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

```
                               {
    free(d->layer1Prefix);
    free(d->layer2Prefix);
    if (d->layer1)
        deleteLayer(d->layer1);
    if (d->layer2)
        deleteLayer(d->layer2);
    memset(d,0,sizeof(Digits));
    free(d);
}
```

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.

```
                                                                  {
    char * tmpFile = (char*)malloc(strlen(directory)+40);
    snprintf(tmpFile,strlen(directory)+40,
             "%s/generate_png_from_font.sh",directory);
    FILE * f = fopen(tmpFile,"w");
    HERE(tmpFile);
    fprintf(f,"#/bin/sh\n");
    fprintf(f,"# this file has been automatically generated\n");
    fprintf(f,"# from c code %s:%d.\n",__FILE__,__LINE__);
    fprintf(f,"set -e\n");
    fprintf(f,"cd `dirname $0`\n");
    fprintf(f,"fonts=(`convert -list font  | grep Font: | grep -vi emoji| sed -e s/Font:// | tr -d ' '`)\n");
    fprintf(f,"fmax=${#fonts[@]}\n");
    fprintf(f,"fmax=$[fmax-1]\n");
    fprintf(f,"if [ $fmax -gt %d ]; then\n",numFonts);
    fprintf(f,"    fmax=%d\n",numFonts);
    fprintf(f,"fi\n");
    fprintf(f,"for f in `seq 0 $fmax`; do\n");
    fprintf(f,"    for i in `seq 1 9`; do\n");
    fprintf(f,"        convert -background white -fill black -font ${fonts[$f]} -size 28x28 -gravity center \"caption:$i\" $[f+1]_$i.png\n");
    fprintf(f,"    done\n");
    fprintf(f,"done\n");
    fclose(f);
    char cmd[99];
    snprintf(cmd,99,"bash %s",tmpFile);
    HERE(cmd);
    if (system(cmd)) {
        ERROR("Command failed, maybe 'convert' not found: ",cmd);
    }
}
```

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

```
                                               {
    // create 6 images 15x15 images with a rotating bar
    Img * allBars[6];
    allBars[0]=newImgVerticalBar(15,3);
    for (int i=1;i<6;++i)
        allBars[i]=imgRotate(allBars[0],30*i);
    // make each image 5x5
    for (int i=0;i<6;++i) {
        Img * tmp = allBars[i];
        allBars[i]=imgDownSampleAvg(tmp,3,3);
        deleteImg(tmp);
    }
    // creates corresponding filter familly
    FilterFam * filterFamilly = newFilterFam(6);
    for (int i=0;i<6;++i) {
        filterFamSetFilter(filterFamilly,
                           i,
                           newFilter(imgInvert(allBars[i]),80));
        // make sure they all have the same weight
        filterSetWeight(filterFamilly->filters[i],1300);
        HERE("filter weight");
        HERED((int)filterFamilly->filters[i]->weight);
        deleteImg(allBars[i]);
    }
    // save the familly on the disk
    char * s = stringAdd(layerPrefix,"_1");
    filterFamWrite(filterFamilly,s);
    // release memory
    free(s);
    deleteFilterFam(filterFamilly);
}
```

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

```
                                               {
    Digits * d = newDigits(layerPrefix);
    if (d->layer1==NULL) {
        ERROR("Generate layer 1 before generating layer 2.","");
    }
    for (int n=1;n<=9;++n) {
        char s[99];
        snprintf(s,99,"%s/digits/reference_%d.png",CFG_DATAROOTDIR,n);
        if (access(s, F_OK) != 0) {
            ERROR("File not found: ",s);
        }
        Img * tmpimg = newImgRead(s);
        Img * img = imgInvert(tmpimg);
        deleteImg(tmpimg);
        ImgFam * firstLayerOutput =
            layerPassImg(d->layer1,img);
        snprintf(s,99,"_layer_%d",n);
        imgFamWrite(firstLayerOutput,s);

        // free allocated memory
        deleteImg(img);
        deleteImgFam(firstLayerOutput);
    }
    deleteDigits(d);
}
```

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

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

```
}
```

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

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

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.

```
                                     {
    int i=1;
    char layerPrefix [digitsGetDefaultLayerPrefixSize()];
    digitsGetDefaultLayerPrefix(layerPrefix);
    while (i<argc) {
        if (strcmp(argv[i],"--create")==0 ||
            strcmp(argv[i],"-c")==0 ) {
            ++i;
            if (i>=argc) {
                digitsUsage(stderr,argv[0]);
                ERROR("-c|--create expects the number of layer to generate","");
            }
            int l = atoi(argv[i]);
            if (l<1 || l>3) {
                digitsUsage(stderr,argv[0]);
                ERROR("-c|--create 1, 2 or 3 as argument","");
            }
            if (l==1) {
                digitsGenerateLayer1(layerPrefix);
            } else if (l==2) {
                digitsGenerateLayer2(layerPrefix);
            } else {
                generateLayer3();
            }
        } else if (strcmp(argv[i],"--test")==0 ||
                   strcmp(argv[i],"-t")==0 ) {
            ++i;
            if (i>=argc) {
                digitsUsage(stderr,argv[0]);
                ERROR("-c|--create expects the number of layer to generate","");
            }
            int t=atoi(argv[i]);
            if (t<1 || t>3) {
                digitsUsage(stderr,argv[0]);
                ERROR("-t|--test 1, 2 or 3 as argument","");
            }
            if (t==1) {
                digitsTestLayer1(layerPrefix);
            } else if (t==2) {

            } else if (t==3) {
               generateTestData();
            }
        } else if (strcmp(argv[i],"--help")==0 ||
                   strcmp(argv[i],"-h")==0 ) {
            digitsUsage(stdout,argv[0]);
            exit(0);
        } else {
            digitsUsage(stderr,argv[0]);
            ERROR("unknown option: ",argv[i]);
        }
        ++i;
    }
    return 0;
}
```

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

4.1.1.8 digitsTestLayer1()

```
void digitsTestLayer1 (
               char * layerPrefix )
```

```
                                          {
    // create 6 images 30x30 images with a rotating bar
    Img * testImg[6];
    Img * base = newImgVerticalBar(30,2);
    for (int i=0;i<6;++i) {
        Img * tmp = imgRotate(base,30*i);
        testImg[i]=imgInvert(tmp);
        deleteImg(tmp);
        char s[99];
        snprintf(s,99,"test_digits_layer_1_%d.png",i);
        imgWrite(testImg[i],s);
    }
    deleteImg(base);

    // read existing filters
    Digits * d = newDigits(layerPrefix);
    if (d->layer1==NULL) {
        ERROR("Layer 1 filters not found, did you generate them?","");
    }

    for (int i=0;i<6;++i) {
        // creates a familly of 6 26x26 images
        ImgFam * firstLayerOutput = layerPassImg(d->layer1,testImg[i]);
        // verify that the maximum output for a bar with
        // and angle i*30 is of the i-th image.
        int maxIndex=-1;
        int maxVal=-1;
        for (int j=0;j<6;++j) {
            int v=imgGetWeight(firstLayerOutput->imgs[j]);
            if (v>maxVal) {
                maxVal=v;
                maxIndex=j;
            }
        }
        if (maxIndex!=i) {
            ERROR("Wrong index.","");
        }
        char s[99];
        snprintf(s,99,"test_digits_l1out_%d",i);
        imgFamWrite(firstLayerOutput,s);
    }


    // create 6 images 30x30 images with a rotating bar
    // of 30*i+15 degrees
    Img * testImgPlus15[6];
    base = newImgVerticalBar(30,3);
    for (int i=0;i<6;++i) {
        Img * tmp = imgRotate(base,15+30*i);
        testImgPlus15[i]=imgInvert(tmp);
        deleteImg(tmp);
        char s[99];
        snprintf(s,99,"test_digits_layer_1_15_%d.png",i);
        imgWrite(testImgPlus15[i],s);
    }
    deleteImg(base);

    // pass the 30x30 images in the convolution filter
    for (int i=0;i<6;++i) {
        // creates a familly of 6 26x26 images
        ImgFam * firstLayerOutput =
            layerPassImg(d->layer1,testImgPlus15[i]);
        char s[99];
        snprintf(s,99,"test_digits_l1out_15_%d",i);
        imgFamWrite(firstLayerOutput,s);


        printf("%d degrees\n",30*i+15);
        int v[6];
        for (int j=0;j<6;++j) {
            v[j]=imgGetWeight(firstLayerOutput->imgs[j]);
            printf("%4d ",v[j]);
        }
        printf("\n");
    }
}
```

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].

```
                                   {
    char * bname=basename(name);
    fprintf(f,"%s usage:\n", bname);
    fprintf(f,"    %s [ options ]\n", bname);
    fprintf(f,"Where option is one of:\n");
    fprintf(f,"    [-c|--create] <n>:\n");
    fprintf(f,"         creates the filters to detect digits for layer <n> \n");
    fprintf(f,"         in directory:\n");
    fprintf(f,"             %s/digits\n",CFG_DATAROOTDIR);
    fprintf(f,"    [-t|--test] <n>:\n");
    fprintf(f,"         tests the filters for digits.\n");
    fprintf(f,"    [-h|--help] :\n");
    fprintf(f,"         display this help message and exits.\n");
}
```

4.1.1.10 generateLayer3()

```
void generateLayer3 ( )
                      {
    char tmpDirName[60];
    //memset(tmpDirName,0,60);
    createTmpDir(tmpDirName,15);
    int numFonts=30;
    digitsGenerateImageOfFont(tmpDirName,numFonts);
    FILE * layer3File = fopen("l3.c","w");
    for (int f=1;f<=numFonts;++f) {
        char font[80];
        snprintf(font,80,"%s/%d",tmpDirName,f);

        char s[99];
        snprintf(s,99,"%s/digits/layer_1",CFG_DATAROOTDIR);
        FilterFam * firstLevelFilters=filterFamRead(s);

        FilterFam * secondLevelFilters[10];
        for (int i=1;i<10;++i) {
            char baseName[99];
            snprintf(baseName,99,"%s/digits/layer_2_Digit%d",CFG_DATAROOTDIR,i);
            secondLevelFilters[i] = filterFamRead(baseName);
        }
        HERE("+++");
        HERE(font);
        unsigned char ** outputFromLevel2 = (unsigned char**)malloc(sizeof(char*)*11);

        for (int digit=0;digit<10;++digit) {
            outputFromLevel2[digit]=(unsigned char*)malloc(sizeof(char)*11);
        }
        for (int digit=1;digit<10;++digit) {
            char s[99];
            snprintf(s,99,"%s_%d.png",font,digit);
            Img * tmpimg = newImgRead(s);
            Img * img = imgInvert(tmpimg);
            deleteImg(tmpimg);

            ImgFam * firstLayerOutput =
                filterFamApplyConvolution(firstLevelFilters,img);

            ImgFam * firstLayerMaxPoolOutput=imgFamDownSampleMax(firstLayerOutput,2,2);
            //ImgFam * avgPool=imgFamDownSampleMax(firstLayerMaxPoolOutput,2,2);
            ImgFam * avgPool = firstLayerMaxPoolOutput;
            HERE("___");
            int maxIndex=-1;
            int maxVal=-1;
            for (int i=1;i<10;++i) {
                Img*singlePixelImg=imgFamScalar(avgPool,secondLevelFilters[i]);

                //HERED(singlePixelImg->data[0]);
                outputFromLevel2[digit][i]=singlePixelImg->data[0];
                printf(" %d\n",singlePixelImg->data[0]);
                if (singlePixelImg->data[0]>maxVal) {
                    maxVal=singlePixelImg->data[0];
                    maxIndex=i;
                }
            }
            if (maxIndex!=digit) {
                fprintf(stderr,
                        "\033[31m%s:%d: ||| wrong index %d should be %d\033[m\n",
                        __FILE__,
                        __LINE__,
                        maxIndex,
                        digit);
            } else {
                HERE("OK");
            }
        }
        for (int digit=1;digit<10;++digit) {
            fprintf(layer3File,"unsigned char tmp_arr_%d_%d[10]={0,",f,digit);
            for (int i=1;i<10;++i) {
                fprintf(layer3File,"%d%s",outputFromLevel2[digit][i],i==9?"":",");
            }
            fprintf(layer3File,"};\n");
        }
    }

    fprintf(layer3File,"unsigned char **digits[%d];\n",numFonts+1);
    fprintf(layer3File,"unsigned char** init_digits() {\n");
    fprintf(layer3File,"    unsigned char** answer = (unsigned char**) malloc(sizeof(char**)*%d*10);\n",numFonts+1);
    for (int f=1;f<=numFonts;++f) {
        fprintf(layer3File,
                "    digits[%d]=answer+10*%d;\n",f,f);
        fprintf(layer3File,
                "    digits[%d][0]=NULL;\n",f);
        for (int digit=1;digit<10;++digit) {
            fprintf(layer3File,
                    // digits[a][b][c] char
                    // digits[a][b]    char*
                    // digits[a]       char**
                    // digits          char***
                    // answer[0]=
                    "    digits[%d][%d]=tmp_arr_%d_%d;\n",
                    f,digit,
                    f,digit);
        }
    }
    fprintf(layer3File,"    return answer;\n");
    fprintf(layer3File,"}\n");
    fclose(layer3File);
    HERE("l3.c written.");
}
```

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

4.1.1.11 generateTestData()

```
void generateTestData ( )
                        {
    char tmpDirName[60];
    //memset(tmpDirName,0,60);
    createTmpDir(tmpDirName,15);
    int numFonts=30;
    digitsGenerateImageOfFont(tmpDirName,numFonts);
    for (int i=1;i<=numFonts;++i) {
        char fontpath[99];
        snprintf(fontpath,99,"%s/%d",tmpDirName,i);
        testFilterForDigit(fontpath);
    }
}
```

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

```
                                        {
    Digits * answer = (Digits*)malloc(sizeof(Digits));
    char layer1location[strlen(layerPrefix)+10];
    snprintf(layer1location,strlen(layerPrefix)+10,"%s_1",layerPrefix);
    answer->layer1Prefix=stringCopy(layer1location);
    if (layerCount(layer1location)>0) {
        answer->layer1 = newLayer("layer1",layer1location,2,2);
    } else {
        answer->layer1=NULL;
    }
    char layer2location[strlen(layerPrefix)+10];
    snprintf(layer2location,strlen(layerPrefix)+10,"%s_2",layerPrefix);
    answer->layer2Prefix=stringCopy(layer2location);
    if (layerCount(layer2location)>0) {
        answer->layer2 = newLayer("layer2",layer2location,2,2);
    } else {
        answer->layer2 = NULL;
    }
    return answer;
}
```

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

4.1.1.13 testFilterForDigit()

```
void testFilterForDigit (
               char * font )
```

```
                                     {

    char s[99];
    snprintf(s,99,"%s/digits/layer_1",CFG_DATAROOTDIR);
    FilterFam * firstLevelFilters=filterFamRead(s);

    FilterFam * secondLevelFilters[10];
    for (int i=1;i<10;++i) {
        char baseName[99];
        snprintf(baseName,99,"%s/digits/layer_2_Digit%d",CFG_DATAROOTDIR,i);
        secondLevelFilters[i] = filterFamRead(baseName);
    }
    HERE("+++");
    HERE(font);
    for (int digit=1;digit<10;++digit) {
        char s[99];
        snprintf(s,99,"%s_%d.png",font,digit);
        Img * tmpimg = newImgRead(s);
        Img * img = imgInvert(tmpimg);
        deleteImg(tmpimg);

        ImgFam * firstLayerOutput =
            filterFamApplyConvolution(firstLevelFilters,img);

        ImgFam * firstLayerMaxPoolOutput=imgFamDownSampleMax(firstLayerOutput,2,2);
        //ImgFam * avgPool=imgFamDownSampleMax(firstLayerMaxPoolOutput,2,2);
        ImgFam * avgPool=firstLayerMaxPoolOutput;
        HERE("___");
        int maxIndex=-1;
        int maxVal=-1;
        for (int i=1;i<10;++i) {
            Img*singlePixelImg=imgFamScalar(avgPool,secondLevelFilters[i]);
            //HERED(singlePixelImg->data[0]);
            printf(" %d\n",singlePixelImg->data[0]);
            if (singlePixelImg->data[0]>maxVal) {
                maxVal=singlePixelImg->data[0];
                maxIndex=i;
            }
        }
        if (maxIndex!=digit) {
            fprintf(stderr,
                    "\033[31m%s:%d: ||| wrong index %d should be %d\033[m\n",
                    __FILE__,
                    __LINE__,
                    maxIndex,
                    digit);
        } else {
            HERE("OK");
        }
    }
}
```

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

```
                          {
    free(d->layer1Prefix);
    free(d->layer2Prefix);
    if (d->layer1)
        deleteLayer(d->layer1);
    if (d->layer2)
        deleteLayer(d->layer2);
    memset(d,0,sizeof(Digits));
    free(d);
}
```

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.

```
                                                            {
    char * tmpFile = (char*)malloc(strlen(directory)+40);
    snprintf(tmpFile,strlen(directory)+40,
             "%s/generate_png_from_font.sh",directory);
    FILE * f = fopen(tmpFile,"w");
    HERE(tmpFile);
    fprintf(f,"#/bin/sh\n");
    fprintf(f,"# this file has been automatically generated\n");
    fprintf(f,"# from c code %s:%d.\n",__FILE__,__LINE__);
    fprintf(f,"set -e\n");
    fprintf(f,"cd ‘dirname $0‘\n");
    fprintf(f,"fonts=(‘convert -list font  | grep Font: | grep -vi emoji| sed -e s/Font:// | tr -d ’
   ’‘)\n");
    fprintf(f,"fmax=${#fonts[@]}\n");
    fprintf(f,"fmax=$[fmax-1]\n");
    fprintf(f,"if [ $fmax -gt %d ]; then\n",numFonts);
    fprintf(f,"    fmax=%d\n",numFonts);
    fprintf(f,"fi\n");
    fprintf(f,"for f in ‘seq 0 $fmax‘; do\n");
    fprintf(f,"    for i in ‘seq 1 9‘; do\n");
    fprintf(f,"        convert -background white -fill black -font ${fonts[$f]} -size 28x28 -gravity
   center \"caption:$i\" $[f+1]_$i.png\n");
    fprintf(f,"    done\n");
    fprintf(f,"done\n");
    fclose(f);
    char cmd[99];
    snprintf(cmd,99,"bash %s",tmpFile);
    HERE(cmd);
    if (system(cmd)) {
        ERROR("Command failed, maybe ’convert’ not found: ",cmd);
    }
}
```

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

```
                                               {
    // create 6 images 15x15 images with a rotating bar
    Img * allBars[6];
    allBars[0]=newImgVerticalBar(15,3);
    for (int i=1;i<6;++i)
        allBars[i]=imgRotate(allBars[0],30*i);
    // make each image 5x5
    for (int i=0;i<6;++i) {
        Img * tmp = allBars[i];
        allBars[i]=imgDownSampleAvg(tmp,3,3);
        deleteImg(tmp);
    }
    // creates corresponding filter familly
    FilterFam * filterFamilly = newFilterFam(6);
    for (int i=0;i<6;++i) {
        filterFamSetFilter(filterFamilly,
                           i,
                            newFilter(imgInvert(allBars[i]),80));
         // make sure they all have the same weight
         filterSetWeight(filterFamilly->filters[i],1300);
        HERE("filter weight");
        HERED((int)filterFamilly->filters[i]->weight);
        deleteImg(allBars[i]);
    }
    // save the familly on the disk
    char * s = stringAdd(layerPrefix,"_1");
    filterFamWrite(filterFamilly,s);
    // release memory
    free(s);
    deleteFilterFam(filterFamilly);
}
```

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

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

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

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

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.

```
                                    {
    int i=1;
    char layerPrefix [digitsGetDefaultLayerPrefixSize()];
    digitsGetDefaultLayerPrefix(layerPrefix);
    while (i<argc) {
        if (strcmp(argv[i],"--create")==0 ||
            strcmp(argv[i],"-c")==0 ) {
            ++i;
            if (i>=argc) {
                digitsUsage(stderr,argv[0]);
                ERROR("-c|--create expects the number of layer to generate","");
            }
            int l = atoi(argv[i]);
            if (l<1 || l>3) {
                digitsUsage(stderr,argv[0]);
                ERROR("-c|--create 1, 2 or 3 as argument","");
            }
            if (l==1) {
                digitsGenerateLayer1(layerPrefix);
            } else if (l==2) {
                digitsGenerateLayer2(layerPrefix);
            } else {
                generateLayer3();
            }
        } else if (strcmp(argv[i],"--test")==0 ||
                   strcmp(argv[i],"-t")==0 ) {
            ++i;
            if (i>=argc) {
                digitsUsage(stderr,argv[0]);
                ERROR("-c|--create expects the number of layer to generate","");
            }
            int t=atoi(argv[i]);
            if (t<1 || t>3) {
                digitsUsage(stderr,argv[0]);
                ERROR("-t|--test 1, 2 or 3 as argument","");
            }
            if (t==1) {
                digitsTestLayer1(layerPrefix);
            } else if (t==2) {

            } else if (t==3) {
               generateTestData();
            }
        } else if (strcmp(argv[i],"--help")==0 ||
                   strcmp(argv[i],"-h")==0 ) {
            digitsUsage(stdout,argv[0]);
            exit(0);
        } else {
            digitsUsage(stderr,argv[0]);
            ERROR("unknown option: ",argv[i]);
        }
        ++i;
    }
    return 0;
}
```

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

4.2.2.7 digitsTestLayer1()

```
void digitsTestLayer1 (
            char * layerPrefix )
```

```
                                    {
    // create 6 images 30x30 images with a rotating bar
    Img * testImg[6];
    Img * base = newImgVerticalBar(30,2);
    for (int i=0;i<6;++i) {
        Img * tmp = imgRotate(base,30*i);
        testImg[i]=imgInvert(tmp);
        deleteImg(tmp);
        char s[99];
        snprintf(s,99,"test_digits_layer_1_%d.png",i);
        imgWrite(testImg[i],s);
    }
    deleteImg(base);

    // read existing filters
    Digits * d = newDigits(layerPrefix);
    if (d->layer1==NULL) {
        ERROR("Layer 1 filters not found, did you generate them?","");
    }

    for (int i=0;i<6;++i) {
        // creates a familly of 6 26x26 images
        ImgFam * firstLayerOutput = layerPassImg(d->layer1,testImg[i]);
        // verify that the maximum output for a bar with
        // and angle i*30 is of the i-th image.
        int maxIndex=-1;
        int maxVal=-1;
        for (int j=0;j<6;++j) {
            int v=imgGetWeight(firstLayerOutput->imgs[j]);
            if (v>maxVal) {
                maxVal=v;
                maxIndex=j;
            }
        }
        if (maxIndex!=i) {
            ERROR("Wrong index.","");
        }
        char s[99];
        snprintf(s,99,"test_digits_l1out_%d",i);
        imgFamWrite(firstLayerOutput,s);
    }


    // create 6 images 30x30 images with a rotating bar
    // of 30*i+15 degrees
    Img * testImgPlus15[6];
    base = newImgVerticalBar(30,3);
    for (int i=0;i<6;++i) {
        Img * tmp = imgRotate(base,15+30*i);
        testImgPlus15[i]=imgInvert(tmp);
        deleteImg(tmp);
        char s[99];
        snprintf(s,99,"test_digits_layer_1_15_%d.png",i);
        imgWrite(testImgPlus15[i],s);
    }
    deleteImg(base);

    // pass the 30x30 images in the convolution filter
    for (int i=0;i<6;++i) {
        // creates a familly of 6 26x26 images
        ImgFam * firstLayerOutput =
            layerPassImg(d->layer1,testImgPlus15[i]);
        char s[99];
        snprintf(s,99,"test_digits_l1out_15_%d",i);
        imgFamWrite(firstLayerOutput,s);


        printf("%d degrees\n",30*i+15);
        int v[6];
        for (int j=0;j<6;++j) {
            v[j]=imgGetWeight(firstLayerOutput->imgs[j]);
            printf("%4d ",v[j]);
        }
        printf("\n");
    }
}
```

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

```
                                       {
    Digits * answer = (Digits*)malloc(sizeof(Digits));
    char layer1location[strlen(layerPrefix)+10];
    snprintf(layer1location,strlen(layerPrefix)+10,"%s_1",layerPrefix);
    answer->layer1Prefix=stringCopy(layer1location);
    if (layerCount(layer1location)>0) {
        answer->layer1 = newLayer("layer1",layer1location,2,2);
    } else {
        answer->layer1=NULL;
    }
    char layer2location[strlen(layerPrefix)+10];
    snprintf(layer2location,strlen(layerPrefix)+10,"%s_2",layerPrefix);
    answer->layer2Prefix=stringCopy(layer2location);
    if (layerCount(layer2location)>0) {
        answer->layer2 = newLayer("layer2",layer2location,2,2);
    } else {
        answer->layer2 = NULL;
    }
    return answer;
}
```

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

4.3 digits.h

Go to the documentation of this file.

```
#ifndef DIGITS_H
#define DIGITS_H

#include "util.h"

// external types
typedef struct layer Layer;

struct digits {
    char * layer1Prefix;
    char * layer2Prefix;
    Layer * layer1;
    Layer * layer2;
};
typedef struct digits Digits;

Digits * newDigits(char*);
void deleteDigits(Digits*);
int digitsGetDefaultLayerPrefixSize();
void digitsGetDefaultLayerPrefix(char* layerPrefix);
void digitsGenerateImageOfFont(char*directory,int numFonts);
void digitsGenerateLayer1 (char * layerPrefix);
void digitsTestLayer1(char * layerPrefix);
int digitsMain(int argc,char**argv);

#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.

```
                              {
    if (f==NULL) return;
    if (f->img)
        deleteImg(f->img);
    f->weight=0;
    f->threshold=0;
    f->maxVal=0;
    if (f->data!=NULL)
        free(f->data);
    memset(f,0,sizeof(Filter));
    free(f);
}
```

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

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

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

```
                                   {
      long int ww=0;
      for(int i = 0; i < f->img->height*f->img->width; i++) {
          ww+=f->img->data[i];
      }
      float ratio = (w+0.)/(ww+0.);
      for(int i = 0; i < f->img->height*f->img->width; i++) {
          f->img->data[i]=INBYTE((int)round(ratio*f->img->data[i]));
      }
      filterUpdateValues(f);
  }
```

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

```
                                  {
    f->weight=0;
    f->threshold=0;
    f->maxVal=f->img->height*f->img->width*255;
    if (f->percent>100 || f->percent<-100) {
        ERROR("percent should be between 0 and 100.","");
    }
    for(int i = 0; i < f->img->height*f->img->width; i++) {
        f->weight+=f->img->data[i]-128;
    }
    f->threshold=f->maxVal*f->percent/100;
}
```

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

```
                                         {
    char s[99];
    // save the picture part
    snprintf(s,99,"%s.png",basename);
    imgWrite(f->img,s);
    // save the data part
    snprintf(s,99,"%s.txt",basename);
    FILE * fi = fopen(s,"w");
    if (fi==NULL)
        ERROR("Could not open file ",s);
    fprintf(fi,"%d\n",f->percent);
    fprintf(fi,"%s\n",f->data);
    fclose(fi);
}
```

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.

```
                                           {
    Filter * answer = (Filter*) malloc(sizeof(struct filter));
    answer->img=img;
    answer->threshold=0;
    answer->percent=percent;
    answer->weight=0;
    answer->maxVal=0;
    answer->data=NULL;
    if (img==NULL) return answer;
    filterUpdateValues(answer);
    return answer;
}
```

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

```
                                           {
    Filter * answer = (Filter*) malloc(sizeof(struct filter));
    char s[99];
    // read the picture part
    snprintf(s,99,"%s.png",basename);
    answer->img=newImgRead(s);
    // read the data part
    snprintf(s,99,"%s.txt",basename);
    FILE * fi = fopen(s,"r");
    if (fi==NULL)
        ERROR("Could not open file ",s);
    fscanf(fi,"%d",&(answer->percent));

    char * line = NULL;
    size_t len = 0;
    if (getline(&line, &len, fi)==-1) {
        WARNING("Error while reading ",s);
    } else {
        answer->data=stringCopy(line);
    }
    if (line) free(line);
    fclose(fi);
    filterUpdateValues(answer);
    return answer;
}
```

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.

```
                             {
    if (f==NULL) return;
    if (f->img)
        deleteImg(f->img);
    f->weight=0;
    f->threshold=0;
    f->maxVal=0;
    if (f->data!=NULL)
        free(f->data);
    memset(f,0,sizeof(Filter));
    free(f);
}
```

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

```
                                      {
    long int ww=0;
    for(int i = 0; i < f->img->height*f->img->width; i++) {
        ww+=f->img->data[i];
    }
    float ratio = (w+0.)/(ww+0.);
    for(int i = 0; i < f->img->height*f->img->width; i++) {
        f->img->data[i]=INBYTE((int)round(ratio*f->img->data[i]));
    }
    filterUpdateValues(f);
}
```

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

```
                                  {
    f->weight=0;
    f->threshold=0;
    f->maxVal=f->img->height*f->img->width*255;
    if (f->percent>100 || f->percent<-100) {
        ERROR("percent should be between 0 and 100.","");
    }
    for(int i = 0; i < f->img->height*f->img->width; i++) {
        f->weight+=f->img->data[i]-128;
    }
    f->threshold=f->maxVal*f->percent/100;
}
```

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

```
                                         {
    char s[99];
    // save the picture part
    snprintf(s,99,"%s.png",basename);
    imgWrite(f->img,s);
    // save the data part
    snprintf(s,99,"%s.txt",basename);
    FILE * fi = fopen(s,"w");
    if (fi==NULL)
        ERROR("Could not open file ",s);
    fprintf(fi,"%d\n",f->percent);
    fprintf(fi,"%s\n",f->data);
    fclose(fi);
}
```

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.

```
                                        {
    Filter * answer = (Filter*) malloc(sizeof(struct filter));
    answer->img=img;
    answer->threshold=0;
    answer->percent=percent;
    answer->weight=0;
    answer->maxVal=0;
    answer->data=NULL;
    if (img==NULL) return answer;
    filterUpdateValues(answer);
    return answer;
}
```

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

```
                                       {
    Filter * answer = (Filter*) malloc(sizeof(struct filter));
    char s[99];
    // read the picture part
    snprintf(s,99,"%s.png",basename);
    answer->img=newImgRead(s);
    // read the data part
    snprintf(s,99,"%s.txt",basename);
    FILE * fi = fopen(s,"r");
    if (fi==NULL)
        ERROR("Could not open file ",s);
    fscanf(fi,"%d",&(answer->percent));

    char * line = NULL;
    size_t len = 0;
    if (getline(&line, &len, fi)==-1) {
        WARNING("Error while reading ",s);
    } else {
        answer->data=stringCopy(line);
    }
    if (line) free(line);
    fclose(fi);
    filterUpdateValues(answer);
    return answer;
}
```

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

4.6 filter.h

Go to the documentation of this file.

```
#ifndef FILTER_H
#define FILTER_H

#include "img.h"

struct filter {
    Img *img;
    long int weight;
    long int threshold;
    long int maxVal;
    int percent;
    char * data;
};


typedef struct filter Filter;

Filter * newFilter(Img*,int);
Filter * newFilterRead(char *basename);
void deleteFilter(Filter*);
void filterSetWeight(Filter*f,int w);
void filterUpdateValues(Filter*);
void filterWrite(Filter*f,char*basename);
#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.

```
                                    {
      if (ff==NULL) return;
      for (int i=0;i<ff->count;++i) {
          deleteFilter(ff->filters[i]);
      }
      memset(ff->filters,0,sizeof(Filter*)*ff->count);
      free(ff->filters);
      memset(ff,0,sizeof(FilterFam));
      free(ff);
  }
```

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.

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

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.

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

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.

```
                                                                           {
    ImgFam * answer = newImgFam (filters->count*imgFam->count);
    for (int i=0;i<filters->count;++i) {
        for (int j=0;j<imgFam->count;++j) {
            int idx=j+i*imgFam->count;
            imgFamSetImg(answer,idx,imgConvolution(imgFam->imgs[j],
                                                   filters->filters[i]));
    }   }
    return answer;
}
```

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.

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

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.

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

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.

```
{
    ImgFam * answer = newImgFam (filters->count*imgFam->count);
    for (int i=0;i<filters->count;++i) {
        for (int j=0;j<imgFam->count;++j) {
            int idx=j+i*imgFam->count;
            imgFamSetImg(answer,idx,imgConvolution(imgFam->imgs[j],
                                                    filters->filters[i]));
    }   }
    return answer;
}
```

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

```
                                         {
    int answer =-1;
    char fname[strlen(convFilterLoc)+10];
    do {
        answer ++;
        snprintf(fname,strlen(convFilterLoc)+10,
                 "%s_%d.png",convFilterLoc,answer);
    } while (access(fname, F_OK) == 0);
    return answer;
}
```

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

```
                                         {
    char s[99];
    int count=0;
    int found=1;
    while (found) {
        snprintf(s,99,"%s_%d.png",basename,count);
        FILE * f = fopen(s,"r");
        if (f!=NULL) {
            fclose(f);
            ++count;
        } else  {
            found=0;
        }
    }
    FilterFam * answer = newFilterFam(count);
    for (int i=0;i<answer->count;++i) {
        snprintf(s,99,"%s_%d",basename,i);
        filterFamSetFilter(answer,i,newFilterRead(s));
    }
    return answer;
}
```

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.

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

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

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

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.

```
                                  {
      FilterFam * answer = (FilterFam*)malloc(sizeof(struct filterfam));
      answer->count=c;
      answer->filters=(Filter**)malloc(sizeof(Filter**)*c);
      memset(answer->filters,0,sizeof(Filter*)*c);
      return answer;
}
```

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.

```
                                      {
    if (ff==NULL) return;
    for (int i=0;i<ff->count;++i) {
        deleteFilter(ff->filters[i]);
    }
    memset(ff->filters,0,sizeof(Filter*)*ff->count);
    free(ff->filters);
    memset(ff,0,sizeof(FilterFam));
    free(ff);
}
```

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.

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

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.

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

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.

```
                                                                           {
    ImgFam * answer = newImgFam (filters->count*imgFam->count);
    for (int i=0;i<filters->count;++i) {
        for (int j=0;j<imgFam->count;++j) {
            int idx=j+i*imgFam->count;
            imgFamSetImg(answer,idx,imgConvolution(imgFam->imgs[j],
                                                   filters->filters[i]));
    }   }
    return answer;
}
```

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.

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

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.

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

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.

```
{
    ImgFam * answer = newImgFam (filters->count*imgFam->count);
    for (int i=0;i<filters->count;++i) {
        for (int j=0;j<imgFam->count;++j) {
            int idx=j+i*imgFam->count;
            imgFamSetImg(answer,idx,imgConvolution(imgFam->imgs[j],
                                                   filters->filters[i]));
    }   }
    return answer;
}
```

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

```
                                      {
    int answer =-1;
    char fname[strlen(convFilterLoc)+10];
    do {
        answer ++;
        snprintf(fname,strlen(convFilterLoc)+10,
                 "%s_%d.png",convFilterLoc,answer);
    } while (access(fname, F_OK) == 0);
    return answer;
}
```

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

```
                                      {
    char s[99];
    int count=0;
    int found=1;
    while (found) {
        snprintf(s,99,"%s_%d.png",basename,count);
        FILE * f = fopen(s,"r");
        if (f!=NULL) {
            fclose(f);
            ++count;
        } else  {
            found=0;
        }
    }
    FilterFam * answer = newFilterFam(count);
    for (int i=0;i<answer->count;++i) {
        snprintf(s,99,"%s_%d",basename,i);
        filterFamSetFilter(answer,i,newFilterRead(s));
    }
    return answer;
}
```

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.

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

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

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

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.

```
                                {
    FilterFam * answer = (FilterFam*)malloc(sizeof(struct filterfam));
    answer->count=c;
    answer->filters=(Filter**)malloc(sizeof(Filter**)*c);
    memset(answer->filters,0,sizeof(Filter*)*c);
    return answer;
}
```

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

4.9 filterfam.h

Go to the documentation of this file.

```
#ifndef FILTERFAM_H
#define FILTERFAM_H

#include "filter.h"


struct filterfam {
    int count;
    Filter ** filters;
};

typedef struct filterfam FilterFam;

// external types
typedef struct imgfam ImgFam;

FilterFam * newFilterFam(int);
ImgFam * filterFamApplyConvolution(FilterFam* filters,Img* img);
ImgFam * filterFamApplyConvolutionDiff(FilterFam* filters,Img* img);
ImgFam * filterFamApplyConvolutionOnFam(FilterFam* filters,ImgFam* imgFam);
ImgFam * filterFamApplyConvolutionSameSize(FilterFam* filters,Img* img);
ImgFam * filterFamApplyConvolutionSameSizeOnFam(FilterFam* filters,
                                                ImgFam* imgFam);
ImgFam * filterFamApplyConvolutionSameSizeDiff(FilterFam* filters,Img* img);
void filterFamSetFilter(FilterFam*filterFam,int i,Filter*filter);
void deleteFilterFam(FilterFam *);
FilterFam * filterFamRead(char*basename);
void filterFamWrite(FilterFam*filterFam,char*basename);
int filterFamCount(char * convFilterLoc);

#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.

```
// file automatically generated by gen_fontlist.sh.
#ifndef FONTNAME_H
#define FONTNAME_H
extern char * fontname[];
extern int fontcount;
#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;

```
{
    FilterFam * answer=newFilterFam(1);

    Img * filter = newImgVerticalBarInRect(5*t,l,t);
    Img * rotatedFilter = NULL;
    if (i==0)
        rotatedFilter=newImgCopy(filter);
    else
        rotatedFilter=imgRotate90(filter);
    Filter * invertedFilter = newFilter(imgInvert(rotatedFilter),p);
    if (gridDumpDebugInfo) {
        HERE("layer2 on filter");
        HERED((int)invertedFilter->weight);
        HERE("__________");
        HERED((int)invertedFilter->maxVal);
    }
    deleteImg(filter);
    deleteImg(rotatedFilter);
    if (gridDumpDebugInfo) {
        char filterName[99];
        snprintf(filterName,99,"%s/grid/grid_filter_%d",CFG_DATAROOTDIR,i);
        filterWrite(invertedFilter,filterName);
    }
    filterFamSetFilter(answer,0,invertedFilter);
    return answer;
}
```

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

```
{
    int n = img->width;
    if (img->height!=1) {
        ERROR("Expected a height of 1","");
    }
    gridIdentifyNPointsCounter++;
    Img * imgs[n];
    int convVal[n];
    int weight[n];
    int max=0;
    int maxIdx=0;
    int lowerBoundOffset[n];
    int upperBoundOffset[n];
    for (int i=n/2;i<n-1;++i) {
        Img * f1 = newImgNDotsHori(N,i);
        Filter * f =newFilter(imgInvert(f1),99);
        deleteImg(f1);
        if (gridDumpDebugInfo) {
            char s[99];
            snprintf(s,99,"conv_filter_%d_%d.png",N,i);
            imgWrite(f->img,s);
        }
        Img * c = imgConvolutionDiff(img,f);
        weight[i]=f->weight;
        deleteFilter(f);
        imgs[i]=c;
        if (gridDumpDebugInfo) {
            char s[99];
            snprintf(s,99,"conv_result_%d_%d.png",gridIdentifyNPointsCounter,i);
            imgWrite(c,s);
        }
        int localMax=-1;
        for (int j=0;j<c->width;++j) {
            if (c->data[j]>localMax) {
                localMax=c->data[j];
                lowerBoundOffset[i]=j+i/2/N;
                upperBoundOffset[i]=j+i-i/2/N;
            }
        }
        if (localMax>max) {
            max=localMax;maxIdx=i;
        }
        convVal[i]=localMax;
    }
    for (int i=n/2;i<n-1;++i) {
        if (max-max/20<convVal[i]) {
            if (gridDumpDebugInfo) {
                HERE("------------");
                HERED(convVal[i]);
                HERED(i);
                //imgPrint(imgs[i]);
            }
        }
        deleteImg(imgs[i]);
    }
    *maxCorrelation=convVal[maxIdx];
    *lowerBound=lowerBoundOffset[maxIdx];
    *upperBound=upperBoundOffset[maxIdx];
    return 0;
}
```

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 gridGetLayerHoriVertFilters. 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

```
{
    Img *layer1HO,*layer1VO;
    gridVertHoriConvo(&layer1HO,&layer1VO,
                      img,
                      img,
                      1,
                      50, /* length */
                      0,  /* threshold */
                      1,1);


    if (gridDumpDebugInfo) {
        imgWrite(layer1HO,"gridLayer1OutputVert.png");
        imgWrite(layer1VO,"gridLayer1OutputHori.png");
    }
    Img *layer5HO,*layer5VO;
    layer5HO=layer1HO;
    layer5VO=layer1VO;
    int sumMinVal=8*img->height*0;
    int sumMaxVal=64*img->height;
    Img * flattenedHori = newImgColor(layer5VO->width,1,0);
    for (int x=0;x<layer5VO->width;++x) {
        int s=0;
        for (int y=0;y<layer5VO->height;++y) {
            s+=layer5VO->data[x+layer5VO->width*y];
        }
        if (s<sumMinVal) s=0;
        else if (s>sumMaxVal) s=255;
        else s=255*(s-sumMinVal)/(sumMaxVal-sumMinVal);
        flattenedHori->data[x]=s;
        printf("%d ",s);
    }

    if (gridDumpDebugInfo) {
        imgWrite(flattenedHori,"flattenedHori.png");

    }
    printf("\n");
    sumMinVal=8*img->width;
    sumMaxVal=64*img->width;
    Img * flattenedVert = newImgColor(layer5VO->height,1,0);
    for (int y=0;y<layer5HO->height;++y) {
        int s=0;
        for (int x=0;x<layer5HO->width;++x) {
            s+=layer5HO->data[x+layer5VO->width*y];
        }
        if (s<sumMinVal) s=0;
        else if (s>sumMaxVal) s=255;
        else s=255*(s-sumMinVal)/(sumMaxVal-sumMinVal);
        flattenedVert->data[y]=s;
        printf("%d ",s);
    }

    if (gridDumpDebugInfo) {
        imgWrite(flattenedVert,"flattenedVert.png");
    }
    printf("\n");

    // the 10 equidistant point should spread on the minimum
    // between img->width and img->height and that value
    // divided by 2.
    int minLength = img->width;
    if (img->height<minLength) minLength=img->height;
    int endRange=minLength;
    int startRange=endRange/2;
    int maxCorrelationVer=0;
    gridIdentifyNPoints(10,flattenedVert,startRange, endRange,ymin,ymax,
                        &maxCorrelationVer);

    if (gridDumpDebugInfo) {
        HERE("+++++++++++++++");
    }
    int maxCorrelationHori=0;
    gridIdentifyNPoints(10,flattenedHori,startRange, endRange,xmin,xmax,
                        &maxCorrelationHori);
    HERE("maxCorrelationHori");
    HERED(maxCorrelationHori);
    HERE("maxCorrelationVer");
    HERED(maxCorrelationVer);

    deleteImg(flattenedVert);
    deleteImg(flattenedHori);

    deleteImg(layer1HO);
    deleteImg(layer1VO);
    return 0;
}
```

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.

```
                                  {
    for (int j=1;j<argc;++j) {
        if (strcmp(argv[j],"-h")==0 || strcmp(argv[j],"--help")==0) {
            gridUsage(stdout,argv[0]);
            exit(0);
        }
    }
    if(argc < 3) {
        gridUsage(stderr,argv[0]);
        ERROR("At least 2 argumens expected, the picture to read, and where tp write the output","");
    }
    /* read input image */
    Img * currentImage=newImgRead(argv[1]);
    int xmin,ymin,xmax,ymax;
    // locate the grid
    if (!gridLocate(currentImage,&xmin,&ymin,&xmax,&ymax)) {
        HERE("Found sudoku grid :");
        printf("%d %d %d %d\n",xmin,ymin,xmax,ymax);
        // draw the grid found
        imgDrawRect(currentImage,xmin,ymin,xmax,ymax);
        // write the image
        imgWrite(currentImage,argv[2]);
    } else {
        ERROR("Could not find a grid in picture: ",argv[1]);
    }
    return 0;
}
```

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].

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

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 deduce image (1 for no reduction)

```
{
    FilterFam * filtersVert=
        gridGetLayerHoriVertFilters(0,length,width,threshold);
    FilterFam * filtersHori=
        gridGetLayerHoriVertFilters(1,length,width,threshold);

    ImgFam * layerVertOutput =
        filterFamApplyConvolutionSameSizeDiff(filtersVert,inputV);
    ImgFam * layerHoriOutput =
        filterFamApplyConvolutionSameSizeDiff(filtersHori,inputH);
    deleteFilterFam(filtersVert);
    deleteFilterFam(filtersHori);

    ImgFam * layerMaxPoolVertOutput=
        imgFamDownSampleMax(layerVertOutput,poolsize,stride);
    ImgFam * layerMaxPoolHoriOutput=
        imgFamDownSampleMax(layerHoriOutput,poolsize,stride);
    deleteImgFam(layerVertOutput);
    deleteImgFam(layerHoriOutput);

    ImgFam * betterContrastHori2 =
        imgFamLuminosityScale(layerMaxPoolHoriOutput);
    ImgFam * betterContrastVert2 =
        imgFamLuminosityScale(layerMaxPoolVertOutput);
    deleteImgFam(layerMaxPoolVertOutput);
    deleteImgFam(layerMaxPoolHoriOutput);

    if (betterContrastHori2->count!=1 ||
         betterContrastVert2->count!=1) {
         ERROR("Wrong size.","");
     }
     // write address the filter to return to caller
     *outputV=betterContrastVert2->imgs[0];
     *outputH=betterContrastHori2->imgs[0];
     // pretend there is no filter in the filterFam
     betterContrastHori2->count=0;
     betterContrastVert2->count=0;
     // delete filterFam allocated data
     deleteImgFam(betterContrastHori2);
     deleteImgFam(betterContrastVert2);
}
```

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 gridGetLayerHoriVertFilters. 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

```
{
    Img *layer1HO,*layer1VO;
    gridVertHoriConvo(&layer1HO,&layer1VO,
                      img,
                      img,
                      1,
                      50, /* length */
                      0,  /* threshold */
                      1,1);


    if (gridDumpDebugInfo) {
        imgWrite(layer1HO,"gridLayer1OutputVert.png");
        imgWrite(layer1VO,"gridLayer1OutputHori.png");
    }
    Img *layer5HO,*layer5VO;
    layer5HO=layer1HO;
    layer5VO=layer1VO;
    int sumMinVal=8*img->height*0;
    int sumMaxVal=64*img->height;
    Img * flattenedHori = newImgColor(layer5VO->width,1,0);
    for (int x=0;x<layer5VO->width;++x) {
        int s=0;
        for (int y=0;y<layer5VO->height;++y) {
            s+=layer5VO->data[x+layer5VO->width*y];
        }
        if (s<sumMinVal) s=0;
        else if (s>sumMaxVal) s=255;
        else s=255*(s-sumMinVal)/(sumMaxVal-sumMinVal);
        flattenedHori->data[x]=s;
        printf("%d ",s);
    }

    if (gridDumpDebugInfo) {
        imgWrite(flattenedHori,"flattenedHori.png");

    }
    printf("\n");
    sumMinVal=8*img->width;
    sumMaxVal=64*img->width;
    Img * flattenedVert = newImgColor(layer5VO->height,1,0);
    for (int y=0;y<layer5HO->height;++y) {
        int s=0;
        for (int x=0;x<layer5HO->width;++x) {
            s+=layer5HO->data[x+layer5VO->width*y];
        }
        if (s<sumMinVal) s=0;
        else if (s>sumMaxVal) s=255;
        else s=255*(s-sumMinVal)/(sumMaxVal-sumMinVal);
        flattenedVert->data[y]=s;
        printf("%d ",s);
    }

    if (gridDumpDebugInfo) {
        imgWrite(flattenedVert,"flattenedVert.png");
    }
    printf("\n");

    // the 10 equidistant point should spread on the minimum
    // between img->width and img->height and that value
    // divided by 2.
    int minLength = img->width;
    if (img->height<minLength) minLength=img->height;
    int endRange=minLength;
    int startRange=endRange/2;
    int maxCorrelationVer=0;
    gridIdentifyNPoints(10,flattenedVert,startRange, endRange,ymin,ymax,
                        &maxCorrelationVer);

    if (gridDumpDebugInfo) {
        HERE("+++++++++++++++");
    }
    int maxCorrelationHori=0;
    gridIdentifyNPoints(10,flattenedHori,startRange, endRange,xmin,xmax,
                        &maxCorrelationHori);
    HERE("maxCorrelationHori");
    HERED(maxCorrelationHori);
    HERE("maxCorrelationVer");
    HERED(maxCorrelationVer);

    deleteImg(flattenedVert);
    deleteImg(flattenedHori);

    deleteImg(layer1HO);
    deleteImg(layer1VO);
    return 0;
}
```

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.

```
                                  {
    for (int j=1;j<argc;++j) {
        if (strcmp(argv[j],"-h")==0 || strcmp(argv[j],"--help")==0) {
            gridUsage(stdout,argv[0]);
            exit(0);
        }
    }
    if(argc < 3) {
        gridUsage(stderr,argv[0]);
        ERROR("At least 2 argumens expected, the picture to read, and where tp write the output","");
    }
    /* read input image */
    Img * currentImage=newImgRead(argv[1]);
    int xmin,ymin,xmax,ymax;
    // locate the grid
    if (!gridLocate(currentImage,&xmin,&ymin,&xmax,&ymax)) {
        HERE("Found sudoku grid :");
        printf("%d %d %d %d\n",xmin,ymin,xmax,ymax);
        // draw the grid found
        imgDrawRect(currentImage,xmin,ymin,xmax,ymax);
        // write the image
        imgWrite(currentImage,argv[2]);
    } else {
        ERROR("Could not find a grid in picture: ",argv[1]);
    }
    return 0;
}
```

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

4.15 grid.h

Go to the documentation of this file.

```
#ifndef GRID_H
#define GRID_H

typedef struct img Img;

int gridLocate(Img * img,
               int * xmin, int * ymin,
               int * xmax, int * ymax);
int gridMain(int argc,char**argv);

#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.

```
                            {
    if (myImg==NULL) return;
    free(myImg->data);
    myImg->width=0;
    myImg->height=0;
    myImg->data=NULL;
    free(myImg);
}
```

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.

```
                                  {
    Img * answer = newImgCopy(in);
    int sq=radius*radius;
    for(int y = 0; y < in->height; y++) {
        for(int x = 0; x < in->width; x++) {
            int v=0;
            for (int xx=-radius;xx<radius;++xx) {
                for (int yy=-radius;yy<radius;++yy) {
                    if (xx*yy<sq)
                        v+=imgGetVal(in,x+xx,y+yy);
                }
            }
            answer->data[x + y * in->width]=INBYTE((int)(v/sq/3.14));
        }
    }
    return answer;
}
```

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.

```
                                          {
    if (in->width<filter->img->width)
        ERROR("Wrong width","");
    if (in->height<filter->img->height)
        ERROR("Wrong height","");
    int aw=in->width-filter->img->width+1;
    int ah=in->height-filter->img->height+1;
    Img * answer = newImgColor(aw,ah,0);
    for(int y = 0; y < ah; y++) {
        for(int x = 0; x < aw; x++) {
            long int v=0;
            for(int yy = 0; yy < filter->img->height; yy++) {
                for(int xx = 0; xx < filter->img->width; xx++) {
                    v+=imgGetVal(in,x+xx,y+yy)*imgGetVal(filter->img,xx,yy);
                }
            }
            answer->data[x+aw*y]=
                (v<filter->threshold)?0
                :(v>filter->maxVal)?255
                :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
        }
    }
    return answer;
}
```

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.

```
                                             {
    if (in->width<filter->img->width)
        ERROR("Wrong width","");
    if (in->height<filter->img->height)
        ERROR("Wrong height","");
    int aw=in->width-filter->img->width+1;
    int ah=in->height-filter->img->height+1;
    Img * answer = newImgColor(aw,ah,0);
    for(int y = 0; y < ah; y++) {
        for(int x = 0; x < aw; x++) {
            long int v=0;
            for(int yy = 0; yy < filter->img->height; yy++) {
                for(int xx = 0; xx < filter->img->width; xx++) {
                    v+=((int)imgGetVal(in,x+xx,y+yy))*
                        (((int)imgGetVal(filter->img,xx,yy))-128);
                }
            }
            answer->data[x+aw*y]=
                (v<filter->threshold)?0
                :(v>filter->maxVal)?255
                :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
        }
    }
    return answer;
}
```

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.

```
                                                  {
    if (in->width<filter->img->width || in->height<filter->img->height)
        ERROR("Wrong size","");
    int wfOver2=filter->img->width/2;
    int hfOver2=filter->img->height/2;
    Img * answer = newImgColor(in->width,in->height,0);
    for(int y = 0; y < in->height; y++) {
        for(int x = 0; x < in->width; x++) {
            long int v=0;
            for(int yy = 0; yy < filter->img->height; yy++) {
                for(int xx = 0; xx < filter->img->width; xx++) {
                    int xxx=x+xx-wfOver2;
                    int yyy=y+yy-hfOver2;
                    if (xxx>=0 && yyy>=0 && xxx<in->width && yyy<in->height) {
                        v+=imgGetVal(in,xxx,yyy)*imgGetVal(filter->img,xx,yy);
                    }
                }
            }
            //HERE("___v,maxval,threshold,percent__________________");
            //HERED((int)v);
            //HERED((int)filter->maxVal);
            //HERED((int)filter->threshold);
            //HERED((int)filter->percent);
            answer->data[x+in->width*y]=
                (v<filter->threshold)?0
                :(v>filter->maxVal)?255
                :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
        }
    }
    return answer;
}
```

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.

```
                                                       {
    if (in->width<filter->img->width || in->height<filter->img->height)
        ERROR("Wrong size","");
    int wfOver2=filter->img->width/2;
    int hfOver2=filter->img->height/2;
    Img * answer = newImgColor(in->width,in->height,0);
    for(int y = 0; y < in->height; y++) {
        for(int x = 0; x < in->width; x++) {
            long int v=0;
            for(int yy = 0; yy < filter->img->height; yy++) {
                for(int xx = 0; xx < filter->img->width; xx++) {
                    int xxx=x+xx-wfOver2;
                    int yyy=y+yy-hfOver2;
                    if (xxx>=0 && yyy>=0 && xxx<in->width && yyy<in->height) {

                        v+=((int)imgGetVal(in,xxx,yyy))*
                            (((int)imgGetVal(filter->img,xx,yy))-128);
                        /*
                        int d = imgGetVal(in,xxx,yyy)
                            -imgGetVal(filter->img,xx,yy);
                        v+=d*d;
                        */
                    }
                }
            }
            /*
            HERE("___v,maxval,threshold,percent__________________");
            HERED((int)v);
            HERED((int)filter->maxVal);
            HERED((int)filter->threshold);
            HERED((int)filter->percent);
            */

            answer->data[x+in->width*y]=
                (v<filter->threshold)?0
                :(v>filter->maxVal)?255
                :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
        }
    }
    return answer;
}
```

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.

```
                              {
    Img * answer = newImgColor(img->width/2,img->height/2,255);
    for (int y=0;y<img->height/2;++y) {
        int o = (img->width»1)*y;
        for (int x=0;x<img->width/2;++x) {
            // compute the average of 4 pixels
            answer->data[x+o] =
                (img->data[(x«1)+img->width*(y«1)] +
                 img->data[(x«1)+1+img->width*(y«1)] +
                 img->data[(x«1)+img->width*((y«1)+1)] +
                 img->data[(x«1)+1+img->width*((y«1)+1)] )»2;
        }
    }
    return answer;
}
```

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.

```
                                                        {
    if (img->width < poolsize)
        ERROR("Wrong size.","");
    if (img->height < poolsize)
        ERROR("Wrong size.","");
    int w =(img->width - poolsize)/stride+1;
    int h =(img->height - poolsize)/stride+1;
    Img* answer = newImgColor(w,h,0);
    for (int x=0;x<w;++x) {
        for (int y=0;y<h;++y) {
            int avg=0;
            for (int xx=0;xx<poolsize;++xx) {
                for (int yy=0;yy<poolsize;++yy) {
                    avg+=img->data[x*stride+xx+img->width*(y*stride+yy)];
                }
            }
            answer->data[x+w*y]=avg/poolsize/poolsize;
        }
    }
    return answer;
}
```

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.

```
                                                        {
    if (img->width < poolsize)
        ERROR("Wrong size, poolsize too large.","");
    if (img->height < poolsize)
        ERROR("Wrong size, poolsize too large.","");
    int w =(img->width - poolsize)/stride+1;
    int h =(img->height - poolsize)/stride+1;
    Img* answer = newImgColor(w,h,0);
    for (int x=0;x<w;++x) {
        for (int y=0;y<h;++y) {
            int max=0;
            for (int xx=0;xx<poolsize;++xx) {
                for (int yy=0;yy<poolsize;++yy) {
                    int v=img->data[x*stride+xx+img->width*(y*stride+yy)];
                    if (v>max)
                        max=v;
                }
            }
            answer->data[x+w*y]=max;
        }
    }
    return answer;
}
```

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

```
                                                                {
    if (xmin<0 || ymin<0 || xmax >myImg->width || myImg->height<ymax ||
        xmin>=xmax || ymin>=ymax ) {
        fprintf(stderr,"rectangle : %d %d %d %d\n",xmin,ymin,xmax,ymax);
        fprintf(stderr,"image size is %d %d\n",myImg->width,myImg->height);
        ERROR("Wrong size.","");
    }
    for(int x = xmin; x <xmax; x++) {
        myImg->data[x+myImg->width*ymin]=(x»2)%2?255:0;
        myImg->data[x+myImg->width*ymax]=(x»2)%2?255:0;
    }
    for(int y = ymin; y <ymax; y++) {
        myImg->data[xmin+myImg->width*y]=(y»2)%2?255:0;
        myImg->data[xmax+myImg->width*y]=(y»2)%2?255:0;
    }
}
```

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.

```
                                                                {
    if (xmin<0 || ymin<0 || xmax >myImg->width || myImg->height<ymax ||
        xmin>=xmax || ymin>=ymax ) {
        fprintf(stderr,"rectangle : %d %d %d %d\n",xmin,ymin,xmax,ymax);
        fprintf(stderr,"image size is %d %d\n",myImg->width,myImg->height);
        ERROR("Wrong size.","");
    }
    int w=xmax-xmin;
    int h=ymax-ymin;
    Img * answer = newImgColor(w,h,255);
    for(int y = 0; y<h; y++) {
        // use memcopy to be faster since horizontal pixels
        // are stored one next to the other.
        memcpy(&answer->data[0+y*w],
               &myImg->data[xmin+myImg->width*(ymin+y)],
               w);
    }
    return answer;
}
```

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.

```
                                {
    Img * answer = newImgCopy(in);
    for(int i = 0; i < in->height*in->width; i++) {
        int m = in->data[i];
        float v = (m-128.0)/128.;
        float w = v*v;
        answer->data[i]=v>0?INBYTE(w*128+128):INBYTE(128-w*128);
    }
    return answer;
}
```

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.

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

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.

```
                         {
    int answer =0;
    for(int i = 0; i < in->height*in->width; i++) {
        answer+=in->data[i];
    }
    return answer;
}
```

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.

```
                       {
    Img * answer = newImgCopy(in);
    for(int i = 0; i < in->height*in->width; i++) {
        answer->data[i]=255-in->data[i];
    }
    return answer;
}
```

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.

```
                                {
    int lumCount[256];
    memset(lumCount,0,sizeof(int)«8);
    Img * answer = newImgCopy(in);
    for(int i = 0; i < in->height*in->width; i++) {
        lumCount[in->data[i]]++;
    }
    // average value is a celle of lumCount should
    // be avg=(in->height*in->width/256
    // we are going to suppress below and above avg/8
    int threshold=(in->height*in->width)»12; // divide by 256*8
    if (threshold<2) threshold=2;
    int topLum=255;
    while (topLum>0 && lumCount[topLum]<threshold) --topLum;
    int botLum=0;
    while (botLum<topLum && lumCount[botLum]<threshold) ++botLum;
    //topLum=100;
    for(int i = 0; i < in->height*in->width; i++) {
        answer->data[i]=
            INBYTE((in->data[i]-botLum)*255/(topLum-botLum));
    }
    return answer;
}
```

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.

```
                                    {
    if (argc==1) {
        imgUsage(stdout,argv[0]);
        exit(0);
    }
    for (int j=1;j<argc;++j) {
        if (strcmp(argv[j],"-h")==0 || strcmp(argv[j],"--help")==0) {
            imgUsage(stdout,argv[0]);
            exit(0);
        }
    }
    int i=1;
    Img * currentImage=NULL;
    if (argv[i][0]!='-') {
        currentImage=newImgRead(argv[1]);
        i=2;
    }
#define NOT_ENOUGH                                        \
    if (i>=argc) {                                        \
        imgUsage(stderr,argv[0]);                         \
        ERROR("not enough arguments.","");                \
    }

    Img * newImage=NULL;
    while (i<argc) {
        if (argv[i][0]!='-') {
            imgWrite(currentImage,argv[i]);
        } else {
            if (strcmp(argv[i],"--blur")==0 ||
                strcmp(argv[i],"-b")==0 ) {
                ++i;
                NOT_ENOUGH;
                int s=atoi(argv[i]);
                newImage=imgBlur(currentImage,s);
            } else if (strcmp(argv[i],"--cross")==0) {
                ++i;
                NOT_ENOUGH;
                int s=atoi(argv[i]);
                ++i;
                NOT_ENOUGH;
                int w=atoi(argv[i]);
                ++i;
                NOT_ENOUGH;
                int t=atoi(argv[i]);
                newImage=newImgCross(s,w,t);
            } else if (strcmp(argv[i],"--square")==0) {
                ++i;
                NOT_ENOUGH;
                int s=atoi(argv[i]);
                ++i;
                NOT_ENOUGH;
                int w=atoi(argv[i]);
                ++i;
                NOT_ENOUGH;
                int t=atoi(argv[i]);
                newImage=newImgSquare(s,w,t);
            } else if (strcmp(argv[i],"--conv")==0) {
                ++i;
                NOT_ENOUGH;
                char * fileName = argv[i];
                ++i;
                NOT_ENOUGH;
                int percent=atoi(argv[i]);
                Filter * f =newFilter(newImgRead(fileName),percent);
                newImage=imgConvolutionSameSizeDiff(currentImage,f);
                deleteFilter(f);
            } else if (strcmp(argv[i],"--9x9dots")==0) {
                ++i;
                NOT_ENOUGH;
                int w=atoi(argv[i]);
                newImage=newImg9By9Dots(w);
            } else if (strcmp(argv[i],"--3x3-edge")==0) {
                newImage=newImg33edge();
            } else if (strcmp(argv[i],"--1d-dots")==0) {
                ++i;
                NOT_ENOUGH;
                int N=atoi(argv[i]);
                ++i;
                NOT_ENOUGH;
                int w=atoi(argv[i]);
                newImage=newImgNDotsHori(N,w);
            } else if (strcmp(argv[i],"--vertical")==0 ||
                strcmp(argv[i],"-v")==0 ) {
                ++i;
                NOT_ENOUGH;
                int s=atoi(argv[i]);
                ++i;
                NOT_ENOUGH;
                int w=atoi(argv[i]);
                newImage=newImgVerticalBar(s,w);
            } else if (strcmp(argv[i],"--scale")==0 ||
                strcmp(argv[i],"-s")==0 ) {
                ++i;
                NOT_ENOUGH;
                int s=atoi(argv[i]);
                newImage=imgScale(currentImage,s);
            } else if (strcmp(argv[i],"--rotate")==0 ||
                strcmp(argv[i],"-r")==0 ) {
                ++i;
                NOT_ENOUGH;
                int s=atoi(argv[i]);
                newImage=imgRotate(currentImage,s);
            } else if (strcmp(argv[i],"--lum")==0 ||
                strcmp(argv[i],"-l")==0 ) {
                newImage=imgLuminosityScale(currentImage);
            } else if (strcmp(argv[i],"--inv")==0 ||
                strcmp(argv[i],"-i")==0 ) {
                newImage=imgInvert(currentImage);
            } else if (strcmp(argv[i],"--contrast")==0 ||
                strcmp(argv[i],"-c")==0 ) {
                newImage=imgRaiseContrast(currentImage);
            } else if (strcmp(argv[i],"--flatten")==0 ||
                strcmp(argv[i],"-f")==0 ) {
                newImage=imgFlattenContrast(currentImage);
            } else if (strcmp(argv[i],"--3d")==0 ||
                strcmp(argv[i],"-3d")==0 ) {
                newImage=imgMake3dEffect(currentImage);
            } else if (strcmp(argv[i],"--print")==0 ||
                strcmp(argv[i],"-p")==0 ) {
                imgPrint(currentImage);
            } else if (strcmp(argv[i],"--sudoku")==0) {
                ++i;
                int sz=atoi(argv[i]);
                newImage=newImgSudoku(sz);
            } else if (strcmp(argv[i],"--version")==0) {
                printf("%s version %s\n",basename(argv[0]), VERSION);
                printf("compiled with %s on %s\n",CFG_CC,__DATE__);
                printf("git hash  %s\n",CFG_GIT_FHASH);
            } else {
                imgUsage(stdout,argv[0]);
                ERROR("Unknown option: ",argv[i]);
            }
            deleteImg(currentImage);
            currentImage=newImage;
            newImage=NULL;
        }
        ++i;
    }
    deleteImg(currentImage);
    return 0;
}
```

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 )
                             {
    Img * answer = newImgColor(in->width,in->height+in->width,255);

    for(int x = 0; x < in->width; x++) {
        for(int y = 0; y < in->height; y++) {
            answer->data[x + (in->width-x-1+y) * answer->width]=
                in->data[x + y * in->width];
        }
    }
    return answer;
}
```

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.

```
                             {
    for(int y = 0; y < myImg->height; y++) {
        for(int x = 0; x < myImg->width; x++) {
            printf("%3d ",imgGetVal(myImg,x,y));
        }
        printf("\n");
    }
}
```

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.

```
                                  {
    Img * answer = newImgCopy(in);
    for(int i = 0; i < in->height*in->width; i++) {
        int m = in->data[i];
        float v =(m-128.0)/128.;
        float w = v<0?sqrtf(-v):sqrt(v);
        answer->data[i]=v>0?INBYTE(w*128+128):INBYTE(128-w*128);
    }
    return answer;
}
```

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

```
                                 {
    Img * answer = newImgColor(img->width,img->height,255);
    float rad=deg*3.14159/180;
    float c = cos(rad);
    float s = sin(rad);
    int w=img->width;
    int h=img->height;
    float eps=0.0001;    // epsilum, considered as zero
    for (int x=0;x<w;++x) {
        for (int y=0;y<h;++y) {
            double xd= (x-w/2)*c - (y-h/2)*s + w/2;
            double yd= (x-w/2)*s + (y-h/2)*c + h/2;
            if (xd>0 && yd>0 && xd<w-1 && yd<h-1) {
                double xf=floor(xd);
                double xc=ceil(xd);
                double yf=floor(yd);
                double yc=ceil(yd);
                double ff=sqrt((xd-xf)*(xd-xf)+(yd-yf)*(yd-yf));
                double fc=sqrt((xd-xf)*(xd-xf)+(yd-yc)*(yd-yc));
                double cf=sqrt((xd-xc)*(xd-xc)+(yd-yf)*(yd-yf));
                double cc=sqrt((xd-xc)*(xd-xc)+(yd-yc)*(yd-yc));
                int v=255;
                if (-eps<ff && ff<eps) {
                    v=img->data[(int)(xf+w*yf)];
                } else if (-eps<fc && fc<eps) {
                    v=img->data[(int)(xf+w*yc)];
                } else if (-eps<cf && cf<eps) {
                    v=img->data[(int)(xc+w*yf)];
                } else if (-eps<cc && cc<eps) {
                    v=img->data[(int)(xc+w*yc)];
                } else {
                    // not too close to any point, so compute average
                    double t=1/ff+1/fc+1/cf+1/cc;
                    v=INBYTE((int)((img->data[(int)(xf+w*yf)]/ff+
                                    img->data[(int)(xf+w*yc)]/fc+
                                    img->data[(int)(xc+w*yf)]/cf+
                                    img->data[(int)(xc+w*yc)]/cc)/t));
                }
                answer->data[x+w*y]=v;
            }
        }
    }
    return answer;
}
```

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

```
                           {
    // exchange height and width
    Img * answer = newImgColor(img->height,img->width,255);
    for (int y=0;y<answer->height;++y) {
        int o = answer->width*y;
        for (int x=0;x<answer->width;++x) {
            answer->data[x+o]=img->data[y+img->width*x];
        }
    }
    return answer;
}
```

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.

```
                                                              {
    if (i1->width<i2->width+xoffset)
        ERROR("Wrong size","");
    if (i1->height<i2->height+yoffset)
        ERROR("Wrong size","");
    long int s=0;
    for (int y=0;y<i2->height;++y) {
        for (int x=0;x<i2->width;++x) {
            int v=i1->data[x+xoffset+(y+yoffset)*i1->width]
                -i2->data[x+y*i1->width];
            s+=v<0?-v:v;
        }
    }
    int max=i2->height*i2->width*50;
    if (max<s) return 0;
    return INBYTE(255-(s*255)/max);
}
```

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.

```
                            {
    Img * answer = newImgColor(in->width*s,in->height*s,255);

    for(int x = 0; x < in->width; x++) {
        for(int y = 0; y < in->height; y++) {
            for(int xx = 0; xx < s; xx++) {
                for(int yy = 0; yy < s; yy++) {
                    answer->data[s*x +xx + (yy+s*y) * answer->width]=
                        in->data[x + y * in->width];
                }
            }
        }
    }
    return answer;
}
```

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].

```
                                  {
    char * bname=basename(name);
    fprintf(f,"%s usage:\n", bname);
    fprintf(f,"    %s [<input-file>] [ options ] <output-file>\n", bname);
    fprintf(f,"Where option is one of:\n");
    fprintf(f,"    [-b|--blur] <r>:\n");
    fprintf(f,"        Blurs the image with a radius of r.\n");
    fprintf(f,"    [-c|--contrast] :\n");
    fprintf(f,"        Raises contrast level.\n");
    fprintf(f,"    --conv <filename> <percent>:\n");
    fprintf(f,"        load image in <filename> and use it as a convolution\n");
    fprintf(f,"        filter with a threshold set at <percent>.\n");
    fprintf(f,"    --cross <n> <w> <t>:\n");
    fprintf(f,"        Generates a black cross on a white background. \n");
    fprintf(f,"        The cross size is n by n pixels in a w by w\n");
    fprintf(f,"        picture. The cross line is <t> pixels thick.\n");
    fprintf(f,"    [-f|--flatten] :\n");
    fprintf(f,"        Flattens the contrast.\n");
    fprintf(f,"    [-h|--help] :\n");
    fprintf(f,"        Displays this help message and leaves.\n");
    fprintf(f,"    [-i|--inv] :\n");
    fprintf(f,"        Inverses the image. White becomes black, black \n");
    fprintf(f,"        becomes white.\n");
    fprintf(f,"    [-l|--lum] :\n");
    fprintf(f,"        Scales luminosity if the image is too bright or too dark.\n");
    fprintf(f,"    [-p|--print] :\n");
    fprintf(f,"        Prints on stdout numerical value of current image.\n");
    fprintf(f,"    [-s|--scale] <n>:\n");
    fprintf(f,"        Scales image by a factor n.\n");
    fprintf(f,"    --square <n> <w> <t>:\n");
    fprintf(f,"        Generates a black square on a white background. \n");
    fprintf(f,"        The square size is n by n pixels in a w by w\n");
    fprintf(f,"        picture. The square line is <t> pixels thick.\n");
    fprintf(f,"    --sudoku <n>:\n");
    fprintf(f,"        Generates a black empty sudoku grid of n pixels large in a \n");
    fprintf(f,"        224 by 224 picture with white background.\n");
    fprintf(f,"    [-r|--rotate] <n>:\n");
    fprintf(f,"        Rotates an image by n degrees.\n");
    fprintf(f,"    [-v|--vertical] <s> <w>:\n");
    fprintf(f,"        Generates a black vertical bar of width <w>\n");
    fprintf(f,"        in a white square of size <s>.\n");
    fprintf(f,"    --version :\n");
    fprintf(f,"        Displays the current version of %s.\n",bname);
    fprintf(f,"    --1d-dots <n> <w>:\n");
    fprintf(f,"        Generates <n> black dots on a white background, \n");
    fprintf(f,"        in one dimension: the generated image size is <w> \n");
    fprintf(f,"        by 1 pixels.\n");
    fprintf(f,"    --3x3-edge :\n");
    fprintf(f,"        Generates a 3x3 image to perform edge detection if \n");
    fprintf(f,"        used as convolution filter.\n");
    fprintf(f,"    [-3d|--3d] :\n");
    fprintf(f,"        Draws an isometric-like view.\n");
    fprintf(f,"        in a 224x224 image.\n");
    fprintf(f,"    --9x9dots <n>:\n");
    fprintf(f,"        Generates 9x9 black dots on a white background. \n");
    fprintf(f,"        The image size is n by n pixels.\n");

    fprintf(f,"Examples:\n");
    fprintf(f,"    %s --sudoku 200 --rotate 5 --blur 2 out.png\n",bname);
    fprintf(f,"        Generates an empty sudoku grid, then rotate it by 5\n");
    fprintf(f,"        degrees, then blurs the image with a radius of 2, then\n");
    fprintf(f,"        save the result in file out.png.\n");
    fprintf(f,"    \n");
    fprintf(f,"    %s --3x3-edge edgefilter.png\n",bname);
    fprintf(f,"        Creates a 3x3 filter to be used for edge detection.\n");
    fprintf(f,"        Save it under edgefilter.png.\n");
    fprintf(f,"    \n");
    fprintf(f,"    %s mypic.png --conv edgefilter.png -10 out.png\n",
            bname);
    fprintf(f,"        Apply the convolution filter edgefilter.png to\n");
    fprintf(f,"        mypic.png and save the result under out.png.\n");
    fprintf(f,"        -10%% of threshold is applied. Put a lower percentage\n");
    fprintf(f,"        for brighter images, a higher one for a darker.\n");
    fprintf(f,"    \n");

    fprintf(f,"    %s my_photo.jpg --lum my_new_photo.png\n",bname);
    fprintf(f,"        Reads file my_photo.jpg increase luminosity to stretch from\n");
    fprintf(f,"        dark to very clear and save the result in my_new_photo.png.\n");
}
```

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.

```
                                        {
    FILE *fp = fopen(filename, "wb");
    if(!fp) {
        ERROR("could not open file ",filename);
    }

    png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
    if (!png) {
        ERROR("could not create png structure","");
    }

    png_infop info = png_create_info_struct(png);
    if (!info) {
        ERROR("could not get info","");
    }

    if (setjmp(png_jmpbuf(png)))  {
        fprintf(stderr,
                "could not jmp to data while trying to write %s\n",
                filename);
    }

    png_init_io(png, fp);

    // Output is 8bit depth, RGBA format.
    png_set_IHDR(png,
                 info,
                 myImg->width, myImg->height,
                 8,
                 PNG_COLOR_TYPE_RGBA,
                 PNG_INTERLACE_NONE,
                 PNG_COMPRESSION_TYPE_DEFAULT,
                 PNG_FILTER_TYPE_DEFAULT
                 );

    png_write_info(png, info);

    // To remove the alpha channel for PNG_COLOR_TYPE_RGB format,
    // Use png_set_filler().
    //png_set_filler(png, 0, PNG_FILLER_AFTER);

    if (!myImg->data) abort();

    unsigned char* row_pointers[myImg->height];
    for(int y = 0; y < myImg->height; y++) {
        row_pointers[y] =
            (unsigned char*)malloc(4*myImg->width);
        unsigned char* row = row_pointers[y];
        for(int x = 0; x < myImg->width; x++) {
            unsigned char * px = &(row[x * 4]);
            px[0]=px[1]=px[2]=myImg->data[x+y*myImg->width];
            px[3]=255;
        }
    }
    png_write_image(png, row_pointers);

    for(int y = 0; y < myImg->height; y++) {
        free(row_pointers[y]);
    }
    png_write_end(png, NULL);
    fclose(fp);
    png_destroy_write_struct(&png, &info);
}
```

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.

```
                                                  {
    /* cinfo->err really points to a jpegerrmgr struct, so coerce pointer */
    MyErrorPtr myerr = (MyErrorPtr) cinfo->err;

    /* Always display the message. */
    /* We could postpone this until after returning, if we chose. */
    (*cinfo->err->output_message) (cinfo);

    /* Return control to the setjmp point */
    longjmp(myerr->setjmp_buffer, 1);
}
```

References jpegerrmgr::setjmp_buffer.

4.16.3.28 newImg33edge()

```
Img * newImg33edge ( )
```

create a 3x3 image to perform edge detection.

Returns

a newly allocated image.

```
                         {
    int c=16;
    int b=c»2;
    unsigned char a[] = {
        128+c-b, 128+c+b, 128+c-b,
        128+c+b, 128-8*c, 128+c+b,
        128+c-b, 128+c+b, 128+c-b
    };
    return newImgFromArray(3,3,a);
}
```

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.

```
                                {
    Img * answer = newImgColor(w,w,255);
    float step=w/9.;
    float stepOverTwo=step/2.;
    float stepOverThree=step/2;
    float stepOverThreeSq=stepOverThree*stepOverThree;
    for (int x=0;x<w;++x) {
        for (int y=0;y<w;++y) {
            float xr = round((x-stepOverTwo)/step)*step+stepOverTwo;
            float yr = round((y-stepOverTwo)/step)*step+stepOverTwo;
            float d = ((x-xr)*(x-xr)+(y-yr)*(y-yr))/stepOverThreeSq;
            if (d<1) {
                answer->data[x+w*y]=INBYTE((int)(d*255));
            }
        }
    }
    return answer;
}
```

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.

```
                                                    {
    Img * answer = (Img *)malloc(sizeof(struct img));
    answer->width=w;
    answer->height=h;
    answer->data=
        (unsigned char*)malloc(answer->height * answer->width);
    memset(answer->data,c,answer->height * answer->width);
    return answer;
}
```

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.

```
                            {
    Img * answer = (Img *) malloc(sizeof(struct img));

    answer->width      = myImg->width       ;
    answer->height     = myImg->height      ;
    answer->data = (unsigned char*)malloc(sizeof(char) *
                                          answer->height*answer->width);
    memcpy(answer->data,
           myImg->data,
           answer->width*answer->height);
    return answer;
}
```

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.

```
                                     {
    Img * answer = newImgColor(w,w,255);
    int m=w/2;
    int count=0;
    while (count<t && count<m) {
        if (s<w-1 && s>0) {
            for (int y=m-s/2;y<m-s/2+s;++y) {
                answer->data[m+count+w*y]=0;
            }
            for (int x=m-s/2;x<m-s/2+s;++x) {
                answer->data[x+w*(m+count)]=0;
            }
            for (int y=m-s/2;y<m-s/2+s;++y) {
                answer->data[m-count+w*y]=0;
            }
            for (int x=m-s/2;x<m-s/2+s;++x) {
                answer->data[x+w*(m-count)]=0;
            }
        }
        count++;
    }
    return answer;
}
```

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

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

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.

```
                                   {
    // create a w by 1 white image
    Img * answer = newImgColor(w,1,255);
    // space between dots is w/N
    float step=((float)w)/((float)N);
    // we begin at step/2 from the border
    float stepOverTwo=step/2.;
    float stepOverFour=step/4;
    float stepOverFourSq=stepOverFour*stepOverFour;
    for (int x=0;x<w;++x) {
        // horizontal position of the closest black dot
        float xr = round((x-stepOverTwo)/step)*step+stepOverTwo;
        // compute distance to the closest black dot
        float d = fabs(xr-x);
        float dmax=3;
        if (d<dmax) {
            // we are close to a black dot so we darken the pixel
            answer->data[x]=INBYTE((int)(d*255/dmax));
        }
    }
    return answer;
}
```

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.

```
                                   {
    int l = strlen(filename);
    if ( (l>4 &&
          filename[l-4]=='.' &&
          (filename[l-3]=='j'|| filename[l-3]=='J')&&
          (filename[l-2]=='p'|| filename[l-2]=='P')&&
          (filename[l-1]=='g'|| filename[l-1]=='G'))
         ||
         (l>5 &&
          filename[l-5]=='.' &&
          (filename[l-4]=='j'|| filename[l-4]=='J')&&
          (filename[l-3]=='p'|| filename[l-3]=='P')&&
          (filename[l-2]=='e'|| filename[l-2]=='E')&&
          (filename[l-1]=='g'|| filename[l-1]=='G'))) {
        return newImgReadJpeg(filename);
    }
    // otherwise we assume it is a png
    return newImgReadPng(filename);
}
```

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.

```
                                            {
    struct jpeg_decompress_struct cinfo;
    struct jpegerrmgr jerr;
    FILE * infile;      /* source file */
    JSAMPARRAY buffer;      /* Output row buffer */
    int row_stride;     /* physical row width in output buffer */

    if ((infile = fopen(filename, "rb")) == NULL) {
        ERROR("can’t open ", filename);
    }

    cinfo.err = jpeg_std_error(&jerr.pub);
    jerr.pub.error_exit = my_error_exit;

    if (setjmp(jerr.setjmp_buffer)) {
        // JPEG code has signaled an error.
        jpeg_destroy_decompress(&cinfo);
        fclose(infile);
        ERROR("Error in jpeg file ", filename);
    }
    jpeg_create_decompress(&cinfo);
    jpeg_stdio_src(&cinfo, infile);

    (void) jpeg_read_header(&cinfo, TRUE);

    (void) jpeg_start_decompress(&cinfo);

    row_stride = cinfo.output_width * cinfo.output_components;
    buffer = (*cinfo.mem->alloc_sarray)
        ((j_common_ptr) &cinfo, JPOOL_IMAGE, row_stride, 1);
    int lineCount=0;
    Img * answer = newImgColor(cinfo.image_width,cinfo.image_height,0);
    // read the image line by line
    while (cinfo.output_scanline < cinfo.output_height) {
        (void) jpeg_read_scanlines(&cinfo, buffer, 1);
        for (int i=0;i<row_stride;++i) {
            int j = (buffer[0][i]+buffer[0][i+1]+buffer[0][i+2])/3;
            answer->data[i/3+lineCount*cinfo.image_width]=j;
        }
        ++lineCount;
    }
    (void) jpeg_finish_decompress(&cinfo);
    jpeg_destroy_decompress(&cinfo);
    fclose(infile);
    return answer;
}
```

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.

```
                                           {
    Img * answer = (Img *)malloc(sizeof(struct img));
    FILE *fp = fopen(filename, "rb");
    if(!fp) {
        ERROR("file not found - ",filename);
    }

    png_structp png = png_create_read_struct(PNG_LIBPNG_VER_STRING,
                                             NULL, NULL, NULL);

    png_infop info = png_create_info_struct(png);
    if(!info) abort();

    if(setjmp(png_jmpbuf(png))) {
        ERROR("Error","");
    }

    png_init_io(png, fp);

    png_read_info(png, info);

    answer->width  = png_get_image_width(png, info);
    answer->height = png_get_image_height(png, info);
    int color_type = png_get_color_type(png, info);
    int bit_depth  = png_get_bit_depth(png, info);

    // Read any color_type into 8bit depth, RGBA format.
    // See http://www.libpng.org/pub/png/libpng-manual.txt

    if(bit_depth == 16)
        png_set_strip_16(png);

    if(color_type == PNG_COLOR_TYPE_PALETTE)
        png_set_palette_to_rgb(png);

    // PNG_COLOR_TYPE_GRAY_ALPHA is always 8 or 16bit depth.
    if(color_type == PNG_COLOR_TYPE_GRAY && bit_depth < 8)
        png_set_expand_gray_1_2_4_to_8(png);

    if(png_get_valid(png, info, PNG_INFO_tRNS))
        png_set_tRNS_to_alpha(png);

    // These color_type don't have an alpha channel then fill it with 0xff.
    if(color_type == PNG_COLOR_TYPE_RGB ||
       color_type == PNG_COLOR_TYPE_GRAY ||
       color_type == PNG_COLOR_TYPE_PALETTE)
        png_set_filler(png, 0xFF, PNG_FILLER_AFTER);

    if(color_type == PNG_COLOR_TYPE_GRAY ||
       color_type == PNG_COLOR_TYPE_GRAY_ALPHA)
        png_set_gray_to_rgb(png);

    png_read_update_info(png, info);

    unsigned char* row_pointers[answer->height];
    answer->data=
        (unsigned char*)malloc(sizeof(char*) * answer->height * answer->width);
    for(int y = 0; y < answer->height; y++) {
        row_pointers[y] =
            (unsigned char*)malloc(png_get_rowbytes(png,info));
    }
    png_read_image(png, row_pointers);
    for(int y = 0; y < answer->height; y++) {
        unsigned char* row = row_pointers[y];
        for(int x = 0; x < answer->width; x++) {
            unsigned char * px = &(row[x * 4]);
            answer->data[x+y*answer->width]=(px[0]+px[1]+px[2])/3;
        }
    }
    for(int y = 0; y < answer->height; y++) {
        free(row_pointers[y]);
    }
    fclose(fp);
    png_destroy_read_struct(&png, &info, NULL);
    return answer;
}
```

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.

```
                                      {
    Img * answer = newImgColor(w,w,255);
    int m=w/2;
    int count=0;
    while (count<t) {
        if (s<w-1 && s>0) {
            for (int y=m-s/2;y<m-s/2+s;++y) {
                answer->data[m-s/2+w*y]=0;
            }
            for (int y=m-s/2;y<m-s/2+s;++y) {
                answer->data[m-s/2+s-1+w*y]=0;
            }
            for (int x=m-s/2;x<m-s/2+s;++x) {
                answer->data[x+w*(m-s/2)]=0;
            }
            for (int x=m-s/2;x<m-s/2+s;++x) {
                answer->data[x+w*(m-s/2+s-1)]=0;
            }
        }
        count++;
        s--;
    }
    return answer;
}
```

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.

```
                            {
    int w=212;
    int h=212;
    Img * answer = newImgColor(w,h,255);
    int xinit=224/2-sz/2;
    int yinit=xinit;
    int step=sz/9;
    for (int x=0;x<10;++x) {
        int xx=xinit+x*step;
        for (int yy=yinit;yy<yinit+step*9-1;++yy) {
            if (xx>=0 && yy>=0 && xx<w && yy<h)
                answer->data[xx+w*yy]=0;
        }
    }
    answer->data[1+xinit+9*step+w*yinit]=0;
    answer->data[xinit+w*(yinit+1+9*step)]=0;
    for (int y=0;y<10;++y) {
        int yy=yinit+y*step;
        for (int xx=xinit;xx<xinit+step*9-1;++xx) {
            if (xx>=0 && yy>=0 && xx<w && yy<h)
                answer->data[xx+w*yy]=0;
        }
    }
    for (int x=0;x<=3;++x) {
        int xx=xinit+x*step*3;
        for (int yy=yinit;yy<yinit+step*9-1;++yy) {
            if (xx>=0 && yy>=0 && xx<w && yy<h)
                answer->data[1+xx+w*(yy+1)]=0;
        }
    }
    for (int y=0;y<=3;++y) {
        int yy=yinit+y*step*3;
        for (int xx=xinit;xx<xinit+step*9-1;++xx) {
            if (xx>=0 && yy>=0 && xx<w && yy<h)
                answer->data[1+xx+w*(yy+1)]=0;
        }
    }
    return answer;
}
```

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.

```
                                      {
    Img * answer = newImgColor(s,s,255);
    int m=s/2;
    for (int x=m-w/2;x<m-w/2+w;++x) {
        for (int y=0;y<s;++y) {
            answer->data[x+s*y]=0;
        }
    }
    return answer;
}
```

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.

```
                                                     {
    Img * answer = newImgColor(sx,sy,255);
    int m=sx/2;
    if (m<w) {
        ERROR("Wrong size.","");
    }
    // put some grey arond the bar so that the
    // equivalent filter is 0
    for (int x=m-w;x<=m+w;++x) {
        for (int y=0;y<sy;++y) {
            answer->data[x+sx*y]=128;
        }
    }
    for (int x=m-w/2;x<m-w/2+w;++x) {
        for (int y=0;y<sy;++y) {
            answer->data[x+sx*y]=0;
        }
    }
    return answer;
}
```

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.

```
                          {
    if (myImg==NULL) return;
    free(myImg->data);
    myImg->width=0;
    myImg->height=0;
    myImg->data=NULL;
    free(myImg);
}
```

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.

```
                                {
    Img * answer = newImgCopy(in);
    int sq=radius*radius;
    for(int y = 0; y < in->height; y++) {
        for(int x = 0; x < in->width; x++) {
            int v=0;
            for (int xx=-radius;xx<radius;++xx) {
                for (int yy=-radius;yy<radius;++yy) {
                    if (xx*yy<sq)
                        v+=imgGetVal(in,x+xx,y+yy);
                }
            }
            answer->data[x + y * in->width]=INBYTE((int)(v/sq/3.14));
        }
    }
    return answer;
}
```

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.

```
                                            {
    if (in->width<filter->img->width)
        ERROR("Wrong width","");
    if (in->height<filter->img->height)
        ERROR("Wrong height","");
    int aw=in->width-filter->img->width+1;
    int ah=in->height-filter->img->height+1;
    Img * answer = newImgColor(aw,ah,0);
    for(int y = 0; y < ah; y++) {
        for(int x = 0; x < aw; x++) {
            long int v=0;
            for(int yy = 0; yy < filter->img->height; yy++) {
                for(int xx = 0; xx < filter->img->width; xx++) {
                    v+=imgGetVal(in,x+xx,y+yy)*imgGetVal(filter->img,xx,yy);
                }
            }
            answer->data[x+aw*y]=
                (v<filter->threshold)?0
                :(v>filter->maxVal)?255
                :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
        }
    }
    return answer;
}
```

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.

```
                                              {
    if (in->width<filter->img->width)
        ERROR("Wrong width","");
    if (in->height<filter->img->height)
        ERROR("Wrong height","");
    int aw=in->width-filter->img->width+1;
    int ah=in->height-filter->img->height+1;
    Img * answer = newImgColor(aw,ah,0);
    for(int y = 0; y < ah; y++) {
        for(int x = 0; x < aw; x++) {
            long int v=0;
            for(int yy = 0; yy < filter->img->height; yy++) {
                for(int xx = 0; xx < filter->img->width; xx++) {
                    v+=((int)imgGetVal(in,x+xx,y+yy))*
                        (((int)imgGetVal(filter->img,xx,yy))-128);
                }
            }
            answer->data[x+aw*y]=
                (v<filter->threshold)?0
                :(v>filter->maxVal)?255
                :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
        }
    }
    return answer;
}
```

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.

```
                                                  {
    if (in->width<filter->img->width || in->height<filter->img->height)
        ERROR("Wrong size","");
    int wfOver2=filter->img->width/2;
    int hfOver2=filter->img->height/2;
    Img * answer = newImgColor(in->width,in->height,0);
    for(int y = 0; y < in->height; y++) {
        for(int x = 0; x < in->width; x++) {
            long int v=0;
            for(int yy = 0; yy < filter->img->height; yy++) {
                for(int xx = 0; xx < filter->img->width; xx++) {
                    int xxx=x+xx-wfOver2;
                    int yyy=y+yy-hfOver2;
                    if (xxx>=0 && yyy>=0 && xxx<in->width && yyy<in->height) {
                        v+=imgGetVal(in,xxx,yyy)*imgGetVal(filter->img,xx,yy);
                    }
                }
            }
            //HERE("___v,maxval,threshold,percent__________________");
            //HERED((int)v);
            //HERED((int)filter->maxVal);
            //HERED((int)filter->threshold);
            //HERED((int)filter->percent);
            answer->data[x+in->width*y]=
                (v<filter->threshold)?0
                :(v>filter->maxVal)?255
                :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
        }
    }
    return answer;
}
```

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.

```
                                                      {
    if (in->width<filter->img->width || in->height<filter->img->height)
        ERROR("Wrong size","");
    int wfOver2=filter->img->width/2;
    int hfOver2=filter->img->height/2;
    Img * answer = newImgColor(in->width,in->height,0);
    for(int y = 0; y < in->height; y++) {
        for(int x = 0; x < in->width; x++) {
            long int v=0;
            for(int yy = 0; yy < filter->img->height; yy++) {
                for(int xx = 0; xx < filter->img->width; xx++) {
                    int xxx=x+xx-wfOver2;
                    int yyy=y+yy-hfOver2;
                    if (xxx>=0 && yyy>=0 && xxx<in->width && yyy<in->height) {

                        v+=((int)imgGetVal(in,xxx,yyy))*
                            (((int)imgGetVal(filter->img,xx,yy))-128);
                        /*
                        int d = imgGetVal(in,xxx,yyy)
                            -imgGetVal(filter->img,xx,yy);
                        v+=d*d;
                        */
                    }
                }
            }
            /*
            HERE("___v,maxval,threshold,percent__________________");
            HERED((int)v);
            HERED((int)filter->maxVal);
            HERED((int)filter->threshold);
            HERED((int)filter->percent);
            */

            answer->data[x+in->width*y]=
                (v<filter->threshold)?0
                :(v>filter->maxVal)?255
                :(255*(v-filter->threshold))/(filter->maxVal-filter->threshold);
        }
    }
    return answer;
}
```

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.

```
                              {
    Img * answer = newImgColor(img->width/2,img->height/2,255);
    for (int y=0;y<img->height/2;++y) {
        int o = (img->width»1)*y;
        for (int x=0;x<img->width/2;++x) {
            // compute the average of 4 pixels
            answer->data[x+o] =
                (img->data[(x«1)+img->width*(y«1)] +
                 img->data[(x«1)+1+img->width*(y«1)] +
                 img->data[(x«1)+img->width*((y«1)+1)] +
                 img->data[(x«1)+1+img->width*((y«1)+1)] )»2;
        }
    }
    return answer;
}
```

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.

```
                                                        {
    if (img->width < poolsize)
        ERROR("Wrong size.","");
    if (img->height < poolsize)
        ERROR("Wrong size.","");
    int w =(img->width - poolsize)/stride+1;
    int h =(img->height - poolsize)/stride+1;
    Img* answer = newImgColor(w,h,0);
    for (int x=0;x<w;++x) {
        for (int y=0;y<h;++y) {
            int avg=0;
            for (int xx=0;xx<poolsize;++xx) {
                for (int yy=0;yy<poolsize;++yy) {
                    avg+=img->data[x*stride+xx+img->width*(y*stride+yy)];
                }
            }
            answer->data[x+w*y]=avg/poolsize/poolsize;
        }
    }
    return answer;
}
```

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.

```
                                                        {
    if (img->width < poolsize)
        ERROR("Wrong size, poolsize too large.","");
    if (img->height < poolsize)
        ERROR("Wrong size, poolsize too large.","");
    int w =(img->width - poolsize)/stride+1;
    int h =(img->height - poolsize)/stride+1;
    Img* answer = newImgColor(w,h,0);
    for (int x=0;x<w;++x) {
        for (int y=0;y<h;++y) {
            int max=0;
            for (int xx=0;xx<poolsize;++xx) {
                for (int yy=0;yy<poolsize;++yy) {
                    int v=img->data[x*stride+xx+img->width*(y*stride+yy)];
                    if (v>max)
                        max=v;
                }
            }
            answer->data[x+w*y]=max;
        }
    }
    return answer;
}
```

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

```
                                                                {
    if (xmin<0 || ymin<0 || xmax >myImg->width || myImg->height<ymax ||
        xmin>=xmax || ymin>=ymax ) {
        fprintf(stderr,"rectangle : %d %d %d %d\n",xmin,ymin,xmax,ymax);
        fprintf(stderr,"image size is %d %d\n",myImg->width,myImg->height);
        ERROR("Wrong size.","");
    }
    for(int x = xmin; x <xmax; x++) {
        myImg->data[x+myImg->width*ymin]=(x»2)%2?255:0;
        myImg->data[x+myImg->width*ymax]=(x»2)%2?255:0;
    }
    for(int y = ymin; y <ymax; y++) {
        myImg->data[xmin+myImg->width*y]=(y»2)%2?255:0;
        myImg->data[xmax+myImg->width*y]=(y»2)%2?255:0;
    }
}
```

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.

```
                                                                {
    if (xmin<0 || ymin<0 || xmax >myImg->width || myImg->height<ymax ||
        xmin>=xmax || ymin>=ymax ) {
        fprintf(stderr,"rectangle : %d %d %d %d\n",xmin,ymin,xmax,ymax);
        fprintf(stderr,"image size is %d %d\n",myImg->width,myImg->height);
        ERROR("Wrong size.","");
    }
    int w=xmax-xmin;
    int h=ymax-ymin;
    Img * answer = newImgColor(w,h,255);
    for(int y = 0; y<h; y++) {
        // use memcopy to be faster since horizontal pixels
        // are stored one next to the other.
        memcpy(&answer->data[0+y*w],
               &myImg->data[xmin+myImg->width*(ymin+y)],
               w);
    }
    return answer;
}
```

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.

```
                                 {
    Img * answer = newImgCopy(in);
    for(int i = 0; i < in->height*in->width; i++) {
        int m = in->data[i];
        float v = (m-128.0)/128.;
        float w = v*v;
        answer->data[i]=v>0?INBYTE(w*128+128):INBYTE(128-w*128);
    }
    return answer;
}
```

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.

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

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.

```
                           {
    int answer =0;
    for(int i = 0; i < in->height*in->width; i++) {
        answer+=in->data[i];
    }
    return answer;
}
```

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.

```
                         {
    Img * answer = newImgCopy(in);
    for(int i = 0; i < in->height*in->width; i++) {
        answer->data[i]=255-in->data[i];
    }
    return answer;
}
```

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.

```
                              {
    int lumCount[256];
    memset(lumCount,0,sizeof(int)«8);
    Img * answer = newImgCopy(in);
    for(int i = 0; i < in->height*in->width; i++) {
        lumCount[in->data[i]]++;
    }
    // average value is a celle of lumCount should
    // be avg=(in->height*in->width/256
    // we are going to suppress below and above avg/8
    int threshold=(in->height*in->width)»12; // divide by 256*8
    if (threshold<2) threshold=2;
    int topLum=255;
    while (topLum>0 && lumCount[topLum]<threshold) --topLum;
    int botLum=0;
    while (botLum<topLum && lumCount[botLum]<threshold) ++botLum;
    //topLum=100;
    for(int i = 0; i < in->height*in->width; i++) {
        answer->data[i]=
            INBYTE((in->data[i]-botLum)*255/(topLum-botLum));
    }
    return answer;
}
```

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 )
                              {
    Img * answer = newImgColor(in->width,in->height+in->width,255);

    for(int x = 0; x < in->width; x++) {
        for(int y = 0; y < in->height; y++) {
            answer->data[x + (in->width-x-1+y) * answer->width]=
                in->data[x + y * in->width];
        }
    }
    return answer;
}
```

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.

```
                              {
    for(int y = 0; y < myImg->height; y++) {
        for(int x = 0; x < myImg->width; x++) {
            printf("%3d ",imgGetVal(myImg,x,y));
        }
        printf("\n");
    }
}
```

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.

```
                              {
    Img * answer = newImgCopy(in);
    for(int i = 0; i < in->height*in->width; i++) {
        int m = in->data[i];
        float v =(m-128.0)/128.;
        float w = v<0?sqrtf(-v):sqrt(v);
        answer->data[i]=v>0?INBYTE(w*128+128):INBYTE(128-w*128);
    }
    return answer;
}
```

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

```
                                 {
    Img * answer = newImgColor(img->width,img->height,255);
    float rad=deg*3.14159/180;
    float c = cos(rad);
    float s = sin(rad);
    int w=img->width;
    int h=img->height;
    float eps=0.0001;    // epsilum, considered as zero
    for (int x=0;x<w;++x) {
        for (int y=0;y<h;++y) {
            double xd= (x-w/2)*c - (y-h/2)*s + w/2;
            double yd= (x-w/2)*s + (y-h/2)*c + h/2;
            if (xd>0 && yd>0 && xd<w-1 && yd<h-1) {
                double xf=floor(xd);
                double xc=ceil(xd);
                double yf=floor(yd);
                double yc=ceil(yd);
                double ff=sqrt((xd-xf)*(xd-xf)+(yd-yf)*(yd-yf));
                double fc=sqrt((xd-xf)*(xd-xf)+(yd-yc)*(yd-yc));
                double cf=sqrt((xd-xc)*(xd-xc)+(yd-yf)*(yd-yf));
                double cc=sqrt((xd-xc)*(xd-xc)+(yd-yc)*(yd-yc));
                int v=255;
                if (-eps<ff && ff<eps) {
                    v=img->data[(int)(xf+w*yf)];
                } else if (-eps<fc && fc<eps) {
                    v=img->data[(int)(xf+w*yc)];
                } else if (-eps<cf && cf<eps) {
                    v=img->data[(int)(xc+w*yf)];
                } else if (-eps<cc && cc<eps) {
                    v=img->data[(int)(xc+w*yc)];
                } else {
                    // not too close to any point, so compute average
                    double t=1/ff+1/fc+1/cf+1/cc;
                    v=INBYTE((int)((img->data[(int)(xf+w*yf)]/ff+
                                    img->data[(int)(xf+w*yc)]/fc+
                                    img->data[(int)(xc+w*yf)]/cf+
                                    img->data[(int)(xc+w*yc)]/cc)/t));
                }
                answer->data[x+w*y]=v;
            }
        }
    }
    return answer;
}
```

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

```
                          {
    // exchange height and width
    Img * answer = newImgColor(img->height,img->width,255);
    for (int y=0;y<answer->height;++y) {
        int o = answer->width*y;
        for (int x=0;x<answer->width;++x) {
            answer->data[x+o]=img->data[y+img->width*x];
        }
    }
    return answer;
}
```

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.

```
                                                            {
    if (i1->width<i2->width+xoffset)
        ERROR("Wrong size","");
    if (i1->height<i2->height+yoffset)
        ERROR("Wrong size","");
    long int s=0;
    for (int y=0;y<i2->height;++y) {
        for (int x=0;x<i2->width;++x) {
            int v=i1->data[x+xoffset+(y+yoffset)*i1->width]
                -i2->data[x+y*i1->width];
            s+=v<0?-v:v;
        }
    }
    int max=i2->height*i2->width*50;
    if (max<s) return 0;
    return INBYTE(255-(s*255)/max);
}
```

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.

```
                            {
    Img * answer = newImgColor(in->width*s,in->height*s,255);

    for(int x = 0; x < in->width; x++) {
        for(int y = 0; y < in->height; y++) {
            for(int xx = 0; xx < s; xx++) {
                for(int yy = 0; yy < s; yy++) {
                    answer->data[s*x +xx + (yy+s*y) * answer->width]=
                        in->data[x + y * in->width];
                }
            }
        }
    }
    return answer;
}
```

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.

```
                                           {
    FILE *fp = fopen(filename, "wb");
    if(!fp) {
        ERROR("could not open file ",filename);
    }

    png_structp png = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL, NULL, NULL);
    if (!png) {
        ERROR("could not create png structure","");
    }

    png_infop info = png_create_info_struct(png);
    if (!info) {
        ERROR("could not get info","");
    }

    if (setjmp(png_jmpbuf(png)))  {
        fprintf(stderr,
                "could not jmp to data while trying to write %s\n",
                filename);
    }

    png_init_io(png, fp);

    // Output is 8bit depth, RGBA format.
    png_set_IHDR(png,
                 info,
                 myImg->width, myImg->height,
                 8,
                 PNG_COLOR_TYPE_RGBA,
                 PNG_INTERLACE_NONE,
                 PNG_COMPRESSION_TYPE_DEFAULT,
                 PNG_FILTER_TYPE_DEFAULT
                 );

    png_write_info(png, info);

    // To remove the alpha channel for PNG_COLOR_TYPE_RGB format,
    // Use png_set_filler().
    //png_set_filler(png, 0, PNG_FILLER_AFTER);

    if (!myImg->data) abort();

    unsigned char* row_pointers[myImg->height];
    for(int y = 0; y < myImg->height; y++) {
        row_pointers[y] =
            (unsigned char*)malloc(4*myImg->width);
        unsigned char* row = row_pointers[y];
        for(int x = 0; x < myImg->width; x++) {
            unsigned char * px = &(row[x * 4]);
            px[0]=px[1]=px[2]=myImg->data[x+y*myImg->width];
            px[3]=255;
        }
    }
    png_write_image(png, row_pointers);

    for(int y = 0; y < myImg->height; y++) {
        free(row_pointers[y]);
    }
    png_write_end(png, NULL);
    fclose(fp);
    png_destroy_write_struct(&png, &info);
}
```

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.

```
                            {
    Img * answer = newImgColor(w,w,255);
    float step=w/9.;
    float stepOverTwo=step/2.;
    float stepOverThree=step/2;
    float stepOverThreeSq=stepOverThree*stepOverThree;
    for (int x=0;x<w;++x) {
        for (int y=0;y<w;++y) {
            float xr = round((x-stepOverTwo)/step)*step+stepOverTwo;
            float yr = round((y-stepOverTwo)/step)*step+stepOverTwo;
            float d = ((x-xr)*(x-xr)+(y-yr)*(y-yr))/stepOverThreeSq;
            if (d<1) {
                answer->data[x+w*y]=INBYTE((int)(d*255));
            }
        }
    }
    return answer;
}
```

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.

```
                                                 {
    Img * answer = (Img *)malloc(sizeof(struct img));
    answer->width=w;
    answer->height=h;
    answer->data=
        (unsigned char*)malloc(answer->height * answer->width);
    memset(answer->data,c,answer->height * answer->width);
    return answer;
}
```

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.

```
                           {
    Img * answer = (Img *) malloc(sizeof(struct img));

    answer->width       = myImg->width       ;
    answer->height      = myImg->height      ;
    answer->data = (unsigned char*)malloc(sizeof(char) *
                                          answer->height*answer->width);
    memcpy(answer->data,
           myImg->data,
           answer->width*answer->height);
    return answer;
}
```

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.

```
                                    {
    Img * answer = newImgColor(w,w,255);
    int m=w/2;
    int count=0;
    while (count<t && count<m) {
        if (s<w-1 && s>0) {
            for (int y=m-s/2;y<m-s/2+s;++y) {
                answer->data[m+count+w*y]=0;
            }
            for (int x=m-s/2;x<m-s/2+s;++x) {
                answer->data[x+w*(m+count)]=0;
            }
            for (int y=m-s/2;y<m-s/2+s;++y) {
                answer->data[m-count+w*y]=0;
            }
            for (int x=m-s/2;x<m-s/2+s;++x) {
                answer->data[x+w*(m-count)]=0;
            }
        }
        count++;
    }
    return answer;
}
```

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

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

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.

```
                                    {
    // create a w by 1 white image
    Img * answer = newImgColor(w,1,255);
    // space between dots is w/N
    float step=((float)w)/((float)N);
    // we begin at step/2 from the border
    float stepOverTwo=step/2.;
    float stepOverFour=step/4;
    float stepOverFourSq=stepOverFour*stepOverFour;
    for (int x=0;x<w;++x) {
        // horizontal position of the closest black dot
        float xr = round((x-stepOverTwo)/step)*step+stepOverTwo;
        // compute distance to the closest black dot
        float d = fabs(xr-x);
        float dmax=3;
        if (d<dmax) {
            // we are close to a black dot so we darken the pixel
            answer->data[x]=INBYTE((int)(d*255/dmax));
        }
    }
    return answer;
}
```

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.

```
                                 {
    int l = strlen(filename);
    if ( (l>4 &&
          filename[l-4]=='.' &&
          (filename[l-3]=='j'|| filename[l-3]=='J')&&
          (filename[l-2]=='p'|| filename[l-2]=='P')&&
          (filename[l-1]=='g'|| filename[l-1]=='G'))
         ||
         (l>5 &&
          filename[l-5]=='.' &&
          (filename[l-4]=='j'|| filename[l-4]=='J')&&
          (filename[l-3]=='p'|| filename[l-3]=='P')&&
          (filename[l-2]=='e'|| filename[l-2]=='E')&&
          (filename[l-1]=='g'|| filename[l-1]=='G'))) {
        return newImgReadJpeg(filename);
    }
    // otherwise we assume it is a png
    return newImgReadPng(filename);
}
```

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.

```
                                       {
    Img * answer = newImgColor(w,w,255);
    int m=w/2;
    int count=0;
    while (count<t) {
        if (s<w-1 && s>0) {
            for (int y=m-s/2;y<m-s/2+s;++y) {
                answer->data[m-s/2+w*y]=0;
            }
            for (int y=m-s/2;y<m-s/2+s;++y) {
                answer->data[m-s/2+s-1+w*y]=0;
            }
            for (int x=m-s/2;x<m-s/2+s;++x) {
                answer->data[x+w*(m-s/2)]=0;
            }
            for (int x=m-s/2;x<m-s/2+s;++x) {
                answer->data[x+w*(m-s/2+s-1)]=0;
            }
        }
        count++;
        s--;
    }
    return answer;
}
```

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.

```
                           {
    int w=212;
    int h=212;
    Img * answer = newImgColor(w,h,255);
    int xinit=224/2-sz/2;
    int yinit=xinit;
    int step=sz/9;
    for (int x=0;x<10;++x) {
        int xx=xinit+x*step;
        for (int yy=yinit;yy<yinit+step*9-1;++yy) {
            if (xx>=0 && yy>=0 && xx<w && yy<h)
                answer->data[xx+w*yy]=0;
        }
    }
    answer->data[1+xinit+9*step+w*yinit]=0;
    answer->data[xinit+w*(yinit+1+9*step)]=0;
    for (int y=0;y<10;++y) {
        int yy=yinit+y*step;
        for (int xx=xinit;xx<xinit+step*9-1;++xx) {
            if (xx>=0 && yy>=0 && xx<w && yy<h)
                answer->data[xx+w*yy]=0;
        }
    }
    for (int x=0;x<=3;++x) {
        int xx=xinit+x*step*3;
        for (int yy=yinit;yy<yinit+step*9-1;++yy) {
            if (xx>=0 && yy>=0 && xx<w && yy<h)
                answer->data[1+xx+w*(yy+1)]=0;
        }
    }
    for (int y=0;y<=3;++y) {
        int yy=yinit+y*step*3;
        for (int xx=xinit;xx<xinit+step*9-1;++xx) {
            if (xx>=0 && yy>=0 && xx<w && yy<h)
                answer->data[1+xx+w*(yy+1)]=0;
        }
    }
    return answer;
}
```

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.

```
                                     {
    Img * answer = newImgColor(s,s,255);
    int m=s/2;
    for (int x=m-w/2;x<m-w/2+w;++x) {
        for (int y=0;y<s;++y) {
            answer->data[x+s*y]=0;
        }
    }
    return answer;
}
```

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.

```
                                                    {
    Img * answer = newImgColor(sx,sy,255);
    int m=sx/2;
    if (m<w) {
        ERROR("Wrong size.","");
    }
    // put some grey arond the bar so that the
    // equivalent filter is 0
    for (int x=m-w;x<=m+w;++x) {
        for (int y=0;y<sy;++y) {
            answer->data[x+sx*y]=128;
        }
    }
    for (int x=m-w/2;x<m-w/2+w;++x) {
        for (int y=0;y<sy;++y) {
            answer->data[x+sx*y]=0;
        }
    }
    return answer;
}
```

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

4.18 img.h

Go to the documentation of this file.

```
#ifndef IMG_H
#define IMG_H

#include "util.h"

struct img {
    int width;
    int height;
    unsigned char * data;
};

typedef struct img Img;

/* external data types */
typedef struct filter Filter;

Img * newImgFromArray(int w, int h, unsigned char *s);
Img * newImgColor(int w, int h, unsigned char c);
Img * newImgRead(char *filename);
Img * newImgCopy(Img*myImg);
Img * newImg9By9Dots(int w);
Img * newImgNDotsHori(int N,int w);
Img * newImgSudoku(int sz);
Img * newImgSquare(int s,int w,int t);
Img * newImgCross(int s,int w,int t);
Img * newImgVerticalBar(int s,int w);
Img * newImgVerticalBarInRect(int sx, int sy, int w);
void deleteImg(Img*myImg);
Img * imgDivideByTwo(Img* img);
unsigned char imgGetVal(Img*p,int x, int y);
void imgPrint(Img*myImg);
void imgWrite(Img*myImg,char *filename);
Img * imgInvert(Img*in);
Img * imgFlattenContrast(Img*in);
Img * imgRaiseContrast(Img*in);
Img * imgBlur(Img*in,int radius);
Img * imgMake3dEffect(Img*in);
Img * imgLuminosityScale(Img*in);
Img * imgConvolution(Img*in,Filter*filter);
Img * imgConvolutionDiff(Img*in,Filter*filter);
Img * imgConvolutionSameSize(Img*in,Filter*filter);
Img * imgConvolutionSameSizeDiff(Img*in,Filter*filter);
Img * imgDownSampleAvg(Img* img,int poolsize,int stride);
Img * imgDownSampleMax(Img* img,int poolsize,int stride);
void imgDrawRect(Img*myImg,int xmin,int ymin,int xmax,int ymax);
unsigned char imgScalar(Img*,Img*,int,int);
Img * imgRotate(Img* img,int deg);
Img * imgRotate90(Img* img);
Img * imgScale(Img*in,int s);
Img * imgExtract(Img*myImg,int xmin,int ymin,int xmax,int ymax);
int imgGetWeight(Img*in);
int imgMain(int,char**);

#endif

```

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.

```
                                 {
    if (ifa==NULL) return;
    for (int i=0;i<ifa->count;++i) {
        deleteImg(ifa->imgs[i]);
    }
    memset(ifa->imgs,0,sizeof(Img*)*ifa->count);
    free(ifa->imgs);
    memset(ifa,0,sizeof(ImgFam));
    free(ifa);
}
```

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.

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

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.

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

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.

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

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

4.19.1.5 imgFamPrint()

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

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

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

```
                                   {
    char s[99];
    int count=0;
    int found=1;
    while (found) {
        snprintf(s,99,"%s_%d.png",basename,count);
        FILE * f = fopen(s,"r");
        if (f!=NULL) {
            fclose(f);
            ++count;
        } else  {
            found=0;
        }
    }
    ImgFam * answer = newImgFam(count);
    for (int i=0;i<answer->count;++i) {
        snprintf(s,99,"%s_%d.png",basename,i);
        imgFamSetImg(answer,i,newImgRead(s));
    }
    return answer;
}
```

References imgfam::count, imgFamSetImg(), newImgFam(), and newImgRead().

4.19.1.7 imgFamScalar()

```
Img * imgFamScalar (
              ImgFam * if1,
              FilterFam * if2 )
```

```
                                              {
    if (if1->count!=if2->count) {
        ERROR("Wrong number of images.","");
    }
    if (if1->imgs[0]->width<if2->filters[0]->img->width) {
        ERROR("Wrong size of images.","");
    }
    if (if1->imgs[0]->height<if2->filters[0]->img->height) {
        ERROR("Wrong size of images.","");
    }
    int w = if1->imgs[0]->width-if2->filters[0]->img->width+1;
    int h = if1->imgs[0]->height-if2->filters[0]->img->height+1;
    Img * answer = newImgColor(w,h,0);
    for (int xoffset=0;xoffset<w;++xoffset) {
        for (int yoffset=0;yoffset<h;++yoffset) {
            int s=0;
            for (int i=0;i<if1->count;++i) {
                s+= imgScalar(if1->imgs[i],if2->filters[i]->img,xoffset,yoffset);
            }
            answer->data[xoffset+yoffset*w]=INBYTE(s/if1->count);
        }
    }
    return answer;
}
```

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.

```
                                               {
    if (i<0 && i>=imgFam->count)
        ERROR("out of bounds.","");
    imgFam->imgs[i]=img;
}
```

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

```
                                         {
      char s[99];
      for (int i=0;i<imgFam->count;++i) {
          snprintf(s,99,"%s_%d.png",basename,i);
          imgWrite(imgFam->imgs[i],s);
       }
}
```

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.

```
                          {
      ImgFam * answer = (ImgFam*)malloc(sizeof(struct imgfam));
      answer->count=c;
      answer->imgs=(Img**)malloc(sizeof(Img**)*c);
      memset(answer->imgs,0,sizeof(Img*)*c);
      return answer;
}
```

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.

```
                                  {
    if (ifa==NULL) return;
    for (int i=0;i<ifa->count;++i) {
        deleteImg(ifa->imgs[i]);
    }
    memset(ifa->imgs,0,sizeof(Img*)*ifa->count);
    free(ifa->imgs);
    memset(ifa,0,sizeof(ImgFam));
    free(ifa);
}
```

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.

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

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.

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

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.

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

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

4.20.2.6 imgFamPrint()

```
void imgFamPrint (
               ImgFam * ifa )
                              {
     for (int i=0;i<ifa->count;++i) {
         printf("=-=-=-=-=-=-=-=-=-=-=-=-=-= %d/%d\n",i,ifa->count);
         imgPrint(ifa->imgs[i]);
     }
}
```

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

```
                                  {
    char s[99];
    int count=0;
    int found=1;
    while (found) {
        snprintf(s,99,"%s_%d.png",basename,count);
        FILE * f = fopen(s,"r");
        if (f!=NULL) {
            fclose(f);
            ++count;
        } else  {
            found=0;
        }
    }
    ImgFam * answer = newImgFam(count);
    for (int i=0;i<answer->count;++i) {
        snprintf(s,99,"%s_%d.png",basename,i);
        imgFamSetImg(answer,i,newImgRead(s));
    }
    return answer;
}
```

References imgfam::count, imgFamSetImg(), newImgFam(), and newImgRead().

4.20.2.8 imgFamScalar()

```
Img * imgFamScalar (
              ImgFam * if1,
              FilterFam * if2 )
```

```
                                             {
    if (if1->count!=if2->count) {
        ERROR("Wrong number of images.","");
    }
    if (if1->imgs[0]->width<if2->filters[0]->img->width) {
        ERROR("Wrong size of images.","");
    }
    if (if1->imgs[0]->height<if2->filters[0]->img->height) {
        ERROR("Wrong size of images.","");
    }
    int w = if1->imgs[0]->width-if2->filters[0]->img->width+1;
    int h = if1->imgs[0]->height-if2->filters[0]->img->height+1;
    Img * answer = newImgColor(w,h,0);
    for (int xoffset=0;xoffset<w;++xoffset) {
        for (int yoffset=0;yoffset<h;++yoffset) {
            int s=0;
            for (int i=0;i<if1->count;++i) {
                s+= imgScalar(if1->imgs[i],if2->filters[i]->img,xoffset,yoffset);
            }
            answer->data[xoffset+yoffset*w]=INBYTE(s/if1->count);
        }
    }
    return answer;
}
```

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.

```
                                                {
    if (i<0 && i>=imgFam->count)
        ERROR("out of bounds.","");
    imgFam->imgs[i]=img;
}
```

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

```
                                           {
    char s[99];
    for (int i=0;i<imgFam->count;++i) {
         snprintf(s,99,"%s_%d.png",basename,i);
         imgWrite(imgFam->imgs[i],s);
     }
}
```

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.

```
                         {
    ImgFam * answer = (ImgFam*)malloc(sizeof(struct imgfam));
    answer->count=c;
    answer->imgs=(Img**)malloc(sizeof(Img**)*c);
    memset(answer->imgs,0,sizeof(Img*)*c);
    return answer;
}
```

References imgfam::count, and imgfam::imgs.

4.21 imgfam.h

Go to the documentation of this file.

```
#ifndef IMGFAM_H
#define IMGFAM_H

#include "img.h"

struct imgfam {
    int count;
    Img ** imgs;
};

typedef struct imgfam ImgFam;

// external types
typedef struct filterfam FilterFam;

ImgFam * newImgFam(int);
void imgFamSetImg(ImgFam*,int,Img*);
ImgFam * imgFamApplyConvolution(ImgFam*,Filter*);
ImgFam * imgFamDownSampleMax(ImgFam*,int,int);
ImgFam * imgFamDownSampleAvg(ImgFam*,int,int);
void imgFamWrite(ImgFam*,char*);
ImgFam * imgFamRead(char*);
Img* imgFamScalar(ImgFam*,FilterFam*);
void imgFamPrint(ImgFam*);
ImgFam* imgFamLuminosityScale(ImgFam*in);
void deleteImgFam(ImgFam *);

#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.

```
                            {
    free(l->name);
    free(l->convFilterLoc);
    deleteFilterFam(l->convFilter);
    memset(l,0,sizeof(Layer));
    free(l);
}
```

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

```
                                   {
    return filterFamCount(convFilterLoc);
}
```

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

```
                                   {
    ImgFam * intermediateOutput =
        filterFamApplyConvolution(l->convFilter,i);
    ImgFam * maxPoolOutput=imgFamDownSampleMax(intermediateOutput,
                                             l->downSamplePoolSize,
                                             l->downSampleStride);
    deleteImgFam(intermediateOutput);
    return maxPoolOutput;
}
```

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

```
                                    {
    ImgFam * intermediateOutput =
        filterFamApplyConvolutionOnFam(l->convFilter,i);
    ImgFam * maxPoolOutput=imgFamDownSampleMax(intermediateOutput,
                                               l->downSamplePoolSize,
                                               l->downSampleStride);
    return maxPoolOutput;
}
```

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.

```
{
    Layer * answer = (Layer*)malloc(sizeof(struct layer));
    answer->name = stringCopy(name);
    answer->convFilterLoc = stringCopy(convFilterLoc);
    answer->convFilter=filterFamRead(convFilterLoc);
    answer->downSamplePoolSize=downSamplePoolSize;
    answer->downSampleStride=downSampleStride;
    return answer;
}
```

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.

```
                                {
    free(l->name);
    free(l->convFilterLoc);
    deleteFilterFam(l->convFilter);
    memset(l,0,sizeof(Layer));
    free(l);
}
```

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

```
                                         {
    return filterFamCount(convFilterLoc);
}
```

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

```
                                     {
    ImgFam * intermediateOutput =
        filterFamApplyConvolution(l->convFilter,i);
    ImgFam * maxPoolOutput=imgFamDownSampleMax(intermediateOutput,
                                               l->downSamplePoolSize,
                                               l->downSampleStride);
    deleteImgFam(intermediateOutput);
    return maxPoolOutput;
}
```

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

```
                                           {
    ImgFam * intermediateOutput =
        filterFamApplyConvolutionOnFam(l->convFilter,i);
    ImgFam * maxPoolOutput=imgFamDownSampleMax(intermediateOutput,
                                               l->downSamplePoolSize,
                                               l->downSampleStride);
    return maxPoolOutput;
}
```

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.

```
{
    Layer * answer = (Layer*)malloc(sizeof(struct layer));
    answer->name = stringCopy(name);
    answer->convFilterLoc = stringCopy(convFilterLoc);
    answer->convFilter=filterFamRead(convFilterLoc);
    answer->downSamplePoolSize=downSamplePoolSize;
    answer->downSampleStride=downSampleStride;
    return answer;
}
```

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.

```
#ifndef LAYER_H
#define LAYER_H

#include "util.h"

// external type
typedef struct filterfam FilterFam;
typedef struct imgfam ImgFam;
typedef struct img Img;

struct layer {
    char * name;
    char * convFilterLoc;
    FilterFam * convFilter;
    int downSamplePoolSize;
    int downSampleStride;
};

typedef struct layer Layer;

Layer * newLayer(char * name,
                 char * convFilterLoc,
                 int downSamplePoolSize,
                 int downSampleStride);
void deleteLayer(Layer*);
ImgFam * layerPassImg(Layer*,Img*);
ImgFam * layerPassImgFam(Layer*,ImgFam*);
int layerCount(char * convFilterLoc);

#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 ( )
        {
int errorCount=0,caseCount=0;
init_digits();
for (int f=1;f<31;++f) {
    // using font f
     int fontError=0;
     for (int d=1;d<10;++d) {
         // trying to guess digit d
         for (int i=1;i<10;++i) {
             // check if i is the highest
             if (d!=i) {
                 if (digits[f][d][i]>=digits[f][d][d]) {
                     // another one is the highest
                     errorCount++;
                     fontError++;
                 }
             }
             caseCount++;
         }
     }
     printf("font %d %s: \t%d errors\n",f,fontname[f],fontError);
 }
 printf("%d/%d\n",errorCount,caseCount);
 return 0;
}
```

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.

```
                                                                      {
    int xstep=(xmax-xmin)/9;
    int ystep=(ymax-ymin)/9;
    int tenPercent=xstep/10;
    for (int i=0;i<9;++i) {
        for (int j=0;j<9;++j) {
            Img * aDigit = imgExtract(myImg,
                                      xmin+i*xstep+tenPercent,
                                      ymin+j*ystep+tenPercent,
                                      xmin+(i+1)*xstep-tenPercent,
                                      ymin+(j+1)*ystep-tenPercent);
            char s [99];
            snprintf(s,99,"extracted_digit_%d_%d.png",i,j);
            imgWrite(aDigit,s);
            deleteImg(aDigit);
        }
    }
}
```

References deleteImg(), imgExtract(), and imgWrite().

4.26.1.2 main()

```
int main (
               int argc,
               char ** argv )
                                {
    char * progName = basename(argv[0]);
    if (strcmp(progName,"img")==0) {
        return imgMain(argc,argv);
    } else if (strcmp(progName,"digits")==0) {
        return digitsMain(argc,argv);
    } else if (strcmp(progName,"grid")==0) {
        return gridMain(argc,argv);
    }

    for (int j=1;j<argc;++j) {
        if (strcmp(argv[j],"-h")==0 || strcmp(argv[j],"--help")==0) {
            usage(stdout,argv[0]);
            exit(0);
        }
    }
    if(argc < 2) {
        usage(stderr,argv[0]);
        ERROR("At least 1 argument expected, the picture to read","");
    }
    /* read input image */
```

```
   Img * rawInputImage=newImgRead(argv[1]);
   Img * goodContastImage=imgLuminosityScale(rawInputImage);
   Img * inverseImage = imgInvert(goodContastImage);
   deleteImg(goodContastImage);
   int scaleFactor=0;
   while (inverseImage->width>400 && inverseImage->height>400) {
       Img * i = imgDivideByTwo(inverseImage);
       deleteImg(inverseImage);
       inverseImage=i;
       scaleFactor++;
   }
   int xmin,ymin,xmax,ymax;
    // locate the grid
    gridLocate(inverseImage,&xmin,&ymin,&xmax,&ymax);
    HERE("Found sudoku grid :");
    printf("%d %d %d %d\n",xmin,ymin,xmax,ymax);

    imgDrawRect(inverseImage,xmin,ymin,xmax,ymax);
    imgWrite(inverseImage,"afterstd.png");
    // draw a square around the grid found in the
    // original picture
    imgDrawRect(rawInputImage,
                xmin«scaleFactor,
                ymin«scaleFactor,
                xmax«scaleFactor,
                ymax«scaleFactor);
    // write the image
    imgWrite(rawInputImage,"out.png");
    HERE("grid detection in out.png");
    cnnExtractDigits(rawInputImage,
                     xmin«scaleFactor,
                     ymin«scaleFactor,
                     xmax«scaleFactor,
                     ymax«scaleFactor);
    // free allocated memory
    deleteImg(inverseImage);
    deleteImg(rawInputImage);
    return 0;
}
```

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].

```
                             {
   char * bname=basename(name);
   fprintf(f,"This is %s version %s on archictecture %s.\n",
           bname, VERSION,CFG_UNAME);
   fprintf(f,"Usage:\n");
   fprintf(f,"    %s <input-file> | <option> \n", bname);
   fprintf(f,"Solves a Sudoku grid given a png or jpeg image as input.\n");
   fprintf(f,"\n");
   fprintf(f,"Where option is one of:\n");
   fprintf(f,"    [-h|--help] :\n");
   fprintf(f,"         Displays this help message and leaves.\n");
   fprintf(f,"\n");
   fprintf(f,"%s comes with 3 friend tools:\n",bname);
   fprintf(f,"    img:\n");
```

```
    fprintf(f,"        Manipulates images. For help type:\n");
    fprintf(f,"        %s/bin/img --help\n",CFG_DEFAULT_PREFIX);
    fprintf(f,"    digits:\n");
    fprintf(f,"        Recognize digits. For help type:\n");
    fprintf(f,"        %s/bin/digits --help\n",CFG_DEFAULT_PREFIX);
    fprintf(f,"    grid:\n");
    fprintf(f,"        Recognize a sudoku grid. For help type:\n");
    fprintf(f,"        %s/bin/grid --help\n",CFG_DEFAULT_PREFIX);
    fprintf(f,"\n");
    fprintf(f,"%s was compiled on %s. Git repo used:\n",bname,__DATE__);
    fprintf(f,"  %s\n",CFG_GIT_REPO);
    fprintf(f,"with hash:\n");
    fprintf(f,"  %s\n",CFG_GIT_FHASH);
}
```

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

```
                                    {
    if (n<10)
        ERROR("n should be at least 10","");
    name[0]=’/’;
    name[1]=’t’;
    name[2]=’m’;
    name[3]=’p’;
    name[4]=’/’;
    writeRandomName(&name[5],n);
    struct stat st = {0};
    if (stat(name, &st) == -1) {
        mkdir(name, 0700);
    } else {
        ERROR("Unlucky name already exists: ",name);
    }
}
```

References ERROR, and writeRandomName().

4.27.1.2 seedRandomNumberGeneratorIfNeeded()

```
void seedRandomNumberGeneratorIfNeeded ( )
```

seed random number generator if not alread done.

```
                                          {
    if (!_seedRandomNumberGenerator) {
        srand ( time(NULL) );
        _seedRandomNumberGenerator=1;
    }
}
```

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

```
                                    {
```

```
    int l = strlen(a)+strlen(b)+1;
    char * answer = (char*)malloc(l);
    snprintf(answer,l,"%s%s",a,b);
    return answer;
}
```

4.27.1.4 stringCopy()

```
char * stringCopy (
              char * s )
```

allocates a new string.

Parameters

s	string to copy.

```
                            {
    int l = strlen(s)+1;
    char * answer = (char*)malloc(l);
    memcpy(answer,s,l);
    return answer;
}
```

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.

```
                                      {
    int i=0;
    seedRandomNumberGeneratorIfNeeded();
    for (i=0;i<n;++i) {
        unsigned char j=rand()%52;
        name[i]=(j<26)?'A'+j:'a'+(j-26);
    }
    name[n]=0;
    if (n>4) {
        name[0]='c';
        name[1]='n';
        name[2]='n';
        name[3]='_';
    }
}
```

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

```
                                {
    if (n<10)
        ERROR("n should be at least 10","");
    name[0]='/';
    name[1]='t';
    name[2]='m';
    name[3]='p';
    name[4]='/';
    writeRandomName(&name[5],n);
    struct stat st = {0};
    if (stat(name, &st) == -1) {
        mkdir(name, 0700);
    } else {
        ERROR("Unlucky name already exists: ",name);
    }
}
```

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

```
                                {
    int l = strlen(a)+strlen(b)+1;
    char * answer = (char*)malloc(l);
    snprintf(answer,l,"%s%s",a,b);
    return answer;
}
```

4.28.2.3 stringCopy()

```
char * stringCopy (
            char * s )
```

allocates a new string.

Parameters

s	string to copy.

```
                            {
    int l = strlen(s)+1;
    char * answer = (char*)malloc(l);
    memcpy(answer,s,l);
    return answer;
}
```

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.

```
                                      {
    int i=0;
    seedRandomNumberGeneratorIfNeeded();
    for (i=0;i<n;++i) {
        unsigned char j=rand()%52;
        name[i]=(j<26)?'A'+j:'a'+(j-26);
    }
    name[n]=0;
    if (n>4) {
        name[0]='c';
        name[1]='n';
        name[2]='n';
        name[3]='_';
    }
}
```

References seedRandomNumberGeneratorIfNeeded().

4.29 util.h

Go to the documentation of this file.

```
#ifndef UTIL_H
#define UTIL_H

#include <libgen.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "config.h"

#define INBYTE(x) ((x)<0?0:(x)>255?255:(x))

#define ERROR(x,y) {fprintf(stderr,"\033[31m%s:%d: %s%s\033[m\n",        \
                            __FILE__,                                    \
                            __LINE__,                                    \
                            x,y); exit(1);}
#define WARNING(x,y) {fprintf(stderr,"\033[31m%s:%d: %s%s\033[m\n",      \
                            __FILE__,                                    \
                            __LINE__,                                    \
                            x,y);}
#define HERE(x) {printf("%s:%d: %s\n",                           \
                        __FILE__,                               \
                        __LINE__,                               \
                        x);}
#define HERED(x) {printf("%s:%d: %d\n",                          \
                         __FILE__,                                \
                         __LINE__,                                \
                         x);}
void writeRandomName(char*,int);
void createTmpDir(char*name,int n);
char * stringCopy(char *);
char * stringAdd(char *,char *);
#endif
```

Index