EECS168-Lab1
Hello World Program

Welcome to C++
An
Object Oriented Programming
Language

University of Kansas
Spring 2018

Announcements

- Login issues? (1001 Eaton Hall)
- Everything you will need (168/169's wiki page): https://wiki.ittc.ku.edu/ittc_wiki/index.php/EECS168
- Please read the wiki page (syllabus, schedule, lab instructions, etc.) carefully
- Office Hours: Tu/Th 11:00 AM – 12:00 PM (3006 Eaton)
- We will use Blackboard for assignment submission
- Feedback will be provided on BB

Announcements

- **<u>Grading Criteria based on the rubric and syllabus (wiki page)</u>**
- Attendance is required for a grade
- If you cannot attend a specific lab, please email me <u>before</u> the lab starts
- Please do not forget to write your name on the attendance sheet
- If the code does not compile, it will not receive any credits
- Labs that are submitted late (within 1 week after the due time), will receive 50% of the credit
- Slides will be posted on Blackboard
- This lab is due one week from the start of your lab. Your submission will be checked according to your Blackboard submission time.

Objectives

- 1. Get familiar with Linux environment.
- 2. Use the command line.
- 3. Compile and execute a program.
- 4. Submit your work.

Get familiar with Linux environment

- Log in to one of the Linux machines in Lab using your **KU** username and password
- After login try to get familiar with the Linux Desktop. Looks similar to Windows/Mac. Locate Start Menu.
- Terminal – Command line interface. Location **Applications -> Terminal**
- Some Linux commands you will be using often:
  - `pwd`
    - print present working directory. Shows you the full name of your current directory.
  - `ls`
    - list: Shows the contents of the current directory.
  - `cd`
    - change directory:
  - `cd ..`
    - Move one directory up
    - A single period denotes the current directory.
  - `cd ~`
    - By default, you start out in your home directory. You can return to your home directory at any time by typing cd~
  - `mkdir`
    - create new directories.
    - `mkdir EECS_168/workspace/Lab1 -p` : Makes three directories at once thanks to the -p option (parent)
    - `cd EECS_168/workspace/Lab1` :Navigate to Lab1. All at one or one directory at a time
    - `ls`
    - `pwd` : A path starting with "/" means absolute path
  - **Arrow keys – Cycle through command history**
  - **Tab key – To auto-complete file names. Type part of file name and hit Tab key**

Get familiar with Linux environment

- Some Linux commands you will be using often (contd):
  - `cd ../../`
    - Move up two directories. Can be used to move up multiple directories.
  - `ls -lt`
    - Shows list of files with time of modification, size, permissions etc in tabular format. Using options `-l` and `-t`
  - `cp` :`cp <the filename of the file you have> <the copied filename>`
    - Copy files
    - `cp libManager.log new_libManager.log`
    - `cp libManager.log  EECS_168/new_libManager.log`
  - `cp -r` : `cp -r <source directory> <destination>`
    - Copy directories
    - `cp -r EECS_168 EECS_168_new`
  - `mv` :`mv <The filenames of the files you want to move> <The target directory>`
    - Cut files and directories
    - `mv libManager.log new_libManager.log`
    - `mv libManager.log EECS_168/new_libManager.log`
    - `mv EECS_168 EECS_168_New`
  - `rm` :`rm <filename>`
    - Delete file
    - `rm new_libManager.log`
  - `rm -r`:`rm -r <directory name>`
    - Delete directory
  - `du`
    - Shows disk usage
    - `du -sh` – Shows the current directory space
  - `man` : `man <command>`
    - `Command Manual`
    - `man ls` : shows manual page for ls command

touch	create file of a given name

Creating a blank file (cpp extension)

- To start coding you will need a blank file with a cpp extension. You will type your code there and save it. Then you can compile the code.
- To create a blank file with or without any kind of extension we have 2 methods (of course, we need .cpp extension here):
- **1- the touch command: touch main.cpp**

- **2- gedit**
  - Source code editor. Richer features than Windows Notepad. gedit is also available for Mac/Windows.
  - Has source code highligting. Useful feature as you will discover very soon.
- Create C++ source file
  - Open up the command Terminal and type the below command
  - `gedit main.cpp &`
  - & keep us from tying up the terminal so we can continue to send commands to the terminal while we edit the file. Please & always!

- Type the code snippet (next slide) given and save.
  - Wait for compilation steps.

First C++ program

Layout of a Simple C++ Program

```
#include <iostream>
using namespace std;

int main()
{
    Variable_Declarations

    Statement_1
    Statement_2
    ...
    Statement_Last

    return 0;
}
```

Include directive. iostream is a library(file)

Names in iostream used in standard way

Program starts here

Program ends here

Ref: Walter Savitch. 2014. Problem Solving with C++ (9th ed.). Addison-Wesley Professional.

using namespace std – Your choice

```
#include <iostream>

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

Command Line Arguments to your program

Think of cout as display. << gives direction of data movement

```
#include <iostream>

using namespace std;

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

; end of statement. Similar to period

{ } are on single line for readability

Comments

- Comments start with `//`
- `//` is line comment.

```
//This is my code!
```

- /* ….comment…. */ for multiple lines of comment

```
/*This
Is
My
Code!*/
```

Compile & Execute (without makefile)

- Select the command Terminal window from where you just typed `gedit main.cpp &`
- First just for fun compile and execute
  - `ls`

```
[nbaruah@cycle1 Lab1]$ ls
main.cpp
```

  - **`g++ main.cpp -o HelloWorld`**
  - `ls`

- C++ compiler is called **g++ (you can see the execution file HelloWorld if you ls once more!)**

```
[nbaruah@cycle1 Lab1]$ ls
HelloWorld  main.cpp
```

- Execute
  - **`./HelloWorld`**

```
[nbaruah@cycle1 Lab1]$ ./HelloWorld
Hello, World!
```

  - `./` Represent the current directory. As your program HelloWorld is in current directory

How important is makefile for this course?

- If your code doesn't compile when we type make it receives a zero!

Makefile and Compilation requirement

This lab also introduces Makefiles. All of your labs are required to provide a Makefile for all your exercises (it's fine to have multiple makefiles per lab - for example this lab will have two folders and each folder will have a main.cpp and a Makefile).

If you need information on Makefiles, I'll refer you the Makefile tutorial. You will only utilize the section about writing a Makefile for a single main.cpp.

If your code doesn't compile when we type *make* it receives a zero!

Source: https://wiki.ittc.ku.edu/ittc_wiki/index.php/EECS168:Lab1

Makefile

1. What is a makefile?

Make is a program that looks for a file called “makefile” or “Makefile”, within the makefile are variables and things called dependencies. The make utility is a software tool for managing and maintaining computer programs consisting many component files. The make utility automatically determines which pieces of a large program need to be recompiled, and issues commands to recompile them.

- Makefile sets a set of rules to determine which parts of a program need to be recompile, and issues command to recompile them.
- Makefile is a way of automating software building procedure and other complex tasks with dependencies.

2. Why are makefiles useful?

- Makefiles can make the compilation procedure much faster.
- The compilation is done using a single command
- Only the files that must be compiled are compiled

makefile and Compilation requirements

- Makefile is needed because if we made changes in main.cpp we would need to retype or find this command again and again:
  - `g++ main.cpp -o HelloWorld`
- By the above command we generated the executable file without any intermediate object code.
- Sometimes (as we will see at the end of the semester) we have thousands of cpp files as parts of a single program. If we want to manually compile these programs every time we make a change to one of the files: Command Line Approach to Compile (example with 3 cpp files for a single program)
  - g++ -c hello.cpp main.cpp factorial.cpp
  - ls *.o
  - g++ -o prog factorial.o hello.o main.o
  - ./ prog
- Suppose we later modified hello.cpp, we need to do these steps again and again:
  - g++ -c hello.cpp
  - g++ -o prog factorial.o hello.o main.o
- Possible with “make” if we have a makefile

makefile

- To compile multiple files at one step by typing command `make` in terminal
- Contains list of compilation commands. Lets make one now.
- `gedit Makefile &`

Makefile and Compilation requirement

- g++ options
  - -o option for output file name
  - -c option for generation object files which will be input to linker
  - -std option to specify the C++ compiler version/standard. We will use std=c++11
  - -g debugging information
  - -Wall Warning options
  - command label : List of file on which command depends.

```
HelloWorld: main.o
    g++ -std=c++11 -g -Wall main.o -o HelloWorld

main.o: main.cpp
    g++ -std=c++11 -g -Wall -c main.cpp

clean:
    rm *.o HelloWorld
```

- **For compiling with Makefile, please type:**

```
make
./HelloWorld
```

makefile

- Lets add command to run program ./Helloworld
  - `$ make run`

```
HelloWorld: main.o
    g++ -std=c++11 -g -Wall main.o -o HelloWorld

main.o: main.cpp
    g++ -std=c++11 -g -Wall -c main.cpp

clean:
    rm *.o HelloWorld

run:
    ./HelloWorld
```

Add this label to your makefile

Exercise 1

- Write a C++ program, "main.cpp", to print the output on your screen. You should create a new folder and a new main.cpp.
- Sample output

```
This is my first lab exercise!
```

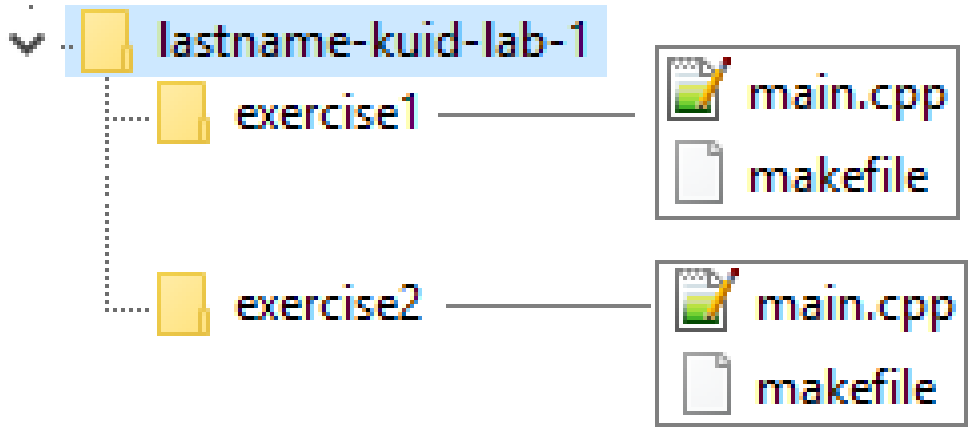
Exercise 2

- Create a new folder and main.cpp to display your name, major, and hobbies. Print each statement on its own line. Your output should be similar to the output below. Remember to use **"\t"** in your output String to cause it to indent. **“\n”** for newline.
- Sample output:

```
My name is John Gibbons.
I am an EECS major.
My hobbies are:
     Coding
     Board games
     Walking my dog
     Cooking
Goodbye
```

Lab Submission

- Folder/directory structure for exercises.



Use small case for folder and file names

Lab Submission

- tar

Output tar file

Input folder to tar

```
$ ls
lastname-kuid-lab-1
$ tar -cvzf lastname-kuid-lab-1.tar.gz lastname-kuid-lab-1
lastname-kuid-lab-1/
lastname-kuid-lab-1/exercise2/
lastname-kuid-lab-1/exercise2/makefile
lastname-kuid-lab-1/exercise2/main.cpp
lastname-kuid-lab-1/exercise1/
lastname-kuid-lab-1/exercise1/makefile
lastname-kuid-lab-1/exercise1/main.cpp
$ ls
lastname-kuid-lab-1  lastname-kuid-lab-1.tar.gz
```

Log of files inside input folder which are being tarred

Output tar file

Please make sure the tarball is not empty!

Pitfalls

- Assuming Your Program Is Correct.
  - Be prepared to go over the program code again and again and again and again….
- Putting a Space Before the include File Name
  - What, if anything, is wrong with the following #include directives?
    - a. `#include <iostream >`
    - b. `#include < iostream>`
    - c. `#include <iostream>`
- Using the Wrong Slash in \n or in \t

Rubrics

- **Rubric**
- [20pts] Email subject matches requirements
- [20pts] Tarball created and only source code files (cpp files and Makefiles, NOT .o or executable files) placed inside
- [10pts] Exercise 0
  - Folders and files created through the terminal (Do this in lab)
- [15pts] Exercise 1
  - Displays as directed
- [25pts] Exercise 2
  - [10pts] Name, major, and hobbies displayed on separate lines
  - [10pts] Use of tab character
- [10pts] Makefiles present for each exercise (for our first lab, you can get some points for at least putting in a Makefile)
- Reminder: To receive credit on your coding exercises they must compile using a Makefile!
- [0pts] Remove .o and executable files then create the tarball through the terminal (Honor system, but it's worth zero points)
- **Rubric will be strictly followed**

Thank you!