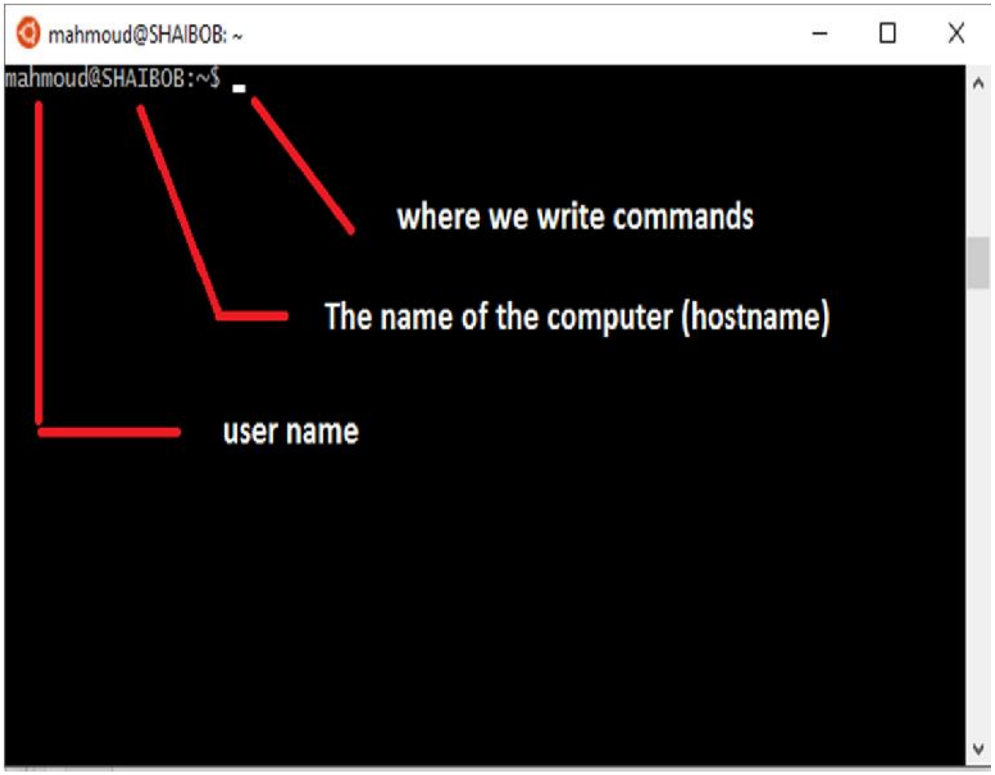
Introduction to Linux and C++

Mahmoud Abdul Galil

Tutorial – Wednesday September 21, 2016

Bash Shell



```
mahmoud@SHAIBOB:~$ _
```

- Bash shell is a UNIX shell and a command processor
- It’s the default command language for Linux consoles (that ugly black window on the left)
- It’s an interface between the user and the OS
- Get used to it, you’ll be seeing it for quite a while ! :P

Useful Command-Line Tools

- “ ls “ command for listing files
- “ cd ” command for changing directories
- “mkdir “ command for creating directories
- “ man “ command for displaying manuals of other commands
- “ cp “ command for copying files
- “ mv ” command for moving and renaming files
- “ cat “ command for showing file contents
- “ nano “ text editor for creating and editing scripts and text files
- “ sudo “ command for administrator priviliges

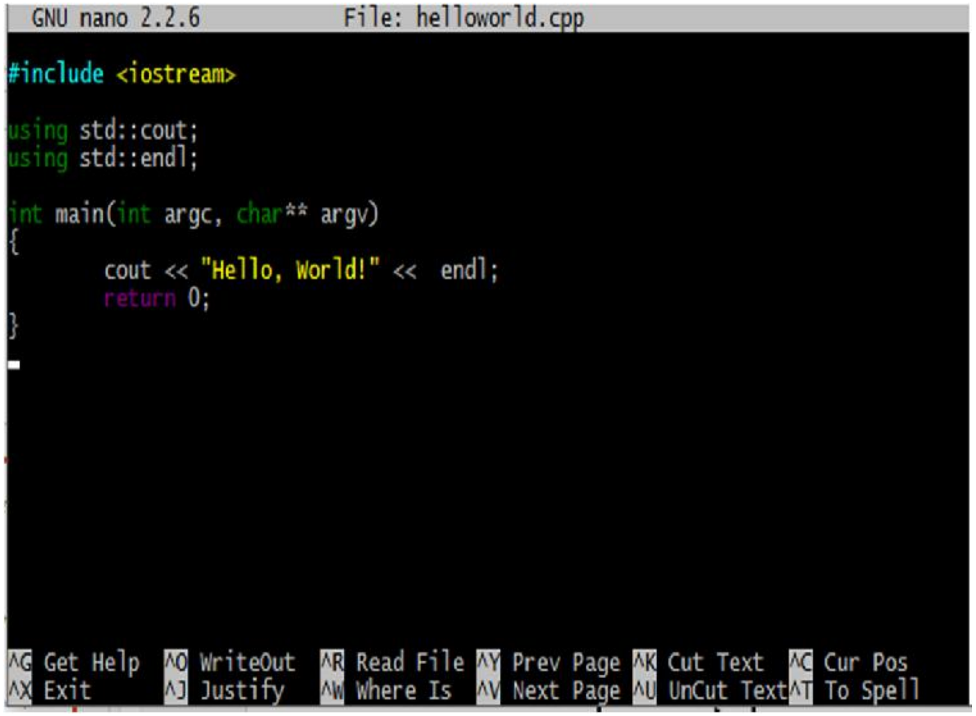
Package Management System: apt-get

- Programs in Linux are called packages
- Packages are installed using a package management system
- Official package management system in Ubuntu is apt-get
- Example use: sudo apt-get install kate
  - sudo => because installing packages requires modifying system files and thus requires system admin privileges
  - apt-get => command-line tool for managing packages: installing & removing .. etc
  - install => option to install a package – alternatives? remove & purge
  - kate => the name of the package we want to install, here it's the text editor kate
- Learn the rest by yourselves =D ➔ http://www.tecmint.com/free-online-linux-learning-guide-for-beginners/

C++ Programming Language

- Programming Languages VS. Scripting Languages
- Programming Concepts:
  - Code: What we usually write such as C++ and Python
  - Compiler: A program that converts the Code into executables
  - Executable: A program that is the output of compiling a code
  - Library: A chunk of code that can be reused in independent executables
  - Linker: A program responsible for linking executables with the libraries they need

C++ First Example: Hello World



```
#include <iostream>

using std::cout;
using std::endl;

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

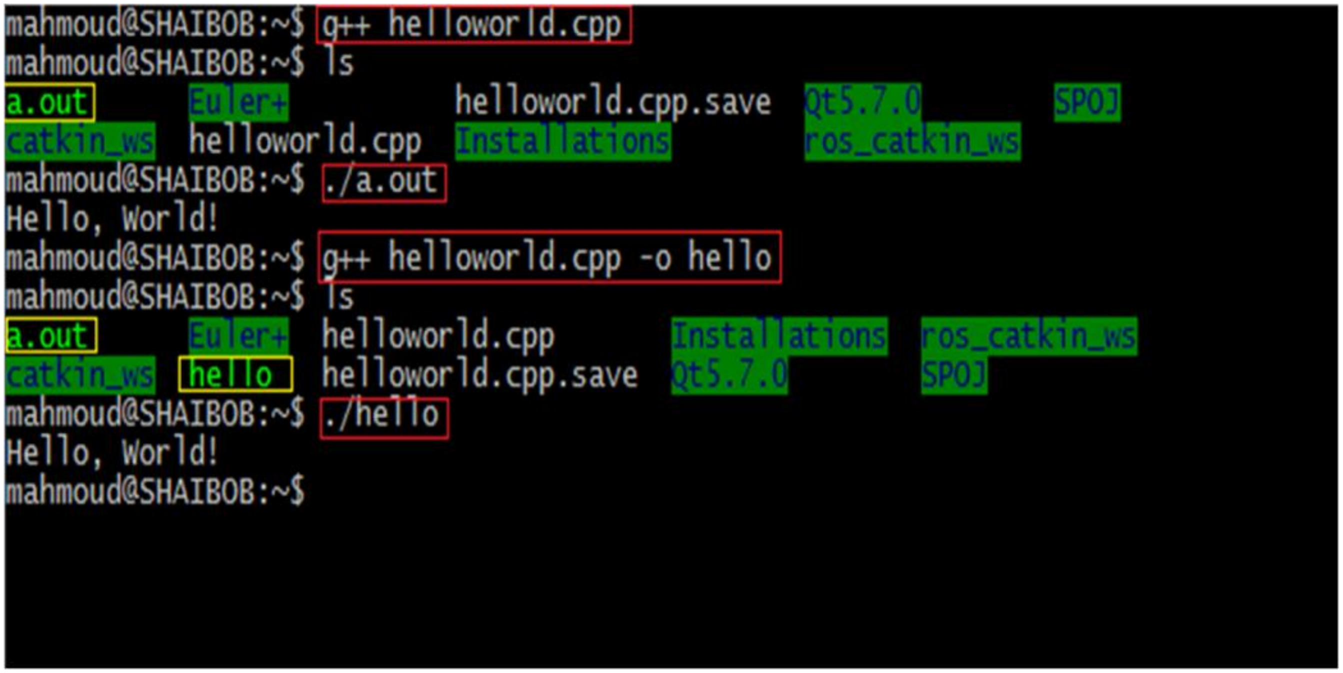
- #include <iostream>
  - Here we include the file called iostream from the standard c++ library to use cout and endl.
- using std::cout;
  - Here we declare that we will use the function cout from the namespace of std
- using std::endl;
  - Here we declare that we will use the function endl from the namespace of std

- int main(int argc, char** argv)
  - Every program must have a main function. The arguments are standard arguments used to pass command line inputs to the executable ➔ more to come on this
- cout << “Hello, World!” << endl;
  - We are using cout to print the string “Hello, World”, exploiting the operator (<<) → check the documentation
- return 0;
  - Here we return 0 as the return value of the function main

General Notes

- Keep an eye on your semi-colons !! (syntax generally)
- Never open a bracket without closing it !!
- Check the documentation of the function you are using
- Data types matter !! → Check each data type for its uses

Compiling our Program



```
mahmoud@SHAIBOB:~$ g++ helloworld.cpp
mahmoud@SHAIBOB:~$ ls
a.out      Euler+          helloworld.cpp.save  Qt5.7.0        SPOJ
catkin_ws  helloworld.cpp  Installations        ros_catkin_ws
mahmoud@SHAIBOB:~$ ./a.out
Hello, World!
mahmoud@SHAIBOB:~$ g++ helloworld.cpp -o hello
mahmoud@SHAIBOB:~$ ls
a.out      Euler+  helloworld.cpp       Installations  ros_catkin_ws
catkin_ws  hello   helloworld.cpp.save  Qt5.7.0        SPOJ
mahmoud@SHAIBOB:~$ ./hello
Hello, World!
mahmoud@SHAIBOB:~$
```

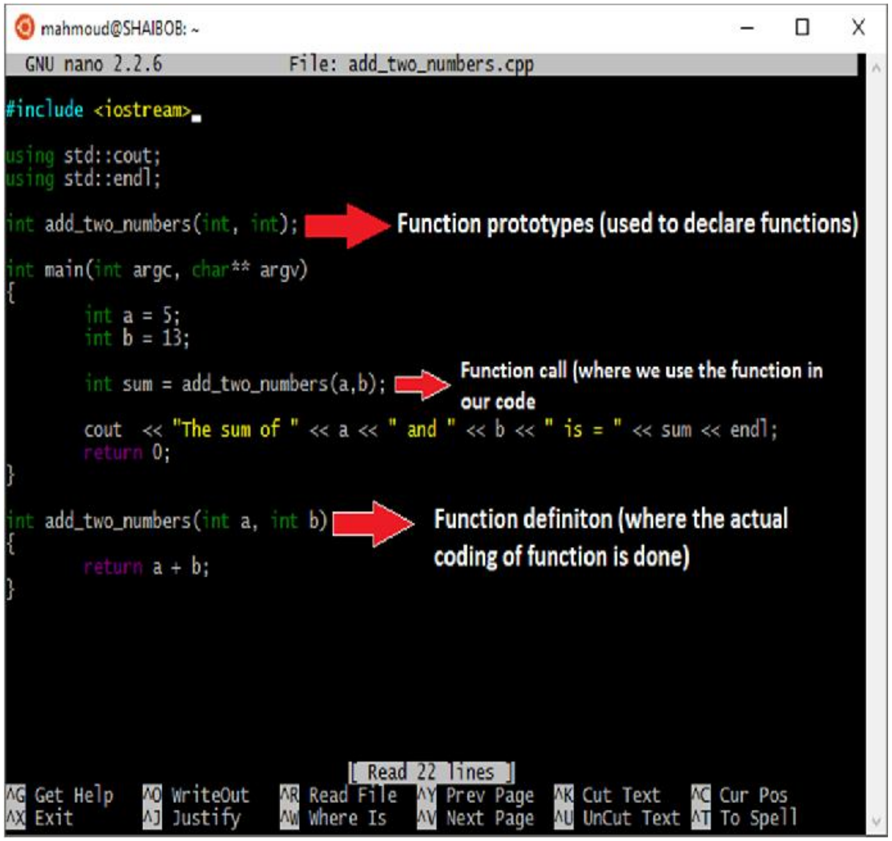
First, we didn't specify the name of executable

Then, we specified the name as "hello" using the option –o

We run an executable by writing its name preceded by " ./ ", without the quotation marks, inside the directory where it resides.

Check the command-line tool "chmod" for info on permissions

C++ Second Example: Add two numbers

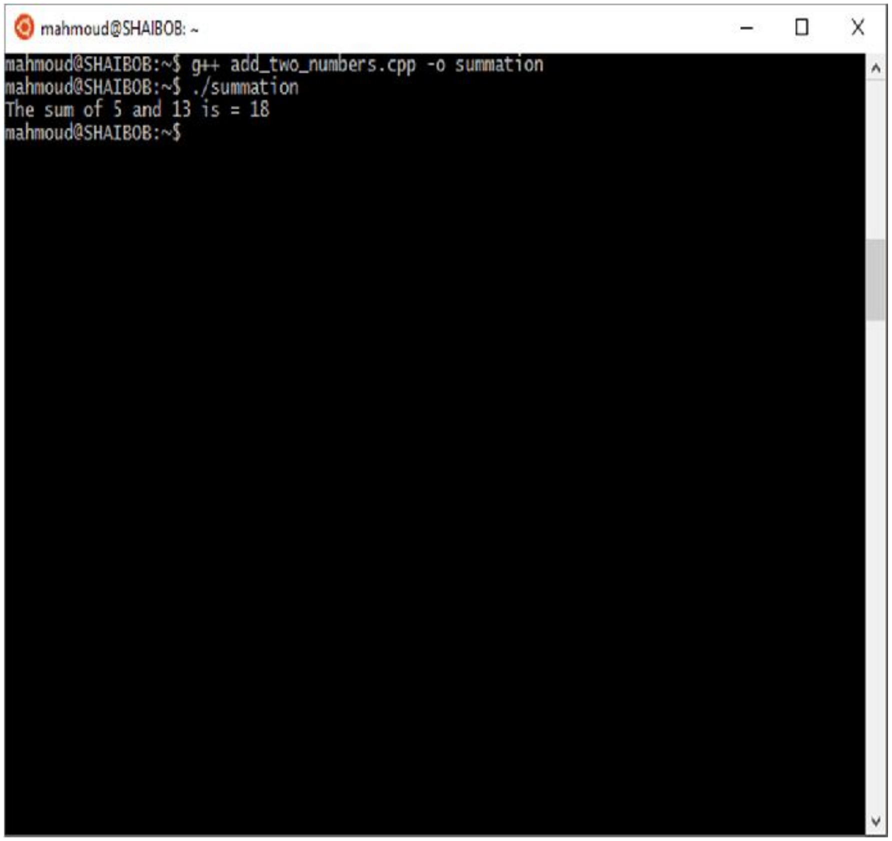

mahmoud@SHAIBOB: ~
GNU nano 2.2.6 File: add_two_numbers.cpp
[Read 22 lines]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

```
#include <iostream>

using std::cout;
using std::endl;

int add_two_numbers(int, int);

int main(int argc, char** argv)
{
        int a = 5;
        int b = 13;

        int sum = add_two_numbers(a,b);

        cout  << "The sum of " << a << " and " << b << " is = " << sum << endl;
        return 0;
}

int add_two_numbers(int a, int b)
{
        return a + b;
}
```

Function prototypes (used to declare functions)

Function call (where we use the function in our code

Function definiton (where the actual coding of function is done)



```
mahmoud@SHAIBOB:~$ g++ add_two_numbers.cpp -o summation
mahmoud@SHAIBOB:~$ ./summation
The sum of 5 and 13 is = 18
mahmoud@SHAIBOB:~$
```

Libraries

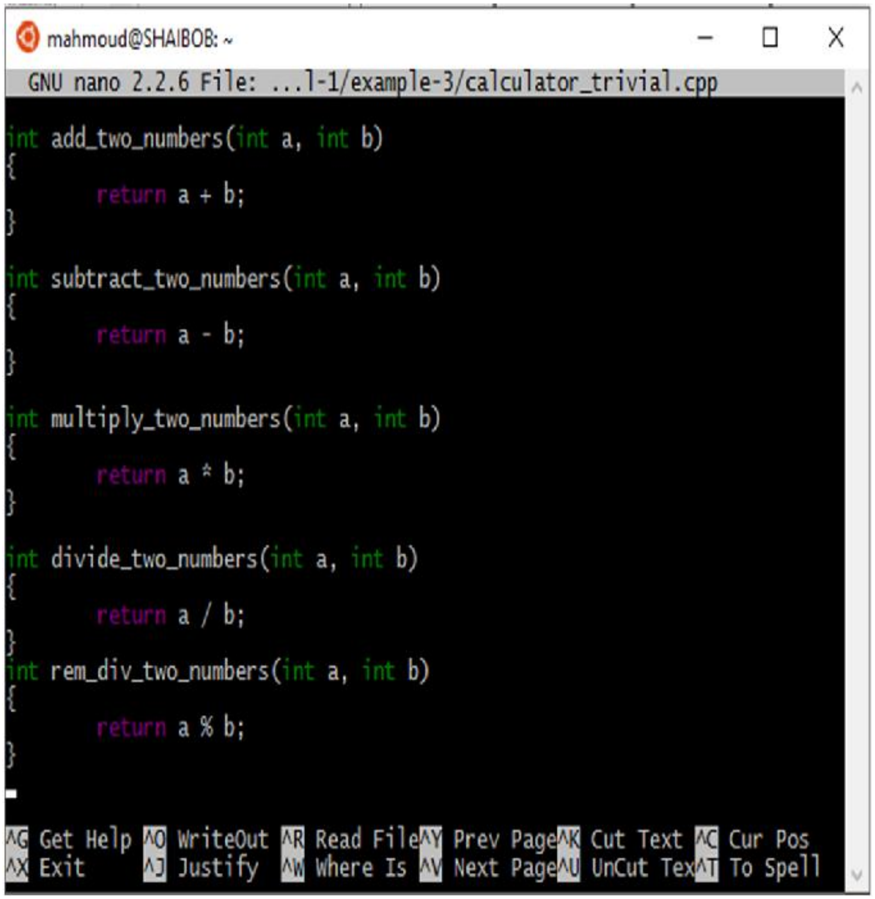
- Suppose we want to use the function add_two_numbers() in a different program.

- Do we have to write it down each time we want to use it ?!

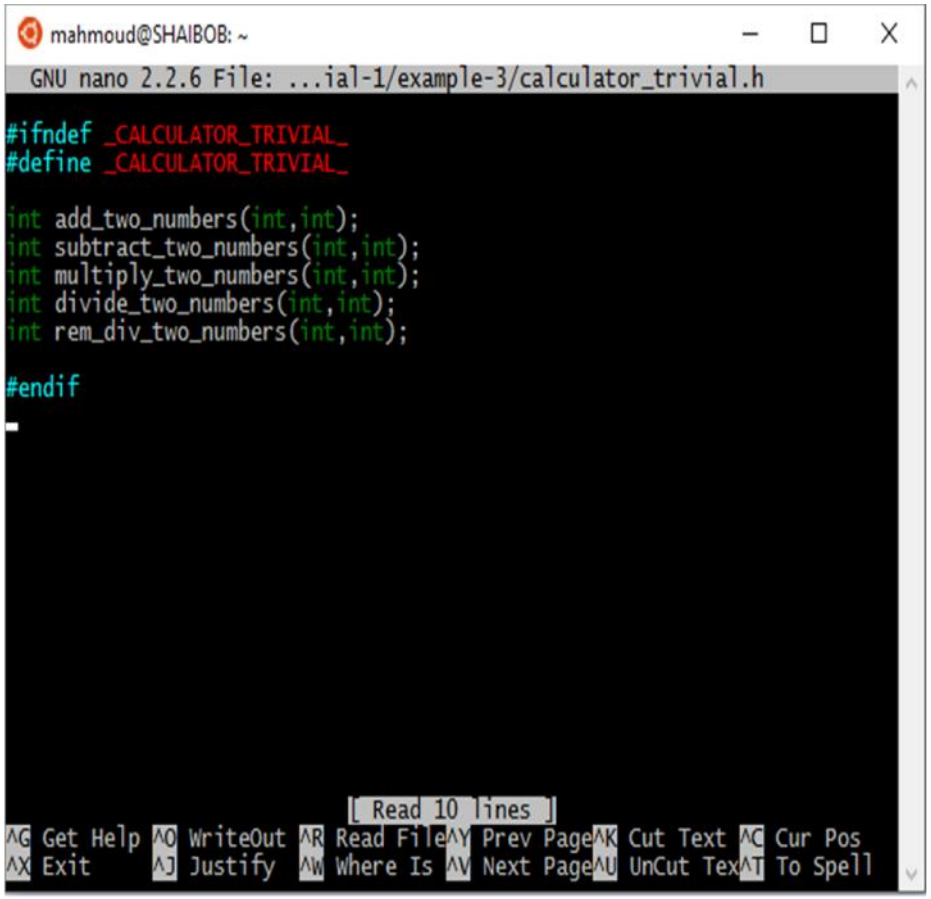
- Let’s define it once and for all, and use it whenever we need

- A Library is a chunk of code compiled in such a way to be reusable by other programs without having to re-invent the wheel each time

C++ Third Example: Trivial Calculator

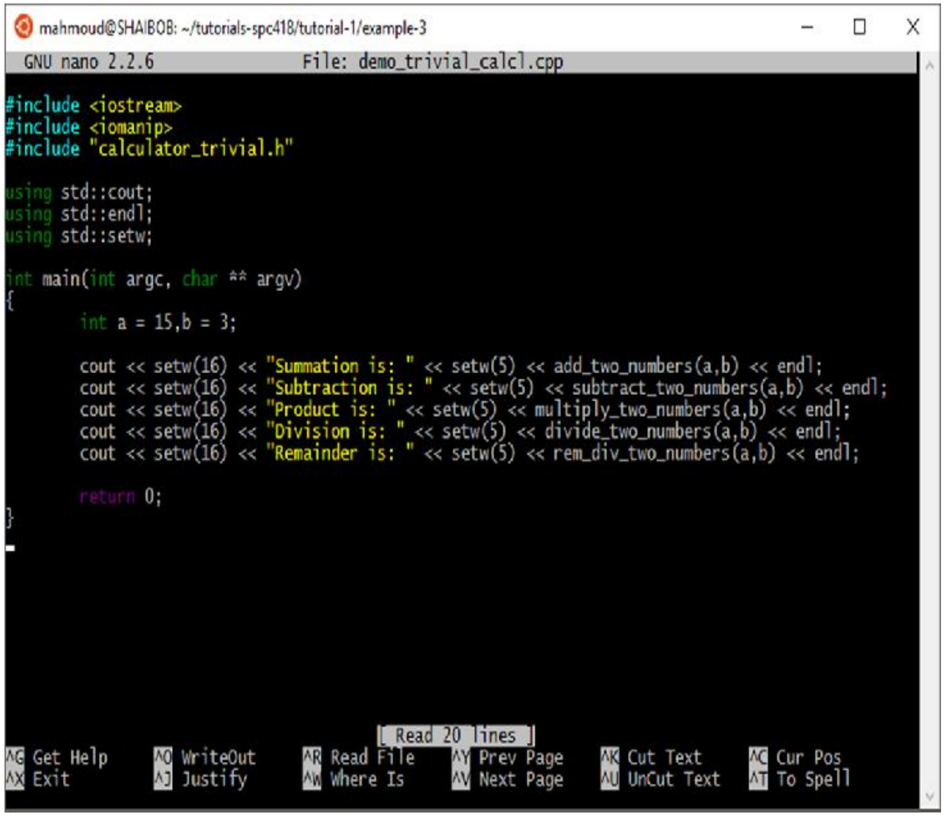


```
  GNU nano 2.2.6 File: ...l-1/example-3/calculator_trivial.cpp

int add_two_numbers(int a, int b)
{
        return a + b;
}

int subtract_two_numbers(int a, int b)
{
        return a - b;
}

int multiply_two_numbers(int a, int b)
{
        return a * b;
}

int divide_two_numbers(int a, int b)
{
        return a / b;
}
int rem_div_two_numbers(int a, int b)
{
        return a % b;
}
```

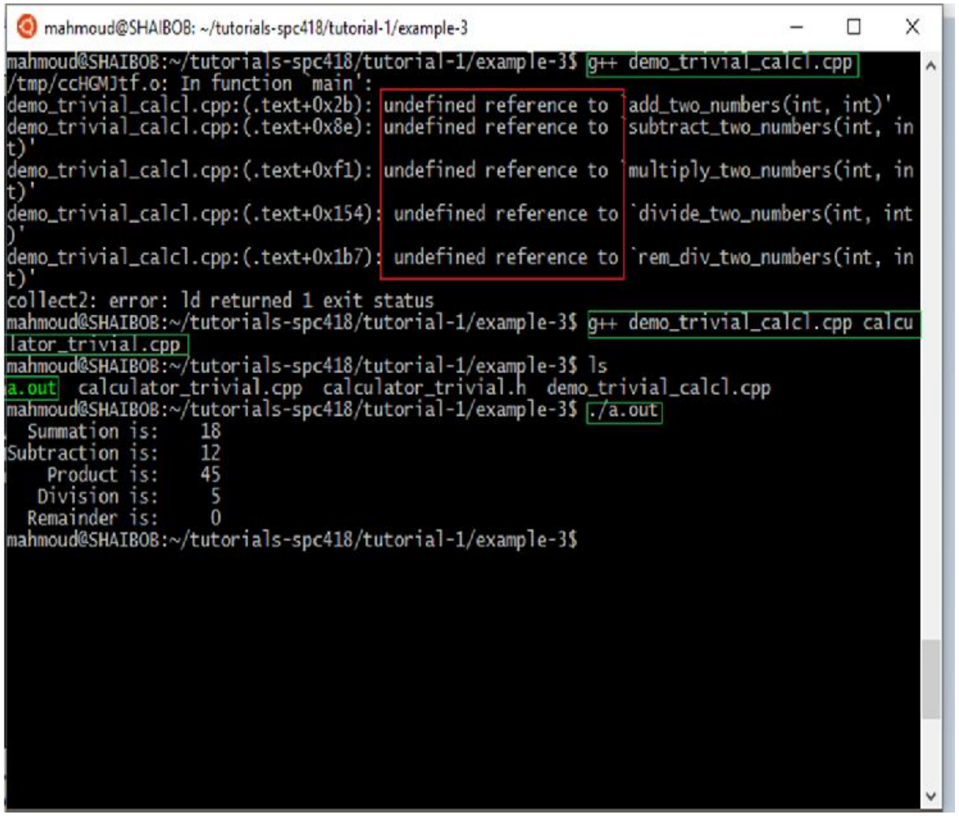


```
  GNU nano 2.2.6 File: ...ial-1/example-3/calculator_trivial.h

#ifndef _CALCULATOR_TRIVIAL_
#define _CALCULATOR_TRIVIAL_

int add_two_numbers(int,int);
int subtract_two_numbers(int,int);
int multiply_two_numbers(int,int);
int divide_two_numbers(int,int);
int rem_div_two_numbers(int,int);

#endif

                         [ Read 10 lines ]
```

C++ Third Example Contd.



```cpp
#include <iostream>
#include <iomanip>
#include "calculator_trivial.h"

using std::cout;
using std::endl;
using std::setw;

int main(int argc, char ** argv)
{
        int a = 15,b = 3;

        cout << setw(16) << "Summation is: " << setw(5) << add_two_numbers(a,b) << endl;
        cout << setw(16) << "Subtraction is: " << setw(5) << subtract_two_numbers(a,b) << endl;
        cout << setw(16) << "Product is: " << setw(5) << multiply_two_numbers(a,b) << endl;
        cout << setw(16) << "Division is: " << setw(5) << divide_two_numbers(a,b) << endl;
        cout << setw(16) << "Remainder is: " << setw(5) << rem_div_two_numbers(a,b) << endl;

        return 0;
}
```



```
mahmoud@SHAIBOB:~/tutorials-spc418/tutorial-1/example-3$ g++ demo_trivial_calcl.cpp
/tmp/ccHGMJtf.o: In function `main':
demo_trivial_calcl.cpp:(.text+0x2b): undefined reference to `add_two_numbers(int, int)'
demo_trivial_calcl.cpp:(.text+0x8e): undefined reference to `subtract_two_numbers(int, in
t)'
demo_trivial_calcl.cpp:(.text+0xf1): undefined reference to `multiply_two_numbers(int, in
t)'
demo_trivial_calcl.cpp:(.text+0x154): undefined reference to `divide_two_numbers(int, int
)'
demo_trivial_calcl.cpp:(.text+0x1b7): undefined reference to `rem_div_two_numbers(int, in
t)'
collect2: error: ld returned 1 exit status
mahmoud@SHAIBOB:~/tutorials-spc418/tutorial-1/example-3$ g++ demo_trivial_calcl.cpp calcu
lator_trivial.cpp
mahmoud@SHAIBOB:~/tutorials-spc418/tutorial-1/example-3$ ls
a.out  calculator_trivial.cpp  calculator_trivial.h  demo_trivial_calcl.cpp
mahmoud@SHAIBOB:~/tutorials-spc418/tutorial-1/example-3$ ./a.out
  Summation is:    18
Subtraction is:    12
    Product is:    45
   Division is:     5
  Remainder is:     0
mahmoud@SHAIBOB:~/tutorials-spc418/tutorial-1/example-3$
```

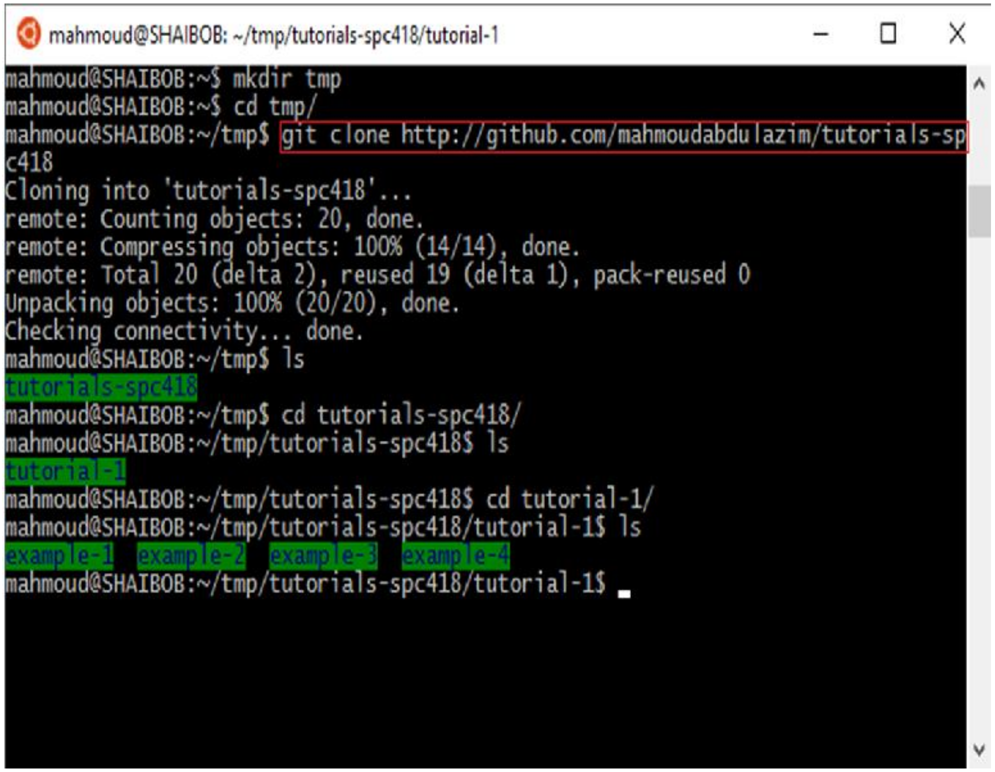
Build Process Manager: CMake

- What if we want to use code that is defined in an external library?
- What if the library provides only the binaries, not the source files?
- What if the project is so big and we have to compile multiple files and multiple libraries and manage to link them appropriately ?
- Things get complicated, and that required for a build process manager that can take care of the details.
- One of the free software available for managing build process is CMake
- CMake is the base for catkin, the official build tool for ROS
- We'll get to know more about catkin and CMake next tutorial isA

Git Version Control

- Git is a version control system made by Linux Torvalds (Father of Linux)
- Github, a collaborative coding platform, uses git to allow opensource software development.
- We will be using git for our course to share codes
- Tutorials repository: https://github.com/mahmoudabdulazim/tutorials-spc418
- We'll use git to download today's tutorials. Open a bash shell! :D

Git Version Control



mahmoud@SHAIBOB: ~/tmp/tutorials-spc418/tutorial-1

```
mahmoud@SHAIBOB:~$ mkdir tmp
mahmoud@SHAIBOB:~$ cd tmp/
mahmoud@SHAIBOB:~/tmp$ git clone http://github.com/mahmoudabdulazim/tutorials-sp
c418
Cloning into 'tutorials-spc418'...
remote: Counting objects: 20, done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 20 (delta 2), reused 19 (delta 1), pack-reused 0
Unpacking objects: 100% (20/20), done.
Checking connectivity... done.
mahmoud@SHAIBOB:~/tmp$ ls
tutorials-spc418
mahmoud@SHAIBOB:~/tmp$ cd tutorials-spc418/
mahmoud@SHAIBOB:~/tmp/tutorials-spc418$ ls
tutorial-1
mahmoud@SHAIBOB:~/tmp/tutorials-spc418$ cd tutorial-1/
mahmoud@SHAIBOB:~/tmp/tutorials-spc418/tutorial-1$ ls
example-1  example-2  example-3  example-4
mahmoud@SHAIBOB:~/tmp/tutorials-spc418/tutorial-1$ _
```

Questions?

Exercise

1) Install Eigen: http://eigen.tuxfamily.org/ and:

- – Write a program to take matrices from the user as input (do it however you like), giving the user the option to :
  - Quit any time and save the output of the operations performed in a text file
    - – Hint: you can ask for that at the beginning of your program
  - Perform any of the essential matrix operations, namely: multiplication, addition, inverse using Eigen.
- – You program should be able to handle invalid inputs and handle exceptions
- – Write the program in two ways:
  - All the code is defined in one file
  - Separate functions are defined in separate files and compiled and linked to the main executable

2) Create a repository and use git command-line tool to upload your code on it

Preparations for Next Tutorial

- Follow the instructions in this link:
  https://github.com/ros-industrial/ros_qtc_plugin/wiki/1.-How-to-Install-(Users)
  to install Qt-IDE for ROS development
- If you still haven't, please follow the instructions in this link:
  http://wiki.ros.org/indigo/Installation/Ubuntu
  to install ROS
- Note, you are free to use whatever version of ROS you'd like, but know that there are some package differences and not everything will work across different ROS versions, so it's better if we unify the version we use as Indigo for many reasons, we'll talk about them in details next tutorial isA