TP2: Tools Gcc and make

1 Getting started

The objective of this lab is to get familiar with the C language and its development chain based on the **gcc** compiler, and teach you how to create and use makefiles when you are developing programs in C.

2 A C program consisting of a single compilation file

1. Create a new directory for this lab inside your personal space and copy the contents of the directory **/share/l3info/CUnix/tp2** to it using the command **cp -r**
2. Inside the copied directory compile the **hms.c** program using the command :
   gcc -Wall filename.c
   What does the options **-Wall** means ?
3. Open the source file with an editor and correct the errors.
4. Recompile the program to verify that the program is now correct.
5. Use the command **ls -ail** to identify the file produced as result of the compilation process. What is this type of file ?
6. To specify a particular executable name for the produced file, we can use the **-o** option of the **gcc** :
   gcc -o progname -Wall filename.c
   Name the executable file of **hms.c** as **hms** (the executable files of Linux do not have the .exe extension as in DOS / Window).
7. Execute the program using **./** and verify that its functionality is correct.

3 A C program consisting of several compilation files

As a general rule, a compilation file consists of a file with extension **.c** containing the source code and a header file with extension **.h**, which allows to export the definitions of the identifiers of the module (data, functions, etc.).

When we compile a compilation file, we use the **-c** option of the compiler. This produces an object file with extension **.o** which is not directly executable.

1. Enter inside the directory **tp_liste**, which you have already copied from the previous exercise. This directory contains the directory **src** where the source files are stored and the directory **include** where the header files are stored. The source files contains the (partial) source code of the implementation of linked lists in C.

2. Compile the files **src/test_list.c** using the command :
   gcc -Wall -c src/test_list.c
3. This produces a compilation error, where does this error come from ? How to correct this error ?
   - Index 1 : we have a header file **list.h** which defines the variables and the Identifiers exported by the **list.c** stored in the directory **include**, and the directive **#include ″file.h″**.
   - Index 2: it is sometimes necessary to indicate to the compiler where to find the header files, if they are not in the same directory. To do so, we can use the **-I** option of the **gcc** :
   gcc -I./Include
   which indicates that you must search the **.h** header files in the **./include** directory.
4. After the problem is solved, compile (with the **-c** option) the **src/list.c** file. What is the output of this command ? Is this file an executable file ?
5. We have to connect the two object files to get the final executable program. Do so by using the command :
   gcc -o test_list test_list.o list.o
6. Execute the program and observe the results on the standard output.
7. The segmentation fault message indicates that the program has tried to access a memory address which was illegal, either because of an array overflow, or the use of an incorrect or not initialized pointer.
8. Recompile the file **test_list.c** using the **-Wall** option. What do you receive as result ?
9. Base on the warnings provided by the compiler, correct the errors in the module **test_list.c** and then check the correct functioning of the program.

4 The tool make

Each time you modify one of the compilation files (**.c** or **.h**), you must recompile all the dependent compilation files, and redo the linking between the object files to get an up-to-date executable file. This task is very time consuming to be done manually, especially when the program includes many compilation files (for instance, the kernel of the Linux operating system which contain several hundred C compilation files.).

The **make** program allows to automate this task by recompiling each time only the files that are required to be compiled.

1. Complete the **makefile** of the **tp3** directory to allow the correct compilation and linking of the compilation files **test_list.c** and **list.c** under the following instructions :
   — The **.o** files produced during compilation are stored in the **./obj** directory
   — The executable program is called **test_list** and it is stored in the **./bin**
   — That the object files and the executable are up-to-date with their dependent **.c** source code and **.h** header files
2. Check your **makefile** for correct operation by modifying one of the files and then re-lancing the **makefile** using the command **make**.
3. In the **makefile** file complete the clean entry so as to delete all object files **.o** and also the produced executable

4. In the **makefile** file complete the listing entry so as to produce a pdf file from the sources of the **.c** and **.h** files, using the sequence of commands :
   a2ps -o listing.ps include/list.h src/list.c src/test_list.c
   ps2pdf listing.ps
   rm listing.ps
5. Check that this rule works correctly by calling it with the command :
   make listing
6. If you want to develop a program using a "debugger", you must compile it (and all its modules) with the **-g** option as shown below :
   gcc -Wall -g -o test_list.o test_list.c
   To avoid having to modify all the makefile rules each time, we can define variables to be more generic. For example, by adding the following line at the beginning of the **makefile**, it allows to define a variable **CFLAGS**.
   CFLAGS = -Wall -g
   Then, we can refer to this variable by writing **$(CFLAGS)** the rules of the **makefile**, as in the example below :
   gcc $(CFLAGS) -c module1.c
   Modify the **makefile** so that calls to gcc are made with the **-g** option.