SUBJECT: **ADVANCED ROBOTICS**	SEMESTER: OT23
ASSIGMENT: **Activity 1.3**	
WORK FORMAT: **Teams**	
DOCUMENT FORMAT: **Digital - Blackboard**	
MADE BY: M.Sc. GILBERTO ALFONSO MONTES RAMIREZ	SEPTEMBER 2023

Instructions I. Report a PDF document with a front cover + the pages with contents of your work. Include the honor code in the cover and consider using IEEE format for references.

Instructions:

1. **Compile and Run C++ Code on the Command Line. Code an small C++ program to obtain the product of two integers (just like we did in C)**
    1. Search for the command line instructions to compile and run C++11 code using
        a. The command **g++**
        b. The command **cmake**
    2. **(OPTIONAL TO YOU)** Investigate how to compile C++11 and .h code (user defined libraries) using
        a. *The command* **g++** (We did it with a C-code example in class, now you do it with your C++ code)
        b. *The command* **cmake**
        c. You may use build a C++ program to obtain de product of two integers withi these file structure **main.cpp yourHeader.h yourHeader.cpp.** Then apply 2.a and 2.b
2. **About Threading in Python**
    a. Investigate how to pass arguments on threat calls & run this small example.
        a. Generate a random integer & display it on the screen in one thread, also pass this number to a second thread where this number is displayed on the screen. You can display something like this from the threads correspondingly:

        i. **The sent number is:**
        ii. **The received number is:**
3. **Interfacing among C, C++, and Python Programming Languajes**
    a. Investigate how to run Python Code from C++
    b. Show a simple program that passes two numbers from C++ to Python where they are added and displayed.
4. **Always use References**
    a. Use IEEE format
5. **Report a document to Blackboard**
    a. Use the link of this activity to report your document.
    b. **NOTE: To report in Blackboard, convert your document to PDF format**

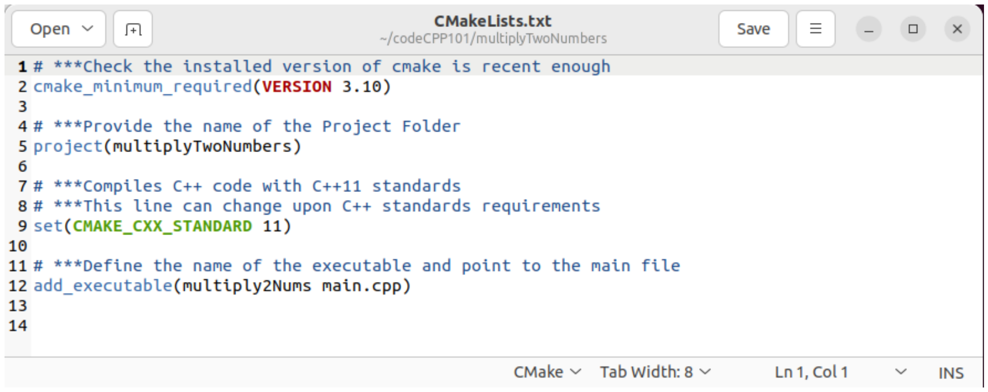
== Compiling C++ via Terminal in N steps

1. Create a Project Folder for you C++ code
2. Create and save the main C++ in that folder
3. Open a Linux Terminal and navigate to the project folder
    a. *(in my pc) gil@Isaiah41:~/codeCPP101/multiplyTwoNumbers*
4. Now, type the Linux shell command
    a. g++ -std=c++11 -o *yourExecutableName* **main.cpp**
    b. Note: Change "yourExecutableName" to the name you want for the executable file.
5. Probe you compile the file correctly
    a. Within the path of the project, type the command ls
    b. *(in my pc) gil@Isaiah41:~/codeCPP101/multiplyTwoNumbers$ ls*

Previous steps should be enough to obtain an executable file of your C++ program.

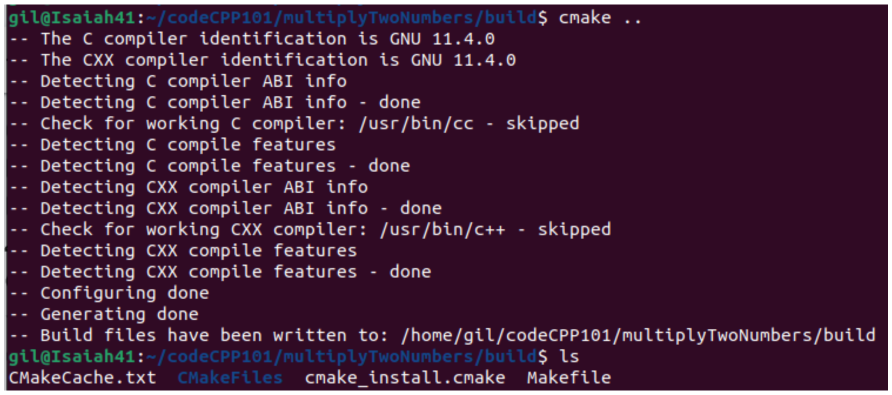
== **Compiling C++ via cmake (make sure you follow up the instructions correctly, be careful with names)**

1. You may need to install **cmake** to your Linux system
   a. …$ sudo apt install cmake
2. You create a Project Folder for the C++ file you are to code
3. Create and save your main.cpp file in such project folder
4. Create and save a file named: CMakeFile.txt
5. Code the following lines in the CMakeFile.txt file



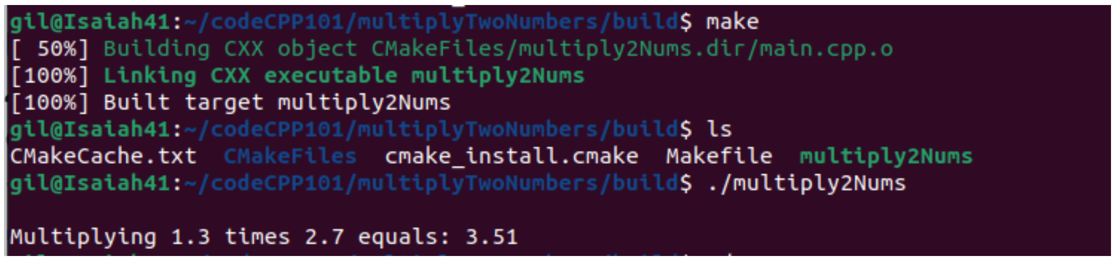
```
# ***Check the installed version of cmake is recent enough
cmake_minimum_required(VERSION 3.10)

# ***Provide the name of the Project Folder
project(multiplyTwoNumbers)

# ***Compiles C++ code with C++11 standards
# ***This line can change upon C++ standards requirements
set(CMAKE_CXX_STANDARD 11)

# ***Define the name of the executable and point to the main file
add_executable(multiply2Nums main.cpp)
```

6. *RECOMMENDED*: Within the same project folder create a new folder called: build
   a. At this point you should have these:
   b. A **project folder** and inside this folder de **main.cpp** file, the **CMakeFile.txt** file along with another folder named **build**
7. In the terminal, navigate to the **build** folder and type **cmake ..**
   a. gil@Isaiah41:~/codeCPP101/multiplyTwoNumbers/build$ cmake ..
   b. Then you should see something like. Check the result with *ls*.



```
gil@Isaiah41:~/codeCPP101/multiplyTwoNumbers/build$ cmake ..
-- The C compiler identification is GNU 11.4.0
-- The CXX compiler identification is GNU 11.4.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/gil/codeCPP101/multiplyTwoNumbers/build
gil@Isaiah41:~/codeCPP101/multiplyTwoNumbers/build$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile
```

8. In the previous step the process you created compilation files. Now it is time to compile the application using the command **make** in this way
   a. *gil@Isaiah41:~/codeCPP101/multiplyTwoNumbers/build$ make*
   b. Checkout the result

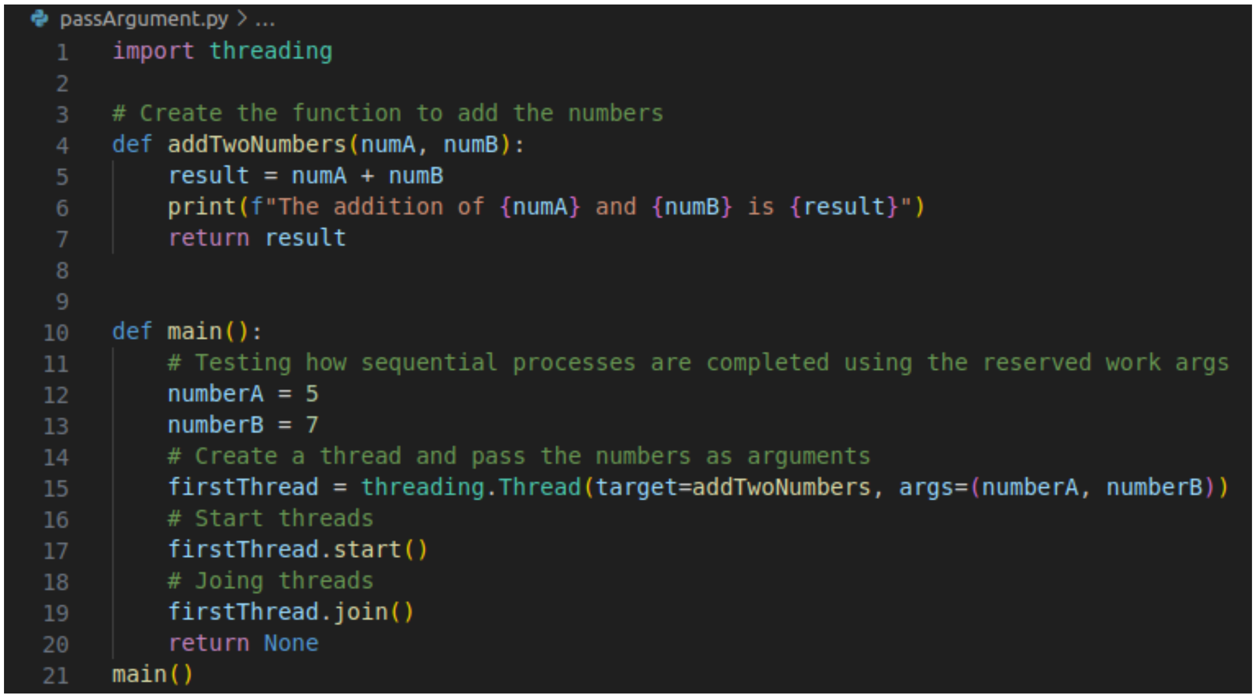


```
gil@Isaiah41:~/codeCPP101/multiplyTwoNumbers/build$ make
[ 50%] Building CXX object CMakeFiles/multiply2Nums.dir/main.cpp.o
[100%] Linking CXX executable multiply2Nums
[100%] Built target multiply2Nums
gil@Isaiah41:~/codeCPP101/multiplyTwoNumbers/build$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile  multiply2Nums
gil@Isaiah41:~/codeCPP101/multiplyTwoNumbers/build$ ./multiply2Nums

Multiplying 1.3 times 2.7 equals: 3.51
```

   c. Observe the green-colored file name that showed up to demonstrate this process has created the executable file we were looking for. Note: the name of this file **multiply2Nums** is the name that was given in the line 12 of the CMakeFile.txt.
9. Finally, you can run the executable with ./ as depicted previously.

== About Threading in Python

IMPORTANT NOTE: The main objective in this section of the activity was to pass arguments on thread calls.



passArgument.py > ...

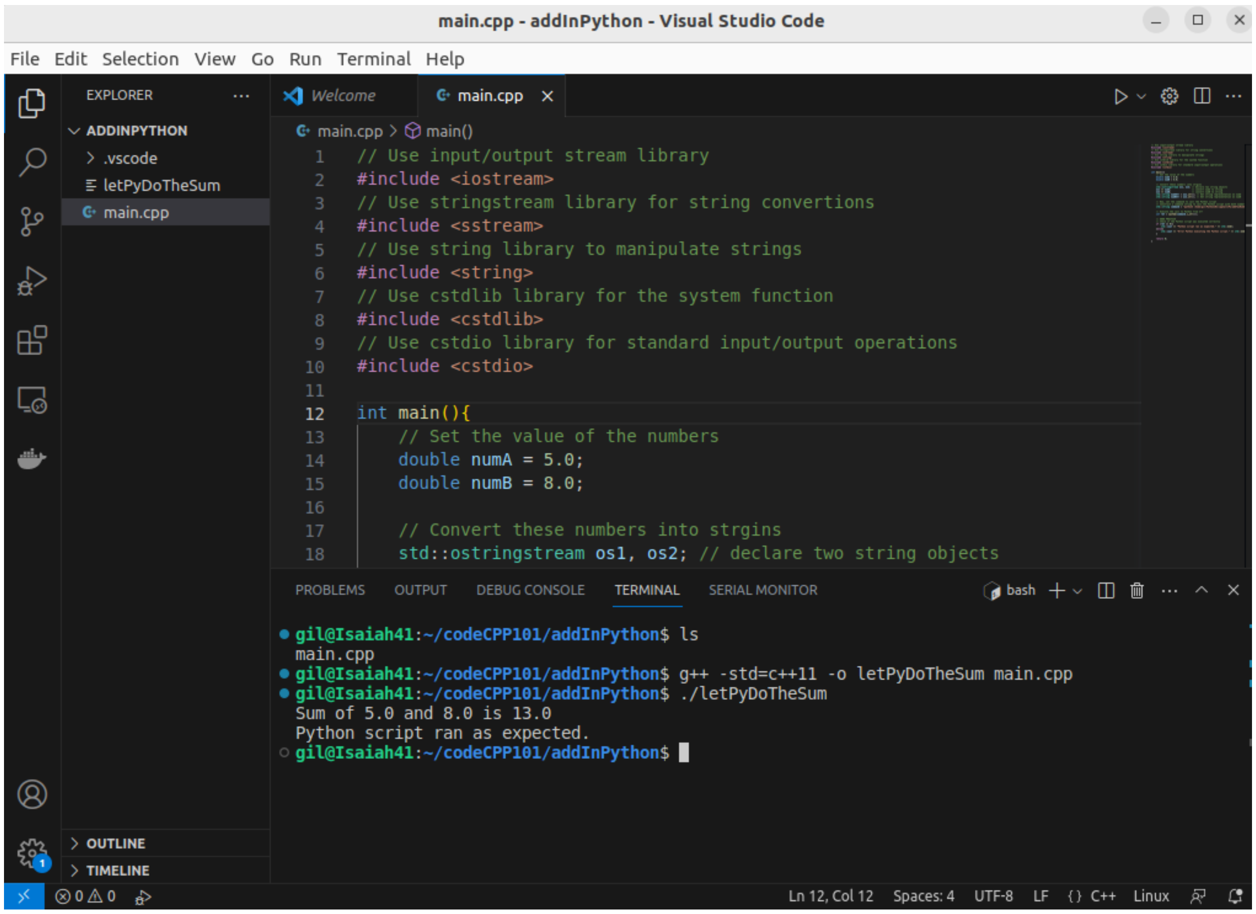
```python
import threading

# Create the function to add the numbers
def addTwoNumbers(numA, numB):
    result = numA + numB
    print(f"The addition of {numA} and {numB} is {result}")
    return result


def main():
    # Testing how sequential processes are completed using the reserved work args
    numberA = 5
    numberB = 7
    # Create a thread and pass the numbers as arguments
    firstThread = threading.Thread(target=addTwoNumbers, args=(numberA, numberB))
    # Start threads
    firstThread.start()
    # Joing threads
    firstThread.join()
    return None
main()
```

For the 1st Partial TEST this will be the knowledge I expect you to show in the case a threading problem comes in the exam. What I mean is that I would not be evaluating something like the instruction I wrote about this exercise. That was my mistake. I apologize to you for this inconvenient. So please, follow up this threading topic in the way I am doing it.

The .py file for this activity is available in Blackboard.

== Interfacing among C, C++, and Python Programming Languages

Check out the programs available in Blackboard. Here I show you all the output after calling Python from C++


main.cpp - addInPython - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
ADDINPYTHON
.vscode
letPyDoTheSum
main.cpp
Welcome
main.cpp
main.cpp > main()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SERIAL MONITOR
bash
OUTLINE
TIMELINE
Ln 12, Col 12 Spaces: 4 UTF-8 LF {} C++ Linux

```cpp
// Use input/output stream library
#include <iostream>
// Use stringstream library for string convertions
#include <sstream>
// Use string library to manipulate strings
#include <string>
// Use cstdlib library for the system function
#include <cstdlib>
// Use cstdio library for standard input/output operations
#include <cstdio>

int main(){
    // Set the value of the numbers
    double numA = 5.0;
    double numB = 8.0;

    // Convert these numbers into strgins
    std::ostringstream os1, os2; // declare two string objects
```

```
gil@Isaiah41:~/codeCPP101/addInPython$ ls
main.cpp
gil@Isaiah41:~/codeCPP101/addInPython$ g++ -std=c++11 -o letPyDoTheSum main.cpp
gil@Isaiah41:~/codeCPP101/addInPython$ ./letPyDoTheSum
Sum of 5.0 and 8.0 is 13.0
Python script ran as expected.
gil@Isaiah41:~/codeCPP101/addInPython$
```