1 Starting out

Create a project on your github account. Call this project **CDDLabs**.

- Clone the labs from github
- Download your new repository onto your PC
- Follow the instructions below to modify the files and complete the exercises.

Each folder will include a folder called **docs**. All generated Doxygen files will be created inside this directory.

2 Lab One - The Toolchain

The purpose of this lab is to introduce you to the Unix (GNU Linux) command line tools. We will specifically use:

1. **Git** for document versioning and management (C++ files in our case)
2. **Emac**s for editing
3. **g++** for compiling
4. **gdb** for debugging
5. **Make** for managing code projects
6. **Doxygen** for documenting code

I will go through the following steps with you:

2.1 Tasks

1. Download the complete set of files using git to clone https://github.com/josephkehoe/CDD101 (If you are reading this then you have done this already ;-)
2. Find and open the file helloThreads.cpp using emacs (the graphical version of emacs is easier to use for beginners)
3. Customise emacs to suit your tastes and examine the code to make sure you understand it
4. Compile the file from the command line using the command:

   ```
   g++ -std=c++11 -pthread helloThreads.cpp
   ```

5. Run the file and check the output making sure you understand what has occured
6. Copy the file **Makefile1** to **Makefile** and examine using **emacs**
7. Reopen **helloThreads.cpp** and try the compile option
8. Copy the file **Makefile2** to **Makefile** and examine using **emacs**

9. Repeat for **Makefile3**

10. Create a file containing the following code:

```
#include <stdio.h>
int main(void)
{
        int i;
        for(i=0;i<10;i++)
                printf("%d",i);
}
```

11. Compile using *-g -O0* switches

12. Step through code using **gdb**

13. Use **Doxygen** to generate documentation for the **helloThreads.cpp** code

3 Lab Two - Signalling with Semaphores

You will need the following files to complete this lab.

Semaphore.h The header file for the *Semaphore* class. This file is provided for your use. Make sure you understand how it works.

Semaphore.cpp The implementation file for the *Semaphore* class. This file is provided for your use. Make sure you understand how it works.

main.cpp The file containing the main function. This main function must create at least two threads where one thread signals the other using a common Semaphore. In this lab this is where all your code will go.

Makefile This is the project file. It contains rules that tell the system how to compile the code and produce a working executable called signal.

Doxyfile This file contains the settings for the Doxygen tool. It is generated by the doxygen program when run for the first time.

README This is a text file describing the project. Every project must have one.

3.1 Tasks

1. Edit the *main.cpp* file so that the two functions (taskOne and taskTwo) are run in seperate threads and a semaphore is used to ensure that taskOne runs and exits before taskTwo.

2. All the code you produce must be properly commented using **Doxygen**, compile without error and be correct.

4 Lab Three - Rendezvous

You will need the following files to complete this lab.

Semaphore.h The header file for the *Semaphore* class. This file is provided for your use. Make sure you understand how it works.

Semaphore.cpp The implementation file for the *Semaphore* class. This file is provided for your use. Make sure you understand how it works.

main.cpp The file containing the main function. This main function must create at least two threads demonstrating a **rendezvous** in action using a common Semaphore. In this lab this is where all your code will go.

Makefile This is the project file. It contains rules that tell the system how to compile the code and produce a working executable called **rendezvous**. The makefile must now also include a "clean" rule that deletes all .o files from the project.

Doxyfile This file contains the settings for the Doxygen tool. It is generated by the doxygen program when run for the first time.

README This is a text file describing the project. Every project must have one.

4.1 Tasks

1. Using the Semaphore class create a program that demonstrates the Rendezvous pattern. A rendezvous occurs between two threads when there is a point (in the code) that both threads must reach before either can continue. (A meeting point or checkpoint)

5 Lab Four - Mutual Exclusion

You will need the following files to complete this lab.

Semaphore.h The header file for the *Semaphore* class. This file is provided for your use. Make sure you understand how it works.

Semaphore.cpp The implementation file for the *Semaphore* class. This file is provided for your use. Make sure you understand how it works.

main.cpp The file containing the main function. This main function must create at least two threads demonstrating **mutual exclusion** in action using a Semaphore. In this lab this is where all your code will go.

Makefile This is the project file. It contains rules that tell the system how to compile the code and produce a working executable called **mutualExclusion**.

Doxyfile This file contains the settings for the Doxygen tool. It is generated by the **Doxygen** program when run for the first time.

README This is a text file describing the project. Every project must have one.

5.1 Tasks

1. Using the Semaphore class create a program that demonstrates Mutual Exclusion.
2. You must incldue the Doxygen settings file and a Makefile.
3. The makefile must now also include a "clean" rule that deletes all .o files from the project and a debug option that allows use of the gdb debugger.

6 Lab Five - Reusable Barrier Class

Files in this Lab

Semaphore.h The header file for the *Semaphore* class. This file is provided for your use. Make sure you understand how it works.

Semaphore.cpp The implementation file for the *Semaphore* class. This file is provided for your use. Make sure you understand how it works.

Barrier.cpp The implementation file for the *Barrier* class.

Barrier.h The header file for the *Barrier* class.

main.cpp The file containing the main function. This main function must create at least two threads demonstrating a **reusable barrier** in action.

Makefile This is the project file. It contains rules that tell the system how to compile the code and produce a working executable called **barrier**.

Doxyfile This file contains the settings for the Doxygen tool. It is generated by the **Doxygen** program when run for the first time.

README This is a text file describing the project. Every project must have one.

6.1 Tasks

1. Create a reusable barrier class that employs the Semaphore Class. It should be a fully working class (i.e. include constructors, destructors and a well defined interface)
2. Include the Doxygen settings file and a Makefile.
3. The makefile must now also include a *clean* rule that deletes all **.o** files from the project and a *debug* rule that allows the use of the gdb debugger.
4. Add the -Wall flag to the list of compiler flags used by the Debug rule.
5. All files should now begin with a file comment that contains the following information:
   - Author Name
   - Date of File Created

  - Licence Employed

6. All code files will include suitable Doxygen comments.
7. The Makefile will also include comments explaining its purpose.
8. A README file must also be included. This is a text file that describes what this solution does, who the author is and the licence employed. It should contain instructions on how to compile the file and run it.
9. The main function should show the barrier in acton in such a way that it is clear that it works.

7 Lab Six - Producers and Consumers

Files in this Lab

Semaphore.h The header file for the *Semaphore* class. This file is provided for your use. Make sure you understand how it works.

Semaphore.cpp The implementation file for the *Semaphore* class. This file is provided for your use. Make sure you understand how it works.

SafeBuffer.cpp The implementation file for the *SafeBuffer* class.

SafeBuffer.h The header file for the *SafeBuffer* class.

main.cpp The file containing the main function. This main function must create at least two threads demonstrating a **reusable barrier** in action.

Makefile This is the project file. It contains rules that tell the system how to compile the code and produce a working executable called **prodCon**.

Doxyfile This file contains the settings for the Doxygen tool. It is generated by the **Doxygen** program when run for the first time.

README This is a text file describing the project. Every project must have one.

7.1 Tasks

Create a program that has two parts. A *producer* and a *consumer.*

Producer The producer generates random characters from ‘a’ to ‘z’ at random intervals (between 0 and 1 second in length). It adds these to a thread safe buffer that has a finite holding capacity of N characters. It generates a preset number of characters (determined at runtime) and when it has finished it add an ‘X’ character to the buffer and exits.

Consumer The consumer takes these letters from the buffer at random time intervals (between 0 and 1 second in length) and records how many of each letter it consumes. Once it sees an ‘X’ in the buffer it adds its character count to a central buffer and exits.

The main file should demonstrate your producer consumer implementation in action by creating a number of consumers and producers and showing them in action. All files must include suitable documentation. The Makefile must contain a rule ('doc') that runs the Doxygen program and generates the documentation. Edit your emacs settings so that it now automatically generates headers for your code files. e.g. see https://www.emacswiki.org/emacs/AutomaticFileHeaders