(v>filter->maxVal)?255
855                 :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
856         }
857     }
858     return answer;
859 }
```

References img::data, ERROR, img::height, filter::img, imgGetVal(), filter::maxVal, newImgColor(), filter::threshold, and img::width.

4.16.3.5 imgConvolutionSameSize()

```
Img * imgConvolutionSameSize (
              Img * in,
              Filter * filter )
```

perform a convolution between an image and a filter applying a threshold given by intFilter and only positive numbers.

The resulting image size is in->width by in->height.

Parameters

in	the input picture
filter	the filter to use

Returns

the newly allocated picture with represents the convolution.

```
870                                                 {
871     if (in->width<filter->img->width || in->height<filter->img->height)
872         ERROR("Wrong size","");
873     int wfOver2=filter->img->width/2;
874     int hfOver2=filter->img->height/2;
875     Img * answer = newImgColor(in->width,in->height,0);
876     for(int y = 0; y < in->height; y++) {
877         for(int x = 0; x < in->width; x++) {
878             long int v=0;
879             for(int yy = 0; yy < filter->img->height; yy++) {
880                 for(int xx = 0; xx < filter->img->width; xx++) {
881                     int xxx=x+xx-wfOver2;
882                     int yyy=y+yy-hfOver2;
883                     if (xxx>=0 && yyy>=0 && xxx<in->width && yyy<in->height) {
884                         v+=imgGetVal(in,xxx,yyy)*imgGetVal(filter->img,xx,yy);
885                     }
886                 }
887             }
888             //HERE("___v,maxval,threshold,percent__________________");
889             //HERED((int)v);
890             //HERED((int)filter->maxVal);
891             //HERED((int)filter->threshold);
892             //HERED((int)filter->percent);
893             answer->data[x+in->width*y]=
894                 (v<filter->threshold)?0
895                 :(v>filter->maxVal)?255
896                 :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
897         }
898     }
899     return answer;
900 }
```

References img::data, ERROR, img::height, filter::img, imgGetVal(), filter::maxVal, newImgColor(), filter::threshold, and img::width.

4.16.3.6 imgConvolutionSameSizeDiff()

```
Img * imgConvolutionSameSizeDiff (
            Img * in,
            Filter * filter )
```

perform a convolution between an image and a filter applying a threshold given by intFilter.

The resulting image size is in->width by in->height.

Parameters

in	the input picture
filter	the filter to use

Returns

the newly allocated picture with represents the convolution.

```
910                                                 {
911     if (in->width<filter->img->width || in->height<filter->img->height)
912         ERROR("Wrong size","");
913     int wfOver2=filter->img->width/2;
914     int hfOver2=filter->img->height/2;
915     Img * answer = newImgColor(in->width,in->height,0);
916     for(int y = 0; y < in->height; y++) {
917         for(int x = 0; x < in->width; x++) {
918             long int v=0;
919             for(int yy = 0; yy < filter->img->height; yy++) {
920                 for(int xx = 0; xx < filter->img->width; xx++) {
921                     int xxx=x+xx-wfOver2;
922                     int yyy=y+yy-hfOver2;
923                     if (xxx>=0 && yyy>=0 && xxx<in->width && yyy<in->height) {
924 
925                         v+=((int)imgGetVal(in,xxx,yyy))*
926                             (((int)imgGetVal(filter->img,xx,yy))-128);
927                         /*
928                         int d = imgGetVal(in,xxx,yyy)
929                             -imgGetVal(filter->img,xx,yy);
930                         v+=d*d;
931                         */
932                     }
933                 }
934             }
935             /*
936             HERE("___v,maxval,threshold,percent__________________");
937             HERED((int)v);
938             HERED((int)filter->maxVal);
939             HERED((int)filter->threshold);
940             HERED((int)filter->percent);
941             */
942 
943             answer->data[x+in->width*y]=
944                 (v<filter->threshold)?0
945                 :(v>filter->maxVal)?255
946                 :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
947         }
948     }
949     return answer;
950 }
```

References img::data, ERROR, img::height, filter::img, imgGetVal(), filter::maxVal, newImgColor(), filter::threshold, and img::width.

4.16.3.7 imgDivideByTwo()

```
Img * imgDivideByTwo (
            Img * img )
```

Divide the size of an image by two.

Parameters

img	an image

Returns

same image as img but with a size scaled down by 2.

```
990                                 {
991     Img * answer = newImgColor(img->width/2,img->height/2,255);
992     for (int y=0;y<img->height/2;++y) {
993         int o = (img->width»1)*y;
994         for (int x=0;x<img->width/2;++x) {
995             // compute the average of 4 pixels
996             answer->data[x+o] =
997                 (img->data[(x«1)+img->width*(y«1)] +
998                  img->data[(x«1)+1+img->width*(y«1)] +
999                  img->data[(x«1)+img->width*((y«1)+1)] +
```

```
1000                 img->data[(x«1)+1+img->width*((y«1)+1)] )»2;
1001         }
1002     }
1003     return answer;
1004 }
```

References img::data, img::height, newImgColor(), and img::width.

4.16.3.8 imgDownSampleAvg()

```
Img * imgDownSampleAvg (
              Img * img,
              int poolsize,
              int stride )
```

Downsample an image using an average pool strategy.

Parameters

img	the image to down scale.
poolsize	size of the square on which the average is computed.
stride	number of pixel by which the square on which the average is computed is moving at each step.

Returns

the down sampled image.

```
1085                                                    {
1086     if (img->width < poolsize)
1087         ERROR("Wrong size.","");
1088     if (img->height < poolsize)
1089         ERROR("Wrong size.","");
1090     int w =(img->width - poolsize)/stride+1;
1091     int h =(img->height - poolsize)/stride+1;
1092     Img* answer = newImgColor(w,h,0);
1093     for (int x=0;x<w;++x) {
1094         for (int y=0;y<h;++y) {
1095             int avg=0;
1096             for (int xx=0;xx<poolsize;++xx) {
1097                 for (int yy=0;yy<poolsize;++yy) {
1098                     avg+=img->data[x*stride+xx+img->width*(y*stride+yy)];
1099                 }
1100             }
1101             answer->data[x+w*y]=avg/poolsize/poolsize;
1102         }
1103     }
1104     return answer;
1105 }
```

References img::data, ERROR, img::height, newImgColor(), and img::width.

4.16.3.9 imgDownSampleMax()

```
Img * imgDownSampleMax (
              Img * img,
              int poolsize,
              int stride )
```

Downsample an image using a maxpool strategy.

Parameters

img	the image to down scale.
poolsize	size of the square on which the maximum is computed.
stride	number of pixel by which the square on which the maximum is computed is moving at each step.

Returns

the down sampled image.

```
961                                                     {
962     if (img->width < poolsize)
963         ERROR("Wrong size, poolsize too large.","");
964     if (img->height < poolsize)
965         ERROR("Wrong size, poolsize too large.","");
966     int w =(img->width - poolsize)/stride+1;
967     int h =(img->height - poolsize)/stride+1;
968     Img* answer = newImgColor(w,h,0);
969     for (int x=0;x<w;++x) {
970         for (int y=0;y<h;++y) {
971             int max=0;
972             for (int xx=0;xx<poolsize;++xx) {
973                 for (int yy=0;yy<poolsize;++yy) {
974                     int v=img->data[x*stride+xx+img->width*(y*stride+yy)];
975                     if (v>max)
976                         max=v;
977                 }
978             }
979             answer->data[x+w*y]=max;
980         }
981     }
982     return answer;
983 }
```

References img::data, ERROR, img::height, newImgColor(), and img::width.

4.16.3.10 imgDrawRect()

```
void imgDrawRect (
              Img * myImg,
              int xmin,
              int ymin,
              int xmax,
              int ymax )
```

draws a rectangle on the image

Parameters

myImg	image on which to draw the rectangle
xmin	horizontal position of lower left corner
ymin	vertical position of lower left corner
xmax	horizontal position of upper right corner
ymax	vertical position of upper right corner

```
526                                                                 {
527     if (xmin<0 || ymin<0 || xmax >myImg->width || myImg->height<ymax ||
528         xmin>=xmax || ymin>=ymax ) {
529         fprintf(stderr,"rectangle : %d %d %d %d\n",xmin,ymin,xmax,ymax);
530         fprintf(stderr,"image size is %d %d\n",myImg->width,myImg->height);
531         ERROR("Wrong size.","");
```

```
532     }
533     for(int x = xmin; x <xmax; x++) {
534         myImg->data[x+myImg->width*ymin]=(x»2)%2?255:0;
535         myImg->data[x+myImg->width*ymax]=(x»2)%2?255:0;
536     }
537     for(int y = ymin; y <ymax; y++) {
538         myImg->data[xmin+myImg->width*y]=(y»2)%2?255:0;
539         myImg->data[xmax+myImg->width*y]=(y»2)%2?255:0;
540     }
541 }
```

References img::data, ERROR, img::height, and img::width.

4.16.3.11 imgExtract()

```
Img * imgExtract (
              Img * myImg,
              int xmin,
              int ymin,
              int xmax,
              int ymax )
```

Extract an image from and image.

Parameters

myImg	image from which we extract the image.
xmin	horizontal position of lower left corner
ymin	vertical position of lower left corner
xmax	horizontal position of upper right corner
ymax	vertical position of upper right corner

Returns

the newly allocated image.

```
552                                                                 {
553     if (xmin<0 || ymin<0 || xmax >myImg->width || myImg->height<ymax ||
554         xmin>=xmax || ymin>=ymax ) {
555         fprintf(stderr,"rectangle : %d %d %d %d\n",xmin,ymin,xmax,ymax);
556         fprintf(stderr,"image size is %d %d\n",myImg->width,myImg->height);
557         ERROR("Wrong size.","");
558     }
559     int w=xmax-xmin;
560     int h=ymax-ymin;
561     Img * answer = newImgColor(w,h,255);
562     for(int y = 0; y<h; y++) {
563         // use memcopy to be faster since horizontal pixels
564         // are stored one next to the other.
565         memcpy(&answer->data[0+y*w],
566                &myImg->data[xmin+myImg->width*(ymin+y)],
567                w);
568     }
569     return answer;
570 }
```

References img::data, ERROR, img::height, newImgColor(), and img::width.

4.16.3.12 imgFlattenContrast()

```
Img * imgFlattenContrast (
            Img * in )
```

Flattens the contrast of an image We perform that by raising to the square the normalized difference with 128.

Parameters

in	the input image to flatten.

Returns

a newly allocated image which is flattened.

```
674                                 {
675     Img * answer = newImgCopy(in);
676     for(int i = 0; i < in->height*in->width; i++) {
677         int m = in->data[i];
678         float v = (m-128.0)/128.;
679         float w = v*v;
680         answer->data[i]=v>0?INBYTE(w*128+128):INBYTE(128-w*128);
681     }
682     return answer;
683 }
```

References img::data, img::height, INBYTE, newImgCopy(), and img::width.

4.16.3.13 imgGetVal()

```
unsigned char imgGetVal (
            Img * p,
            int x,
            int y )
```

Get the value a pixel for a given color. Zero is returned if the pixel is out of the picture.

Parameters

p	the picture
x	the horizontal position
y	the vertical position
ch	the channel/color : 0 for red, 1 for green, 2 for blue.

Returns

the value of the pixel.

```
501                                         {
502     return (y<0 || y>=p->height)?0:(x<0 || x>=p->width)?0:p->data[y*p->width+x];
503 }
```

References img::data, img::height, and img::width.

4.16.3.14 imgGetWeight()

```
int imgGetWeight (
            Img * in )
```

Return the sum of all pixels in the picture.

Parameters

in	a picture

Returns

the sum of all pixels.

```
646                         {
647     int answer =0;
648     for(int i = 0; i < in->height*in->width; i++) {
649         answer+=in->data[i];
650     }
651     return answer;
652 }
```

References img::data, img::height, and img::width.

4.16.3.15 imgInvert()

```
Img * imgInvert (
            Img * in )
```

Inverts an image.

Parameters

in	the input image to invert.

Returns

a newly allocated image which is the invese of image in.

```
659                         {
660     Img * answer = newImgCopy(in);
661     for(int i = 0; i < in->height*in->width; i++) {
662         answer->data[i]=255-in->data[i];
663     }
664     return answer;
665 }
```

References img::data, img::height, newImgCopy(), and img::width.

4.16.3.16 imgLuminosityScale()

```
Img * imgLuminosityScale (
            Img * in )
```

Spread luminosity in the picture.

Parameters

in	the picture to spread luminosity

Returns

the newly allocated picture with spreaded luminosity.

```
766                                 {
767     int lumCount[256];
768     memset(lumCount,0,sizeof(int)*8);
769     Img * answer = newImgCopy(in);
770     for(int i = 0; i < in->height*in->width; i++) {
771         lumCount[in->data[i]]++;
772     }
773     // average value is a celle of lumCount should
774     // be avg=(in->height*in->width/256
775     // we are going to suppress below and above avg/8
776     int threshold=(in->height*in->width)>>12; // divide by 256*8
777     if (threshold<2) threshold=2;
778     int topLum=255;
779     while (topLum>0 && lumCount[topLum]<threshold) --topLum;
780     int botLum=0;
781     while (botLum<topLum && lumCount[botLum]<threshold) ++botLum;
782     //topLum=100;
783     for(int i = 0; i < in->height*in->width; i++) {
784         answer->data[i]=
785             INBYTE((in->data[i]-botLum)*255/(topLum-botLum));
786     }
787     return answer;
788 }
```

References img::data, img::height, INBYTE, newImgCopy(), and img::width.

4.16.3.17 imgMain()

```
int imgMain (
              int argc,
              char * argv[] )
```

What to do when directly called from the command line.

Parameters

argc	number of arguments given
argv	value of arguments.

Returns

0 if no error occurs.

```
1226                                    {
1227     if (argc==1) {
1228         imgUsage(stdout,argv[0]);
1229         exit(0);
1230     }
1231     for (int j=1;j<argc;++j) {
1232         if (strcmp(argv[j],"-h")==0 || strcmp(argv[j],"--help")==0) {
1233             imgUsage(stdout,argv[0]);
1234             exit(0);
1235         }
1236     }
1237     int i=1;
1238     Img * currentImage=NULL;
```

```
1239     if (argv[i][0]!='-') {
1240         currentImage=newImgRead(argv[1]);
1241         i=2;
1242     }
1243 #define NOT_ENOUGH                                          \
1244     if (i>=argc) {                                          \
1245         imgUsage(stderr,argv[0]);                           \
1246         ERROR("not enough arguments.","");                  \
1247     }
1248
1249     Img * newImage=NULL;
1250     while (i<argc) {
1251         if (argv[i][0]!='-') {
1252             imgWrite(currentImage,argv[i]);
1253         } else {
1254             if (strcmp(argv[i],"--blur")==0 ||
1255                 strcmp(argv[i],"-b")==0 ) {
1256                 ++i;
1257                 NOT_ENOUGH;
1258                 int s=atoi(argv[i]);
1259                 newImage=imgBlur(currentImage,s);
1260             } else if (strcmp(argv[i],"--cross")==0) {
1261                 ++i;
1262                 NOT_ENOUGH;
1263                 int s=atoi(argv[i]);
1264                 ++i;
1265                 NOT_ENOUGH;
1266                 int w=atoi(argv[i]);
1267                 ++i;
1268                 NOT_ENOUGH;
1269                 int t=atoi(argv[i]);
1270                 newImage=newImgCross(s,w,t);
1271             } else if (strcmp(argv[i],"--square")==0) {
1272                 ++i;
1273                 NOT_ENOUGH;
1274                 int s=atoi(argv[i]);
1275                 ++i;
1276                 NOT_ENOUGH;
1277                 int w=atoi(argv[i]);
1278                 ++i;
1279                 NOT_ENOUGH;
1280                 int t=atoi(argv[i]);
1281                 newImage=newImgSquare(s,w,t);
1282             } else if (strcmp(argv[i],"--conv")==0) {
1283                 ++i;
1284                 NOT_ENOUGH;
1285                 char * fileName = argv[i];
1286                 ++i;
1287                 NOT_ENOUGH;
1288                 int percent=atoi(argv[i]);
1289                 Filter * f =newFilter(newImgRead(fileName),percent);
1290                 newImage=imgConvolutionSameSizeDiff(currentImage,f);
1291                 deleteFilter(f);
1292             } else if (strcmp(argv[i],"--9x9dots")==0) {
1293                 ++i;
1294                 NOT_ENOUGH;
1295                 int w=atoi(argv[i]);
1296                 newImage=newImg9By9Dots(w);
1297             } else if (strcmp(argv[i],"--3x3-edge")==0) {
1298                 newImage=newImg33edge();
1299             } else if (strcmp(argv[i],"--1d-dots")==0) {
1300                 ++i;
1301                 NOT_ENOUGH;
1302                 int N=atoi(argv[i]);
1303                 ++i;
1304                 NOT_ENOUGH;
1305                 int w=atoi(argv[i]);
1306                 newImage=newImgNDotsHori(N,w);
1307             } else if (strcmp(argv[i],"--vertical")==0 ||
1308                 strcmp(argv[i],"-v")==0 ) {
1309                 ++i;
1310                 NOT_ENOUGH;
1311                 int s=atoi(argv[i]);
1312                 ++i;
1313                 NOT_ENOUGH;
1314                 int w=atoi(argv[i]);
1315                 newImage=newImgVerticalBar(s,w);
1316             } else if (strcmp(argv[i],"--scale")==0 ||
1317                 strcmp(argv[i],"-s")==0 ) {
1318                 ++i;
1319                 NOT_ENOUGH;
1320                 int s=atoi(argv[i]);
1321                 newImage=imgScale(currentImage,s);
1322             } else if (strcmp(argv[i],"--rotate")==0 ||
1323                 strcmp(argv[i],"-r")==0 ) {
1324                 ++i;
1325                 NOT_ENOUGH;
```

```
1326                 int s=atoi(argv[i]);
1327                 newImage=imgRotate(currentImage,s);
1328             } else if (strcmp(argv[i],"--lum")==0 ||
1329                 strcmp(argv[i],"-l")==0 ) {
1330                 newImage=imgLuminosityScale(currentImage);
1331             } else if (strcmp(argv[i],"--inv")==0 ||
1332                 strcmp(argv[i],"-i")==0 ) {
1333                 newImage=imgInvert(currentImage);
1334             } else if (strcmp(argv[i],"--contrast")==0 ||
1335                 strcmp(argv[i],"-c")==0 ) {
1336                 newImage=imgRaiseContrast(currentImage);
1337             } else if (strcmp(argv[i],"--flatten")==0 ||
1338                 strcmp(argv[i],"-f")==0 ) {
1339                 newImage=imgFlattenContrast(currentImage);
1340             } else if (strcmp(argv[i],"--3d")==0 ||
1341                 strcmp(argv[i],"-3d")==0 ) {
1342                 newImage=imgMake3dEffect(currentImage);
1343             } else if (strcmp(argv[i],"--print")==0 ||
1344                 strcmp(argv[i],"-p")==0 ) {
1345                 imgPrint(currentImage);
1346             } else if (strcmp(argv[i],"--sudoku")==0) {
1347                 ++i;
1348                 int sz=atoi(argv[i]);
1349                 newImage=newImgSudoku(sz);
1350             } else if (strcmp(argv[i],"--version")==0) {
1351                 printf("%s version %s\n",basename(argv[0]), VERSION);
1352                 printf("compiled with %s on %s\n",CFG_CC,__DATE__);
1353                 printf("git hash  %s\n",CFG_GIT_FHASH);
1354             } else {
1355                 imgUsage(stdout,argv[0]);
1356                 ERROR("Unknown option: ",argv[i]);
1357             }
1358             deleteImg(currentImage);
1359             currentImage=newImage;
1360             newImage=NULL;
1361         }
1362         ++i;
1363     }
1364     deleteImg(currentImage);
1365     return 0;
1366 }
```

References deleteFilter(), deleteImg(), ERROR, imgBlur(), imgConvolutionSameSizeDiff(), imgFlattenContrast(), imgInvert(), imgLuminosityScale(), imgMake3dEffect(), imgPrint(), imgRaiseContrast(), imgRotate(), imgScale(), imgUsage(), imgWrite(), newFilter(), newImg33edge(), newImg9By9Dots(), newImgCross(), newImgNDotsHori(), newImgRead(), newImgSquare(), newImgSudoku(), newImgVerticalBar(), and NOT_ENOUGH.

4.16.3.18 imgMake3dEffect()

```
Img * imgMake3dEffect (
              Img * in )
704                             {
705     Img * answer = newImgColor(in->width,in->height+in->width,255);
706
707     for(int x = 0; x < in->width; x++) {
708         for(int y = 0; y < in->height; y++) {
709             answer->data[x + (in->width-x-1+y) * answer->width]=
710                 in->data[x + y * in->width];
711         }
712     }
713     return answer;
714 }
```

References img::data, img::height, newImgColor(), and img::width.

4.16.3.19 imgPrint()

```
void imgPrint (
              Img * myImg )
```

Displays an image with numerical values.

Parameters

myImg	image to display in the terminal.

```
509                         {
510     for(int y = 0; y < myImg->height; y++) {
511         for(int x = 0; x < myImg->width; x++) {
512             printf("%3d ",imgGetVal(myImg,x,y));
513         }
514         printf("\n");
515     }
516 }
```

References img::height, imgGetVal(), and img::width.

4.16.3.20 imgRaiseContrast()

```
Img * imgRaiseContrast (
              Img * in )
```

Raises the contrast of an image We perform that by computing the square root of normalized difference with 128.

Parameters

in	the input image to flatten.

Returns

a newly allocated image which is flattened.

```
692                             {
693     Img * answer = newImgCopy(in);
694     for(int i = 0; i < in->height*in->width; i++) {
695         int m = in->data[i];
696         float v =(m-128.0)/128.;
697         float w = v<0?sqrtf(-v):sqrt(v);
698         answer->data[i]=v>0?INBYTE(w*128+128):INBYTE(128-w*128);
699     }
700     return answer;
701 }
```

References img::data, img::height, INBYTE, newImgCopy(), and img::width.

4.16.3.21 imgRotate()

```
Img * imgRotate (
              Img * img,
              int deg )
```

Rotates an image.

Parameters

img	to rotate
deg	degrees to rotate

Returns

the down sampled image.

See also

http://www.leptonica.org/rotation.html

```
1032                                {
1033     Img * answer = newImgColor(img->width,img->height,255);
1034     float rad=deg*3.14159/180;
1035     float c = cos(rad);
1036     float s = sin(rad);
1037     int w=img->width;
1038     int h=img->height;
1039     float eps=0.0001;    // epsilum, considered as zero
1040     for (int x=0;x<w;++x) {
1041         for (int y=0;y<h;++y) {
1042             double xd= (x-w/2)*c - (y-h/2)*s + w/2;
1043             double yd= (x-w/2)*s + (y-h/2)*c + h/2;
1044             if (xd>0 && yd>0 && xd<w-1 && yd<h-1) {
1045                 double xf=floor(xd);
1046                 double xc=ceil(xd);
1047                 double yf=floor(yd);
1048                 double yc=ceil(yd);
1049                 double ff=sqrt((xd-xf)*(xd-xf)+(yd-yf)*(yd-yf));
1050                 double fc=sqrt((xd-xf)*(xd-xf)+(yd-yc)*(yd-yc));
1051                 double cf=sqrt((xd-xc)*(xd-xc)+(yd-yf)*(yd-yf));
1052                 double cc=sqrt((xd-xc)*(xd-xc)+(yd-yc)*(yd-yc));
1053                 int v=255;
1054                 if (-eps<ff && ff<eps) {
1055                     v=img->data[(int)(xf+w*yf)];
1056                 } else if (-eps<fc && fc<eps) {
1057                     v=img->data[(int)(xf+w*yc)];
1058                 } else if (-eps<cf && cf<eps) {
1059                     v=img->data[(int)(xc+w*yf)];
1060                 } else if (-eps<cc && cc<eps) {
1061                     v=img->data[(int)(xc+w*yc)];
1062                 } else {
1063                     // not too close to any point, so compute average
1064                     double t=1/ff+1/fc+1/cf+1/cc;
1065                     v=INBYTE((int)((img->data[(int)(xf+w*yf)]/ff+
1066                                     img->data[(int)(xf+w*yc)]/fc+
1067                                     img->data[(int)(xc+w*yf)]/cf+
1068                                     img->data[(int)(xc+w*yc)]/cc)/t));
1069                 }
1070                 answer->data[x+w*y]=v;
1071             }
1072         }
1073     }
1074     return answer;
1075 }
```

References img::data, img::height, INBYTE, newImgColor(), and img::width.

4.16.3.22 imgRotate90()

```
Img * imgRotate90 (
            Img * img )
```

Rotates an image by 90 degrees.

Resulting image exchange height and width with original image.

Parameters

img	to rotate

Returns

rotated image

```
1013                            {
1014     // exchange height and width
1015     Img * answer = newImgColor(img->height,img->width,255);
1016     for (int y=0;y<answer->height;++y) {
1017         int o = answer->width*y;
1018         for (int x=0;x<answer->width;++x) {
1019             answer->data[x+o]=img->data[y+img->width*x];
1020         }
1021     }
1022     return answer;
1023 }
```

References img::data, img::height, newImgColor(), and img::width.

4.16.3.23 imgScalar()

```
unsigned char imgScalar (
              Img * i1,
              Img * i2,
              int xoffset,
              int yoffset )
```

Given a large image i1 and a smaller one i2 compute the scaler product of i2 with the subset of i1 at offset (xoffset,yoffset).

Parameters

i1	the large image.
i2	the small image
xoffset	horizontal offset to apply.
yoffset	vertical offset to apply.

Returns

the scalar product of i2 with a sub image of i1.

```
1117                                                                {
1118     if (i1->width<i2->width+xoffset)
1119         ERROR("Wrong size","");
1120     if (i1->height<i2->height+yoffset)
1121         ERROR("Wrong size","");
1122     long int s=0;
1123     for (int y=0;y<i2->height;++y) {
1124         for (int x=0;x<i2->width;++x) {
1125             int v=i1->data[x+xoffset+(y+yoffset)*i1->width]
1126                 -i2->data[x+y*i1->width];
1127             s+=v<0?-v:v;
1128         }
1129     }
1130     int max=i2->height*i2->width*50;
1131     if (max<s) return 0;
1132     return INBYTE(255-(s*255)/max);
1133 }
```

References img::data, ERROR, img::height, INBYTE, and img::width.

4.16.3.24 imgScale()

```
Img * imgScale (
            Img * in,
            int s )
```

Scales an image to a larger image.

Parameters

in	the picture to blur
s	factor to scale

Returns

the newly allocated picture.

```
721                             {
722     Img * answer = newImgColor(in->width*s,in->height*s,255);
723 
724     for(int x = 0; x < in->width; x++) {
725         for(int y = 0; y < in->height; y++) {
726             for(int xx = 0; xx < s; xx++) {
727                 for(int yy = 0; yy < s; yy++) {
728                     answer->data[s*x +xx + (yy+s*y) * answer->width]=
729                         in->data[x + y * in->width];
730                 }
731             }
732         }
733     }
734     return answer;
735 }
```

References img::data, img::height, newImgColor(), and img::width.

4.16.3.25 imgUsage()

```
void imgUsage (
            FILE * f,
            char * name )
```

Tells how to use this program.

Parameters

f	where to write the info (stdout or stderr).
name	the value of argv[0].

```
1140                                {
1141     char * bname=basename(name);
1142     fprintf(f,"%s usage:\n", bname);
1143     fprintf(f,"    %s [<input-file>] [ options ] <output-file>\n", bname);
1144     fprintf(f,"Where option is one of:\n");
1145     fprintf(f,"    [-b|--blur] <r>:\n");
1146     fprintf(f,"        Blurs the image with a radius of r.\n");
1147     fprintf(f,"    [-c|--contrast] :\n");
1148     fprintf(f,"        Raises contrast level.\n");
1149     fprintf(f,"    --conv <filename> <percent>:\n");
1150     fprintf(f,"        load image in <filename> and use it as a convolution\n");
1151     fprintf(f,"        filter with a threshold set at <percent>.\n");
1152     fprintf(f,"    --cross <n> <w> <t>:\n");
```

```
1153     fprintf(f,"        Generates a black cross on a white background. \n");
1154     fprintf(f,"        The cross size is n by n pixels in a w by w\n");
1155     fprintf(f,"        picture. The cross line is <t> pixels thick.\n");
1156     fprintf(f,"    [-f|--flatten] :\n");
1157     fprintf(f,"        Flattens the contrast.\n");
1158     fprintf(f,"    [-h|--help] :\n");
1159     fprintf(f,"        Displays this help message and leaves.\n");
1160     fprintf(f,"    [-i|--inv] :\n");
1161     fprintf(f,"        Inverses the image. White becomes black, black \n");
1162     fprintf(f,"        becomes white.\n");
1163     fprintf(f,"    [-l|--lum] :\n");
1164     fprintf(f,"        Scales luminosity if the image is too bright or too dark.\n");
1165     fprintf(f,"    [-p|--print] :\n");
1166     fprintf(f,"        Prints on stdout numerical value of current image.\n");
1167     fprintf(f,"    [-s|--scale] <n>:\n");
1168     fprintf(f,"        Scales image by a factor n.\n");
1169     fprintf(f,"    --square <n> <w> <t>:\n");
1170     fprintf(f,"        Generates a black square on a white background. \n");
1171     fprintf(f,"        The square size is n by n pixels in a w by w\n");
1172     fprintf(f,"        picture. The square line is <t> pixels thick.\n");
1173     fprintf(f,"    --sudoku <n>:\n");
1174     fprintf(f,"        Generates a black empty sudoku grid of n pixels large in a \n");
1175     fprintf(f,"        224 by 224 picture with white background.\n");
1176     fprintf(f,"    [-r|--rotate] <n>:\n");
1177     fprintf(f,"        Rotates an image by n degrees.\n");
1178     fprintf(f,"    [-v|--vertical] <s> <w>:\n");
1179     fprintf(f,"        Generates a black vertical bar of width <w>\n");
1180     fprintf(f,"        in a white square of size <s>.\n");
1181     fprintf(f,"    --version :\n");
1182     fprintf(f,"        Displays the current version of %s.\n",bname);
1183     fprintf(f,"    --1d-dots <n> <w>:\n");
1184     fprintf(f,"        Generates <n> black dots on a white background, \n");
1185     fprintf(f,"        in one dimension: the generated image size is <w> \n");
1186     fprintf(f,"        by 1 pixels.\n");
1187     fprintf(f,"    --3x3-edge :\n");
1188     fprintf(f,"        Generates a 3x3 image to perform edge detection if \n");
1189     fprintf(f,"        used as convolution filter.\n");
1190     fprintf(f,"    [-3d|--3d] :\n");
1191     fprintf(f,"        Draws an isometric-like view.\n");
1192     fprintf(f,"        in a 224x224 image.\n");
1193     fprintf(f,"    --9x9dots <n>:\n");
1194     fprintf(f,"        Generates 9x9 black dots on a white background. \n");
1195     fprintf(f,"        The image size is n by n pixels.\n");
1196 
1197     fprintf(f,"Examples:\n");
1198     fprintf(f,"    %s --sudoku 200 --rotate 5 --blur 2 out.png\n",bname);
1199     fprintf(f,"        Generates an empty sudoku grid, then rotate it by 5\n");
1200     fprintf(f,"        degrees, then blurs the image with a radius of 2, then\n");
1201     fprintf(f,"        save the result in file out.png.\n");
1202     fprintf(f,"    \n");
1203     fprintf(f,"    %s --3x3-edge edgefilter.png\n",bname);
1204     fprintf(f,"        Creates a 3x3 filter to be used for edge detection.\n");
1205     fprintf(f,"        Save it under edgefilter.png.\n");
1206     fprintf(f,"    \n");
1207     fprintf(f,"    %s mypic.png --conv edgefilter.png -10 out.png\n",
1208             bname);
1209     fprintf(f,"        Apply the convolution filter edgefilter.png to\n");
1210     fprintf(f,"        mypic.png and save the result under out.png.\n");
1211     fprintf(f,"        -10%% of threshold is applied. Put a lower percentage\n");
1212     fprintf(f,"        for brighter images, a higher one for a darker.\n");
1213     fprintf(f,"    \n");
1214 
1215     fprintf(f,"    %s my_photo.jpg --lum my_new_photo.png\n",bname);
1216     fprintf(f,"        Reads file my_photo.jpg increase luminosity to stretch from\n");
1217     fprintf(f,"        dark to very clear and save the result in my_new_photo.png.\n");
1218 }
```

4.16.3.26 imgWrite()

```
void imgWrite (
            Img * myImg,
            char * filename )
```

Writes a Img struct to a png file.

Parameters

myImg	an existing image to save to a file.
filename	name of the png to write.

```
577                                     {
578     FILE *fp = fopen(filename, "wb");
579     if(!fp) {
580         ERROR("could not open file ",filename);
581     }
582
583     png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
584     if (!png) {
585         ERROR("could not create png structure","");
586     }
587
588     png_infop info = png_create_info_struct(png);
589     if (!info) {
590         ERROR("could not get info","");
591     }
592
593     if (setjmp(png_jmpbuf(png)))  {
594         fprintf(stderr,
595                 "could not jmp to data while trying to write %s\n",
596                 filename);
597     }
598
599     png_init_io(png, fp);
600
601     // Output is 8bit depth, RGBA format.
602     png_set_IHDR(png,
603                  info,
604                  myImg->width, myImg->height,
605                  8,
606                  PNG_COLOR_TYPE_RGBA,
607                  PNG_INTERLACE_NONE,
608                  PNG_COMPRESSION_TYPE_DEFAULT,
609                  PNG_FILTER_TYPE_DEFAULT
610                  );
611
612     png_write_info(png, info);
613
614     // To remove the alpha channel for PNG_COLOR_TYPE_RGB format,
615     // Use png_set_filler().
616     //png_set_filler(png, 0, PNG_FILLER_AFTER);
617
618     if (!myImg->data) abort();
619
620     unsigned char* row_pointers[myImg->height];
621     for(int y = 0; y < myImg->height; y++) {
622         row_pointers[y] =
623             (unsigned char*)malloc(4*myImg->width);
624         unsigned char* row = row_pointers[y];
625         for(int x = 0; x < myImg->width; x++) {
626             unsigned char * px = &(row[x * 4]);
627             px[0]=px[1]=px[2]=myImg->data[x+y*myImg->width];
628             px[3]=255;
629         }
630     }
631     png_write_image(png, row_pointers);
632
633     for(int y = 0; y < myImg->height; y++) {
634         free(row_pointers[y]);
635     }
636     png_write_end(png, NULL);
637     fclose(fp);
638     png_destroy_write_struct(&png, &info);
639 }
```

References img::data, ERROR, img::height, and img::width.

4.16.3.27 my_error_exit()

```
void my_error_exit (
             j_common_ptr cinfo )
```

used to manage jpeg errors.

```
288                                             {
289     /* cinfo->err really points to a jpegerrmgr struct, so coerce pointer */
290     MyErrorPtr myerr = (MyErrorPtr) cinfo->err;
291
292     /* Always display the message. */
293     /* We could postpone this until after returning, if we chose. */
294     (*cinfo->err->output_message) (cinfo);
295
296     /* Return control to the setjmp point */
297     longjmp(myerr->setjmp_buffer, 1);
298 }
```

References jpegerrmgr::setjmp_buffer.

4.16.3.28 newImg33edge()

```
Img * newImg33edge ( )
```

create a 3x3 image to perform edge detection.

Returns

a newly allocated image.

```
102                         {
103     int c=16;
104     int b=c»2;
105     unsigned char a[] = {
106         128+c-b, 128+c+b, 128+c-b,
107         128+c+b, 128-8*c, 128+c+b,
108         128+c-b, 128+c+b, 128+c-b
109     };
110     return newImgFromArray(3,3,a);
111 }
```

References newImgFromArray().

4.16.3.29 newImg9By9Dots()

```
Img * newImg9By9Dots (
              int w )
```

create an image with 9x9 black dots on a picture of size w times w.

Parameters

w	side of the picture

Returns

a newly allocated image.

```
184                             {
185     Img * answer = newImgColor(w,w,255);
186     float step=w/9.;
187     float stepOverTwo=step/2.;
188     float stepOverThree=step/2;
```

```
189     float stepOverThreeSq=stepOverThree*stepOverThree;
190     for (int x=0;x<w;++x) {
191         for (int y=0;y<w;++y) {
192             float xr = round((x-stepOverTwo)/step)*step+stepOverTwo;
193             float yr = round((y-stepOverTwo)/step)*step+stepOverTwo;
194             float d = ((x-xr)*(x-xr)+(y-yr)*(y-yr))/stepOverThreeSq;
195             if (d<1) {
196                 answer->data[x+w*y]=INBYTE((int)(d*255));
197             }
198         }
199     }
200     return answer;
201 }
```

References img::data, INBYTE, and newImgColor().

4.16.3.30 newImgColor()

```
Img * newImgColor (
            int w,
            int h,
            unsigned char c )
```

create an image of a given color

Parameters

filename	name of the file to read

Returns

a newly allocated image.

```
43                                                  {
44     Img * answer = (Img *)malloc(sizeof(struct img));
45     answer->width=w;
46     answer->height=h;
47     answer->data=
48         (unsigned char*)malloc(answer->height * answer->width);
49     memset(answer->data,c,answer->height * answer->width);
50     return answer;
51 }
```

References img::data, img::height, and img::width.

4.16.3.31 newImgCopy()

```
Img * newImgCopy (
            Img * myImg )
```

create an image from an exiting Img instance

Parameters

myImg	an existing image to copy

Returns

a newly allocated image.

```
464                             {
465     Img * answer = (Img *) malloc(sizeof(struct img));
466
467     answer->width      = myImg->width      ;
468     answer->height     = myImg->height     ;
469     answer->data = (unsigned char*)malloc(sizeof(char) *
470                                           answer->height*answer->width);
471     memcpy(answer->data,
472            myImg->data,
473            answer->width*answer->height);
474     return answer;
475 }
```

References img::data, img::height, and img::width.

4.16.3.32 newImgCross()

```
Img * newImgCross (
              int s,
              int w,
              int t )
```

create an image with a black cross on a w by w white background.

Parameters

s	size of the cross
w	side of the image
t	thickness of the cross

Returns

a newly allocated image.

```
121                                     {
122     Img * answer = newImgColor(w,w,255);
123     int m=w/2;
124     int count=0;
125     while (count<t && count<m) {
126         if (s<w-1 && s>0) {
127             for (int y=m-s/2;y<m-s/2+s;++y) {
128                 answer->data[m+count+w*y]=0;
129             }
130             for (int x=m-s/2;x<m-s/2+s;++x) {
131                 answer->data[x+w*(m+count)]=0;
132             }
133             for (int y=m-s/2;y<m-s/2+s;++y) {
134                 answer->data[m-count+w*y]=0;
135             }
136             for (int x=m-s/2;x<m-s/2+s;++x) {
137                 answer->data[x+w*(m-count)]=0;
138             }
139         }
140         count++;
141     }
142     return answer;
143 }
```

References img::data, and newImgColor().

4.16.3.33 newImgFromArray()

```
Img * newImgFromArray (
            int w,
            int h,
            unsigned char * buffer )
```

Creates an image from an array of unsigned char.

Parameters

w	width of the picture in pixels
h	height of the picture in pixels
buffer	a buffer of w times h grey pixels.

Returns

corresponding structure to the data in the buffer

```
32                                                          {
33     Img * answer = newImgColor(w,h,0);
34     memcpy(answer->data,buffer,w*h);
35     return answer;
36 }
```

References img::data, and newImgColor().

4.16.3.34 newImgNDotsHori()

```
Img * newImgNDotsHori (
            int N,
            int w )
```

Creates an image with equaly separated N black dots on a picture of size w by 1 pixel.

First point is located at w/N/2 from the left border of the picture, then second at w/2/N+w/N, third at w/2/N+2∗w/N, N-th point is located at w/2/N+(N-1)∗w/N=w-w/2/N. So N-th point is at w/N/2 from the right border of the picture.

Parameters

N	number of back dots to put
w	width of the picture

Returns

a newly allocated image.

```
217                                 {
218     // create a w by 1 white image
219     Img * answer = newImgColor(w,1,255);
220     // space between dots is w/N
221     float step=((float)w)/((float)N);
222     // we begin at step/2 from the border
223     float stepOverTwo=step/2.;
224     float stepOverFour=step/4;
```

```
225     float stepOverFourSq=stepOverFour*stepOverFour;
226     for (int x=0;x<w;++x) {
227         // horizontal position of the closest black dot
228         float xr = round((x-stepOverTwo)/step)*step+stepOverTwo;
229         // compute distance to the closest black dot
230         float d = fabs(xr-x);
231         float dmax=3;
232         if (d<dmax) {
233             // we are close to a black dot so we darken the pixel
234             answer->data[x]=INBYTE((int)(d*255/dmax));
235         }
236     }
237     return answer;
238 }
```

References img::data, INBYTE, and newImgColor().

4.16.3.35 newImgRead()

```
Img * newImgRead (
            char * filename )
```

create an image from a png or jpeg file

Parameters

filename	name of the file to read

Returns

a newly allocated image.

```
439                                 {
440     int l = strlen(filename);
441     if ( (l>4 &&
442           filename[l-4]=='.' &&
443           (filename[l-3]=='j'|| filename[l-3]=='J')&&
444           (filename[l-2]=='p'|| filename[l-2]=='P')&&
445           (filename[l-1]=='g'|| filename[l-1]=='G'))
446          ||
447          (l>5 &&
448           filename[l-5]=='.' &&
449           (filename[l-4]=='j'|| filename[l-4]=='J')&&
450           (filename[l-3]=='p'|| filename[l-3]=='P')&&
451           (filename[l-2]=='e'|| filename[l-2]=='E')&&
452           (filename[l-1]=='g'|| filename[l-1]=='G'))) {
453         return newImgReadJpeg(filename);
454     }
455     // otherwise we assume it is a png
456     return newImgReadPng(filename);
457 }
```

References newImgReadJpeg(), and newImgReadPng().

4.16.3.36 newImgReadJpeg()

```
static Img * newImgReadJpeg (
            char * filename )  [static]
```

create an image from a jpeg file

Parameters

filename	name of the file to read

Returns

a newly allocated image.

```
305                                         {
306     struct jpeg_decompress_struct cinfo;
307     struct jpegerrmgr jerr;
308     FILE * infile;      /* source file */
309     JSAMPARRAY buffer;      /* Output row buffer */
310     int row_stride;     /* physical row width in output buffer */
311 
312     if ((infile = fopen(filename, "rb")) == NULL) {
313         ERROR("can't open ", filename);
314     }
315 
316     cinfo.err = jpeg_std_error(&jerr.pub);
317     jerr.pub.error_exit = my_error_exit;
318 
319     if (setjmp(jerr.setjmp_buffer)) {
320         // JPEG code has signaled an error.
321         jpeg_destroy_decompress(&cinfo);
322         fclose(infile);
323         ERROR("Error in jpeg file ", filename);
324     }
325     jpeg_create_decompress(&cinfo);
326     jpeg_stdio_src(&cinfo, infile);
327 
328     (void) jpeg_read_header(&cinfo, TRUE);
329 
330     (void) jpeg_start_decompress(&cinfo);
331 
332     row_stride = cinfo.output_width * cinfo.output_components;
333     buffer = (*cinfo.mem->alloc_sarray)
334         ((j_common_ptr) &cinfo, JPOOL_IMAGE, row_stride, 1);
335     int lineCount=0;
336     Img * answer = newImgColor(cinfo.image_width,cinfo.image_height,0);
337     // read the image line by line
338     while (cinfo.output_scanline < cinfo.output_height) {
339         (void) jpeg_read_scanlines(&cinfo, buffer, 1);
340         for (int i=0;i<row_stride;++i) {
341             int j = (buffer[0][i]+buffer[0][i+1]+buffer[0][i+2])/3;
342             answer->data[i/3+lineCount*cinfo.image_width]=j;
343         }
344         ++lineCount;
345     }
346     (void) jpeg_finish_decompress(&cinfo);
347     jpeg_destroy_decompress(&cinfo);
348     fclose(infile);
349     return answer;
350 }
```

References img::data, ERROR, my_error_exit(), newImgColor(), jpegerrmgr::pub, and jpegerrmgr::setjmp_buffer.

4.16.3.37 newImgReadPng()

```
static Img * newImgReadPng (
             char * filename )  [static]
```

create an image from a png file

Parameters

filename	name of the file to read

Returns

a newly allocated image.

```
357                                             {
358     Img * answer = (Img *)malloc(sizeof(struct img));
359     FILE *fp = fopen(filename, "rb");
360     if(!fp) {
361         ERROR("file not found - ",filename);
362     }
363 
364     png_structp png = png_create_read_struct(PNG_LIBPNG_VER_STRING,
365                                              NULL, NULL, NULL);
366 
367     png_infop info = png_create_info_struct(png);
368     if(!info) abort();
369 
370     if(setjmp(png_jmpbuf(png))) {
371         ERROR("Error","");
372     }
373 
374     png_init_io(png, fp);
375 
376     png_read_info(png, info);
377 
378     answer->width  = png_get_image_width(png, info);
379     answer->height = png_get_image_height(png, info);
380     int color_type = png_get_color_type(png, info);
381     int bit_depth  = png_get_bit_depth(png, info);
382 
383     // Read any color_type into 8bit depth, RGBA format.
384     // See http://www.libpng.org/pub/png/libpng-manual.txt
385 
386     if(bit_depth == 16)
387         png_set_strip_16(png);
388 
389     if(color_type == PNG_COLOR_TYPE_PALETTE)
390         png_set_palette_to_rgb(png);
391 
392     // PNG_COLOR_TYPE_GRAY_ALPHA is always 8 or 16bit depth.
393     if(color_type == PNG_COLOR_TYPE_GRAY && bit_depth < 8)
394         png_set_expand_gray_1_2_4_to_8(png);
395 
396     if(png_get_valid(png, info, PNG_INFO_tRNS))
397         png_set_tRNS_to_alpha(png);
398 
399     // These color_type don't have an alpha channel then fill it with 0xff.
400     if(color_type == PNG_COLOR_TYPE_RGB ||
401        color_type == PNG_COLOR_TYPE_GRAY ||
402        color_type == PNG_COLOR_TYPE_PALETTE)
403         png_set_filler(png, 0xFF, PNG_FILLER_AFTER);
404 
405     if(color_type == PNG_COLOR_TYPE_GRAY ||
406        color_type == PNG_COLOR_TYPE_GRAY_ALPHA)
407         png_set_gray_to_rgb(png);
408 
409     png_read_update_info(png, info);
410 
411     unsigned char* row_pointers[answer->height];
412     answer->data=
413         (unsigned char*)malloc(sizeof(char*) * answer->height * answer->width);
414     for(int y = 0; y < answer->height; y++) {
415         row_pointers[y] =
416             (unsigned char*)malloc(png_get_rowbytes(png,info));
417     }
418     png_read_image(png, row_pointers);
419     for(int y = 0; y < answer->height; y++) {
420         unsigned char* row = row_pointers[y];
421         for(int x = 0; x < answer->width; x++) {
422             unsigned char * px = &(row[x * 4]);
423             answer->data[x+y*answer->width]=(px[0]+px[1]+px[2])/3;
424         }
425     }
426     for(int y = 0; y < answer->height; y++) {
427         free(row_pointers[y]);
428     }
429     fclose(fp);
430     png_destroy_read_struct(&png, &info, NULL);
431     return answer;
432 }
```

References img::data, ERROR, img::height, and img::width.

4.16.3.38 newImgSquare()

```
Img * newImgSquare (
            int s,
            int w,
            int t )
```

create an image with a black square on a w by w white background.

Parameters

s	side of the square in pixel
w	side of the image
t	thickness

Returns

a newly allocated image.

```
153                                     {
154     Img * answer = newImgColor(w,w,255);
155     int m=w/2;
156     int count=0;
157     while (count<t) {
158         if (s<w-1 && s>0) {
159             for (int y=m-s/2;y<m-s/2+s;++y) {
160                 answer->data[m-s/2+w*y]=0;
161             }
162             for (int y=m-s/2;y<m-s/2+s;++y) {
163                 answer->data[m-s/2+s-1+w*y]=0;
164             }
165             for (int x=m-s/2;x<m-s/2+s;++x) {
166                 answer->data[x+w*(m-s/2)]=0;
167             }
168             for (int x=m-s/2;x<m-s/2+s;++x) {
169                 answer->data[x+w*(m-s/2+s-1)]=0;
170             }
171         }
172         count++;
173         s--;
174     }
175     return answer;
176 }
```

References img::data, and newImgColor().

4.16.3.39 newImgSudoku()

```
Img * newImgSudoku (
            int sz )
```

create an image of a given color

Parameters

filename	name of the file to read

Returns

a newly allocated image.

```
245                             {
246     int w=212;
247     int h=212;
248     Img * answer = newImgColor(w,h,255);
249     int xinit=224/2-sz/2;
250     int yinit=xinit;
251     int step=sz/9;
252     for (int x=0;x<10;++x) {
253         int xx=xinit+x*step;
254         for (int yy=yinit;yy<yinit+step*9-1;++yy) {
255             if (xx>=0 && yy>=0 && xx<w && yy<h)
256                 answer->data[xx+w*yy]=0;
257         }
258     }
259     answer->data[1+xinit+9*step+w*yinit]=0;
260     answer->data[xinit+w*(yinit+1+9*step)]=0;
261     for (int y=0;y<10;++y) {
262         int yy=yinit+y*step;
263         for (int xx=xinit;xx<xinit+step*9-1;++xx) {
264             if (xx>=0 && yy>=0 && xx<w && yy<h)
265                 answer->data[xx+w*yy]=0;
266         }
267     }
268     for (int x=0;x<=3;++x) {
269         int xx=xinit+x*step*3;
270         for (int yy=yinit;yy<yinit+step*9-1;++yy) {
271             if (xx>=0 && yy>=0 && xx<w && yy<h)
272                 answer->data[1+xx+w*(yy+1)]=0;
273         }
274     }
275     for (int y=0;y<=3;++y) {
276         int yy=yinit+y*step*3;
277         for (int xx=xinit;xx<xinit+step*9-1;++xx) {
278             if (xx>=0 && yy>=0 && xx<w && yy<h)
279                 answer->data[1+xx+w*(yy+1)]=0;
280         }
281     }
282     return answer;
283 }
```

References img::data, and newImgColor().

4.16.3.40 newImgVerticalBar()

```
Img * newImgVerticalBar (
            int s,
            int w )
```

create an image of a vertical black bar on a square white background

Parameters

s	size of the image
w	width of the back bar

Returns

a newly allocated image.

```
59                                      {
60     Img * answer = newImgColor(s,s,255);
61     int m=s/2;
62     for (int x=m-w/2;x<m-w/2+w;++x) {
63         for (int y=0;y<s;++y) {
64             answer->data[x+s*y]=0;
```

```
65          }
66      }
67      return answer;
68  }
```

References img::data, and newImgColor().

4.16.3.41 newImgVerticalBarInRect()

```
Img * newImgVerticalBarInRect (
              int sx,
              int sy,
              int w )
```

create an image of a vertical black bar on a rectangular white background

Parameters

sx	horizontal size of the image
sy	vertical size of the image
w	width of the back bar

Returns

a newly allocated image.

```
78                                                      {
79      Img * answer = newImgColor(sx,sy,255);
80      int m=sx/2;
81      if (m<w) {
82          ERROR("Wrong size.","");
83      }
84      // put some grey arond the bar so that the
85      // equivalent filter is 0
86      for (int x=m-w;x<=m+w;++x) {
87          for (int y=0;y<sy;++y) {
88              answer->data[x+sx*y]=128;
89          }
90      }
91      for (int x=m-w/2;x<m-w/2+w;++x) {
92          for (int y=0;y<sy;++y) {
93              answer->data[x+sx*y]=0;
94          }
95      }
96      return answer;
97  }
```

References img::data, ERROR, and newImgColor().

4.17 img.h File Reference

```
#include "util.h"
```

Classes

- struct img

  A structure to store greyscale image data.

Typedefs

- typedef struct img Img

  Short name for 'struct img'.
- typedef struct filter Filter

Functions

- Img ∗ newImgFromArray (int w, int h, unsigned char ∗s)

  Creates an image from an array of unsigned char.
- Img ∗ newImgColor (int w, int h, unsigned char c)

  create an image of a given color
- Img ∗ newImgRead (char ∗filename)

  create an image from a png or jpeg file
- Img ∗ newImgCopy (Img ∗myImg)

  create an image from an exiting Img instance
- Img ∗ newImg9By9Dots (int w)

  create an image with 9x9 black dots on a picture of size w times w.
- Img ∗ newImgNDotsHori (int N, int w)

  Creates an image with equaly separated N black dots on a picture of size w by 1 pixel.
- Img ∗ newImgSudoku (int sz)

  create an image of a given color
- Img ∗ newImgSquare (int s, int w, int t)

  create an image with a black square on a w by w white background.
- Img ∗ newImgCross (int s, int w, int t)

  create an image with a black cross on a w by w white background.
- Img ∗ newImgVerticalBar (int s, int w)

  create an image of a vertical black bar on a square white background
- Img ∗ newImgVerticalBarInRect (int sx, int sy, int w)

  create an image of a vertical black bar on a rectangular white background
- void deleteImg (Img ∗myImg)

  Deletes an existing image.
- Img ∗ imgDivideByTwo (Img ∗img)

  Divide the size of an image by two.
- unsigned char imgGetVal (Img ∗p, int x, int y)

  Get the value a pixel for a given color. Zero is returned if the pixel is out of the picture.
- void imgPrint (Img ∗myImg)

  Displays an image with numerical values.
- void imgWrite (Img ∗myImg, char ∗filename)

  Writes a Img struct to a png file.
- Img ∗ imgInvert (Img ∗in)

  Inverts an image.
- Img ∗ imgFlattenContrast (Img ∗in)

  Flattens the contrast of an image We perform that by raising to the square the normalized difference with 128.
- Img ∗ imgRaiseContrast (Img ∗in)

  Raises the contrast of an image We perform that by computing the square root of normalized difference with 128.
- Img ∗ imgBlur (Img ∗in, int radius)

  Blurs a picture.
- Img ∗ imgMake3dEffect (Img ∗in)
- Img ∗ imgLuminosityScale (Img ∗in)

Spread luminosity in the picture.

- Img * imgConvolution (Img *in, Filter *filter)

  perform a convolution between an image and a filter with unsigned char.
- Img * imgConvolutionDiff (Img *in, Filter *filter)

  perform a convolution between an image and a filter with unsigned char.
- Img * imgConvolutionSameSize (Img *in, Filter *filter)

  perform a convolution between an image and a filter applying a threshold given by intFilter and only positive numbers.
- Img * imgConvolutionSameSizeDiff (Img *in, Filter *filter)

  perform a convolution between an image and a filter applying a threshold given by intFilter.
- Img * imgDownSampleAvg (Img *img, int poolsize, int stride)

  Downsample an image using an average pool strategy.
- Img * imgDownSampleMax (Img *img, int poolsize, int stride)

  Downsample an image using a maxpool strategy.
- void imgDrawRect (Img *myImg, int xmin, int ymin, int xmax, int ymax)

  draws a rectangle on the image
- unsigned char imgScalar (Img *, Img *, int, int)

  Given a large image i1 and a smaller one i2 compute the scaler product of i2 with the subset of i1 at offset (xoffset,yoffset).
- Img * imgRotate (Img *img, int deg)

  Rotates an image.
- Img * imgRotate90 (Img *img)

  Rotates an image by 90 degrees.
- Img * imgScale (Img *in, int s)

  Scales an image to a larger image.
- Img * imgExtract (Img *myImg, int xmin, int ymin, int xmax, int ymax)

  Extract an image from and image.
- int imgGetWeight (Img *in)

  Return the sum of all pixels in the picture.
- int imgMain (int, char **)

4.17.1 Typedef Documentation

4.17.1.1 Filter

```
typedef struct filter Filter
```

4.17.1.2 Img

```
typedef struct img Img
```

Short name for 'struct img'.

4.17.2 Function Documentation

4.17.2.1 deleteImg()

```
void deleteImg (
            Img * myImg )
```

Deletes an existing image.

Parameters

myImg	an existing image to delete Nothing happens if myImg is NULL.

```
482                         {
483     if (myImg==NULL) return;
484     free(myImg->data);
485     myImg->width=0;
486     myImg->height=0;
487     myImg->data=NULL;
488     free(myImg);
489 }
```

References img::data, img::height, and img::width.

4.17.2.2 imgBlur()

```
Img * imgBlur (
            Img * in,
            int radius )
```

Blurs a picture.

Parameters

in	the picture to blur
radius	intensity of the blur

Returns

the newly allocated picture.

```
743                             {
744     Img * answer = newImgCopy(in);
745     int sq=radius*radius;
746     for(int y = 0; y < in->height; y++) {
747         for(int x = 0; x < in->width; x++) {
748             int v=0;
749             for (int xx=-radius;xx<radius;++xx) {
750                 for (int yy=-radius;yy<radius;++yy) {
751                     if (xx*yy<sq)
752                         v+=imgGetVal(in,x+xx,y+yy);
753                 }
754             }
755             answer->data[x + y * in->width]=INBYTE((int)(v/sq/3.14));
756         }
```

```
757     }
758     return answer;
759 }
```

References img::data, img::height, imgGetVal(), INBYTE, newImgCopy(), and img::width.

4.17.2.3 imgConvolution()

```
Img * imgConvolution (
              Img * in,
              Filter * filter )
```

perform a convolution between an image and a filter with unsigned char.

The resulting image size is in->width-filter->img->width+1 by in->height-filter->img->height+1.

Parameters

in	the input picture
filter	the filter to use

Returns

the newly allocated picture with represents the convolution.

```
800                                                 {
801     if (in->width<filter->img->width)
802         ERROR("Wrong width","");
803     if (in->height<filter->img->height)
804         ERROR("Wrong height","");
805     int aw=in->width-filter->img->width+1;
806     int ah=in->height-filter->img->height+1;
807     Img * answer = newImgColor(aw,ah,0);
808     for(int y = 0; y < ah; y++) {
809         for(int x = 0; x < aw; x++) {
810             long int v=0;
811             for(int yy = 0; yy < filter->img->height; yy++) {
812                 for(int xx = 0; xx < filter->img->width; xx++) {
813                     v+=imgGetVal(in,x+xx,y+yy)*imgGetVal(filter->img,xx,yy);
814                 }
815             }
816             answer->data[x+aw*y]=
817                 (v<filter->threshold)?0
818                 :(v>filter->maxVal)?255
819                 :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
820         }
821     }
822     return answer;
823 }
```

References img::data, ERROR, img::height, filter::img, imgGetVal(), filter::maxVal, newImgColor(), filter::threshold, and img::width.

4.17.2.4 imgConvolutionDiff()

```
Img * imgConvolutionDiff (
              Img * in,
              Filter * filter )
```

perform a convolution between an image and a filter with unsigned char.

The resulting image size is in->width-filter->img->width+1 by in->height-filter->img->height+1.

Parameters

in	the input picture
filter	the filter to use

Returns

the newly allocated picture with represents the convolution.

```
835                                                 {
836     if (in->width<filter->img->width)
837         ERROR("Wrong width","");
838     if (in->height<filter->img->height)
839         ERROR("Wrong height","");
840     int aw=in->width-filter->img->width+1;
841     int ah=in->height-filter->img->height+1;
842     Img * answer = newImgColor(aw,ah,0);
843     for(int y = 0; y < ah; y++) {
844         for(int x = 0; x < aw; x++) {
845             long int v=0;
846             for(int yy = 0; yy < filter->img->height; yy++) {
847                 for(int xx = 0; xx < filter->img->width; xx++) {
848                     v+=((int)imgGetVal(in,x+xx,y+yy))*
849                         (((int)imgGetVal(filter->img,xx,yy))-128);
850                 }
851             }
852             answer->data[x+aw*y]=
853                 (v<filter->threshold)?0
854                 :(v>filter->maxVal)?255
855                 :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
856         }
857     }
858     return answer;
859 }
```

References img::data, ERROR, img::height, filter::img, imgGetVal(), filter::maxVal, newImgColor(), filter::threshold, and img::width.

4.17.2.5 imgConvolutionSameSize()

```
Img * imgConvolutionSameSize (
              Img * in,
              Filter * filter )
```

perform a convolution between an image and a filter applying a threshold given by intFilter and only positive numbers.

The resulting image size is in->width by in->height.

Parameters

in	the input picture
filter	the filter to use

Returns

the newly allocated picture with represents the convolution.

```
870                                                 {
871     if (in->width<filter->img->width || in->height<filter->img->height)
872         ERROR("Wrong size","");
```

```
873     int wfOver2=filter->img->width/2;
874     int hfOver2=filter->img->height/2;
875     Img * answer = newImgColor(in->width,in->height,0);
876     for(int y = 0; y < in->height; y++) {
877         for(int x = 0; x < in->width; x++) {
878             long int v=0;
879             for(int yy = 0; yy < filter->img->height; yy++) {
880                 for(int xx = 0; xx < filter->img->width; xx++) {
881                     int xxx=x+xx-wfOver2;
882                     int yyy=y+yy-hfOver2;
883                     if (xxx>=0 && yyy>=0 && xxx<in->width && yyy<in->height) {
884                         v+=imgGetVal(in,xxx,yyy)*imgGetVal(filter->img,xx,yy);
885                     }
886                 }
887             }
888             //HERE("___v,maxval,threshold,percent__________________");
889             //HERED((int)v);
890             //HERED((int)filter->maxVal);
891             //HERED((int)filter->threshold);
892             //HERED((int)filter->percent);
893             answer->data[x+in->width*y]=
894                 (v<filter->threshold)?0
895                 :(v>filter->maxVal)?255
896                 :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
897         }
898     }
899     return answer;
900 }
```

References img::data, ERROR, img::height, filter::img, imgGetVal(), filter::maxVal, newImgColor(), filter::threshold, and img::width.

4.17.2.6 imgConvolutionSameSizeDiff()

```
Img * imgConvolutionSameSizeDiff (
              Img * in,
              Filter * filter )
```

perform a convolution between an image and a filter applying a threshold given by intFilter.

The resulting image size is in->width by in->height.

Parameters

in	the input picture
filter	the filter to use

Returns

the newly allocated picture with represents the convolution.

```
910                                                     {
911     if (in->width<filter->img->width || in->height<filter->img->height)
912         ERROR("Wrong size","");
913     int wfOver2=filter->img->width/2;
914     int hfOver2=filter->img->height/2;
915     Img * answer = newImgColor(in->width,in->height,0);
916     for(int y = 0; y < in->height; y++) {
917         for(int x = 0; x < in->width; x++) {
918             long int v=0;
919             for(int yy = 0; yy < filter->img->height; yy++) {
920                 for(int xx = 0; xx < filter->img->width; xx++) {
921                     int xxx=x+xx-wfOver2;
922                     int yyy=y+yy-hfOver2;
923                     if (xxx>=0 && yyy>=0 && xxx<in->width && yyy<in->height) {
924
925                         v+=((int)imgGetVal(in,xxx,yyy))*
926                             (((int)imgGetVal(filter->img,xx,yy))-128);
```

```
927                         /*
928                         int d = imgGetVal(in,xxx,yyy)
929                             -imgGetVal(filter->img,xx,yy);
930                         v+=d*d;
931                         */
932                     }
933                 }
934             }
935             /*
936             HERE("___v,maxval,threshold,percent__________________");
937             HERED((int)v);
938             HERED((int)filter->maxVal);
939             HERED((int)filter->threshold);
940             HERED((int)filter->percent);
941             */
942
943             answer->data[x+in->width*y]=
944                 (v<filter->threshold)?0
945                 :(v>filter->maxVal)?255
946                 :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
947         }
948     }
949     return answer;
950 }
```

References img::data, ERROR, img::height, filter::img, imgGetVal(), filter::maxVal, newImgColor(), filter::threshold, and img::width.

4.17.2.7 imgDivideByTwo()

```
Img * imgDivideByTwo (
            Img * img )
```

Divide the size of an image by two.

Parameters

img	an image

Returns

same image as img but with a size scaled down by 2.

```
990                             {
991     Img * answer = newImgColor(img->width/2,img->height/2,255);
992     for (int y=0;y<img->height/2;++y) {
993         int o = (img->width»1)*y;
994         for (int x=0;x<img->width/2;++x) {
995             // compute the average of 4 pixels
996             answer->data[x+o] =
997                 (img->data[(x«1)+img->width*(y«1)] +
998                  img->data[(x«1)+1+img->width*(y«1)] +
999                  img->data[(x«1)+img->width*((y«1)+1)] +
1000                  img->data[(x«1)+1+img->width*((y«1)+1)] )»2;
1001         }
1002     }
1003     return answer;
1004 }
```

References img::data, img::height, newImgColor(), and img::width.

4.17.2.8 imgDownSampleAvg()

```
Img * imgDownSampleAvg (
              Img * img,
              int poolsize,
              int stride )
```

Downsample an image using an average pool strategy.

Parameters

img	the image to down scale.
poolsize	size of the square on which the average is computed.
stride	number of pixel by which the square on which the average is computed is moving at each step.

Returns

the down sampled image.

```
1085                                                    {
1086     if (img->width < poolsize)
1087         ERROR("Wrong size.","");
1088     if (img->height < poolsize)
1089         ERROR("Wrong size.","");
1090     int w =(img->width - poolsize)/stride+1;
1091     int h =(img->height - poolsize)/stride+1;
1092     Img* answer = newImgColor(w,h,0);
1093     for (int x=0;x<w;++x) {
1094         for (int y=0;y<h;++y) {
1095             int avg=0;
1096             for (int xx=0;xx<poolsize;++xx) {
1097                 for (int yy=0;yy<poolsize;++yy) {
1098                     avg+=img->data[x*stride+xx+img->width*(y*stride+yy)];
1099                 }
1100             }
1101             answer->data[x+w*y]=avg/poolsize/poolsize;
1102         }
1103     }
1104     return answer;
1105 }
```

References img::data, ERROR, img::height, newImgColor(), and img::width.

4.17.2.9 imgDownSampleMax()

```
Img * imgDownSampleMax (
              Img * img,
              int poolsize,
              int stride )
```

Downsample an image using a maxpool strategy.

Parameters

img	the image to down scale.
poolsize	size of the square on which the maximum is computed.
stride	number of pixel by which the square on which the maximum is computed is moving at each step.

Returns

the down sampled image.

```
961                                                     {
962     if (img->width < poolsize)
963         ERROR("Wrong size, poolsize too large.","");
964     if (img->height < poolsize)
965         ERROR("Wrong size, poolsize too large.","");
966     int w =(img->width - poolsize)/stride+1;
967     int h =(img->height - poolsize)/stride+1;
968     Img* answer = newImgColor(w,h,0);
969     for (int x=0;x<w;++x) {
970         for (int y=0;y<h;++y) {
971             int max=0;
972             for (int xx=0;xx<poolsize;++xx) {
973                 for (int yy=0;yy<poolsize;++yy) {
974                     int v=img->data[x*stride+xx+img->width*(y*stride+yy)];
975                     if (v>max)
976                         max=v;
977                 }
978             }
979             answer->data[x+w*y]=max;
980         }
981     }
982     return answer;
983 }
```

References img::data, ERROR, img::height, newImgColor(), and img::width.

4.17.2.10 imgDrawRect()

```
void imgDrawRect (
              Img * myImg,
              int xmin,
              int ymin,
              int xmax,
              int ymax )
```

draws a rectangle on the image

Parameters

myImg	image on which to draw the rectangle
xmin	horizontal position of lower left corner
ymin	vertical position of lower left corner
xmax	horizontal position of upper right corner
ymax	vertical position of upper right corner

```
526                                                                     {
527     if (xmin<0 || ymin<0 || xmax >myImg->width || myImg->height<ymax ||
528         xmin>=xmax || ymin>=ymax ) {
529         fprintf(stderr,"rectangle : %d %d %d %d\n",xmin,ymin,xmax,ymax);
530         fprintf(stderr,"image size is %d %d\n",myImg->width,myImg->height);
531         ERROR("Wrong size.","");
532     }
533     for(int x = xmin; x <xmax; x++) {
534         myImg->data[x+myImg->width*ymin]=(x»2)%2?255:0;
535         myImg->data[x+myImg->width*ymax]=(x»2)%2?255:0;
536     }
537     for(int y = ymin; y <ymax; y++) {
538         myImg->data[xmin+myImg->width*y]=(y»2)%2?255:0;
539         myImg->data[xmax+myImg->width*y]=(y»2)%2?255:0;
540     }
541 }
```

References img::data, ERROR, img::height, and img::width.

4.17.2.11 imgExtract()

```
Img * imgExtract (
              Img * myImg,
              int xmin,
              int ymin,
              int xmax,
              int ymax )
```

Extract an image from and image.

Parameters

myImg	image from which we extract the image.
xmin	horizontal position of lower left corner
ymin	vertical position of lower left corner
xmax	horizontal position of upper right corner
ymax	vertical position of upper right corner

Returns

the newly allocated image.

```
552                                                                 {
553     if (xmin<0 || ymin<0 || xmax >myImg->width || myImg->height<ymax ||
554         xmin>=xmax || ymin>=ymax ) {
555         fprintf(stderr,"rectangle : %d %d %d %d\n",xmin,ymin,xmax,ymax);
556         fprintf(stderr,"image size is %d %d\n",myImg->width,myImg->height);
557         ERROR("Wrong size.","");
558     }
559     int w=xmax-xmin;
560     int h=ymax-ymin;
561     Img * answer = newImgColor(w,h,255);
562     for(int y = 0; y<h; y++) {
563         // use memcopy to be faster since horizontal pixels
564         // are stored one next to the other.
565         memcpy(&answer->data[0+y*w],
566                &myImg->data[xmin+myImg->width*(ymin+y)],
567                w);
568     }
569     return answer;
570 }
```

References img::data, ERROR, img::height, newImgColor(), and img::width.

4.17.2.12 imgFlattenContrast()

```
Img * imgFlattenContrast (
              Img * in )
```

Flattens the contrast of an image We perform that by raising to the square the normalized difference with 128.

Parameters

in	the input image to flatten.

Returns

a newly allocated image which is flattened.

```
674                                 {
675     Img * answer = newImgCopy(in);
676     for(int i = 0; i < in->height*in->width; i++) {
677         int m = in->data[i];
678         float v = (m-128.0)/128.;
679         float w = v*v;
680         answer->data[i]=v>0?INBYTE(w*128+128):INBYTE(128-w*128);
681     }
682     return answer;
683 }
```

References img::data, img::height, INBYTE, newImgCopy(), and img::width.

4.17.2.13 imgGetVal()

```
unsigned char imgGetVal (
              Img * p,
              int x,
              int y )
```

Get the value a pixel for a given color. Zero is returned if the pixel is out of the picture.

Parameters

p	the picture
x	the horizontal position
y	the vertical position
ch	the channel/color : 0 for red, 1 for green, 2 for blue.

Returns

the value of the pixel.

```
501                                                 {
502     return (y<0 || y>=p->height)?0:(x<0 || x>=p->width)?0:p->data[y*p->width+x];
503 }
```

References img::data, img::height, and img::width.

4.17.2.14 imgGetWeight()

```
int imgGetWeight (
              Img * in )
```

Return the sum of all pixels in the picture.

Parameters

in	a picture

Returns

the sum of all pixels.

```
646                         {
647     int answer =0;
648     for(int i = 0; i < in->height*in->width; i++) {
649         answer+=in->data[i];
650     }
651     return answer;
652 }
```

References img::data, img::height, and img::width.

4.17.2.15 imgInvert()

```
Img * imgInvert (
            Img * in )
```

Inverts an image.

Parameters

in	the input image to invert.

Returns

a newly allocated image which is the invese of image in.

```
659                         {
660     Img * answer = newImgCopy(in);
661     for(int i = 0; i < in->height*in->width; i++) {
662         answer->data[i]=255-in->data[i];
663     }
664     return answer;
665 }
```

References img::data, img::height, newImgCopy(), and img::width.

4.17.2.16 imgLuminosityScale()

```
Img * imgLuminosityScale (
            Img * in )
```

Spread luminosity in the picture.

Parameters

in	the picture to spread luminosity

Returns

the newly allocated picture with spreaded luminosity.

```
766                                 {
767     int lumCount[256];
768     memset(lumCount,0,sizeof(int)«8);
769     Img * answer = newImgCopy(in);
770     for(int i = 0; i < in->height*in->width; i++) {
771         lumCount[in->data[i]]++;
772     }
773     // average value is a celle of lumCount should
774     // be avg=(in->height*in->width/256
775     // we are going to suppress below and above avg/8
776     int threshold=(in->height*in->width)»12; // divide by 256*8
777     if (threshold<2) threshold=2;
778     int topLum=255;
779     while (topLum>0 && lumCount[topLum]<threshold) --topLum;
780     int botLum=0;
781     while (botLum<topLum && lumCount[botLum]<threshold) ++botLum;
782     //topLum=100;
783     for(int i = 0; i < in->height*in->width; i++) {
784         answer->data[i]=
785             INBYTE((in->data[i]-botLum)*255/(topLum-botLum));
786     }
787     return answer;
788 }
```

References img::data, img::height, INBYTE, newImgCopy(), and img::width.

4.17.2.17 imgMain()

```
int imgMain (
              int ,
              char **  )
```

4.17.2.18 imgMake3dEffect()

```
Img * imgMake3dEffect (
              Img * in )
704                             {
705     Img * answer = newImgColor(in->width,in->height+in->width,255);
706
707     for(int x = 0; x < in->width; x++) {
708         for(int y = 0; y < in->height; y++) {
709             answer->data[x + (in->width-x-1+y) * answer->width]=
710                 in->data[x + y * in->width];
711         }
712     }
713     return answer;
714 }
```

References img::data, img::height, newImgColor(), and img::width.

4.17.2.19 imgPrint()

```
void imgPrint (
              Img * myImg )
```

Displays an image with numerical values.

Parameters

myImg	image to display in the terminal.

```
509                         {
510     for(int y = 0; y < myImg->height; y++) {
511         for(int x = 0; x < myImg->width; x++) {
512             printf("%3d ",imgGetVal(myImg,x,y));
513         }
514         printf("\n");
515     }
516 }
```

References img::height, imgGetVal(), and img::width.

4.17.2.20 imgRaiseContrast()

```
Img * imgRaiseContrast (
              Img * in )
```

Raises the contrast of an image We perform that by computing the square root of normalized difference with 128.

Parameters

in	the input image to flatten.

Returns

a newly allocated image which is flattened.

```
692                             {
693     Img * answer = newImgCopy(in);
694     for(int i = 0; i < in->height*in->width; i++) {
695         int m = in->data[i];
696         float v =(m-128.0)/128.;
697         float w = v<0?sqrtf(-v):sqrt(v);
698         answer->data[i]=v>0?INBYTE(w*128+128):INBYTE(128-w*128);
699     }
700     return answer;
701 }
```

References img::data, img::height, INBYTE, newImgCopy(), and img::width.

4.17.2.21 imgRotate()

```
Img * imgRotate (
              Img * img,
              int deg )
```

Rotates an image.

Parameters

img	to rotate
deg	degrees to rotate

Returns

the down sampled image.

See also

http://www.leptonica.org/rotation.html

```
1032                                {
1033     Img * answer = newImgColor(img->width,img->height,255);
1034     float rad=deg*3.14159/180;
1035     float c = cos(rad);
1036     float s = sin(rad);
1037     int w=img->width;
1038     int h=img->height;
1039     float eps=0.0001;    // epsilum, considered as zero
1040     for (int x=0;x<w;++x) {
1041         for (int y=0;y<h;++y) {
1042             double xd= (x-w/2)*c - (y-h/2)*s + w/2;
1043             double yd= (x-w/2)*s + (y-h/2)*c + h/2;
1044             if (xd>0 && yd>0 && xd<w-1 && yd<h-1) {
1045                 double xf=floor(xd);
1046                 double xc=ceil(xd);
1047                 double yf=floor(yd);
1048                 double yc=ceil(yd);
1049                 double ff=sqrt((xd-xf)*(xd-xf)+(yd-yf)*(yd-yf));
1050                 double fc=sqrt((xd-xf)*(xd-xf)+(yd-yc)*(yd-yc));
1051                 double cf=sqrt((xd-xc)*(xd-xc)+(yd-yf)*(yd-yf));
1052                 double cc=sqrt((xd-xc)*(xd-xc)+(yd-yc)*(yd-yc));
1053                 int v=255;
1054                 if (-eps<ff && ff<eps) {
1055                     v=img->data[(int)(xf+w*yf)];
1056                 } else if (-eps<fc && fc<eps) {
1057                     v=img->data[(int)(xf+w*yc)];
1058                 } else if (-eps<cf && cf<eps) {
1059                     v=img->data[(int)(xc+w*yf)];
1060                 } else if (-eps<cc && cc<eps) {
1061                     v=img->data[(int)(xc+w*yc)];
1062                 } else {
1063                     // not too close to any point, so compute average
1064                     double t=1/ff+1/fc+1/cf+1/cc;
1065                     v=INBYTE((int)((img->data[(int)(xf+w*yf)]/ff+
1066                                     img->data[(int)(xf+w*yc)]/fc+
1067                                     img->data[(int)(xc+w*yf)]/cf+
1068                                     img->data[(int)(xc+w*yc)]/cc)/t));
1069                 }
1070                 answer->data[x+w*y]=v;
1071             }
1072         }
1073     }
1074     return answer;
1075 }
```

References img::data, img::height, INBYTE, newImgColor(), and img::width.

4.17.2.22 imgRotate90()

```
Img * imgRotate90 (
            Img * img )
```

Rotates an image by 90 degrees.

Resulting image exchange height and width with original image.

Parameters

img	to rotate

Returns

rotated image

```
1013                            {
1014     // exchange height and width
1015     Img * answer = newImgColor(img->height,img->width,255);
1016     for (int y=0;y<answer->height;++y) {
1017         int o = answer->width*y;
1018         for (int x=0;x<answer->width;++x) {
1019             answer->data[x+o]=img->data[y+img->width*x];
1020         }
1021     }
1022     return answer;
1023 }
```

References img::data, img::height, newImgColor(), and img::width.

4.17.2.23 imgScalar()

```
unsigned char imgScalar (
              Img * i1,
              Img * i2,
              int xoffset,
              int yoffset )
```

Given a large image i1 and a smaller one i2 compute the scaler product of i2 with the subset of i1 at offset (xoffset,yoffset).

Parameters

i1	the large image.
i2	the small image
xoffset	horizontal offset to apply.
yoffset	vertical offset to apply.

Returns

the scalar product of i2 with a sub image of i1.

```
1117                                                                    {
1118     if (i1->width<i2->width+xoffset)
1119         ERROR("Wrong size","");
1120     if (i1->height<i2->height+yoffset)
1121         ERROR("Wrong size","");
1122     long int s=0;
1123     for (int y=0;y<i2->height;++y) {
1124         for (int x=0;x<i2->width;++x) {
1125             int v=i1->data[x+xoffset+(y+yoffset)*i1->width]
1126                 -i2->data[x+y*i1->width];
1127             s+=v<0?-v:v;
1128         }
1129     }
1130     int max=i2->height*i2->width*50;
1131     if (max<s) return 0;
1132     return INBYTE(255-(s*255)/max);
1133 }
```

References img::data, ERROR, img::height, INBYTE, and img::width.

4.17.2.24 imgScale()

```
Img * imgScale (
            Img * in,
            int s )
```

Scales an image to a larger image.

Parameters

in	the picture to blur
s	factor to scale

Returns

the newly allocated picture.

```
721                             {
722     Img * answer = newImgColor(in->width*s,in->height*s,255);
723
724     for(int x = 0; x < in->width; x++) {
725         for(int y = 0; y < in->height; y++) {
726             for(int xx = 0; xx < s; xx++) {
727                 for(int yy = 0; yy < s; yy++) {
728                     answer->data[s*x +xx + (yy+s*y) * answer->width]=
729                         in->data[x + y * in->width];
730                 }
731             }
732         }
733     }
734     return answer;
735 }
```

References img::data, img::height, newImgColor(), and img::width.

4.17.2.25 imgWrite()

```
void imgWrite (
            Img * myImg,
            char * filename )
```

Writes a Img struct to a png file.

Parameters

myImg	an existing image to save to a file.
filename	name of the png to write.

```
577                                     {
578     FILE *fp = fopen(filename, "wb");
579     if(!fp) {
580         ERROR("could not open file ",filename);
581     }
582
583     png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
584     if (!png) {
585         ERROR("could not create png structure","");
586     }
587
588     png_infop info = png_create_info_struct(png);
589     if (!info) {
```

```
590         ERROR("could not get info","");
591     }
592
593     if (setjmp(png_jmpbuf(png)))  {
594         fprintf(stderr,
595                 "could not jmp to data while trying to write %s\n",
596                 filename);
597     }
598
599     png_init_io(png, fp);
600
601     // Output is 8bit depth, RGBA format.
602     png_set_IHDR(png,
603                  info,
604                  myImg->width, myImg->height,
605                  8,
606                  PNG_COLOR_TYPE_RGBA,
607                  PNG_INTERLACE_NONE,
608                  PNG_COMPRESSION_TYPE_DEFAULT,
609                  PNG_FILTER_TYPE_DEFAULT
610                  );
611
612     png_write_info(png, info);
613
614     // To remove the alpha channel for PNG_COLOR_TYPE_RGB format,
615     // Use png_set_filler().
616     //png_set_filler(png, 0, PNG_FILLER_AFTER);
617
618     if (!myImg->data) abort();
619
620     unsigned char* row_pointers[myImg->height];
621     for(int y = 0; y < myImg->height; y++) {
622         row_pointers[y] =
623             (unsigned char*)malloc(4*myImg->width);
624         unsigned char* row = row_pointers[y];
625         for(int x = 0; x < myImg->width; x++) {
626             unsigned char * px = &(row[x * 4]);
627             px[0]=px[1]=px[2]=myImg->data[x+y*myImg->width];
628             px[3]=255;
629         }
630     }
631     png_write_image(png, row_pointers);
632
633     for(int y = 0; y < myImg->height; y++) {
634         free(row_pointers[y]);
635     }
636     png_write_end(png, NULL);
637     fclose(fp);
638     png_destroy_write_struct(&png, &info);
639 }
```

References img::data, ERROR, img::height, and img::width.

4.17.2.26 newImg9By9Dots()

```
Img * newImg9By9Dots (
            int w )
```

create an image with 9x9 black dots on a picture of size w times w.

Parameters

w	side of the picture

Returns

a newly allocated image.

```
184                             {
185     Img * answer = newImgColor(w,w,255);
```

```
186     float step=w/9.;
187     float stepOverTwo=step/2.;
188     float stepOverThree=step/2;
189     float stepOverThreeSq=stepOverThree*stepOverThree;
190     for (int x=0;x<w;++x) {
191         for (int y=0;y<w;++y) {
192             float xr = round((x-stepOverTwo)/step)*step+stepOverTwo;
193             float yr = round((y-stepOverTwo)/step)*step+stepOverTwo;
194             float d = ((x-xr)*(x-xr)+(y-yr)*(y-yr))/stepOverThreeSq;
195             if (d<1) {
196                 answer->data[x+w*y]=INBYTE((int)(d*255));
197             }
198         }
199     }
200     return answer;
201 }
```

References img::data, INBYTE, and newImgColor().

4.17.2.27 newImgColor()

```
Img * newImgColor (
            int w,
            int h,
            unsigned char c )
```

create an image of a given color

Parameters

filename	name of the file to read

Returns

a newly allocated image.

```
43                                                  {
44     Img * answer = (Img *)malloc(sizeof(struct img));
45     answer->width=w;
46     answer->height=h;
47     answer->data=
48         (unsigned char*)malloc(answer->height * answer->width);
49     memset(answer->data,c,answer->height * answer->width);
50     return answer;
51 }
```

References img::data, img::height, and img::width.

4.17.2.28 newImgCopy()

```
Img * newImgCopy (
            Img * myImg )
```

create an image from an exiting Img instance

Parameters

myImg	an existing image to copy

Returns

a newly allocated image.

```
464                             {
465     Img * answer = (Img *) malloc(sizeof(struct img));
466
467     answer->width      = myImg->width      ;
468     answer->height     = myImg->height     ;
469     answer->data = (unsigned char*)malloc(sizeof(char) *
470                                           answer->height*answer->width);
471     memcpy(answer->data,
472            myImg->data,
473            answer->width*answer->height);
474     return answer;
475 }
```

References img::data, img::height, and img::width.

4.17.2.29 newImgCross()

```
Img * newImgCross (
              int s,
              int w,
              int t )
```

create an image with a black cross on a w by w white background.

Parameters

s	size of the cross
w	side of the image
t	thickness of the cross

Returns

a newly allocated image.

```
121                                     {
122     Img * answer = newImgColor(w,w,255);
123     int m=w/2;
124     int count=0;
125     while (count<t && count<m) {
126         if (s<w-1 && s>0) {
127             for (int y=m-s/2;y<m-s/2+s;++y) {
128                 answer->data[m+count+w*y]=0;
129             }
130             for (int x=m-s/2;x<m-s/2+s;++x) {
131                 answer->data[x+w*(m+count)]=0;
132             }
133             for (int y=m-s/2;y<m-s/2+s;++y) {
134                 answer->data[m-count+w*y]=0;
135             }
136             for (int x=m-s/2;x<m-s/2+s;++x) {
137                 answer->data[x+w*(m-count)]=0;
138             }
139         }
140         count++;
141     }
142     return answer;
143 }
```

References img::data, and newImgColor().

4.17.2.30 newImgFromArray()

```
Img * newImgFromArray (
            int w,
            int h,
            unsigned char * buffer )
```

Creates an image from an array of unsigned char.

Parameters

w	width of the picture in pixels
h	height of the picture in pixels
buffer	a buffer of w times h grey pixels.

Returns

corresponding structure to the data in the buffer

```
32                                                      {
33     Img * answer = newImgColor(w,h,0);
34     memcpy(answer->data,buffer,w*h);
35     return answer;
36 }
```

References img::data, and newImgColor().

4.17.2.31 newImgNDotsHori()

```
Img * newImgNDotsHori (
            int N,
            int w )
```

Creates an image with equaly separated N black dots on a picture of size w by 1 pixel.

First point is located at w/N/2 from the left border of the picture, then second at w/2/N+w/N, third at w/2/N+2∗w/N, N-th point is located at w/2/N+(N-1)∗w/N=w-w/2/N. So N-th point is at w/N/2 from the right border of the picture.

Parameters

N	number of back dots to put
w	width of the picture

Returns

a newly allocated image.

```
217                                 {
218     // create a w by 1 white image
219     Img * answer = newImgColor(w,1,255);
220     // space between dots is w/N
221     float step=((float)w)/((float)N);
222     // we begin at step/2 from the border
223     float stepOverTwo=step/2.;
224     float stepOverFour=step/4;
```

```
225     float stepOverFourSq=stepOverFour*stepOverFour;
226     for (int x=0;x<w;++x) {
227         // horizontal position of the closest black dot
228         float xr = round((x-stepOverTwo)/step)*step+stepOverTwo;
229         // compute distance to the closest black dot
230         float d = fabs(xr-x);
231         float dmax=3;
232         if (d<dmax) {
233             // we are close to a black dot so we darken the pixel
234             answer->data[x]=INBYTE((int)(d*255/dmax));
235         }
236     }
237     return answer;
238 }
```

References img::data, INBYTE, and newImgColor().

4.17.2.32 newImgRead()

```
Img * newImgRead (
            char * filename )
```

create an image from a png or jpeg file

Parameters

filename	name of the file to read

Returns

a newly allocated image.

```
439                                 {
440     int l = strlen(filename);
441     if ( (l>4 &&
442           filename[l-4]=='.' &&
443           (filename[l-3]=='j'|| filename[l-3]=='J')&&
444           (filename[l-2]=='p'|| filename[l-2]=='P')&&
445           (filename[l-1]=='g'|| filename[l-1]=='G'))
446          ||
447          (l>5 &&
448           filename[l-5]=='.' &&
449           (filename[l-4]=='j'|| filename[l-4]=='J')&&
450           (filename[l-3]=='p'|| filename[l-3]=='P')&&
451           (filename[l-2]=='e'|| filename[l-2]=='E')&&
452           (filename[l-1]=='g'|| filename[l-1]=='G'))) {
453         return newImgReadJpeg(filename);
454     }
455     // otherwise we assume it is a png
456     return newImgReadPng(filename);
457 }
```

References newImgReadJpeg(), and newImgReadPng().

4.17.2.33 newImgSquare()

```
Img * newImgSquare (
            int s,
            int w,
            int t )
```

create an image with a black square on a w by w white background.

Parameters

s	side of the square in pixel
w	side of the image
t	thickness

Returns

a newly allocated image.

```
153                                     {
154     Img * answer = newImgColor(w,w,255);
155     int m=w/2;
156     int count=0;
157     while (count<t) {
158         if (s<w-1 && s>0) {
159             for (int y=m-s/2;y<m-s/2+s;++y) {
160                 answer->data[m-s/2+w*y]=0;
161             }
162             for (int y=m-s/2;y<m-s/2+s;++y) {
163                 answer->data[m-s/2+s-1+w*y]=0;
164             }
165             for (int x=m-s/2;x<m-s/2+s;++x) {
166                 answer->data[x+w*(m-s/2)]=0;
167             }
168             for (int x=m-s/2;x<m-s/2+s;++x) {
169                 answer->data[x+w*(m-s/2+s-1)]=0;
170             }
171         }
172         count++;
173         s--;
174     }
175     return answer;
176 }
```

References img::data, and newImgColor().

4.17.2.34 newImgSudoku()

```
Img * newImgSudoku (
              int sz )
```

create an image of a given color

Parameters

filename	name of the file to read

Returns

a newly allocated image.

```
245                             {
246     int w=212;
247     int h=212;
248     Img * answer = newImgColor(w,h,255);
249     int xinit=224/2-sz/2;
250     int yinit=xinit;
251     int step=sz/9;
252     for (int x=0;x<10;++x) {
253         int xx=xinit+x*step;
254         for (int yy=yinit;yy<yinit+step*9-1;++yy) {
255             if (xx>=0 && yy>=0 && xx<w && yy<h)
256                 answer->data[xx+w*yy]=0;
```

```
257         }
258     }
259     answer->data[1+xinit+9*step+w*yinit]=0;
260     answer->data[xinit+w*(yinit+1+9*step)]=0;
261     for (int y=0;y<10;++y) {
262         int yy=yinit+y*step;
263         for (int xx=xinit;xx<xinit+step*9-1;++xx) {
264             if (xx>=0 && yy>=0 && xx<w && yy<h)
265                 answer->data[xx+w*yy]=0;
266         }
267     }
268     for (int x=0;x<=3;++x) {
269         int xx=xinit+x*step*3;
270         for (int yy=yinit;yy<yinit+step*9-1;++yy) {
271             if (xx>=0 && yy>=0 && xx<w && yy<h)
272                 answer->data[1+xx+w*(yy+1)]=0;
273         }
274     }
275     for (int y=0;y<=3;++y) {
276         int yy=yinit+y*step*3;
277         for (int xx=xinit;xx<xinit+step*9-1;++xx) {
278             if (xx>=0 && yy>=0 && xx<w && yy<h)
279                 answer->data[1+xx+w*(yy+1)]=0;
280         }
281     }
282     return answer;
283 }
```

References img::data, and newImgColor().

4.17.2.35 newImgVerticalBar()

```
Img * newImgVerticalBar (
            int s,
            int w )
```

create an image of a vertical black bar on a square white background

Parameters

s	size of the image
w	width of the back bar

Returns

a newly allocated image.

```
59                                      {
60     Img * answer = newImgColor(s,s,255);
61     int m=s/2;
62     for (int x=m-w/2;x<m-w/2+w;++x) {
63         for (int y=0;y<s;++y) {
64             answer->data[x+s*y]=0;
65         }
66     }
67     return answer;
68 }
```

References img::data, and newImgColor().

4.17.2.36 newImgVerticalBarInRect()

```
Img * newImgVerticalBarInRect (
              int sx,
              int sy,
              int w )
```

create an image of a vertical black bar on a rectangular white background

Parameters

sx	horizontal size of the image
sy	vertical size of the image
w	width of the back bar

Returns

a newly allocated image.

```
78                                                      {
79     Img * answer = newImgColor(sx,sy,255);
80     int m=sx/2;
81     if (m<w) {
82         ERROR("Wrong size.","");
83     }
84     // put some grey arond the bar so that the
85     // equivalent filter is 0
86     for (int x=m-w;x<=m+w;++x) {
87         for (int y=0;y<sy;++y) {
88             answer->data[x+sx*y]=128;
89         }
90     }
91     for (int x=m-w/2;x<m-w/2+w;++x) {
92         for (int y=0;y<sy;++y) {
93             answer->data[x+sx*y]=0;
94         }
95     }
96     return answer;
97 }
```

References img::data, ERROR, and newImgColor().

4.18 img.h

Go to the documentation of this file.

```
1 #ifndef IMG_H
2 #define IMG_H
3
4 #include "util.h"
5
9 struct img {
11     int width;
13     int height;
17     unsigned char * data;
18 };
19
23 typedef struct img Img;
24
25 /* external data types */
26 typedef struct filter Filter;
27
28 Img * newImgFromArray(int w, int h, unsigned char *s);
29 Img * newImgColor(int w, int h, unsigned char c);
30 Img * newImgRead(char *filename);
31 Img * newImgCopy(Img*myImg);
32 Img * newImg9By9Dots(int w);
33 Img * newImgNDotsHori(int N,int w);
34 Img * newImgSudoku(int sz);
```

```
35 Img * newImgSquare(int s,int w,int t);
36 Img * newImgCross(int s,int w,int t);
37 Img * newImgVerticalBar(int s,int w);
38 Img * newImgVerticalBarInRect(int sx, int sy, int w);
39 void deleteImg(Img*myImg);
40 Img * imgDivideByTwo(Img* img);
41 unsigned char imgGetVal(Img*p,int x, int y);
42 void imgPrint(Img*myImg);
43 void imgWrite(Img*myImg,char *filename);
44 Img * imgInvert(Img*in);
45 Img * imgFlattenContrast(Img*in);
46 Img * imgRaiseContrast(Img*in);
47 Img * imgBlur(Img*in,int radius);
48 Img * imgMake3dEffect(Img*in);
49 Img * imgLuminosityScale(Img*in);
50 Img * imgConvolution(Img*in,Filter*filter);
51 Img * imgConvolutionDiff(Img*in,Filter*filter);
52 Img * imgConvolutionSameSize(Img*in,Filter*filter);
53 Img * imgConvolutionSameSizeDiff(Img*in,Filter*filter);
54 Img * imgDownSampleAvg(Img* img,int poolsize,int stride);
55 Img * imgDownSampleMax(Img* img,int poolsize,int stride);
56 void imgDrawRect(Img*myImg,int xmin,int ymin,int xmax,int ymax);
57 unsigned char imgScalar(Img*,Img*,int,int);
58 Img * imgRotate(Img* img,int deg);
59 Img * imgRotate90(Img* img);
60 Img * imgScale(Img*in,int s);
61 Img * imgExtract(Img*myImg,int xmin,int ymin,int xmax,int ymax);
62 int imgGetWeight(Img*in);
63 int imgMain(int,char**);
64
65 #endif
66
```

4.19 imgfam.c File Reference

```
#include <limits.h>
#include "imgfam.h"
#include "filterfam.h"
```

Functions

- ImgFam * newImgFam (int c)

  create a a new familly of images.
- void deleteImgFam (ImgFam *ifa)

  Deletes a famelly of images.
- void imgFamSetImg (ImgFam *imgFam, int i, Img *img)

  set the i-th image of a familly of images.
- ImgFam * imgFamLuminosityScale (ImgFam *imgFam)

  applies the imgLuminosityScale function to all images in a familly.
- ImgFam * imgFamDownSampleMax (ImgFam *imgFam, int poolsize, int stride)

  dowm sample all images in a familly using a max pooling strategy.
- ImgFam * imgFamDownSampleAvg (ImgFam *imgFam, int poolsize, int stride)

  dowm sample all images in a familly using an average pooling strategy.
- void imgFamWrite (ImgFam *imgFam, char *basename)

  Saves as png files the image familly.
- ImgFam * imgFamRead (char *basename)

  Read a set of png files to return an image familly.
- Img * imgFamScalar (ImgFam *if1, FilterFam *if2)
- void imgFamPrint (ImgFam *ifa)

4.19.1 Function Documentation

4.19.1.1 deleteImgFam()

```
void deleteImgFam (
            ImgFam * ifa )
```

Deletes a famelly of images.

Parameters

ifa	the familly to delete.

```
21                                  {
22     if (ifa==NULL) return;
23     for (int i=0;i<ifa->count;++i) {
24         deleteImg(ifa->imgs[i]);
25     }
26     memset(ifa->imgs,0,sizeof(Img*)*ifa->count);
27     free(ifa->imgs);
28     memset(ifa,0,sizeof(ImgFam));
29     free(ifa);
30 }
```

References imgfam::count, deleteImg(), and imgfam::imgs.

4.19.1.2 imgFamDownSampleAvg()

```
ImgFam * imgFamDownSampleAvg (
            ImgFam * imgFam,
            int poolsize,
            int stride )
```

dowm sample all images in a familly using an average pooling strategy.

Parameters

imgFam	the familly of images in which we are going to down sample.
poolsize	size of the square in which we are taking the average
stride	steps between two squares to get average.

```
80                                                                  {
81     ImgFam * answer = newImgFam (imgFam->count);
82     for (int i=0;i<imgFam->count;++i) {
83         Img * nimg = imgDownSampleAvg(imgFam->imgs[i],poolsize,stride);
84         imgFamSetImg(answer,i,nimg);
85     }
86     return answer;
87 }
```

References imgfam::count, imgDownSampleAvg(), imgFamSetImg(), imgfam::imgs, and newImgFam().

4.19.1.3 imgFamDownSampleMax()

```
ImgFam * imgFamDownSampleMax (
              ImgFam * imgFam,
              int poolsize,
              int stride )
```

dowm sample all images in a familly using a max pooling strategy.

Parameters

imgFam	the familly of images in which we are going to down sample.
poolsize	size of the square in which we are taking the max.
stride	steps between two squares to get maximum.

```
65                                                                  {
66     ImgFam * answer = newImgFam (imgFam->count);
67     for (int i=0;i<imgFam->count;++i) {
68         Img * nimg = imgDownSampleMax(imgFam->imgs[i],poolsize,stride);
69         imgFamSetImg(answer,i,nimg);
70     }
71     return answer;
72 }
```

References imgfam::count, imgDownSampleMax(), imgFamSetImg(), imgfam::imgs, and newImgFam().

4.19.1.4 imgFamLuminosityScale()

```
ImgFam * imgFamLuminosityScale (
              ImgFam * imgFam )
```

applies the imgLuminosityScale function to all images in a familly.

Parameters

imgFam	the familly on which we want to apply imgLuminosityScale.

Returns

the newly allocated familly where imgLuminosityScale has been applied.

```
51                                          {
52     ImgFam * answer = newImgFam(imgFam->count);
53     for (int i=0;i<imgFam->count;++i) {
54         imgFamSetImg(answer,i,imgLuminosityScale(imgFam->imgs[i]));
55     }
56     return answer;
57 }
```

References imgfam::count, imgFamSetImg(), imgLuminosityScale(), imgfam::imgs, and newImgFam().

4.19.1.5 imgFamPrint()

```
void imgFamPrint (
              ImgFam * ifa )
```

```
160                                 {
161     for (int i=0;i<ifa->count;++i) {
162         printf("=-=-=-=-=-=-=-=-=-=-=-= %d/%d\n",i,ifa->count);
163         imgPrint(ifa->imgs[i]);
164     }
165 }
```

References imgfam::count, imgPrint(), and imgfam::imgs.

4.19.1.6 imgFamRead()

```
ImgFam * imgFamRead (
              char * basename )
```

Read a set of png files to return an image familly.

Parameters

basename	the base name for all files read. names read have the form basename_i.png where i is the number of the picture in the familly.

Returns

a newly allocated image familly.

See also

imgFamWrite

```
113                                     {
114     char s[99];
115     int count=0;
116     int found=1;
117     while (found) {
118         snprintf(s,99,"%s_%d.png",basename,count);
119         FILE * f = fopen(s,"r");
120         if (f!=NULL) {
121             fclose(f);
122             ++count;
123         } else  {
124             found=0;
125         }
126     }
127     ImgFam * answer = newImgFam(count);
128     for (int i=0;i<answer->count;++i) {
129         snprintf(s,99,"%s_%d.png",basename,i);
130         imgFamSetImg(answer,i,newImgRead(s));
131     }
132     return answer;
133 }
```

References imgfam::count, imgFamSetImg(), newImgFam(), and newImgRead().

4.19.1.7 imgFamScalar()

```
Img * imgFamScalar (
              ImgFam * if1,
              FilterFam * if2 )
```

```
135                                                 {
136     if (if1->count!=if2->count) {
137         ERROR("Wrong number of images.","");
138     }
139     if (if1->imgs[0]->width<if2->filters[0]->img->width) {
140         ERROR("Wrong size of images.","");
141     }
142     if (if1->imgs[0]->height<if2->filters[0]->img->height) {
143         ERROR("Wrong size of images.","");
144     }
145     int w = if1->imgs[0]->width-if2->filters[0]->img->width+1;
146     int h = if1->imgs[0]->height-if2->filters[0]->img->height+1;
147     Img * answer = newImgColor(w,h,0);
148     for (int xoffset=0;xoffset<w;++xoffset) {
149         for (int yoffset=0;yoffset<h;++yoffset) {
150             int s=0;
151             for (int i=0;i<if1->count;++i) {
152                 s+= imgScalar(if1->imgs[i],if2->filters[i]->img,xoffset,yoffset);
153             }
154             answer->data[xoffset+yoffset*w]=INBYTE(s/if1->count);
155         }
156     }
157     return answer;
158 }
```

References filterfam::count, imgfam::count, img::data, ERROR, filterfam::filters, img::height, filter::img, imgfam::imgs, imgScalar(), INBYTE, newImgColor(), and img::width.

4.19.1.8 imgFamSetImg()

```
void imgFamSetImg (
              ImgFam * imgFam,
              int i,
              Img * img )
```

set the i-th image of a familly of images.

Parameters

imgFam	the familly of images in which we are going to set the image.
i	index of the image to be in the collection.
img	the image that will be at i-th location in imgFam.

```
38                                                  {
39     if (i<0 && i>=imgFam->count)
40         ERROR("out of bounds.","");
41     imgFam->imgs[i]=img;
42 }
```

References imgfam::count, ERROR, and imgfam::imgs.

4.19.1.9 imgFamWrite()

```
void imgFamWrite (
              ImgFam * imgFam,
              char * basename )
```

Saves as png files the image familly.

Parameters

imgFam	the familly to save.
basename	the base name for all files save. The final name of each file will be basename_i.png where i is the number of the picture in the familly.

See also

imgFamRead

```
97                                              {
98     char s[99];
99     for (int i=0;i<imgFam->count;++i) {
100         snprintf(s,99,"%s_%d.png",basename,i);
101         imgWrite(imgFam->imgs[i],s);
102     }
103 }
```

References imgfam::count, imgfam::imgs, and imgWrite().

4.19.1.10 newImgFam()

```
ImgFam * newImgFam (
            int c )
```

create a a new familly of images.

Parameters

c	the size of the familly.

```
9                           {
10     ImgFam * answer = (ImgFam*)malloc(sizeof(struct imgfam));
11     answer->count=c;
12     answer->imgs=(Img**)malloc(sizeof(Img**)*c);
13     memset(answer->imgs,0,sizeof(Img*)*c);
14     return answer;
15 }
```

References imgfam::count, and imgfam::imgs.

4.20 imgfam.h File Reference

```
#include "img.h"
```

Classes

- struct imgfam

  A structure to store a collection (also called familly) of images.

Typedefs

- typedef struct imgfam ImgFam
  Short name for 'struct imgfam'.
- typedef struct filterfam FilterFam

Functions

- ImgFam * newImgFam (int)
  create a a new familly of images.
- void imgFamSetImg (ImgFam *, int, Img *)
  set the i-th image of a familly of images.
- ImgFam * imgFamApplyConvolution (ImgFam *, Filter *)
- ImgFam * imgFamDownSampleMax (ImgFam *, int, int)
  dowm sample all images in a familly using a max pooling strategy.
- ImgFam * imgFamDownSampleAvg (ImgFam *, int, int)
  dowm sample all images in a familly using an average pooling strategy.
- void imgFamWrite (ImgFam *, char *)
  Saves as png files the image familly.
- ImgFam * imgFamRead (char *)
  Read a set of png files to return an image familly.
- Img * imgFamScalar (ImgFam *, FilterFam *)
- void imgFamPrint (ImgFam *)
- ImgFam * imgFamLuminosityScale (ImgFam *in)
  applies the imgLuminosityScale function to all images in a familly.
- void deleteImgFam (ImgFam *)
  Deletes a famelly of images.

4.20.1 Typedef Documentation

4.20.1.1 FilterFam

```
typedef struct filterfam FilterFam
```

4.20.1.2 ImgFam

```
typedef struct imgfam ImgFam
```

Short name for 'struct imgfam'.

4.20.2 Function Documentation

4.20.2.1 deleteImgFam()

```
void deleteImgFam (
            ImgFam * ifa )
```

Deletes a famelly of images.

Parameters

ifa	the familly to delete.

```
21                                  {
22     if (ifa==NULL) return;
23     for (int i=0;i<ifa->count;++i) {
24         deleteImg(ifa->imgs[i]);
25     }
26     memset(ifa->imgs,0,sizeof(Img*)*ifa->count);
27     free(ifa->imgs);
28     memset(ifa,0,sizeof(ImgFam));
29     free(ifa);
30 }
```

References imgfam::count, deleteImg(), and imgfam::imgs.

4.20.2.2 imgFamApplyConvolution()

```
ImgFam * imgFamApplyConvolution (
              ImgFam * ,
              Filter *  )
```

4.20.2.3 imgFamDownSampleAvg()

```
ImgFam * imgFamDownSampleAvg (
              ImgFam * imgFam,
              int poolsize,
              int stride )
```

dowm sample all images in a familly using an average pooling strategy.

Parameters

imgFam	the familly of images in which we are going to down sample.
poolsize	size of the square in which we are taking the average
stride	steps between two squares to get average.

```
80                                                                  {
81     ImgFam * answer = newImgFam (imgFam->count);
82     for (int i=0;i<imgFam->count;++i) {
83         Img * nimg = imgDownSampleAvg(imgFam->imgs[i],poolsize,stride);
84         imgFamSetImg(answer,i,nimg);
85     }
86     return answer;
87 }
```

References imgfam::count, imgDownSampleAvg(), imgFamSetImg(), imgfam::imgs, and newImgFam().

4.20.2.4 imgFamDownSampleMax()

```
ImgFam * imgFamDownSampleMax (
              ImgFam * imgFam,
```

```
            int poolsize,
            int stride )
```

dowm sample all images in a familly using a max pooling strategy.

Parameters

imgFam	the familly of images in which we are going to down sample.
poolsize	size of the square in which we are taking the max.
stride	steps between two squares to get maximum.

```
65                                                                  {
66     ImgFam * answer = newImgFam (imgFam->count);
67     for (int i=0;i<imgFam->count;++i) {
68         Img * nimg = imgDownSampleMax(imgFam->imgs[i],poolsize,stride);
69         imgFamSetImg(answer,i,nimg);
70     }
71     return answer;
72 }
```

References imgfam::count, imgDownSampleMax(), imgFamSetImg(), imgfam::imgs, and newImgFam().

4.20.2.5 imgFamLuminosityScale()

```
ImgFam * imgFamLuminosityScale (
            ImgFam * imgFam )
```

applies the imgLuminosityScale function to all images in a familly.

Parameters

imgFam	the familly on which we want to apply imgLuminosityScale.

Returns

the newly allocated familly where imgLuminosityScale has been applied.

```
51                                              {
52     ImgFam * answer = newImgFam(imgFam->count);
53     for (int i=0;i<imgFam->count;++i) {
54         imgFamSetImg(answer,i,imgLuminosityScale(imgFam->imgs[i]));
55     }
56     return answer;
57 }
```

References imgfam::count, imgFamSetImg(), imgLuminosityScale(), imgfam::imgs, and newImgFam().

4.20.2.6 imgFamPrint()

```
void imgFamPrint (
            ImgFam * ifa )
160                             {
161     for (int i=0;i<ifa->count;++i) {
162         printf("=-=-=-=-=-=-=-=-=-=-=-= %d/%d\n",i,ifa->count);
163         imgPrint(ifa->imgs[i]);
164     }
165 }
```

References imgfam::count, imgPrint(), and imgfam::imgs.

4.20.2.7 imgFamRead()

```
ImgFam * imgFamRead (
            char * basename )
```

Read a set of png files to return an image familly.

Parameters

basename	the base name for all files read. names read have the form basename_i.png where i is the number of the picture in the familly.

Returns

a newly allocated image familly.

See also

imgFamWrite

```
113                                     {
114     char s[99];
115     int count=0;
116     int found=1;
117     while (found) {
118         snprintf(s,99,"%s_%d.png",basename,count);
119         FILE * f = fopen(s,"r");
120         if (f!=NULL) {
121             fclose(f);
122             ++count;
123         } else  {
124             found=0;
125         }
126     }
127     ImgFam * answer = newImgFam(count);
128     for (int i=0;i<answer->count;++i) {
129         snprintf(s,99,"%s_%d.png",basename,i);
130         imgFamSetImg(answer,i,newImgRead(s));
131     }
132     return answer;
133 }
```

References imgfam::count, imgFamSetImg(), newImgFam(), and newImgRead().

4.20.2.8 imgFamScalar()

```
Img * imgFamScalar (
            ImgFam * if1,
            FilterFam * if2 )
135                                                 {
136     if (if1->count!=if2->count) {
137         ERROR("Wrong number of images.","");
138     }
139     if (if1->imgs[0]->width<if2->filters[0]->img->width) {
140         ERROR("Wrong size of images.","");
141     }
142     if (if1->imgs[0]->height<if2->filters[0]->img->height) {
143         ERROR("Wrong size of images.","");
144     }
145     int w = if1->imgs[0]->width-if2->filters[0]->img->width+1;
146     int h = if1->imgs[0]->height-if2->filters[0]->img->height+1;
147     Img * answer = newImgColor(w,h,0);
148     for (int xoffset=0;xoffset<w;++xoffset) {
149         for (int yoffset=0;yoffset<h;++yoffset) {
```

```
150             int s=0;
151             for (int i=0;i<if1->count;++i) {
152                 s+= imgScalar(if1->imgs[i],if2->filters[i]->img,xoffset,yoffset);
153             }
154             answer->data[xoffset+yoffset*w]=INBYTE(s/if1->count);
155         }
156     }
157     return answer;
158 }
```

References filterfam::count, imgfam::count, img::data, ERROR, filterfam::filters, img::height, filter::img, imgfam::imgs, imgScalar(), INBYTE, newImgColor(), and img::width.

4.20.2.9 imgFamSetImg()

```
void imgFamSetImg (
              ImgFam * imgFam,
              int i,
              Img * img )
```

set the i-th image of a familly of images.

Parameters

imgFam	the familly of images in which we are going to set the image.
i	index of the image to be in the collection.
img	the image that will be at i-th location in imgFam.

```
38                                                  {
39     if (i<0 && i>=imgFam->count)
40         ERROR("out of bounds.","");
41     imgFam->imgs[i]=img;
42 }
```

References imgfam::count, ERROR, and imgfam::imgs.

4.20.2.10 imgFamWrite()

```
void imgFamWrite (
              ImgFam * imgFam,
              char * basename )
```

Saves as png files the image familly.

Parameters

imgFam	the familly to save.
basename	the base name for all files save. The final name of each file will be basename_i.png where i is the number of the picture in the familly.

See also

imgFamRead

```
97                                              {
98     char s[99];
99     for (int i=0;i<imgFam->count;++i) {
100         snprintf(s,99,"%s_%d.png",basename,i);
101         imgWrite(imgFam->imgs[i],s);
102     }
103 }
```

References imgfam::count, imgfam::imgs, and imgWrite().

4.20.2.11 newImgFam()

```
ImgFam * newImgFam (
            int c )
```

create a a new familly of images.

Parameters

c	the size of the familly.

```
9                           {
10     ImgFam * answer = (ImgFam*)malloc(sizeof(struct imgfam));
11     answer->count=c;
12     answer->imgs=(Img**)malloc(sizeof(Img**)*c);
13     memset(answer->imgs,0,sizeof(Img*)*c);
14     return answer;
15 }
```

References imgfam::count, and imgfam::imgs.

4.21 imgfam.h

Go to the documentation of this file.

```
1 #ifndef IMGFAM_H
2 #define IMGFAM_H
3
4 #include "img.h"
5
10 struct imgfam {
12     int count;
14     Img ** imgs;
15 };
16
20 typedef struct imgfam ImgFam;
21
22 // external types
23 typedef struct filterfam FilterFam;
24
25 ImgFam * newImgFam(int);
26 void imgFamSetImg(ImgFam*,int,Img*);
27 ImgFam * imgFamApplyConvolution(ImgFam*,Filter*);
28 ImgFam * imgFamDownSampleMax(ImgFam*,int,int);
29 ImgFam * imgFamDownSampleAvg(ImgFam*,int,int);
30 void imgFamWrite(ImgFam*,char*);
31 ImgFam * imgFamRead(char*);
32 Img* imgFamScalar(ImgFam*,FilterFam*);
33 void imgFamPrint(ImgFam*);
34 ImgFam* imgFamLuminosityScale(ImgFam*in);
35 void deleteImgFam(ImgFam *);
36
37 #endif
```

4.22 layer.c File Reference

```
#include "layer.h"
#include "filterfam.h"
#include "imgfam.h"
```

Functions

- Layer ∗ newLayer (char ∗name, char ∗convFilterLoc, int downSamplePoolSize, int downSampleStride)

  Allocates memory for a new layer.
- void deleteLayer (Layer ∗l)

  free space previously allocated for a layer.
- int layerCount (char ∗convFilterLoc)

  Get the number of images in the filter.
- ImgFam ∗ layerPassImg (Layer ∗l, Img ∗i)

  Passes an image through a layer.
- ImgFam ∗ layerPassImgFam (Layer ∗l, ImgFam ∗i)

  Passes a familly of images through a layer.

4.22.1 Function Documentation

4.22.1.1 deleteLayer()

```
void deleteLayer (
            Layer * l )
```

free space previously allocated for a layer.

```
29                          {
30     free(l->name);
31     free(l->convFilterLoc);
32     deleteFilterFam(l->convFilter);
33     memset(l,0,sizeof(Layer));
34     free(l);
35 }
```

References layer::convFilter, layer::convFilterLoc, deleteFilterFam(), and layer::name.

4.22.1.2 layerCount()

```
int layerCount (
            char * convFilterLoc )
```

Get the number of images in the filter.

Parameters

convFilterLoc	location of the pictures. Name of pictures will be convFilterLoc_<num>.png

Returns

number of elements in this filter

```
43                                      {
44     return filterFamCount(convFilterLoc);
45 }
```

References filterFamCount().

4.22.1.3 layerPassImg()

```
ImgFam * layerPassImg (
            Layer * l,
            Img * i )
```

Passes an image through a layer.

Parameters

l	a layer
i	an image

Returns

output of i through l

```
53                                      {
54     ImgFam * intermediateOutput =
55         filterFamApplyConvolution(l->convFilter,i);
56     ImgFam * maxPoolOutput=imgFamDownSampleMax(intermediateOutput,
57                                                l->downSamplePoolSize,
58                                                l->downSampleStride);
59     deleteImgFam(intermediateOutput);
60     return maxPoolOutput;
61 }
```

References layer::convFilter, deleteImgFam(), layer::downSamplePoolSize, layer::downSampleStride, filterFamApplyConvolution(), and imgFamDownSampleMax().

4.22.1.4 layerPassImgFam()

```
ImgFam * layerPassImgFam (
            Layer * l,
            ImgFam * i )
```

Passes a familly of images through a layer.

Parameters

l	a layer
i	a familly of images

Returns

output of i through l

```
69                                          {
70     ImgFam * intermediateOutput =
71         filterFamApplyConvolutionOnFam(l->convFilter,i);
72     ImgFam * maxPoolOutput=imgFamDownSampleMax(intermediateOutput,
73                                                l->downSamplePoolSize,
74                                                l->downSampleStride);
75     return maxPoolOutput;
76 }
```

References layer::convFilter, layer::downSamplePoolSize, layer::downSampleStride, filterFamApplyConvolutionOnFam(), and imgFamDownSampleMax().

4.22.1.5 newLayer()

```
Layer * newLayer (
            char * name,
            char * convFilterLoc,
            int downSamplePoolSize,
            int downSampleStride )
```

Allocates memory for a new layer.

Parameters

name	name of this layer (a filename)
convFilterLoc	name of the convolution filter to load
downSamplePoolSize	pool size value for downsizing.
downSampleStride	stride value for downsizing.

```
16 {
17     Layer * answer = (Layer*)malloc(sizeof(struct layer));
18     answer->name = stringCopy(name);
19     answer->convFilterLoc = stringCopy(convFilterLoc);
20     answer->convFilter=filterFamRead(convFilterLoc);
21     answer->downSamplePoolSize=downSamplePoolSize;
22     answer->downSampleStride=downSampleStride;
23     return answer;
24 }
```

References layer::convFilter, layer::convFilterLoc, layer::downSamplePoolSize, layer::downSampleStride, filterFamRead(), layer::name, and stringCopy().

4.23 layer.h File Reference

```
#include "util.h"
```

Classes

- struct layer

Typedefs

- typedef struct filterfam FilterFam
- typedef struct imgfam ImgFam
- typedef struct img Img
- typedef struct layer Layer

Functions

- Layer * newLayer (char *name, char *convFilterLoc, int downSamplePoolSize, int downSampleStride)

  Allocates memory for a new layer.
- void deleteLayer (Layer *)

  free space previously allocated for a layer.
- ImgFam * layerPassImg (Layer *, Img *)

  Passes an image through a layer.
- ImgFam * layerPassImgFam (Layer *, ImgFam *)

  Passes a familly of images through a layer.
- int layerCount (char *convFilterLoc)

  Get the number of images in the filter.

4.23.1 Typedef Documentation

4.23.1.1 FilterFam

```
typedef struct filterfam FilterFam
```

4.23.1.2 Img

```
typedef struct img Img
```

4.23.1.3 ImgFam

```
typedef struct imgfam ImgFam
```

4.23.1.4 Layer

```
typedef struct layer Layer
```

4.23.2 Function Documentation

4.23.2.1 deleteLayer()

```
void deleteLayer (
            Layer * l )
```

free space previously allocated for a layer.

```
29                          {
30     free(l->name);
31     free(l->convFilterLoc);
32     deleteFilterFam(l->convFilter);
33     memset(l,0,sizeof(Layer));
34     free(l);
35 }
```

References layer::convFilter, layer::convFilterLoc, deleteFilterFam(), and layer::name.

4.23.2.2 layerCount()

```
int layerCount (
            char * convFilterLoc )
```

Get the number of images in the filter.

Parameters

convFilterLoc	location of the pictures. Name of pictures will be convFilterLoc_<num>.png

Returns

number of elements in this filter

```
43                                  {
44     return filterFamCount(convFilterLoc);
45 }
```

References filterFamCount().

4.23.2.3 layerPassImg()

```
ImgFam * layerPassImg (
            Layer * l,
            Img * i )
```

Passes an image through a layer.

Parameters

l	a layer
i	an image

Returns

output of i through l

```
53                                      {
54     ImgFam * intermediateOutput =
55         filterFamApplyConvolution(l->convFilter,i);
56     ImgFam * maxPoolOutput=imgFamDownSampleMax(intermediateOutput,
57                                                l->downSamplePoolSize,
58                                                l->downSampleStride);
59     deleteImgFam(intermediateOutput);
60     return maxPoolOutput;
61 }
```

References layer::convFilter, deleteImgFam(), layer::downSamplePoolSize, layer::downSampleStride, filterFamApplyConvolution(), and imgFamDownSampleMax().

4.23.2.4 layerPassImgFam()

```
ImgFam * layerPassImgFam (
             Layer * l,
             ImgFam * i )
```

Passes a familly of images through a layer.

Parameters

l	a layer
i	a familly of images

Returns

output of i through l

```
69                                          {
70     ImgFam * intermediateOutput =
71         filterFamApplyConvolutionOnFam(l->convFilter,i);
72     ImgFam * maxPoolOutput=imgFamDownSampleMax(intermediateOutput,
73                                                l->downSamplePoolSize,
74                                                l->downSampleStride);
75     return maxPoolOutput;
76 }
```

References layer::convFilter, layer::downSamplePoolSize, layer::downSampleStride, filterFamApplyConvolutionOnFam(), and imgFamDownSampleMax().

4.23.2.5 newLayer()

```
Layer * newLayer (
             char * name,
```

```
            char * convFilterLoc,
            int downSamplePoolSize,
            int downSampleStride )
```

Allocates memory for a new layer.

Parameters

name	name of this layer (a filename)
convFilterLoc	name of the convolution filter to load
downSamplePoolSize	pool size value for downsizing.
downSampleStride	stride value for downsizing.

```
16 {
17     Layer * answer = (Layer*)malloc(sizeof(struct layer));
18     answer->name = stringCopy(name);
19     answer->convFilterLoc = stringCopy(convFilterLoc);
20     answer->convFilter=filterFamRead(convFilterLoc);
21     answer->downSamplePoolSize=downSamplePoolSize;
22     answer->downSampleStride=downSampleStride;
23     return answer;
24 }
```

References layer::convFilter, layer::convFilterLoc, layer::downSamplePoolSize, layer::downSampleStride, filterFamRead(), layer::name, and stringCopy().

4.24 layer.h

Go to the documentation of this file.

```
1 #ifndef LAYER_H
2 #define LAYER_H
3
4 #include "util.h"
5
6 // external type
7 typedef struct filterfam FilterFam;
8 typedef struct imgfam ImgFam;
9 typedef struct img Img;
10
14 struct layer {
16     char * name;
18     char * convFilterLoc;
20     FilterFam * convFilter;
22     int downSamplePoolSize;
24     int downSampleStride;
25 };
26
27 typedef struct layer Layer;
28
29 Layer * newLayer(char * name,
30                  char * convFilterLoc,
31                  int downSamplePoolSize,
32                  int downSampleStride);
33 void deleteLayer(Layer*);
34 ImgFam * layerPassImg(Layer*,Img*);
35 ImgFam * layerPassImgFam(Layer*,ImgFam*);
36 int layerCount(char * convFilterLoc);
37
38 #endif
```

4.25 layer3.c File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include "l3.c"
#include "fontname.h"
```

Functions

- int main ()

4.25.1 Function Documentation

4.25.1.1 main()

```
int main ( )
5             {
6     int errorCount=0,caseCount=0;
7     init_digits();
8     for (int f=1;f<31;++f) {
9         // using font f
10         int fontError=0;
11         for (int d=1;d<10;++d) {
12             // trying to guess digit d
13             for (int i=1;i<10;++i) {
14                 // check if i is the highest
15                 if (d!=i) {
16                     if (digits[f][d][i]>=digits[f][d][d]) {
17                         // another one is the highest
18                         errorCount++;
19                         fontError++;
20                     }
21                 }
22                 caseCount++;
23             }
24         }
25         printf("font %d %s: \t%d errors\n",f,fontname[f],fontError);
26     }
27     printf("%d/%d\n",errorCount,caseCount);
28     return 0;
29 }
```

References fontname.

4.26 main.c File Reference

```
#include "img.h"
#include "digits.h"
#include "grid.h"
```

Functions

- void cnnExtractDigits (Img *myImg, int xmin, int ymin, int xmax, int ymax)

  Extract digits from a sudoku grid.
- void usage (FILE *f, char *name)

  Tells how to use this program.
- int main (int argc, char **argv)

4.26.1 Function Documentation

4.26.1.1 cnnExtractDigits()

```
void cnnExtractDigits (
              Img * myImg,
              int xmin,
              int ymin,
              int xmax,
              int ymax )
```

Extract digits from a sudoku grid.

Parameters

myImg	image from which we extract the digits.
xmin	horizontal position of lower left corner of the sudoku grid.
ymin	vertical position of lower left corner of the sudoku grid.
xmax	horizontal position of upper right corner of the sudoku grid.
ymax	vertical position of upper right corner of the sudoku grid.

```
14                                                                  {
15     int xstep=(xmax-xmin)/9;
16     int ystep=(ymax-ymin)/9;
17     int tenPercent=xstep/10;
18     for (int i=0;i<9;++i) {
19         for (int j=0;j<9;++j) {
20             Img * aDigit = imgExtract(myImg,
21                                       xmin+i*xstep+tenPercent,
22                                       ymin+j*ystep+tenPercent,
23                                       xmin+(i+1)*xstep-tenPercent,
24                                       ymin+(j+1)*ystep-tenPercent);
25             char s [99];
26             snprintf(s,99,"extracted_digit_%d_%d.png",i,j);
27             imgWrite(aDigit,s);
28             deleteImg(aDigit);
29         }
30     }
31 }
```

References deleteImg(), imgExtract(), and imgWrite().

4.26.1.2 main()

```
int main (
              int argc,
              char ** argv )
67                                {
68     char * progName = basename(argv[0]);
69     if (strcmp(progName,"img")==0) {
70         return imgMain(argc,argv);
71     } else if (strcmp(progName,"digits")==0) {
72         return digitsMain(argc,argv);
73     } else if (strcmp(progName,"grid")==0) {
74         return gridMain(argc,argv);
75     }
76
77     for (int j=1;j<argc;++j) {
78         if (strcmp(argv[j],"-h")==0 || strcmp(argv[j],"--help")==0) {
79             usage(stdout,argv[0]);
80             exit(0);
81         }
82     }
83     if(argc < 2) {
84         usage(stderr,argv[0]);
85         ERROR("At least 1 argument expected, the picture to read","");
86     }
87     /* read input image */
```

```
88      Img * rawInputImage=newImgRead(argv[1]);
89      Img * goodContastImage=imgLuminosityScale(rawInputImage);
90      Img * inverseImage = imgInvert(goodContastImage);
91      deleteImg(goodContastImage);
92      int scaleFactor=0;
93      while (inverseImage->width>400 && inverseImage->height>400) {
94          Img * i = imgDivideByTwo(inverseImage);
95          deleteImg(inverseImage);
96          inverseImage=i;
97          scaleFactor++;
98      }
99      int xmin,ymin,xmax,ymax;
100      // locate the grid
101      gridLocate(inverseImage,&xmin,&ymin,&xmax,&ymax);
102      HERE("Found sudoku grid :");
103      printf("%d %d %d %d\n",xmin,ymin,xmax,ymax);
104
105      imgDrawRect(inverseImage,xmin,ymin,xmax,ymax);
106      imgWrite(inverseImage,"afterstd.png");
107      // draw a square around the grid found in the
108      // original picture
109      imgDrawRect(rawInputImage,
110                  xmin«scaleFactor,
111                  ymin«scaleFactor,
112                  xmax«scaleFactor,
113                  ymax«scaleFactor);
114      // write the image
115      imgWrite(rawInputImage,"out.png");
116      HERE("grid detection in out.png");
117      cnnExtractDigits(rawInputImage,
118                       xmin«scaleFactor,
119                       ymin«scaleFactor,
120                       xmax«scaleFactor,
121                       ymax«scaleFactor);
122      // free allocated memory
123      deleteImg(inverseImage);
124      deleteImg(rawInputImage);
125      return 0;
126  }
```

References cnnExtractDigits(), deleteImg(), digitsMain(), ERROR, gridLocate(), gridMain(), img::height, HERE, imgDivideByTwo(), imgDrawRect(), imgInvert(), imgLuminosityScale(), imgMain(), imgWrite(), newImgRead(), usage(), and img::width.

4.26.1.3 usage()

```
void usage (
              FILE * f,
              char * name )
```

Tells how to use this program.

Parameters

f	where to write the info.
name	the value of argv[0].

```
38                              {
39      char * bname=basename(name);
40      fprintf(f,"This is %s version %s on archictecture %s.\n",
41              bname, VERSION,CFG_UNAME);
42      fprintf(f,"Usage:\n");
43      fprintf(f,"    %s <input-file> | <option> \n", bname);
44      fprintf(f,"Solves a Sudoku grid given a png or jpeg image as input.\n");
45      fprintf(f,"\n");
46      fprintf(f,"Where option is one of:\n");
47      fprintf(f,"    [-h|--help] :\n");
48      fprintf(f,"         Displays this help message and leaves.\n");
49      fprintf(f,"\n");
50      fprintf(f,"%s comes with 3 friend tools:\n",bname);
51      fprintf(f,"    img:\n");
```

```
52     fprintf(f,"         Manipulates images. For help type:\n");
53     fprintf(f,"         %s/bin/img --help\n",CFG_DEFAULT_PREFIX);
54     fprintf(f,"    digits:\n");
55     fprintf(f,"         Recognize digits. For help type:\n");
56     fprintf(f,"         %s/bin/digits --help\n",CFG_DEFAULT_PREFIX);
57     fprintf(f,"    grid:\n");
58     fprintf(f,"         Recognize a sudoku grid. For help type:\n");
59     fprintf(f,"         %s/bin/grid --help\n",CFG_DEFAULT_PREFIX);
60     fprintf(f,"\n");
61     fprintf(f,"%s was compiled on %s. Git repo used:\n",bname,__DATE__);
62     fprintf(f,"  %s\n",CFG_GIT_REPO);
63     fprintf(f,"with hash:\n");
64     fprintf(f,"  %s\n",CFG_GIT_FHASH);
65 }
```

4.27 util.c File Reference

```
#include "util.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <time.h>
```

Functions

- void seedRandomNumberGeneratorIfNeeded ()

  seed random number generator if not alread done.
- void writeRandomName (char *name, int n)

  writes a random name of size n.
- void createTmpDir (char *name, int n)

  creates a random directory
- char * stringCopy (char *s)

  allocates a new string.
- char * stringAdd (char *a, char *b)

  allocates a new string which is a concatenated with b.

Variables

- int _seedRandomNumberGenerator =0

4.27.1 Function Documentation

4.27.1.1 createTmpDir()

```
void createTmpDir (
            char * name,
            int n )
```

creates a random directory

Parameters

name	an allocated buffer of size n+1.
n	size of the created dir

```
46                                      {
47      if (n<10)
48          ERROR("n should be at least 10","");
49      name[0]='/';
50      name[1]='t';
51      name[2]='m';
52      name[3]='p';
53      name[4]='/';
54      writeRandomName(&name[5],n);
55      struct stat st = {0};
56      if (stat(name, &st) == -1) {
57          mkdir(name, 0700);
58      } else {
59          ERROR("Unlucky name already exists: ",name);
60      }
61  }
```

References ERROR, and writeRandomName().

4.27.1.2 seedRandomNumberGeneratorIfNeeded()

```
void seedRandomNumberGeneratorIfNeeded ( )
```

seed random number generator if not alread done.

```
13                                              {
14      if (!_seedRandomNumberGenerator) {
15          srand ( time(NULL) );
16          _seedRandomNumberGenerator=1;
17      }
18  }
```

References _seedRandomNumberGenerator.

4.27.1.3 stringAdd()

```
char * stringAdd (
            char * a,
            char * b )
```

allocates a new string which is a concatenated with b.

Parameters

a	a string
b	a string

Returns

concatenation of a and b

```
80                                      {
```

```
81     int l = strlen(a)+strlen(b)+1;
82     char * answer = (char*)malloc(l);
83     snprintf(answer,l,"%s%s",a,b);
84     return answer;
85 }
```

4.27.1.4 stringCopy()

```
char * stringCopy (
            char * s )
```

allocates a new string.

Parameters

s	string to copy.

```
67                              {
68     int l = strlen(s)+1;
69     char * answer = (char*)malloc(l);
70     memcpy(answer,s,l);
71     return answer;
72 }
```

4.27.1.5 writeRandomName()

```
void writeRandomName (
            char * name,
            int n )
```

writes a random name of size n.

Parameters

name	an allocated buffer of size n+1.
n	size of the random name.

```
25                                      {
26     int i=0;
27     seedRandomNumberGeneratorIfNeeded();
28     for (i=0;i<n;++i) {
29         unsigned char j=rand()%52;
30         name[i]=(j<26)?'A'+j:'a'+(j-26);
31     }
32     name[n]=0;
33     if (n>4) {
34         name[0]='c';
35         name[1]='n';
36         name[2]='n';
37         name[3]='_';
38     }
39 }
```

References seedRandomNumberGeneratorIfNeeded().

4.27.2 Variable Documentation

4.27.2.1 _seedRandomNumberGenerator

```
int _seedRandomNumberGenerator =0
```

4.28 util.h File Reference

```
#include <libgen.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "config.h"
```

Macros

- #define INBYTE(x) ((x)<0?0:(x)>255?255:(x))
- #define ERROR(x, y)
- #define WARNING(x, y)
- #define HERE(x)
- #define HERED(x)

Functions

- void writeRandomName (char *, int)

  writes a random name of size n.
- void createTmpDir (char *name, int n)

  creates a random directory
- char * stringCopy (char *)

  allocates a new string.
- char * stringAdd (char *, char *)

  allocates a new string which is a concatenated with b.

4.28.1 Macro Definition Documentation

4.28.1.1 ERROR

```
#define ERROR(
              x,
              y )
```

Value:

```
{fprintf(stderr,"\033[31m%s:%d: %s%s\033[m\n",       \
__FILE__,                                            \
__LINE__,                                            \
x,y); exit(1);}
```

4.28.1.2 HERE

```
#define HERE(
             x )
```

Value:

```
{printf("%s:%d: %s\n",                          \
__FILE__,                                 \
__LINE__,                                 \
x);}
```

4.28.1.3 HERED

```
#define HERED(
             x )
```

Value:

```
{printf("%s:%d: %d\n",                          \
__FILE__,                                 \
__LINE__,                                 \
x);}
```

4.28.1.4 INBYTE

```
#define INBYTE(
             x ) ((x)<0?0:(x)>255?255:(x))
```

4.28.1.5 WARNING

```
#define WARNING(
             x,
             y )
```

Value:

```
{fprintf(stderr,"\033[31m%s:%d: %s%s\033[m\n",     \
__FILE__,                                  \
__LINE__,                                  \
x,y);}
```

4.28.2 Function Documentation

4.28.2.1 createTmpDir()

```
void createTmpDir (
            char * name,
            int n )
```

creates a random directory

Parameters

name	an allocated buffer of size n+1.
n	size of the created dir

```
46                                      {
47     if (n<10)
48         ERROR("n should be at least 10","");
49     name[0]='/';
50     name[1]='t';
51     name[2]='m';
52     name[3]='p';
53     name[4]='/';
54     writeRandomName(&name[5],n);
55     struct stat st = {0};
56     if (stat(name, &st) == -1) {
57         mkdir(name, 0700);
58     } else {
59         ERROR("Unlucky name already exists: ",name);
60     }
61 }
```

References ERROR, and writeRandomName().

4.28.2.2 stringAdd()

```
char * stringAdd (
              char * a,
              char * b )
```

allocates a new string which is a concatenated with b.

Parameters

a	a string
b	a string

Returns

concatenation of a and b

```
80                                      {
81     int l = strlen(a)+strlen(b)+1;
82     char * answer = (char*)malloc(l);
83     snprintf(answer,l,"%s%s",a,b);
84     return answer;
85 }
```

4.28.2.3 stringCopy()

```
char * stringCopy (
              char * s )
```

allocates a new string.

Parameters

s	string to copy.

```
67                              {
68     int l = strlen(s)+1;
69     char * answer = (char*)malloc(l);
70     memcpy(answer,s,l);
71     return answer;
72 }
```

4.28.2.4 writeRandomName()

```
void writeRandomName (
             char * name,
             int n )
```

writes a random name of size n.

Parameters

name	an allocated buffer of size n+1.
n	size of the random name.

```
25                                      {
26     int i=0;
27     seedRandomNumberGeneratorIfNeeded();
28     for (i=0;i<n;++i) {
29         unsigned char j=rand()%52;
30         name[i]=(j<26)?'A'+j:'a'+(j-26);
31     }
32     name[n]=0;
33     if (n>4) {
34         name[0]='c';
35         name[1]='n';
36         name[2]='n';
37         name[3]='_';
38     }
39 }
```

References seedRandomNumberGeneratorIfNeeded().

4.29 util.h

Go to the documentation of this file.

```
1 #ifndef UTIL_H
2 #define UTIL_H
3
4 #include <libgen.h>
5 #include <stdlib.h>
6 #include <stdio.h>
7 #include <string.h>
8 #include "config.h"
9
10 #define INBYTE(x) ((x)<0?0:(x)>255?255:(x))
11
12 #define ERROR(x,y) {fprintf(stderr,"\033[31m%s:%d: %s%s\033[m\n",       \
13                             __FILE__,                                   \
14                             __LINE__,                                   \
15                             x,y); exit(1);}
16 #define WARNING(x,y) {fprintf(stderr,"\033[31m%s:%d: %s%s\033[m\n",     \
17                             __FILE__,                                   \
18                             __LINE__,                                   \
19                             x,y);}
20 #define HERE(x) {printf("%s:%d: %s\n",                                  \
```

```
21                          __FILE__,                                   \
22                          __LINE__,                                   \
23                          x);}
24 #define HERED(x) {printf("%s:%d: %d\n",                               \
25                           __FILE__,                                   \
26                           __LINE__,                                   \
27                           x);}
28 void writeRandomName(char*,int);
29 void createTmpDir(char*name,int n);
30 char * stringCopy(char *);
31 char * stringAdd(char *,char *);
32 #endif
```

Index