Biomedical Imaging Group

C Coding Guidelines

Document

Date: March 1, 1999

Author: Daniel Sage
Swiss Federal Institute of Technology - Lausanne
Biomedical Imaging Group

Address EPFL, DMT/IOA
BM-Ecublens
CH-1015 Lausanne
Switzerland

Contents

Objective

The objective of the Biomedical Imaging Group is to build an image processing software library that has the following key points:

- Code ready to be distributed
- Consistence and readability
- Maintainable by the staff, other than the author
- Multi-platforms
- Multi-programmers environment

The way to achieve this goal is to have some guidelines to help programming and some directives to approve the code It is not exhaustive but gives some practical rules that can be applied immediately.

Rules of approval

To be approved, a routine should satisfy five criteria.

Approval 1 *General Interest*

A function of the public library should have at least one of the following points:

- an interest for the rest of the group;
- an interest to be distributed outside of the group;
- prove a result or implement an algorithm that is published.

Approval 2 *Follow the programming requirements*

There are three levels for programming rules:

- Requirement: It should be absolutely respected, no exception allowed.
- Recommendation: Most of the time, it should be respected, but exceptions are allowed. Every time a rule is broken, this must be justified and documented.
- Advice: Propositions and suggestions.

Approval 3 *Follow the documentation requirements*

In the same spirit, there will be a guideline for the documentation.

Approval 4 *Stabilized*

Stabilized state, no future evolution or parameters to add.

Approval 5 *Free of bugs!*

The routine to be integrated is free of bugs, or at least there is no known bug. There is at least one described and reproducible test or demonstration. So, the final code could be associated with a set of data to test it.

1. Management of dependencies

Required

No explicit path

Use the option of the compiler to give the include path.

KO

```
#include "../include/spline.h"

fp = fopen("/disk/data/lena.tiff");
```

Required

Always a header file

A source file (.c) is always associated to a header file (.h).

Define the external functions in the header file.

Recommended

Include

To have a maximal visibility, the include directives are put at the top of the files, first the C standard inclusion, then the specific inclusion.

OK

```
#include <stdio.h>
#include "getput.h"
```

Required

No global variables

No global variables to a project.

No global variables to a module. This allows to extract routines one by one.

The variables are always local to a function.

Required

Static or external functions

Local functions are always prefixed with `static` keyword in the declaration and in the definition.

Public functions are always prefixed with `extern` keyword in the declaration and in the definition.

Recommended

User-defined types

The user-defined types (structure, enum, union) should be put into a separate file (`*.h`). The filename and the defined type are the same.

OK

```
/* ------------------------------------------------------------
    Filename: tboundaryconditions.h
    …
------------------------------------------------------------*/
enum    TBoundaryConditions
{
        Mirror,
        Periodic,
};
```

Required

No cascade of include

No cascade of include, i.e., no `#include` in a header file.

Required

Never use `#include "*.c"`

➢ *Advice*

Few public functions

A module has only few public functions (once it is good), except for files that contain very short functions.

When there is just one public function in a module, the name of the function is the same than the filename.

The public functions are prefixed by the file name.

OK

```
/* ---------------------------------------
    Filename: setcolortable.c
    …
----------------------------------------*/
#include "setcolortable.h"
extern int SetColorTableGray( void)
{
...
}
extern int SetColorTableRainbow( void)
{
...
}
```

2. Naming convention

➢ *Advice*

Meaningful names

A variable or routine name should fully and accurately describe the entity. Meaningful names give a self-documentation.

Recommended

English

All names, variable, routines and comments are in English.

➢ *Advice*

Function naming

If possible, the function should begin by a verb describing the action.

OK

```
GetRow()
IsLetter()
RotateImage()
InsertLine()
```

Required

File naming

The length of the file names is limited to 32 characters, including the extension.

The current extensions are:
- .c for the source file
- .h for the header file
- .txt for a documentation file containing only text (ASCII)
- .doc for a documentation written with a Word Processor
- .html for a documentation file in HTML

The mandatory header file ".h" file matches its ".c" file.

A user-defined ".h" file for types (structure, enum, union) has the same name than the name of the type, except that it is all lowercase.

OK

```
/*    Filename: boundarycondition.h    */
…
enum TboundaryCondition {
         Mirror,
         Periodic
};
…
```

Recommended

Typing convention

		Good examples

Local variables	First letter of words in uppercase or Lowercase letters	`i`
		`sum2`
		`NumIteration`

Routine names	First letter of words in uppercase	`DisplayMessage()`
Types	First letter of words in uppercase and a prefix **T**	`TboundaryCondition`
Fields of structure	First letter of words in uppercase	`ColorRed`
Values of enumeration	First letter of words in uppercase	`ColorRed`
Defines	Uppercase letters	`COLORRED` `COLOR_RED`
Macros	Uppercase letters	`MAX`

Required

Typing convention

Filenames	Lowercase letters	`getimage.c`

Recommended

Acronyms

Acronyms are tolerated for routines and variables if they are common and well known. They are typed in uppercase mixed with words. Less common acronyms are forbidden.

OK

```
ComputeFFT()
```

Recommended **Abbreviation**

Choose common and well-known abbreviations like:

OK
```
Num, Min, Max, Sum, Vol …
```

The non-common abbreviation should be explained in an abbreviation tables in the begin of the file.

OK
```
/*    Abbreviation Table                          */
/*        Imgr    Image Plane Red                 */
/*        H       Filter lowpass                 */
```

Do not use phonetic abbreviations.

KO
```
Float2Double
```

OK
```
FloatToDouble
```

➢ *Advice* **Length of the names**

The optimal length of a name is from 6 to 24 characters.

One rule of thumb: "*The larger the scope, the longer the name"*.

Recommended **Variant naming**

The qualifying term of a different variant should be a suffix.

OK
```
AllocateLineFloat
AllocateLineDouble
AllocateLineShort
```

KO
```
FloatAllocateLine
DoubleAllocateLine
ShortAllocateLine
```

3. C coding

Required **Strict ANSI C**

Use the option of the compiler to guarantee that the code is compiled ANSI.

Required **No warnings allowed**

The code should generate no warnings. All issues must be addressed such an unused arguments, unused variables, missing declarations, etc.

Recommended **Compilation options**

Recommended compilation options for gcc (Unix)

-ansi –pedantic –Wall -Wcast-qual -Wcast-align -Wwrite-strings -Wstrict-prototypes -Wmissing-prototypes -Wmissing-declarations -Wredundant-decls -Wnested-externs -Winline

Required **No old C style to pass parameters**

KO

```
static int function1()
double *px, double *py;
{
…
}
```

Required **Functions have a type**

All functions are typed, `void` if they return nothing.

Function parameters are typed, `void` if there is none.

OK

```
static void DoNothing(void)
{
…
}
```

Required **Allocated memory**

The same function that allocates variables, has to free them.

When a value is a pointer it should be compared to NULL instead of zero.

Always cast NULL.

KO

```
double *ptr;

    if( ptr == (double *)NULL) {
    …
    }
```

Required **No system call using the standard input/output**

The system calls using standard input/output are forbidden. The function printf() should be never used, prefer the functions of the toolbox, for example there is a function in the toolbox: MessageDisplay(const char *Message).

Recommended **No macros**

Avoid macros, prefer functions, even short ones.

Recommended **Don't use register**

➢ *Advice* **No numeric values**

Avoid numeric values, use symbolic values instead.

➢ *Advice* **Variables**

All local variables are defined at the beginning of the function, including loop indexes.

Remove all unused variables.

➢ *Advice* **Readability rather than speed**

Make a code easy to understand rather than a code that runs quickly, except for the basic functions or for the toolbox functions.

Recommended

Type of data

The type of the data processed is:

- **long** for integer data, if the amount of data is huge the type could be **short**;
- **double** for real data, if the amount of data is huge the type could be **float**.

All arithmetic operations and logical operations should be written with variables of the same type.

Do not mix types in expressions, logical conditions or arithmetic operations.

Avoid implicit conversion of type. Be explicit instead.

KO

```
long  kx, n;
float *p;
float a;

a = sqrt(2.0);          /* sqrt() return a double          */
                        /* and a is float                  */
…
for(kx=0;kx<n;kx) {     /* 0 is an int and kx a long */
    …
}

*p = 1.0;               /* 1. is a double and px a float   */
…
if ( n < 3)   {         /* 3 is a int and n is a long      */
    …
}
```

OK

```
long   kx, n;
float  *p;
double a;
float  b;

    a = sqrt(2.0);
    b = (float)sqrt(2.0);
    …
    for(kx=0L;kx<n;kx) {
        …
    }

    *p = 1.0F;

    if ( n < 3L) {
        …
    }
```

Recommended

Order of the parameters

Pass first the input parameters, then pass the output parameters.

OK

```
    Parameters
        In     - input  – data to be inverted
        Size   - input  – length of the input data
        Result – output – Result of the inversion

------------------------------------------------------------*/
Invert(short *In, long Size, short *Result)
```

➢ *Advice*

Special control statements

No goto, no continue.

No break without switch.

The body of the loop does not modify a loop index.

If you need a break statement in a for( ) loop, consider using a while( ) or a do while() loop instead.

Recommended

Default case in a switch

A switch has a default case.

Recommended

Define

Any define symbol should first be undefined.

OK

```
#undef SPACE
#define SPACE ((char)20)
```

Recommended

Boolean

The boolean constants TRUE and FALSE are defined in "`configs.h`", use them for the conditions and flags.

OK

```
#undef   FALSE
#define FALSE   ((int)(0 != 0))
#undef   TRUE
#define TRUE    (!FALSE)
```

Recommended

Free unused block memory

Don't keep unused block of memory; rather, deallocate them as soon as possible.

4. Image processing coding

➢ *Advice*

Data

The input of an image processing function is a volume in **short** or **float.**

A volume is referenced by a pointer; its data are organized follow as a raster.

OK

```
/* How to access to a point in a volume   */
/* The size of the Volume is [nx,ny,nz]   */
/* The coordinate of the point is [i,j,k] */
Value = Volume[i + nx * (j + ny*k)]
```

All internal computations are in **double.** All sizes of data are in **long.**

➢ *Advice*

3D image processing

An image processing should be designed to work with 3D data from the beginning on.

➢ *Advice*

Arrays vs. pointers

When giving types to the parameters of a routine, use arrays for 1D vectors and pointers for 3D volumes.

OK

```
GetRow( float *Vol, double Row[]);
```

Recommended

Toolbox

The toolbox is a set a basic routines that are provided. If a function is available in the toolbox, use it.

5. Layout

Recommended

File organization

For a source file (*.c)

1. Prologue (file summary comment)
2. Inclusion (#include)
3. Declaration of local functions (static)
4. Definition of public functions (extern)
5. Definition of local functions (static)

There is no user-defined type section in the source file because it is put into a separate file.

For a header file (*.h)

1. Prologue
2. Define
3. Declaration of public functions (extern)

➢ *Advice*

Sorting the functions

According to the precedent rule, the first criteria to sort the functions in a file is the category – external or static. The second sorting criteria is strictly alphabetical.

➢ *Advice*

One statement per line

Put only one statement per line. There are exceptions; in some trivial statements, the layout is improved with several statements in the same line.

OK

```
switch(nb) {
    case 0:  a=0; b=1; break;
    case 1:  a=1; b=5; break;
    case 2:  a=3; b=7; break;
    default: a=0; b=0; break;
}
```

OK

```
if (a == 0L) a1 = 1.0;
if (b == 0L) b1 = 2.0;
```

➢ *Advice*

Large statement

When a statement is long, split it onto separate lines and try to maintain the logical structure of the statement. Put operators at the start of the line, if possible.

OK

```
if (    (Data.Scale == 2L)
     && (strcmp(Data.Filter, "Spline") == 0L)
     && (Data.Order == 3L) ) {
```

OK

```
ExpandImage( Data.Input, Data.NxIn, Data.Ny,
             Data.Output, Data.NxOut, Data.NyOut,
             Data.Scale);
```

➢ *Advice*

Lines

Maxi 80 characters / line.

Use the tab character rather than blanks (spaces). Tab is set to 4 characters.

Use variables to store intermediate results when there is a large formula.

Avoid stair step indentation, the embedded block depth is limited to 5.

OK

```
/* Example to limit step indentation */
for (x = 0L; x < xmax; x++)
for (y = 0L; y < ymax; y++)
for (z = 0L; z < zmax; z++) {
    if ( x > max ) {
       vol = 0.0;
    }
```

6. Error management

Consideration **Why error management?**

The goal is to report an error and to tell in which function the error happens.

Required **Graceful exit**

After an error is detected, the exit should not provoke errors in the operating system. The early exit must clearly identify what happens in which function, free all allocated memory and close all open files.

Required **System error checking**

The following errors should always be checked in all functions:
- memory allocation;
- file access;

Recommended **Computation error checking**

The following errors should be also checked in external functions:
- mathematical errors;
- validity of input parameters;
- coherence of parameters.

Required **Return status**

Functions that may produce an error must be declared with type (`int`). The returned value is either (!ERROR) when there is no error, or (ERROR) when there is one. The symbol (ERROR) is defined in "configs.h".

Alternatively, a negative error code is allowed; success is always (!ERROR), and positive values are reserved for future use.

A function that can generate system errors (such as memory-related, file-related, IO-related errors, etc.) must have as its last parameter a status with type (`int *`). The value of this status must duplicate the value returned by the function itself.

A procedure p1() that calls a function f1(..., &status) must test for (status) and must perform a graceful exit whenever an error is detected.

A function that can generate errors (e.g., math) but cannot generate system errors should be declared with type (`int`) and should return (!ERROR) or (ERROR).

Required **Check the return status**

When a function f1() returns a status as last parameter (system error) the function that calls f1() should check the return status of f1().

Deallocate memory in reverse order from allocation.

Deallocate memory as soon as possible when an error is detected.

➢ *Advice*

Flowchart

Provoke early returns when there are errors; in this way, the last statement is return (!ERROR). When there is an early return, don't forget to close every open file, to free allocated memory, to report information and to set the status variable; by this way, the exit is graceful.

OK

```
int Do(int *Status)
{
long n;
…
px=(float *)malloc((size_t)(n*(long)sizeof(float)));
if ( px == (float *)NULL) {
     WRITE_ERROR(Do, "Unable to allocate memory");
     *Status = ERROR;
     return(ERROR);
}
…
py=(float *)malloc((size_t)(n*(long)sizeof(float)));
if ( py == (float *)NULL) {
     free(px);
     WRITE_ERROR(Do, "Unable to allocate memory");
     *Status = ERROR;
     return(ERROR);
}
…
if ( delta <= 0.0) {
     free(py);
     free(px);
     WRITE_ERROR(Do, "No real roots");
     *Status = ERROR;
     return(ERROR);
}
…
ComputeSomething(px, py, Status);
if (*Status == ERROR) {
     free(py);
     free(px);
     WRITE_ERROR(Do, "Error in ComputeSomething");
     return(ERROR);
}
…
/* The following test is an advice,
   it is not an requirement */
if (ComputeSomethingElse(px, py) == ERROR) {
     free(py);
     free(px);
     WRITE_ERROR(Do, "Error in ComputeSomething");
     *Status = ERROR;
     return(ERROR);
}

free(py);
free(px);
*Status = !ERROR;
return(!ERROR);
}
```

7. Comment

Consideration

Comment

The comment is one of the elements of the software documentation.

The three contributions to the documentation are:
- self-documentation by a good naming of variables and routines;
- internal documentation: comments;
- external documentation: the documentation put into separate files.

Required

No C++ comments (//)

Advice

Inline comment

Inline comments are small comments in right of the piece of code. They can serve to describe:
- the variables and the parameters (if the name is not meaningful enough);
- a part difficult to understand (a trick of coding).

Excess of inline comment can give a harder code to read and to maintain.

OK

```
long    Nx;    /* Size of the input signal */
double  Coef;  /* First coefficient of the filter */
long    Index;
```

Recommended

File summary

Every file should begin by a summary comment block to explain what the file contains. A file summary gives also the general information, author, date and organization. Indicate clearly the origin of the code “Swiss Federal Institute of Technology Lausanne, Biomedical Imaging Group”.

OK

```
/* ------------------------------------------------------------
    Filename:
        smooth.c

    Project:
        Biomedical Imaging Library

    Purpose:
        Smooth an input data …

    Author:
        Mister Unknown
        Swiss Federal Institute of Technology - Lausanne
        Biomedical Imaging Group
        EPFL, BM-Ecublens
        CH-1015 Lausanne, Switzerland

    Date:
        Jan 1, 2000

    External functions:
        Smooth()
------------------------------------------------------------*/
```

Recommended

Function summary

Every function should have a large comment summary at the start of the function. It summarizes the purpose and describes the implementation. The input and output parameters should be clearly described. If the general information (author, date, note…) is different from the file summary, you could repeat it in the function summary.

OK

```
/* ------------------------------------------------------
    Function: SmoothGauss

    Purpose:  Apply smoothing using the Gaussian filter

    Author:   Mister Unknown

    Parameters:
        Source — input  — data to be processed
        Result — output — result of the function
        Sigma  - input  - sigma of the Gaussian
------------------------------------------------------*/
static void SmoothGauss(
                    float *Source,
                    float *Result,
                    double Sigma)
```

➢ *Advice*

Step marker

When a function is long (more than 20 lines), put some comments that play the role of step markers and give the different steps of the processing.

OK

```
    /* --- Initialization --- */
    Sigma       = 3.0;
    Lambda      = Sigma * sqrt(2.0);
    NbIteration = 4L;
    Sum         = 0.0;
    dy[0L]      = 0.0;
    dy[ny-1L]   = 0.0;

    /* --- Row processing --- */
    for ( j = 1L; j < ny-1L; j++)
        dy[j] = dx[j+1L] — dx[j-1L];

    /* --- Best filter ---*/
    if (Filter == BESTONE) {
        for ( j = 0L; j < ny; j++)
            Sum = Sum + dx[j];
    }
    /* --- Other filters ---*/
    else {
        Sum = dx[0];
    }
```

References

L. W. Cannon & Co., "Recommended C style and coding standards", Indian Hill C Style and Coding Standards paper, 1995.

Steve McConnell, "Code complete, a practical handbook of software construction", Microsoft Press, 1993.

Roedy Green , "How To Write Unmaintainable Code", http://mindprod.com/unmain.html, Canadian Mind Products, January 1999.

Rodolphe Nicolaï, "Règles de programmation applicables au langage C", http://www.planete.net/~rnicolai/codage_c.html, 1997.

Example

The following listings show the implementation of a simple routine named FlipVolume(). This example includes a user-defined type put into a separate file (tflipenum.h), the header file (flipvolume.h) and the source file (flipvolume.c).

This source file is given in three versions of style coding:

- Bad version
- Average version
- Good version

```
#ifdef BAD
/* @ this program is fully functional; however... @ */
/* @ no meta information @ */
#include     <stdio.h>
#include     <stddef.h>               /* @ stddef not necessary @ */
#include     <stdlib.h>               /* @ stdlib not necessary @ */
#include     "configs.h"
#include     "getput.h"
#include     "tflipvolume.h"
#include     "flipvolume.h"

static void FlipIntensity(float *Volume, long Nx, long Ny, long Nz);
static void FlipLine (double *Line, long N);
static   voidFlipLine ( double        Line[],long N  ) {
/* @ wrong place for procedure definition @ */
/* @ declaration *Line and definition Line[] do not match @ */
/* @ deficient stylistic coherence (declaration/definition) in the use of space and tab @ */
/* @ trailing spaces at the end of the definition line @ */
/* @ sequence <space><tab> is to be avoided @ */
    double       Swap;
    long     K;
    K = 0L;
    while (K < --N) {
        Swap = Line[K];  Line[K++] = Line[N];
/* @ multistatement line @ */
        Line[N] = Swap;
}}
/* @ indentation @ */

/*.....................................................................
    Purpose: To flip a (float)volume around one of the four main axis
                There are three axis of space and one of intensity
.....................................................................*/
/* @ insufficient description of the routine @ */
extern   int     FlipVolume (
                        float        *Volume,
                        long     Nx,
                        long     Ny,
                        long     Nz,
                        enum TflipEnum
                                     FlipAxis,
                        int          *Status) {

    double       *Line;
    long     i, j, k;
/* @ unused variables @ */
    int          Kx, Ky, Kz;
/* @ type int has not enough capacity for indexing @ */
/* @ additional problems arise when mixing long and int (e.g., see (2)) @ */

    switch (FlipAxis) {
        case FlipAxisX:
            AllocateLineDouble(&Line, Nx, Status);
/* @ missing error management @ */
            Kx = 0L;
/* @ (1) invalid mixing of types: assignment of a long to an int @ */
            for (Kz = 0L; (Kz < Nz); Kz++)
            for (Ky = 0L; (Ky < Ny); Ky++) {
                GetxFloatToDouble(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nx);
/* @ (2) invalid mixing of types: assignment of an int to a long @ */
/* @ this is the complementary problem to (1) @ */
                    FlipLine(Line, Nx);
                        PutxDoubleToFloat(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nx);
                }
/* @ incoherent indentation @ */
            FreeLineDouble(&Line);
```

```
            break;
        case FlipAxisY:
            AllocateLineDouble(&Line, Ny, Status);
/* @ missing error management @ */
            Ky = 0L;
            for (Kz = 0L; (Kz < Nz); Kz++)
                for (Kx = 0L; (Kx < Nx); Kx++) {
                    GetyFloatToDouble(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Ny);
                    FlipLine(Line, Ny);
                    PutyDoubleToFloat(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Ny);
                }
            FreeLineDouble(&Line);
            break;
        case FlipAxisZ:
            AllocateLineDouble(&Line, Nz, Status);
            if (*Status) return (-1);
/* @ the test is not explicit enough @ */
/* @ returned value does not follow the conventions ERROR or !ERROR @ */
            Kz = 0L;
            for (Ky = 0L; (Ky < Ny); Ky++)
                for (Kx = 0L; (Kx < Nx); Kx++) {
                    GetzFloatToDouble(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nz);
                    FlipLine(Line, Nz);
                    PutzDoubleToFloat(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nz);
                }
            break;
/* @ missing deallocation @ */
        case FlipValues:
            FlipIntensity(Volume, Nx, Ny, Nz);
            break;
/* @ missing default case (because there is no validity check at routine entry) @ */
    }
    return(!ERROR);
/* @ even if an error arised in FlipAxisY, the error is not returned @ */
}

voidFlipIntensity (
/* @ missing qualifier static or extern @ */
                        float       *Volume,
                        long    Nx,
                        long    Ny,
                        long    Nz)
{
    float       *P;
    float       Max, Min, Sum;
    long    Kx, Ky, Kz;

    Max = *Volume;
    Min = *Volume;
    P = Volume;
    for (Kz = 0L; (Kz < Nz); Kz++)
        for (Ky = 0L; (Ky < Ny); Ky++)
            for (Kx = 0L; (Kx < Nx); P++, Kx++) {
                if (Max < *P)
                    Max = *P;
                if (*P < Min)
                    Min = *P;
            }
    Sum = Max + Min;
    P = Volume;
    for (Kz = 0L; (Kz < Nz); Kz++)
        for (Ky = 0L; (Ky < Ny); Ky++)
            for (Kx = 0L; (Kx < Nx); P++, Kx++)
                *P = Sum - *P;
}
#endif
```

```
#ifdef AVERAGE
/*.....................................................................

    Filename:    flipvolume.c

    Project: Biomedical Imaging Library

    Author:      Daniel Sage
                 Swiss Federal Institute of Technology--Lausanne
                 Biomedical Imaging Group
                 EPFL/DMT/IOA
                 BM-Ecublens
                 CH-1015 Lausanne
                 Switzerland

    Date:        February 4, 1999

    Purpose: Implementation of FlipVolume()
.....................................................................*/

#include     <stdio.h>
#include     <limits.h>
#include     <string.h>
#include     "configs.h"
#include     "messagedisplay.h"
#include     "error.h"
#include     "debug.h"
#include     "getput.h"
#include     "tflipvolume.h"
#include     "flipvolume.h"

/*---------------------------------------------------------------------------*/
static   voidFlipIntensity (
                         float        *Volume,
                         long     Nx,
                         long     Ny,
                         long     Nz);

/*---------------------------------------------------------------------------*/
static   voidFlipLine (
                         double       Line[],
                         long     N);

/*---------------------------------------------------------------------------*/
/*.....................................................................
    Procedure:   FlipVolume(Volume, Nx, Ny, Nz, FlipAxis, &Status)

    Purpose: To flip a (float)volume around one of the four main axis
                 There are three axis of space and one of intensity

    Parameters:  Volume
                     float *
                     in-place processing
                     reference parameter
                     pointer to a block of float memory
                 Nx
                     long
                     input
                     value parameter
                     horizontal size of 'Volume', in voxel units
                 Ny
                     long
                     input
                     value parameter
                     vertical size of 'Volume', in voxel units
                 Nz
```

```
                    long
                    input
                    value parameter
                    transverse size of 'Volume', in voxel units
                FlipAxis
                    enum TflipEnum
                    input
                    value parameter
                    selection of the axis along which the flip will occur
                Status
                    int *
                    output
                    reference parameter
                    duplicates the returned value
    Return:     int
                ERROR
                    Something went wrong
                !ERROR
                    Nothing went wrong
.....................................................................*/
extern  int     FlipVolume (
                        float       *Volume, /* In- and Output volume      */
                        long    Nx,          /* In- and Output width       */
                        long    Ny,          /* In- and Output height      */
                        long    Nz,          /* In- and Output depth       */
                        enum TflipEnum
                                    FlipAxis,    /* Flip Axis              */
                        int         *Status) /* Error management           */

{

    double      *Line;
    long    Kx, Ky, Kz;

    *Status = !ERROR;

/***************************
 * Check for input validity *
 ***************************/
/**/DEBUG_CHECK_NULL_POINTER(FlipVolume, Volume, *Status,
/**/"No volume")
/**/DEBUG_CHECK_RANGE_LONG(FlipVolume, Nx, 1L, LONG_MAX, *Status,
/**/"Invalid width (should be strictly positive)")
/**/DEBUG_CHECK_RANGE_LONG(FlipVolume, Ny, 1L, LONG_MAX, *Status,
/**/"Invalid height (should be strictly positive)")
/**/DEBUG_CHECK_RANGE_LONG(FlipVolume, Nz, 1L, LONG_MAX, *Status,
/**/"Invalid depth (should be strictly positive)")
/**/DEBUG_RETURN_ON_ERROR(FlipVolume, *Status)
    switch (FlipAxis) {
        case FlipAxisX:
        case FlipAxisY:
        case FlipAxisZ:
        case FlipValues:
            break;
        default:
            WRITE_ERROR(FlipVolume,
                "Invalid flip FlipAxis (should be one of Flip-{AxisX,AxisY,AxisZ,Values})")
/**/    DEBUG_WRITE_LEAVING(FlipVolume,
/**/        "Done with FlipVolume")
            return(ERROR);
    }

/********************
 * Perform the task *
 ********************/
    switch (FlipAxis) {
```

```
        case FlipAxisX:
            AllocateLineDouble(&Line, Nx, Status);
            if (*Status == ERROR) return(ERROR);
            Kx = 0L;
            for (Kz = 0L; (Kz < Nz); Kz++)
                for (Ky = 0L; (Ky < Ny); Ky++) {
                    GetxFloatToDouble(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nx);
                    FlipLine(Line, Nx);
                    PutxDoubleToFloat(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nx);
                }
            FreeLineDouble(&Line);
            break;
        case FlipAxisY:
            AllocateLineDouble(&Line, Ny, Status);
            if (*Status == ERROR) return(ERROR);
            Ky = 0L;
            for (Kz = 0L; (Kz < Nz); Kz++)
                for (Kx = 0L; (Kx < Nx); Kx++) {
                    GetyFloatToDouble(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Ny);
                    FlipLine(Line, Ny);
                    PutyDoubleToFloat(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Ny);
                }
            FreeLineDouble(&Line);
            break;
        case FlipAxisZ:
            AllocateLineDouble(&Line, Nz, Status);
            if (*Status == ERROR) return(ERROR);
            Kz = 0L;
            for (Ky = 0L; (Ky < Ny); Ky++)
                for (Kx = 0L; (Kx < Nx); Kx++) {
                    GetzFloatToDouble(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nz);
                    FlipLine(Line, Nz);
                    PutzDoubleToFloat(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nz);
                }
            FreeLineDouble(&Line);
            break;
        case FlipValues:
            FlipIntensity(Volume, Nx, Ny, Nz);
            break;
    }
    return(!ERROR);
}

/*-------------------------------------------------------------------------*/
/*.........................................................................
    Procedure:   FlipIntensity(Volume, Nx, Ny, Nz)

    Purpose: To flip the intensities of a volume
                The maximum value becomes a minimum
                The minimum value becomes a maximum

    Parameters:  Volume
                    float *
                    in-place processing
                    reference parameter
                    pointer to a block of double memory
                Nx
                    long
                    input
                    value parameter
                    horizontal size of 'Volume', in voxel units
                Ny
                    long
                    input
                    value parameter
                    vertical size of 'Volume', in voxel units
```

```
                Nz
                    long
                    input
                    value parameter
                    transverse size of 'Volume', in voxel units
    Return:     None
.....................................................................*/
static  voidFlipIntensity (
                        float       *Volume, /* In- and Output volume  */
                        long    Nx,          /* In- and Output width       */
                        long    Ny,          /* In- and Output height  */
                        long    Nz)          /* In- and Output depth       */

{

    float       *P;
    float       Max, Min, Sum;
    long    Kx, Ky, Kz;

/*************************
 * Find maximum & minimum *
 *************************/
    Max = *Volume;
    Min = *Volume;
    P = Volume;
    for (Kz = 0L; (Kz < Nz); Kz++)
        for (Ky = 0L; (Ky < Ny); Ky++)
            for (Kx = 0L; (Kx < Nx); P++, Kx++) {
                if (Max < *P)
                    Max = *P;
                if (*P < Min)
                    Min = *P;
            }

/********************
 * Perform the task *
 ********************/
    Sum = Max + Min;
    P = Volume;
    for (Kz = 0L; (Kz < Nz); Kz++)
        for (Ky = 0L; (Ky < Ny); Ky++)
            for (Kx = 0L; (Kx < Nx); P++, Kx++)
                *P = Sum - *P;
}

/*---------------------------------------------------------------------*/
/*.....................................................................
    Procedure:   FlipLine(Line, N)

    Purpose: To flip a (double)line

    Parameters:  Line
                    double[]
                    in-place processing
                    reference parameter
                    pointer to a block of double memory
                N
                    input
                    value parameter
                    size of 'Line', in voxel units
    Return:     None
.....................................................................*/
static  voidFlipLine (
                        double      Line[],     /* In- and Output line        */
                        long    N)          /* line length                */
```

```
{

    double       Swap;
    long     K;

/*******************
 * Perform the task *
 *******************/
    K = 0L;
    while (K < --N) {
         Swap = Line[K];
         Line[K++] = Line[N];
         Line[N] = Swap;
    }
}
#endif
```

```
#ifdef GOOD
/*.....................................................................

    Filename:    flipvolume.c

    Project: Biomedical Imaging Library

    Author:      Daniel Sage
                 Swiss Federal Institute of Technology--Lausanne
                 Biomedical Imaging Group
                 EPFL/DMT/IOA
                 BM-Ecublens
                 CH-1015 Lausanne
                 Switzerland

    Date:        February 4, 1999

    Purpose: Implementation of FlipVolume()
.....................................................................*/


/*************************************************************************/
/*  System includes                                                      */
/*************************************************************************/
#include     <stdio.h>


/*************************************************************************/
/*  Toolbox includes                                                     */
/*************************************************************************/
#include     "configs.h"
#include     "messagedisplay.h"
#include     "error.h"
#include     "debug.h"
#include     "getput.h"


/*************************************************************************/
/*  Conditional includes                                                 */
/*************************************************************************/
#ifdef DEBUG
#include     <limits.h>
#include     <string.h>
#endif


/*************************************************************************/
/*  Private includes                                                     */
/*************************************************************************/
#include     "tflipvolume.h"
#include     "flipvolume.h"


/*************************************************************************/
/*  Declaration of static procedures                                     */
/*************************************************************************/

/*-----------------------------------------------------------------------*/
static   int      FlipIntensity (
                          float            *Volume,
                          long         Nx,
                          long         Ny,
                          long         Nz);

/*-----------------------------------------------------------------------*/
static   int      FlipLine (
```

```
                        double           Line[],
                        long        N);


/*************************************************************************/
/*  Definition of extern procedures                                      */
/*************************************************************************/

/*-----------------------------------------------------------------------*/
/*.......................................................................
    Procedure:   FlipVolume(Volume, Nx, Ny, Nz, FlipAxis, &Status)

    Purpose: To flip a (float)volume around one of the four main axis
                There are three axis of space and one of intensity

    Parameters:  Volume
                     float *
                     in-place processing
                     reference parameter
                     pointer to a block of float memory
                 Nx
                     long
                     input
                     value parameter
                     horizontal size of 'Volume', in voxel units
                 Ny
                     long
                     input
                     value parameter
                     vertical size of 'Volume', in voxel units
                 Nz
                     long
                     input
                     value parameter
                     transverse size of 'Volume', in voxel units
                 FlipAxis
                     enum TflipEnum
                     input
                     value parameter
                     selection of the axis along which the flip will occur
                 Status
                     int *
                     output
                     reference parameter
                     duplicates the returned value
    Return:      int
                 ERROR
                     Something went wrong
                 !ERROR
                     Nothing went wrong
.......................................................................*/
extern   int     FlipVolume (
                        float       *Volume, /* In- and Output volume  */
                        long    Nx,          /* In- and Output width       */
                        long    Ny,          /* In- and Output height  */
                        long    Nz,          /* In- and Output depth       */
                        enum TflipEnum
                                    FlipAxis,    /* Flip Axis              */
                        int         *Status) /* Error management           */

{ /* begin FlipVolume */

    double       *Line;
    long    Kx, Ky, Kz;

    *Status = !ERROR;
```

```
/****************************
 * Check for input validity *
 ****************************/
/**/DEBUG_CHECK_NULL_POINTER(FlipVolume, Volume, *Status,
/**/"No volume")
/**/DEBUG_CHECK_RANGE_LONG(FlipVolume, Nx, 1L, LONG_MAX, *Status,
/**/"Invalid width (should be strictly positive)")
/**/DEBUG_CHECK_RANGE_LONG(FlipVolume, Ny, 1L, LONG_MAX, *Status,
/**/"Invalid height (should be strictly positive)")
/**/DEBUG_CHECK_RANGE_LONG(FlipVolume, Nz, 1L, LONG_MAX, *Status,
/**/"Invalid depth (should be strictly positive)")
/**/DEBUG_RETURN_ON_ERROR(FlipVolume, *Status)
    switch (FlipAxis) {
        case FlipAxisX:
        case FlipAxisY:
        case FlipAxisZ:
        case FlipValues:
            break;
        default:
            WRITE_ERROR(FlipVolume,
                "Invalid flip operation (should be one of
{FlipAxisX,FlipAxisY,FlipAxisZ,FlipValues})")
/**/    DEBUG_WRITE_LEAVING(FlipVolume,
/**/        "Done with FlipVolume")
            return(*Status);
    }
/**/DEBUG_WRITE_ENTERING(FlipVolume,
/**/"About to flip a volume")

/********************
 * Perform the task *
 ********************/
    switch (FlipAxis) {
        case FlipAxisX:
            if (AllocateLineDouble(&Line, Nx, Status) == ERROR) {
/**/        DEBUG_WRITE_LEAVING(FlipVolume,
/**/            "Done with FlipVolume")
                return(ERROR);
            }
            Kx = 0L;
            for (Kz = 0L; (Kz < Nz); Kz++)
                for (Ky = 0L; (Ky < Ny); Ky++) {
                    if (GetxFloatToDouble(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nx) == ERROR) {
                        *Status = FreeLineDouble(&Line);
/**/                    DEBUG_WRITE_LEAVING(FlipVolume,
/**/                        "Done with FlipVolume")
                        return(ERROR);
                    }
                    if (FlipLine(Line, Nx) == ERROR) {
                        *Status = FreeLineDouble(&Line);
/**/                    DEBUG_WRITE_LEAVING(FlipVolume,
/**/                        "Done with FlipVolume")
                        return(ERROR);
                    }
                    if (PutxDoubleToFloat(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nx) == ERROR) {
                        *Status = FreeLineDouble(&Line);
/**/                    DEBUG_WRITE_LEAVING(FlipVolume,
/**/                        "Done with FlipVolume")
                        return(ERROR);
                    }
                }
            if (FreeLineDouble(&Line) == ERROR) {
/**/        DEBUG_WRITE_LEAVING(FlipVolume,
/**/            "Done with FlipVolume")
                return(ERROR);
```

```
            }
            break;
        case FlipAxisY:
            if (AllocateLineDouble(&Line, Ny, Status) == ERROR) {
/**/        DEBUG_WRITE_LEAVING(FlipVolume,
/**/            "Done with FlipVolume")
                return(ERROR);
            }
            Ky = 0L;
            for (Kz = 0L; (Kz < Nz); Kz++)
                for (Kx = 0L; (Kx < Nx); Kx++) {
                    if (GetyFloatToDouble(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Ny) == ERROR) {
                        *Status = FreeLineDouble(&Line);
/**/                    DEBUG_WRITE_LEAVING(FlipVolume,
/**/                        "Done with FlipVolume")
                        return(ERROR);
                    }
                    if (FlipLine(Line, Ny) == ERROR) {
                        *Status = FreeLineDouble(&Line);
/**/                    DEBUG_WRITE_LEAVING(FlipVolume,
/**/                        "Done with FlipVolume")
                        return(ERROR);
                    }
                    if (PutyDoubleToFloat(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Ny) == ERROR) {
                        *Status = FreeLineDouble(&Line);
/**/                    DEBUG_WRITE_LEAVING(FlipVolume,
/**/                        "Done with FlipVolume")
                        return(ERROR);
                    }
                }
            if (FreeLineDouble(&Line) == ERROR) {
/**/        DEBUG_WRITE_LEAVING(FlipVolume,
/**/            "Done with FlipVolume")
                return(ERROR);
            }
            break;
        case FlipAxisZ:
            if (AllocateLineDouble(&Line, Nz, Status) == ERROR) {
/**/        DEBUG_WRITE_LEAVING(FlipVolume,
/**/            "Done with FlipVolume")
                return(ERROR);
            }
            Kz = 0L;
            for (Ky = 0L; (Ky < Ny); Ky++)
                for (Kx = 0L; (Kx < Nx); Kx++) {
                    if (GetzFloatToDouble(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nz) == ERROR) {
                        *Status = FreeLineDouble(&Line);
/**/                    DEBUG_WRITE_LEAVING(FlipVolume,
/**/                        "Done with FlipVolume")
                        return(ERROR);
                    }
                    if (FlipLine(Line, Nz) == ERROR) {
                        *Status = FreeLineDouble(&Line);
/**/                    DEBUG_WRITE_LEAVING(FlipVolume,
/**/                        "Done with FlipVolume")
                        return(ERROR);
                    }
                    if (PutzDoubleToFloat(Volume, Nx, Ny, Nz, Kx, Ky, Kz, Line, Nz) == ERROR) {
                        *Status = FreeLineDouble(&Line);
/**/                    DEBUG_WRITE_LEAVING(FlipVolume,
/**/                        "Done with FlipVolume")
                        return(ERROR);
                    }
                }
            if (FreeLineDouble(&Line) == ERROR) {
/**/        DEBUG_WRITE_LEAVING(FlipVolume,
```

```
/**/             "Done with FlipVolume")
                 return(ERROR);
             }
             break;
         case FlipValues:
             if (FlipIntensity(Volume, Nx, Ny, Nz) == ERROR) {
/**/         DEBUG_WRITE_LEAVING(FlipVolume,
/**/             "Done with FlipVolume")
                 return(ERROR);
             }
             break;
    }

/********
 * Exit *
 ********/
/**/DEBUG_WRITE_LEAVING(FlipVolume,
/**/"Done with FlipVolume")
    return(*Status);
} /* end FlipVolume */


/*****************************************************************************/
/*  Definition of static procedures                                          */
/*****************************************************************************/

/*---------------------------------------------------------------------------*/
/*...........................................................................
    Procedure:   FlipIntensity(Volume, Nx, Ny, Nz)

    Purpose: To flip the intensities of a volume
                 The maximum value becomes a minimum
                 The minimum value becomes a maximum

    Parameters:  Volume
                     in-place processing
                     reference parameter
                     pointer to a block of double memory
                 Nx
                     input
                     value parameter
                     horizontal size of 'Volume', in voxel units
                 Ny
                     input
                     value parameter
                     vertical size of 'Volume', in voxel units
                 Nz
                     input
                     value parameter
                     transverse size of 'Volume', in voxel units
    Return:      int
                 ERROR
                     Something went wrong
                 !ERROR
                     Nothing went wrong
...........................................................................*/
static   int     FlipIntensity (
                         float       *Volume, /* In- and Output volume  */
                         long    Nx,          /* In- and Output width       */
                         long    Ny,          /* In- and Output height  */
                         long    Nz)          /* In- and Output depth       */

{ /* begin FlipIntensity */

    float        *P;
    float        Max, Min, Sum;
```

```
    long     Kx, Ky, Kz;
    int          Status = !ERROR;

/*****************************
 * Check for input validity *
 *****************************/
/**/DEBUG_CHECK_NULL_POINTER(FlipIntensity, Volume, Status,
/**/"No volume")
/**/DEBUG_CHECK_RANGE_LONG(FlipIntensity, Nx, 1L, LONG_MAX, Status,
/**/"Invalid width (should be strictly positive)")
/**/DEBUG_CHECK_RANGE_LONG(FlipIntensity, Ny, 1L, LONG_MAX, Status,
/**/"Invalid height (should be strictly positive)")
/**/DEBUG_CHECK_RANGE_LONG(FlipIntensity, Nz, 1L, LONG_MAX, Status,
/**/"Invalid depth (should be strictly positive)")
/**/DEBUG_RETURN_ON_ERROR(FlipIntensity, Status)
/**/DEBUG_WRITE_ENTERING(FlipIntensity,
/**/"About to invert the values of a volume")

/*************************
 * Find maximum & minimum *
 *************************/
    Max = *Volume;
    Min = *Volume;
    P = Volume;
    for (Kz = 0L; (Kz < Nz); Kz++)
         for (Ky = 0L; (Ky < Ny); Ky++)
              for (Kx = 0L; (Kx < Nx); P++, Kx++) {
                   if (Max < *P)
                        Max = *P;
                   if (*P < Min)
                        Min = *P;
              }

/********************
 * Perform the task *
 ********************/
    Sum = Max + Min;
    P = Volume;
    for (Kz = 0L; (Kz < Nz); Kz++)
         for (Ky = 0L; (Ky < Ny); Ky++)
              for (Kx = 0L; (Kx < Nx); P++, Kx++)
                   *P = Sum - *P;

/********
 * Exit *
 ********/
/**/DEBUG_WRITE_LEAVING(FlipIntensity,
/**/"Done with FlipIntensity")
    return(Status);
} /* FlipIntensity */

/*---------------------------------------------------------------------------*/
/*...........................................................................
    Procedure:   FlipLine(Line, N)

    Purpose: To flip a (double)line

    Parameters:  Line
                      in-place processing
                      reference parameter
                      pointer to a block of double memory
                 N
                      input
                      value parameter
                      size of 'Line', in voxel units
    Return:      int
```

```
                ERROR
                    Something went wrong
                !ERROR
                    Nothing went wrong
.....................................................................*/
static   int     FlipLine (
                        double       Line[],      /* In- and Output line        */
                        long     N)           /* line length                */

{ /* begin FlipLine */

    double       Swap;
    long     K;
    int          Status = !ERROR;

/***************************
 * Check for input validity *
 ***************************/
/**/DEBUG_CHECK_NULL_POINTER(FlipLine, Line, Status,
/**/"No line")
/**/DEBUG_CHECK_RANGE_LONG(FlipLine, N, 1L, LONG_MAX, Status,
/**/"Invalid size (should be strictly positive)")
/**/DEBUG_RETURN_ON_ERROR(FlipLine, Status)
/**/DEBUG_WRITE_ENTERING(FlipLine,
/**/"About to flip a line")

/********************
 * Perform the task *
 ********************/
    K = 0L;
    while (K < --N) {
         Swap = Line[K];
         Line[K++] = Line[N];
         Line[N] = Swap;
    }

/********
 * Exit *
 ********/
/**/DEBUG_WRITE_LEAVING(FlipLine,
/**/"Done with FlipLine")
    return(Status);
} /* FlipLine */
#endif
```