INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
RENNES

C language 2

Labs
S5 EII

2020-2021

Contents

LAB 3

Postfix expressions and pointers of function

Objectives : (*1h30*)

- — use the stack of int to evaluate postfix expressions ;
- — write a new solution based on pointers of functions ;
- — write a Makefile to handle a C project with several modules ;
- — practice basic unit testing.

Requirement :

- — read the whole practical subject ;
- — understand the postfix expression notation ;
- — read the course document on pointers of function ;
- — write the Makefile for the project **before this lab**.

You will need the stack of *int* of the previous lab.

3.1 Evaluation of postfix expressions

The usual arithmetic expressions are expressed in the infix notation. For example : 1 - 2 + 3 + 4 + 5 = 11.

This notation requires parenthesis to define priority.
For example : 3 + 9 - 3 * 4 / 3 = 8 and 3 + (9 - 3) * 4 / 3 = 11.

The postfix notation does not suffer from this drawback : the operator is written after the operands. For example, the infix expression 3 + (9 -3)*4 /3 = 11 is written in postfix notation 3 9 3 - 4 * 3 / + = 11. Such a postfix expression has to be read from left to right. Each time an operator is met, it is applied on the two last operands of the expression.

A stack can be used to help the evaluation of such an expression. When an operator is met, the 2 last values of the stack are popped and the result of the operation is pushed. The Figure 3.1 details the stack state when evaluating 3 9 3 - 4 * 3 / +.

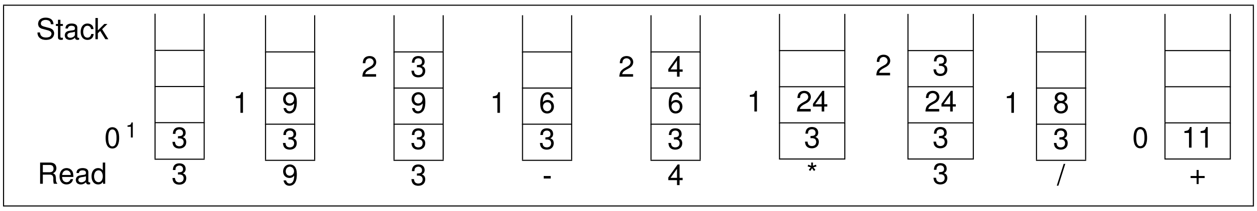


FIGURE 3.1 – Evaluation of the postfix expression 3 9 3 - 4 * 3 / + using a stack

3.1.A An implementation of the postfix evaluation using a stack

An implementation of such a use of the stack ("evaluation.c/.h") is provided, only with the operators ’+’ and ’-’. Copy the code and Makefile from the previous lab and add the provided code. Update your Makefile of the previous lab to compile and test the postfix evaluation code by calling *test_eval()* in your *main()* function.

Why is the function *binary_operator()* static ? Why should it not be declared in the header "evaluation.h" ?

Add the operators ’*’ and ’/’ in the evaluation functionalities and test them to evaluate 3 9 3 - 4 * 3 / +

3.1.B Improve the implementation by using function pointers

The current code suffers from a drawback : the switch in the function *binary_operator()* is not efficient and may be hard to maintain if a lot of operators are handled. Moreover, with this switch solution, the behaviour of the operator ’+’ is defined in the same code block than the operator ’*’ while they are not related. The use of function pointers will provide a solution to avoid this switch and its resulting disadvantages.

Code the function *binary_operator_pf()* that takes as a parameter directly the function to be applied to the operands. Its prototype is :

```
Boolean binary_operator_pf(Stack *s, int (*op)(int,int));
```

Modify the add, substract,... functions to use this function instead of *binary_operator()* and check if the evaluation of the postfix expression is still right.

3.1.C Evaluate the whole postfix expression directly

In order to ease the evaluation of an expression, it is required to compute the result of an expression given in a string. An implementation of such a function can be :

```
static int isANumber(char c)
{    return (c>=’0’) && (c <= ’9’);    }

int evaluateExpression(char *str){
  Stack s;
  int i = 0;
  init_stack(&s);
  while(str[i] != ’\0’)
  {
    if(isANumber(str[i]))
        push(&s, str[i] - ’0’);
    else
    {
        switch(str[i])
        {
            case '+' : add(&s); break;
            case '-' : substract(&s); break;
            case '*' : multiply(&s); break;
            case '/' : divide(&s); break;
            default : fprintf(stderr,"unknown operator\n"); return 0;
        }
    }
    ++i;
  }
  return pop(&s);
}
```

This code is provided in "evaluationFromString.c". Update your Makefile to compile it and test the evaluation of *expr1* in the *test_eval_string()* function using the provided unit test functions.

Once again, the switch is not very smart. Therefore, we would like an array that links a character ('+','-','*','/') to the action to perform when this character is met. This link is registered in such a structure :

```
typedef struct{
    char c;                     //<! operator code ('+', '-', '*', '/',...)
    Boolean (*act) (Stack *);   //<! pointer of function associated to the code
}Map_element;
```

Add in "evaluation.h" the *Map_element* structure and in "evaluationFromString.c" an array of *Map_element* structures with every association (for '+','-','*','/').

In "evaluationFromString.c", code *evaluateExpression_pf()* using this array instead of the switch thanks to the pointers of function stored in the *Map_element* structure.

```
int evaluateExpression_pf(char *str);
```

Test it in the *test_eval_string()* function using the provided unit test functions.

3.1.D Add unary operators

On the model of the binary operators, it is required to add the square-root operator $\sqrt{x}$ and the cube operator x^3. They are unary operators (require only one operand). The character associated to the square-root will be 's' and the one for cube will be 'c'.

You will have to add the option -lm to the LDFLAGS option in the Makefile in order to link with the *sqrt* function binary code[2] :

```
        LDFLAGS=-lm
```

Modify your code to handle the proposed unary operators.

[2] Il s'agit de la librairie "math" standard en C.

LAB 4

Dynamic memory allocation : Image structure

Objectives : (*1h30*)
- Practice dynamic memory allocation ;
- Implement a classical image structure ;
- Link with third-party binary code ;
- Write a Makefile with third-party binary code.

Requirement :
- read the course document on dynamic allocation ;
- the module "test" from the previous lab ;
- the module "fichier" from the previous labs ;
- read the whole practical subject ;
- write the Makefile for the project **before this lab** ;
- read Appendix 6 to be able to configure Clion with a Makefile.

From now, you can use Clion as IDE.

4.1 Image structure

This practical work is about handling an image structure. An image will be considered in our case as in Figure 4.1, *ie* a matrix of pixels of dimension $N \times M$ where :
- N is the number of rows of pixels : the height of the image ;
- M is the number of columns of pixel : the width of the image.

There are several ways to represent this 2D matrix : a 2D array or a 1D array. We will use this latter one. In this case, data (*ie* pixels) is stored in an array representing the grid of pixels, where rows are stored continuously. This choice is often used in the image processing community.

This is illustrated by the Figure 4.1 :

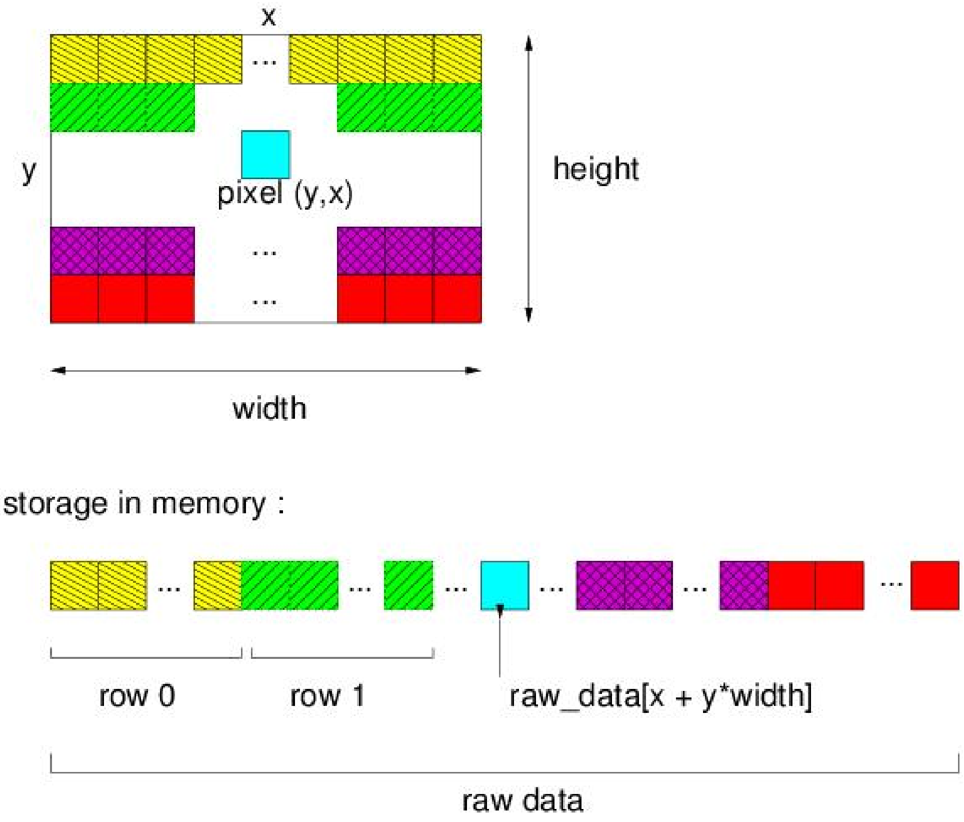


FIGURE 4.1 – The grid of pixels in the Image structure

Besides this, each value of the grid is a pixel and pixels are represented by several bytes to handle colour. For example, for a RBG (Red Blue Green) colour space, you need 3 bytes, one per component.

It leads to the following declaration of the image structure :

```
/** struct typedef to define a Pixel using RGB colour space
* A pixel is defined in RGB space : a byte for each color coponent (Blue,
   Green, Red)
*/
typedef struct{
      unsigned char blue;   /*!<the blue component */
      unsigned char green;  /*!<the green component */
      unsigned char red;    /*!<the red component */
}Pixel;

/** struct typedef to define an Image using RBG colour space
* An image is defined as a grid of pixels.
* It needs to set its number of columns (width), number of rows (heigth).
* The pixel data is stored continuously in memory, row by row.
*/
typedef struct{
      int width;    /*!<the number of columns in the pixel grid*/
      int height;   /*!<the number of lines in the pixel grid*/
      Pixel * raw_data; /*!<the pixel grid*/
}Image;
```

In order to use this structure properly and simply, it is necessary to provide basic functionalities, such as create an image, fill it, display it, ...

The following questions guide you step by step. For each functionality, you will find a more detailed documentation in the provided *TP3-doc* directory, for example by opening the *index.html* file.

Question 0 : Download the files "image.h" and "main.c". Write the Makefile related to your project. Then create a *clion*project using a Makefile following in the Appendix 6 how to use the Makefile within *clion*. Test the build (the main program does nothing for the moment as everything is commented).

Question 1 : Add a file "image.c" in your project. Update the Makefile. Implement in this file "image.c" the function *allocateImage* that allocates the memory for an image of size $h \times w$:

```c
Image * allocateImage(int h, int w);
```

Question 2 : To test the function allocateImage, we need to display the resulting image. It is inconvenient to display the image structure on the console output. So we provide a "BMPFile_teacher.o" file that contains binary code to store an image in the BMP format[1]. Download the files "BMPFile/fichier.h" and "BMPFile_teacher/fichier_teacher.o" in the project repertory. Update your Makefile to build the program. To check the image you created with your function *allocateImage*, you can use the provided *writeBMPFile_teacher* function like this :

```c
#include "BMPfile.h"
#include "image.h"
/*!\def MY_FILENAME_MAX
* Taille maximum du nom d’un fichier*/
#define MY_FILENAME_MAX 64
int main()
{
    char filename[MY_FILENAME_MAX];
    Image* f = NULL;
    f = allocateImage(200,100);
    sprintf(filename,"image_test.bmp");
    writeBMPFile_teacher(filename, f,0);
    return 0;
}
```

The three arguments of the provided *writeBMPFile_teacher* function are :

— the filename where to write the image contents

— the image structure to be saved in a file

— a verbose flag : 1 to display every information about the writing process, 0 otherwise.

Allocate an image of size 200×100 and save it in a file. Check you can open this new file. Check also its properties : it should be 200×100.

Question 3 : if you have a function to allocate a structure, you need to provide a function that deletes the structure. Implement this function :

```c
void freeImage(Image * i);
```

[1] This format will be described in the next lab session, as you will implement it.

Question 4 : For the moment, you have only allocated an image structure. It is time to fill it.

As presented in Figure 4.1, you can access to the pixel at location (y,x) in the image I with :

```
/*if x is the column index and y the row index, the pixel (y,x) is given by */
I.raw_data[x + y*I.width]
```

and the i^{th} row address is given by

```
&(I.raw_data[i*I.width])
```

Write the two following functions that enable to modify a pixel in the image (*setPixelxy*) or to get its value (*getPixelxy*) :

```
void setPixelxy(Image *image, int x, int y , Pixel p);
Pixel getPixelxy(Image *image, int x, int y);
```

Question 5 : Implement a function that replaces every pixel of color *pixel_initial* in an image by a pixel of color *pixel_final*. Therefore, this function modifies the image given as parameter. The prototype of the function is :

```
void modifyImage(Image * image, Pixel pix_initial, Pixel pix_final);
```

To test this function, do te following in the main function :

```
  Image * f = readBMPFile_teacher("mask.bmp",0);
  modifyImage(f,{0,0,0},{255,255,255});
  writeBMPFile_teacher("mask_unmasked.bmp",f,0);
```

This code loads in a structure Image the content of the provided image "mask.bmp" using the function *readBMPFile_teacher*.Then it substitutes every black pixel (R=0,G=0,B=0) with a white one (R=255,G=255,B=255). When saving the modified image in "mask_unmasked.bmp", you should get a text, one of my favourites.

Question 6 : Implement a function that creates an image of size 100×100 like on Figure 4.2 : Use the function "writeBMPFILE_teacher" to see the result.

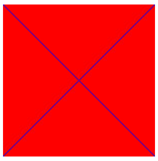


FIGURE 4.2 – An image with a red background and a blue cross

LAB 5

Binary files : bitmap format

Objectives : (*1h30*)
- Read and write in a binary file ;
- Discover the Bitmap format for image files ;
- Discover two examples of padding.

Requirement :
- read the course document on binary files ;
- read the whole practical subject ;
- the previous lab

This practical work is about reading and writing image BMP file format. The BMP file format is a raster graphics image file format, which means that data is stored in a matrix representing the grid of pixels. Pixels are represented by several bytes to handle colour. For example, for a RBG (Red Blue Green) colour space, you need 3 bytes, one per component.

The content of the read file will be stored in the image structure of the previous lab. Therefore, copy the project of the previous lab and work on this copy.

5.1 Bitmap file format

You will practice binary file reading and writing on bitmap image file. When storing data of an image in a file, you may choose different kinds of formats, such as bitmap (.bmp), jpeg, png,...

The BMP file format can handle many pixel formats however we will focus only on the RBG one for this practical work.

A BMP file is divided into 3 parts[1] :
- information about the file : size, signature,... This part is called the file header and its length is 14 bytes.
- information about the image data : width, height, resolution, colour space,... This part is called the DIB (Device Independent Bitmap) header whose length is variable.
- image data : pixels stored row by row, starting by the bottom row of the image and ending with the top one. To better align data in memory, the size of each row is rounded up to a multiple of 4 bytes by padding. Some help will be given on the padding when it will be necessary.

[1] optional parts can be added but they will not be considered in this practical work. See wikipedia for further information.

This is illustrated in the Figure 5.1.

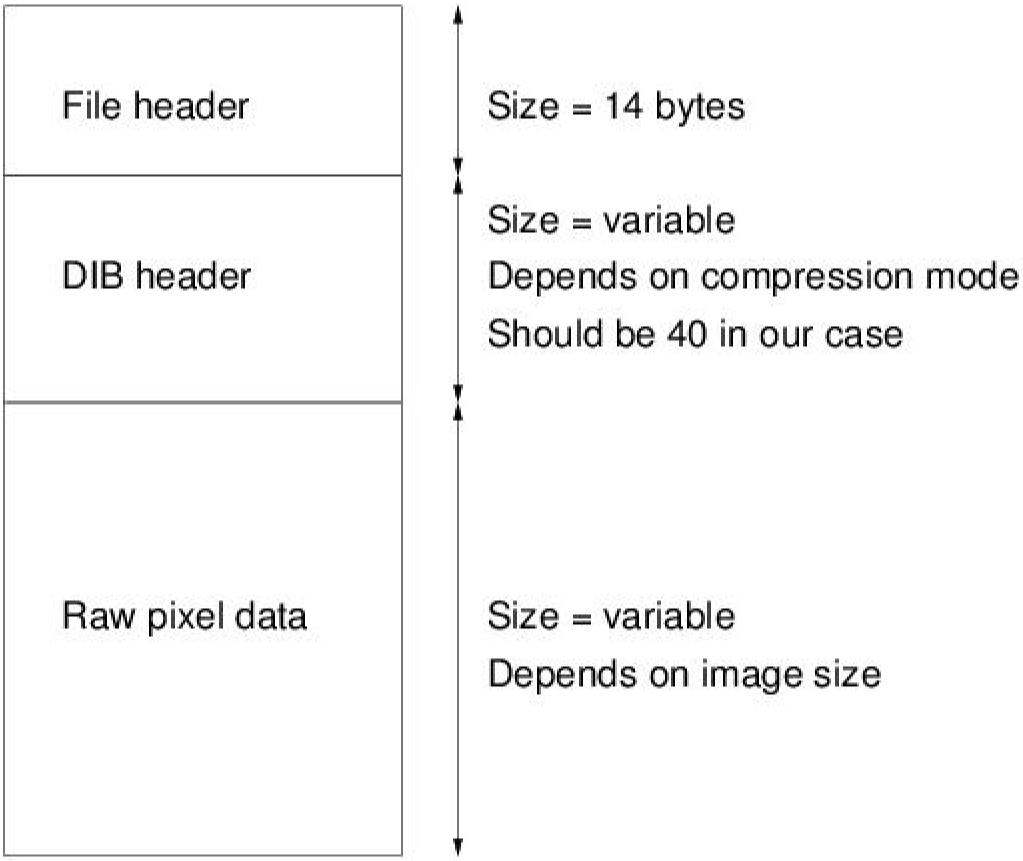


FIGURE 5.1 – The BMP file format

Here is a further description of the file header in a C structure :

```
/** A struct typedef to store the content of the BMP file header
*/
typedef struct {
 char identity[2];          /*!< 'B''M' for a Windows BitMap*/
 uint32_t  file_size;       /*!< 4 bytes for an unsigned int*/
 char application_id[4];    /*!< 4 bytes for the application id which creates
    the file. Ignore its value for this practical work*/
 uint32_t raster_address;   /*!< 4 bytes corresponding to the adress in the
    file where the image raw data begins*/
}FileHeader;
```

Here is a further description of the DIB header in a C structure :

```
/** A struct typedef to store the content of the DIB  header*/
typedef struct {
  uint32_t size_DIBHeader;     /*!<DIB Header size*/
  uint32_t image_width;        /*!<Image width in pixels = number of columns*/
  uint32_t image_height;       /*!<Image height in pixels = number of rows*/
  uint16_t nbColorPlanes;      /*!<set to 1*/
  uint16_t nbBitPerPixel;      /*!<set to 24 for this lab (3 bytes for RGB)*/
  uint32_t typeCompression;    /*!<set to  0 for RGB colour space*/
  uint32_t size_raw_image;     /*!<total data size*/
  int32_t hResolution;         /*!<horizontal resolution = 2800 for this PW*/
  int32_t vResolution;         /*!<vertical resolution = 2800 for this PW*/
  uint32_t nbUsedColours;      /*!<set to 0 for this lab*/
  uint32_t nbImportantColours;/*!<set to 0 for this lab*/
}DIBHeader;
```

The order of the structure fields is important : it will be storedu using the same order in the file, so do not modify them.
You can also notice that *uint32_t* type is used rather than *int*. Indeed, the *int* size depends on your OS (32 bits or 64 bits for example). Since bitmap files use *int* values on 4 bytes, it is better to use the *uint32_t* type so the declaration works for all OS.

You will now implement the write and read BMP file functions in a new file called "BMPFile.c". The declarations are provided in the "BMPFile.h" file of the previous lab.

5.1.A Write a bitmap file

The first work is about writing a BMP file to save an image. You will have to implement the *writeBMPFile* function :

```
void writeBMPFile(char * filename,Image* image);
```

The *writeBMPFile* function aims to write in a file the binary information relative to the given image. To do so, you will use a FileHeader and a DIBHeader structure, fill them according to the image content. Once they are filled, you will write in a file the binary information in the order required by the BMP file format :

- the file header
- the DIB header
- the raw data

Question 1 : First test your code on an image whose size is 200×200

Question 2 : When writing the raw data, you have to take into account the padding required by the BMP file format (a row size in the file should be a multiple of 4 bytes), you will have to estimate the size of a row (width * sizeof(Pixel)). If this latter is not a multiple of 4 bytes, you will have to add one, two or three bytes (or characters) at the end of each row. To test your padding management, try with a cross of dimension 151×151.

The provided *main* function should work by replacing *writeBMPFile_teacher* by *writeBMPFile* in the *main* function.

You can use the display functions available to test a part of your code :

```
void displayFileHeader(FileHeader* eFichier);
void displayDIBHeader(DIBHeader* eImage);
```

5.1.B Read a bitmap file

The last work is about reading a BMP file to create an image structure. You will have to implement the *readBMPFile* function :

```
Image* readBMPFile(char * filename, int verbose);
```

It will read information from a BMP file to create an Image structure so the program can do some image processing on it for example. Here is the process step by step to code the *readBMPFile* function :

Question 3 : First you will fill the FileHeader structure according to the file content by coding the following function

```
    void readFileHeader(FILE* fp, FileHeader* eFichier);
```

As it can be understood when analysing the prototype, the file is supposed to be already opened and handled by *fp*.

Question 4 : Then you will fill the DIBHeader structure according to the file content by coding the following function

```
    void readDIBHeader(FILE* fp,  DIBHeader* eImage);
```

Question 5 : For this next step, you will create the Image structure according to the file content by coding the following function

```
    Image* readRawImage(FILE* fp,unsigned int address, int l, int h);
```

Question 6 : At last, by calling the 3 previous functions in the right order, you can code the *readBMPFile* function to achieve its goal.

You can read the provided documentation for further information on these function parameters.

Now the provided *main* function should work by replacing *readBMPFile_teacher* by *readBMPFile* in the *main* function.

In prder to help you, we provide functions to display the FileHeader and DIBHeader structures.

5.2 The padding in memory management (optionnal this year)

At the beginning of file "BMPFile.h", there are the following comment and macro :

```
/**
To avoid padding (issue with sizeof(FileHeader) otherwise -> if padding
    enabled, replace sizeof(FileHeader) with the value 14 )
*/
#pragma pack(1)
```

However it is not recommended to disable the padding memory for the sake of performance. Indeed, the data structure alignment is an optimization performed to improve memory access[2]. Indeed, the processor is linked to the memory by a bus whose size (for example 32 bits) is bigger than the smaller variable (for example a char whose size is 8 bits). It is more time consuming to extract a char inside a 32 bit word than to read the char at the beginning of the 32 bit word. The padding consists in adding bytes to ensure that data is located at the beginning of a word read in the memory.

For the structure FileHeader

```
/** A struct typedef to store the content of the BMP file header
*/
typedef struct {
 char identity[2];          /*!< 'B''M' for a Windows BitMap*/
 uint32_t  file_size;       /*!< 4 bytes for an unsigned int*/
 char application_id[4];    /*!< 4 bytes for the application id which creates
     the file. Ignore its value for this practical work*/
 uint32_t raster_address;   /*!< 4 bytes corresponding to the adress in the
     file where the image raw data begins*/
}FileHeader;
```

this mechanism leads to the memory allocation described in Figure 5.2 for a 32 bit OS since addresses will be 4-bytes aligned (*ie* multiple of 4 bytes).

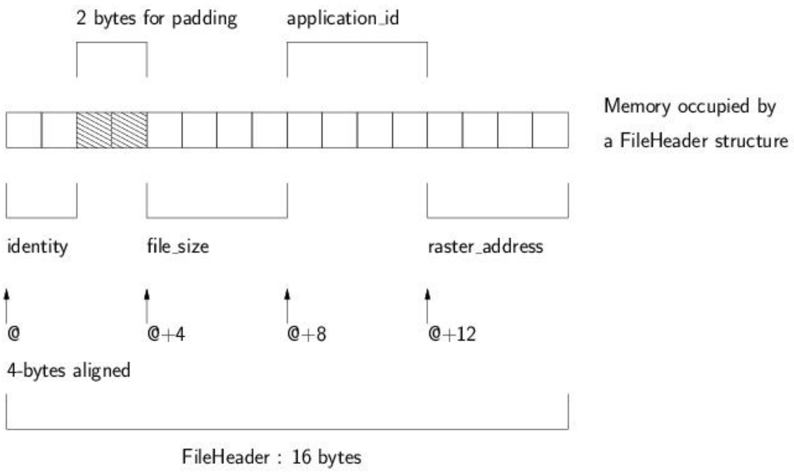


FIGURE 5.2 – The memory allocation for a FileHeader Structure when the addresses are 4-bytes aligned

Therefore when padding is enabled, *sizeof(FileHeader)* returns 16.

In our case, we need to fit the memory occupation and the binary data in the file in order to import/export it correctly. The 4-bytes alignment would imply a wrong binary reading/writing in

[2] An explanation in French can be found at https://fr.wikipedia.org/wiki/Alignement_en_m%C3%A9moire

a file since there are only 14 bytes stored in the file for the file header. This is why padding is disabled in this lab by setting the alignment value to 1 (one byte per one byte) using the macro *pack*. It leads to the memory allocation described in Figure 5.3.

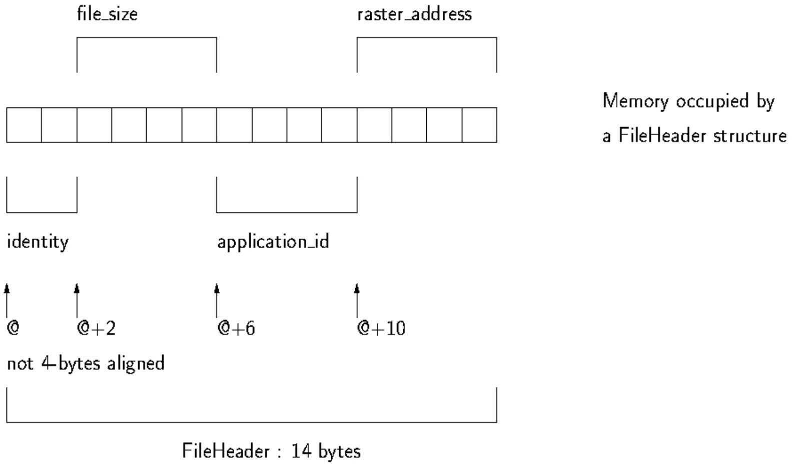


FIGURE 5.3 – The memory allocation for a FileHeader Structure when padding is disabled

question 7 Comment the line code with the pragma. If your program is not working any more, change your code to read each field of the FileHeader one by one or using a constant instead of the *sizeof(FileHeader)*.

LAB 6

Clion with Makefile

In order to get familiar with the build process and build tools, you will write your own Makefile for all practical works. It would have been possible to use a simple text editor to code and to compile using the make command. However tools provided by an IDE are very useful, such as the debugger, code completion,... You can also call some of them from the terminal but it is not user friendly. The solution at this level of experience in coding is to use an IDE (Integrated Development Environment) as Code::Blocks. IDE usually uses their internal configuration for compiling project, however *clion*can use a Makefile. So you will use *clion*[1] configured to use a Makefile you will write. Here are the steps to configure *clion*with a Makefile :

— If *clion*is not installed yet, install it with the following command :

```
$ sudo snap install clion
```

The following will present

1. first how to open a Makefile project with clion
2. then how it has to be use ;
3. finally, you will see how to add a target to run the program build with a Clion Makefile project.

6.1 Create a Clion project from a Makefile project

1. Go in the directory with your Makefile. Check there are the phony rules *all* and *clean* ;
2. Then click right on the Makefile and choose "Open with Clion"
3. Click on "OK" to accept the *clean* rule to clean the project
4. If the Makefile is well-written, the project is created.

6.2 How to use a Makefile under Clion

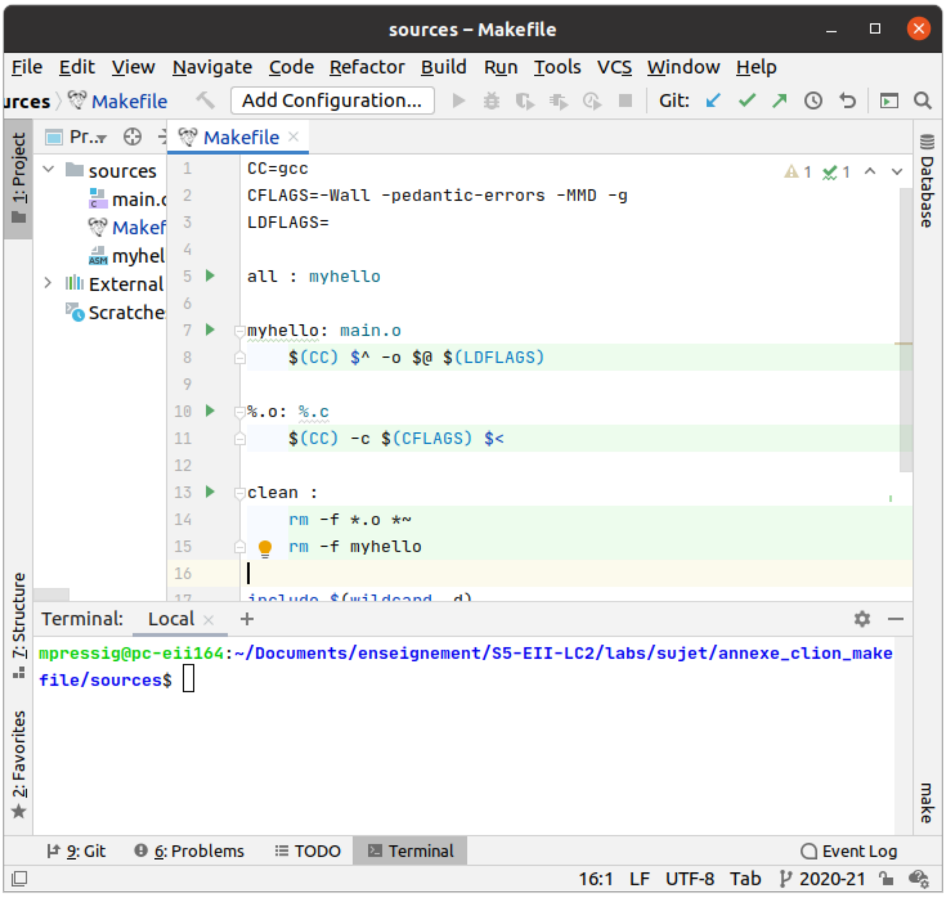
When editing the Makefile of a clion project, you can see green triangles in the margin at the left of the target name.

If you click on it, the action of making it is proposed and you can click on it.

Once you have done it at least one, you can run this make target by choosing it in the Configuration list and click on the "Run" button.

If you change the Makefile, do not forget to reload the project. You can click on the notification taht will appear on the Makefile when editing it. Another solution is to enable the auto-reload : choose " Go to Settings / Preferences | Build, Execution, Deployment | Build Tool.

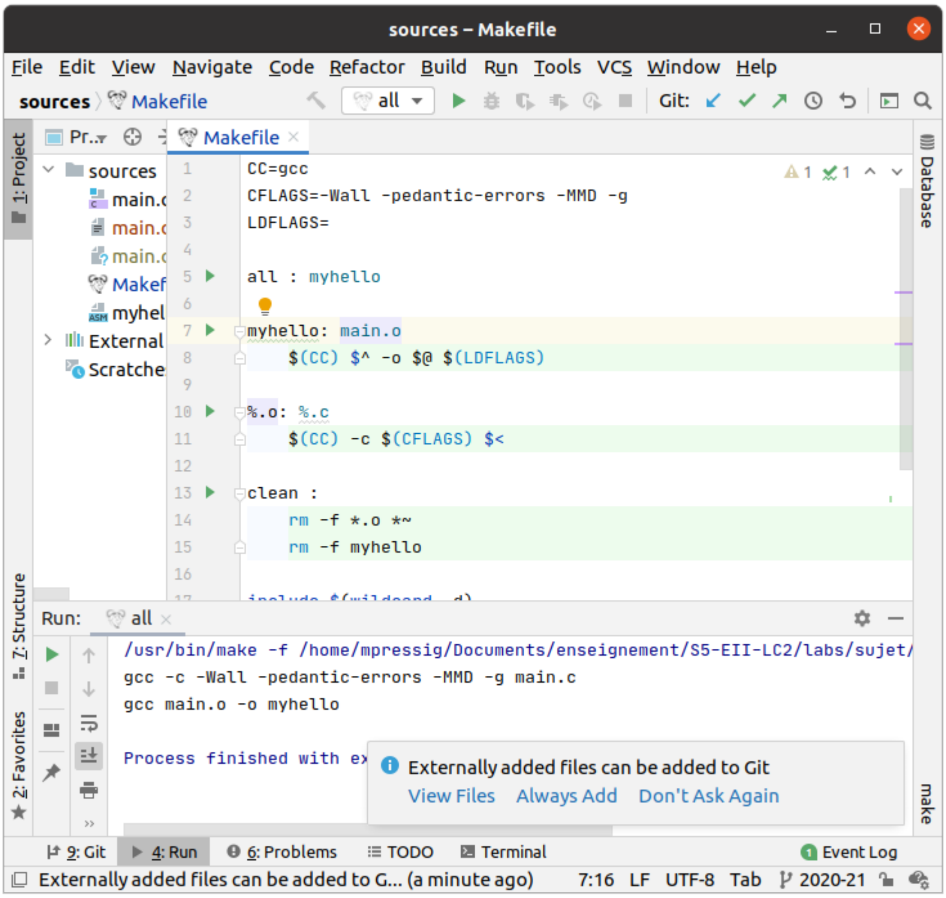
[1] which enables you to prepare your practical lesson on a Windows OS



```
CC=gcc
CFLAGS=-Wall -pedantic-errors -MMD -g
LDFLAGS=

all : myhello

myhello: main.o
	$(CC) $^ -o $@ $(LDFLAGS)

%.o: %.c
	$(CC) -c $(CFLAGS) $<

clean :
	rm -f *.o *~
	rm -f myhello

include $(wildcard *.d)
```

```
mpressig@pc-eii164:~/Documents/enseignement/S5-EII-LC2/labs/sujet/annexe_clion_make
file/sources$ 
```

FIGURE 6.1 – When the clion project with Makefile is loaded, it is possible to run each rule.



```
CC=gcc
CFLAGS=-Wall -pedantic-errors -MMD -g
LDFLAGS=

all : myhello

myhello: main.o
	$(CC) $^ -o $@ $(LDFLAGS)

%.o: %.c
	$(CC) -c $(CFLAGS) $<

clean :
	rm -f *.o *~
	rm -f myhello

include $(wildcard *.d)
```

```
/usr/bin/make -f /home/mpressig/Documents/enseignement/S5-EII-LC2/labs/sujet/
gcc -c -Wall -pedantic-errors -MMD -g main.c
gcc main.o -o myhello

Process finished with ex
```

Externally added files can be added to Git

Externally added files can be added to G... (a minute ago)

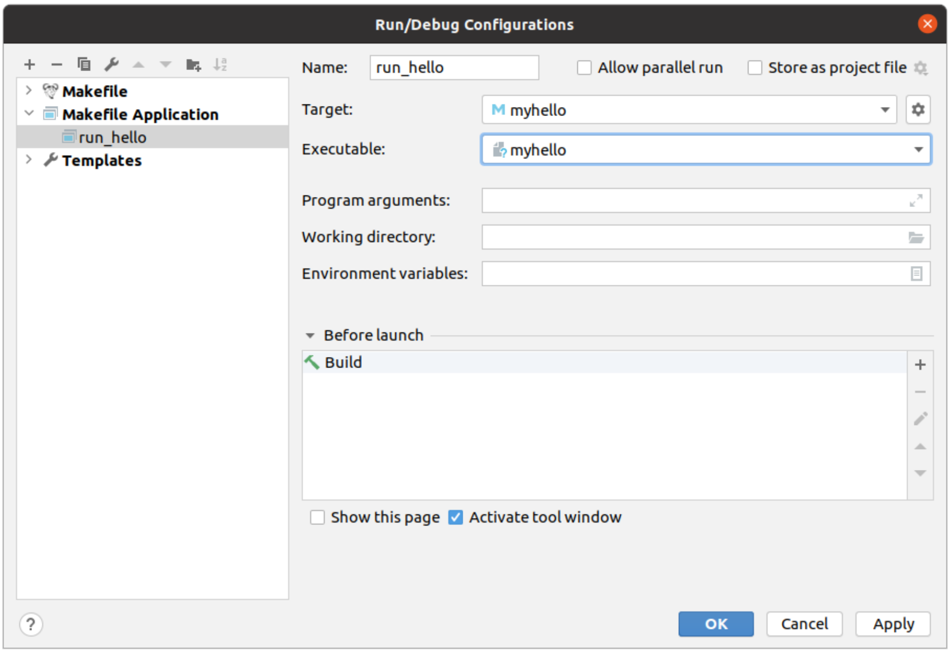
FIGURE 6.2 – The "all" rule is selected in the configuration list.

Now you know how to create a Makefile project under clion and how to run te make target. The last section is about how to launch a program that is build from the Makefile.

6.3 How to create a Configuration to run a program build from a Makefile

Go in the Configurations list and choose to edit them. lick on the "+" symbol and add a Makefile Application. You can configure like this :

1. Build the program
2. Choose a name for the configuration, for example "run_programName".
3. For the target, choose the one that builds the program.
4. For the executable, choose the one that has been build.
5. Click "OK"



Run/Debug Configurations

- Makefile
- Makefile Application
  - run_hello
- Templates

Name: run_hello □ Allow parallel run □ Store as project file

Target: myhello

Executable: myhello

Program arguments:

Working directory:

Environment variables:

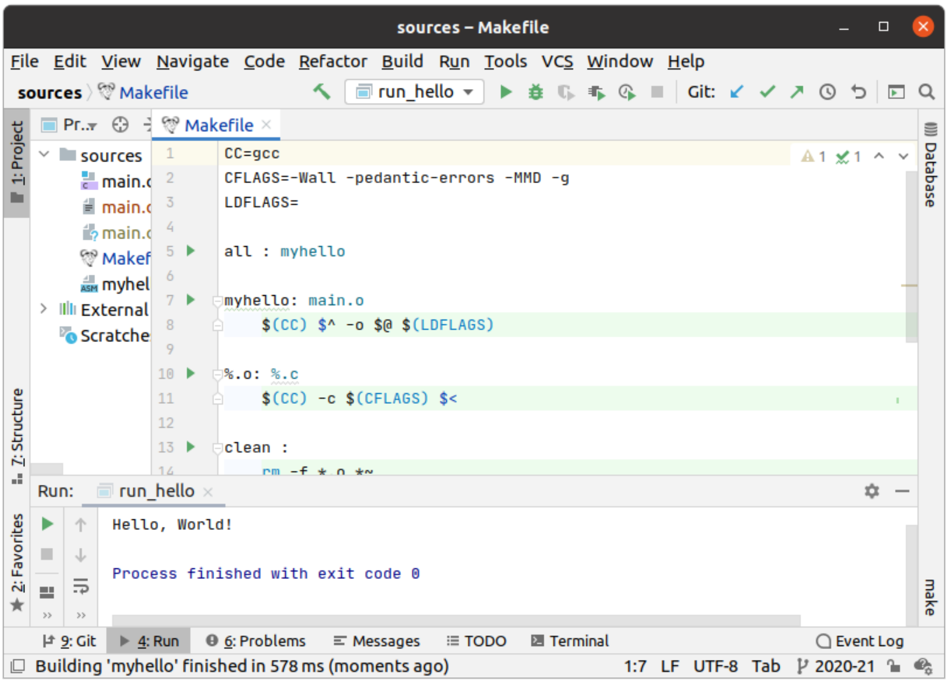
Before launch

Build

□ Show this page ☑ Activate tool window

FIGURE 6.3 – The configuration of the target to run *myhello*.

Now when choosing this configuration, you can run and debug the program. If you want to debug, do not forget the '-g' compilation option in your Makefile.



```
CC=gcc
CFLAGS=-Wall -pedantic-errors -MMD -g
LDFLAGS=

all : myhello

myhello: main.o
	$(CC) $^ -o $@ $(LDFLAGS)

%.o: %.c
	$(CC) -c $(CFLAGS) $<

clean :
	rm -f *.o *~
```

```
Hello, World!

Process finished with exit code 0
```

Building 'myhello' finished in 578 ms (moments ago)

FIGURE 6.4 – Once the target to run the prgram is done, it can be run and debugged.

INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7

Tél. +33 (0) 2 23 23 82 00
Fax +33 (0) 2 23 23 83 96

www.insa-rennes.fr

INSA

UNIVERSITÉ EUROPÉENNE DE BRETAGNE

Commission des Titres d'Ingénieur

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE