Foreword

In order to do this lab you must have access to the school's virtual machine (see wiki on Moodle).

In this exercise, you will cover the following topics:

- Writing a first C++ program ;
- Compiling and linking a program with g++ ;
- Study the variables and their type;
- Interacting with the command line ***[BONUS]***.

Advice

- Use `man` an especially `man 3` for the development pages ;
- The website `https://en.cppreference.com` is your most valuable friend for the Teaching Unit;

Part I
Compilation - Linking - Automation

1 Compiling a basic C++ file

Use the example file `hello.cpp` from the first lab that simply displayed "Hello World!" as a starting point.

```cpp
#include <iostream>

int main(int argc, char* argv[]) {
  std::cout << "Hello World!" <<std::endl;
  return 0;
}
```

Instruction

Compile the program with : `g++ -c hello.cpp -Wall`

Question 1

- What type of file is produced by the above command line ?
- What does the argument `-Wall` on the command line correspond to?

Instruction

Link the program with : `g++ -o hello hello.o` and run it with : `./hello`

Question 2

- What type of file is produced by the above command line ?

2 Linking against external libraries

Up to this point, our program has only be linked against the standard C and C++ libraries. However, sometime, the binary code for a given function is not included in these standard libraries and you have to link against an external one. One simple example of that is the Math library that contains the binary code for some useful math functions.

In this section we want to write a new program that compute the cosine function applied to a specific value (see expected output below).

```
$./cosinus
Cos(3.14) = -0.999999
```

We start by looking at the manual page for the cos function using `man 3 cos` (see extract below).

```
COS(3)      Linux Programmer's Manual     COS(3)

NAME
       cos, cosf, cosl - cosine function

SYNOPSIS
       #include <math.h>

       double cos(double x);
       float cosf(float x);
       long double cosl(long double x);

       Link with -lm.

   [...]

DESCRIPTION
       These functions return the cosine of x, where x is given in radians.

RETURN VALUE
       On success, these functions return the cosine of x.
```

Question 3

- What types are accepted as parameters for the cosine function ?
- Which header needs to be included to have the definition of the cosine function ?
- What argument need to be added to the linking command line to link against the math library ?

Instruction

Write the `cosinus.cpp` program that declares a double variable and compute the cosine of this variable[a].

Compile it with the usual g++ command line : `g++ -c cosinus.cpp -Wall`.

Link it against the math library using the correct command line option previously found.

[a]to have a similar output initialize the variable to 3.14

3 Automation

As C++ is a compiled language, the two steps compile & link are always needed before any program can be executed. For example, in the above code we need to compile the program each time the variable changes. It can be cumbersome to perform those task especially when the program relies on multiple classes that can be compiled separately. In this section we will introduce the concept of makefiles and the `make` utility that can automate some of those tasks. From the manual page of the `make` utility one can read :

```
DESCRIPTION
       The make utility will determine automatically which pieces of a large program need
          to be recompiled, and issue the commands to recompile them.  The manual
          describes the GNU implementation of make, which was  written by Richard
          Stallman and Roland McGrath, and is currently maintained by Paul Smith.  Our
          examples show C programs, since they are very common, but you can use make with
           any programming language whose compiler can  be run  with  a  shell command.
          In fact, make is not limited to programs.  You can use it to describe any task
          where some files must be updated automatically from others whenever the others
          change.

       To prepare to use make, you must write a file called the makefile that  describes
          the  relationships  among files in your program, and the states the commands
          for updating each file.  In a program, typically the executable file is updated
           from object files, which are in turn made by compiling source files.

       Once a suitable makefile exists, each time you change some source files, this
          simple shell command:

              make

       suffices to perform all necessary recompilations.  The make program uses the
          makefile  description  and  the last-modification  times  of  the  files to
          decide which of the files need to be updated.  For each of those files, it
          issues the commands recorded in the makefile.

       make executes commands in the makefile to update one or more target names, where
          name is  typically  a  program.   If no -f option is present, make will look
          for the makefiles GNUmakefile, makefile, and Makefile, in that order.
```

From now on we will rely on the make utility to build our applications as they become more and more complex.

To do so, it is important to understand the structure of the makefile. A typical makefile is organized as follow :

```
[target1]: [dependence1] [dependence2] ...
  [build instruction]
[target2] : [dependence3] ...
  [build instruction]
...
```

It is however important to notice that in that organization, a dependence for one target can be another target in the makefile. In order to give you a simple example, we will build the cosine program using the make utility.

Instruction

Create the following makefile :

```
cosinus: cosinus.o
  g++ -o cosinus cosinus.o -lm
cosinus.o: cosinus.cpp
  g++ -c cosinus.cpp -Wall
clean:
  rm -rf *.o cosinus
```

build your program with the make command :

```
$ make
g++ -c cosinus.cpp -Wall
g++ -o cosinus cosinus.o -lm
```

Question 4

- Why does the build starts with the second target of the makefile?
- What is the purpose of the clean target ?

Part II

BONUS - Command line - additional types

4 Reading the command line

Even if we can simplify the compilation & linking of our program thank to the `make` utility, it still needs to be edited each time we want to compute the cosine function over another value. One of the way to interact with a program is to add arguments to the command line. For example with the command `ls -l` the argument `-l` allows you to display an organized list of files in a directory.

As a matter of facts, every C++ program can read such arguments and use them if the `main` function is properly declared. In this bonus part we will learn how to read and use command line arguments.

Question 5

- What are the parameters of the `main` function ?
- What are their types ?

Instruction

Modify the first "Hello World" program so that the output shows the number of arguments on the command line (see examples below).

```
$ ./hello titi toto
The program has 3 arguments
$ ./hello titi toto tata
The program has 4 arguments
```

Question 6

- Why is there always one more argument [a] ?

[a]here in the example the arguments are "titi" and "toto" and the program outputs 3

Instruction

Modify the program so that the given arguments are printed on the console (see example below).

```
$ ./hello titi toto tata
The program has 4 arguments
Argument 0 : ./hello
Argument 1 : titi
Argument 2 : toto
Argument 3 : tata
```

5 Using the command line

Now that we know how to read the command line, we want to use some of the arguments as variables in our program. Specifically, we want to use the first argument (after the program name) to be the number of time our program will print a name provided as the second argument (see exemple below).

```
$ ./hello 3 toto
Hello toto
Hello toto
Hello toto
```

Question 7

- Which variable points to the number of times we want to print the name ?
- What is the type of this variable ?
- From the precedent answers, which operation has to be performed in order to iterate over the first argument ?

Instruction

Write the corresponding program[a].

[a] check the manual for the `atoi()` function

6 Merge everything

Instruction

Rewrite the cosinus program so that it uses the first argument of the command line as the variable on which the cosine is computed[a].

```
$./cosinus 3.14
Cos(3.14) = -0.999999
```

[a] check the manual for the `atof()` function

Warning

Always perform sanity checks before parsing user input data as it can lead to serious security issues and unforeseen consequences. To illustrate this try to run your cosine program without an argument, it should trigger a segmentation fault error (as shown below).

```
$./cosinus
Segmentation fault (core dumped)
```

Part III
Appendices : Useful commands

```
$man COMMAND                #display the manual page for the given COMMAND
$man 3 FUNCTION             #display the developer manual for the given FUNCTION
$g++                        #GNU project C and C++ compiler
$make                       #GNU make utility to maintain groups of programs
$ldd                        #print shared object dependencies
$strace                     #trace system calls and signals
$strings                    #print the sequences of printable characters in files
$gdb                        #The GNU Debugger
```