LAB 1 : MakeFile

I . Compilation

The following files are being provided to you.
hello.cpp : Hello World Program
fibonacci.cpp : Program to generate n fibonacci numbers
To run the program, you need to compile it into an executable.

Task:

- Compile the **hello.cpp file** into an executable by running the following command
  $ g++ <cpp-file> -o <name of your executable>

- Run the executable
  $ ./<name of your executable>

II. Compilation Flags

General Structure for Compilation :
$ g++ <flags> <cpp files> -o <executable name>

Warning Flags:

-Wall : It will cause the compiler to warn you about technically legal but potentially problematic syntax. Example : Uninitialized and unused variables

-Werror : It will force the compiler to treat all compiler warnings as errors. Code won't be compiled unless these errors are fixed.

-Wextra : It adds a few more warnings that are not covered by -Wall. Example: Unused function parameters, empty if else statements. **Note:** Try to use this with -Wall and without -Wextra to understand the concepts better

Task:

1. Compile the fibonacci.cpp file by adding the above warning flags in the command. Fix the errors that come up.

Sanitizers:

Problems like memory leak, stack/heap corruption are not caught by warning flags. Hence, it is a good coding practice to use sanitizers.

-g : It provides more specific debugging information.

Optimization Flags:

These flags speed up the run-time of the code. Example: -O0, -O1, -O2, -O3

Note: Read more about Sanitizers and Optimization Flags.

III. MakeFiles

Structure of Makefile :

<target>: <dependencies>
[tab] <shell_command>

Target : Name of an output file generated by this rule
Dependencies: Files or other targets that this target depends on.
Shell command: Command you want to run when the target/dependencies are out of date.

Command to run Makefile: make <target>.

Task: (After every Task (2-7), run the make command)

1. Create an empty Makefile
2. **Link** the files factorial.cpp hello.cpp and main.cpp into a single executable file

   Example :

   ```
   myTarget :  program1.cpp program2.cpp
           g++  program1.cpp program2.cpp -o hello
   ```

3. **Variables in Makefiles:** Define variables so that you can reuse flags and names that are commonly used. Ex: **VAR_NAME = "myVariable"** will define a variable that can be used as **$(VAR_NAME) or ${VAR_NAME}** in your rules

   Use variables for declaring the CXX compiler(g++),CXXFLAGS(-Wall etc), OBJECTS(for all .o files - Example : OBJECTS = program1.o program2.o)

4. **Automatic Variables:** Special variables that can have a different value for each rule in a Makefile.

   $@ -> current rule's target.
   $^-> names of all of the current rule's dependencies including the spaces between them.

   Example: hello: main.cpp program1.cpp
   g++ $^ -o $@

   Use automatic variables to simplify the Makefile further

5. **Simplifying the linking process:** It will be much faster to generate intermediate .o files or object files and separately link the .o files together into an executable.
   In the Makefile, create targets main.o factorial.o and helloworld.o
   Example:

```
hello:  main.o program1.o
        g++  main.o program1.o -o hello
main.o: main.cpp
        g++  -c main.cpp
```

6. **Phony Targets**: Targets that themselves create no files.They provide shortcuts for common operations, like making all the targets in our Makefile or getting rid of all the executables that we made.

   Include this line before any targets in your Makefile to mark targets as phony :

   .PHONY: <target_name>

   Here are some common phony targets that we'll be using in this course:

   a) **all target**

   To make all of the executables in our project simultaneously. There is no need to include shell commands for all, because by including each target (target1, target2) as dependencies for the all target, the Makefile will automatically build those targets in order to fulfill the requirements of all.

   Example: all: hello

b) clean target

To get rid of all the executables (and other files we created with make) in our project. Example:

clean:
    rm -f <my_executable_name> *.o

Note: Do not include .cpp or .h files in the rm -f command.

7. Adding more files and modifying the makefile:

a) Try to compile squareroot.cpp file from the command line.
b) Fix the error by including the required cmath library using the command:
   g++ -c squareroot.cpp -include cmath
c) Include squareroot.cpp file into the final executable created with **Task 2-7**
   Hint : Add declaration of the squareroot function in the header file
   Call the squareroot function from main file
   Create squareroot.o target in the makefile **(Refer to Task 5)**