
EEE8126: Computer Systems and Microprocessors

Lab 1: Review of functions, arrays, loops and introduction to profiling/debugging

Module Instructor: Dr Rishad Shafik

[[You are welcome to discuss your Exercises of Session 1 during this lab session]]

Exercise I: Review of functions and parameters

Recommended Time: 20 Mins Maximum

Aims:

- To understand how functions are used for modular and structured C++ programs
- To understand how call by value is made using C++ functions

Follow the instructions below and try to do accordingly-

- DOWNLOAD** the source code of [check_prime.cpp](#).
- REVIEW** the source code of `check_prime.cpp` using Notepad++ (Start->type "Notepad++")
Go through each line to understand how the code is organized. Check the following:
 - An integer data type *number* will be used to read and store the target number
 - The standard IO functions *cin* and *cout* will allow input and output from the console.
 - The *number* will then be used as a parameter of `check_prime` function; note how `check_prime` function is first declared as a prototype; and then also defined separately.
 - Note that `check_prime` function is capable of taking some parameters and capable of generating a return type as well.
- COMPILE** the source code of `check_prime.cpp`:
 - Start Cygwin command shell through Start->All Programs->Cygwin->Cygwin Bash Shell
 - In the Cygwin shell type: `g++ -Wall check_prime.cpp -o check_prime`
The `-Wall` option enables all the warnings.
The `-o` option enables specification of the output executable

Your compilation should generate an executable called `check_prime` (`check_prime.exe` in Cygwin)

- EXECUTE** the `check_prime` executable by typing the following in the Cygwin shell
`./check_prime`

(or `./check_prime.exe` in Cygwin)

- OBSERVE** the output with different inputs and see how `check_prime()` function evaluates the prime numbers through the flag

Exercise II: Review of array and loop

Recommended Time: 25 Mins Maximum

Aims:

- To understand how arrays are used for organised data storage

b. To understand how loops can be used in conjunction with arrays to ensure modular data processing

Follow the instructions below and try to do accordingly-

6. **DOWNLOAD** the source code of [matrix_mult.cpp](#).
7. **REVIEW** the source code of `matrix_mult.cpp` using Notepad++ (Start->type "Notepad++")

Go through each line to understand how the code is organized. Check the following:

- Two arrays `a` and `b` will be used to store the input matrices; Array `C++` will be used to store the output matrix
- Using loops these arrays and their elements can be initialised or updated

8. **COMPILE** the source code of `matrix_mult.cpp` by
 - a. Start Cygwin command shell through Start->All Programs->Cygwin->Cygwin Bash Shell
 - b. In the Cygwin shell type: `g++ -Wall matrix_mult.cpp -o matrix_mult`

The `-Wall` option enables all the warnings.

The `-o` option enables specification of the output executable

Your compilation should generate an executable called `matrix_mult`

9. **EXECUTE** the `matrix_mult` executable by
`./matrix_mult`
10. **OBSERVE** the output and see how `a`, `b`, `C++` arrays are used by the loops to instantiate values and to evaluate the outputs

Exercise III: Review of C++ functions and array

Recommended Tim: 45 Mins Maximum

Aims:

- a. To be able to write a simple C++ program using function and loops
- b. To be able to incorporate arrays when needed

Follow the instructions below and try to do accordingly-

11. **DOWNLOAD** the source code of [example3.cpp](#).
12. **REVIEW** the source code of `example3.cpp` using Notepad++ (Start->type "Notepad++")
Go through each line and comment to see where you need to insert your codes and expressions
13. **COMPILE** the completed source code of `example3.cpp` by
`g++ -Wall example3.cpp -o example3`
Your compilation should generate an executable called `example3`
14. **EXECUTE** the `example3` executable by
`./example3`
15. **OBSERVE** the output and compare that with the source code of `find_maximum.c`

Exercise IV: Profiling C++ programs

Recommended Tim: 25 Mins Maximum

Aims:

a. To learn basic C++ gC++ profiling (using gprof)

16. **DOWNLOAD** the source code of [gprof_example.cpp](#).

17. **REVIEW** the source code of `gprof_example.cpp` using Notepad++ (Start->type "Notepad++")
Three functions with three different execution times

18. **COMPILE AND EXECUTE** the source code of `gprof_example.cpp` by

```
g++ -pg -Wall gprof_example.cpp -o gprof_example  
./gprof_example
```

You can now list the files generated and you will see a `gmon.out` file, which is new, using
`ls -l`

19. **PROFILE** your executable by using this `gmon.out` file by

```
gprof gprof_example > analysis.txt  
less analysis.txt
```

[Cygwin may need `.exe` extension for the executables]

20. **EXAMINE** the analysis information to find out how different function calls have been used in your code; see `gprof -help` for more information

21. Check the overall execution time and cross-validate using

```
time gprof_example
```

Exercise V: Debugging C++ programs (take home task)

Self-study this link (<http://tinyurl.com/ybuverka>) and see if you can carry out debugging on Example I-III using `gdb` or not.