Lab 03 – Introduction to C

This lab explores the C language and the classic **`gcc`** software tool. Throughout this lab, you will review some programs in C as well as compile, test, debug and fix them.

By the end of this lab, you will be able to:

- Compile code in C using the `gcc` tool;
- Generate executable code from source code in C using the gcc tool;
- Test, debug and fix source code in C;
- Compile and link C code from two or more source code files;
- Use the C language to express code solutions to simple problem specifications;
- push your source code from your local repository to a remote repository.

Copy the lab files into your git repo

- If you have cloned your repo already, use git pull to make sure it is up to date.
- If you have not cloned your repo already, do so (the instructions are found in the week 2 lab writeup). You may need the following command (remember to substitute your own netlink id):

  ```
  git clone ssh://NETLINKID@git.seng.uvic.ca/seng265/NETLINKID
  ```

Make a Lab3 directory in your repo, enter it, and copy over the files from Roberto's home directory:

```
cp -R /home/rbittencourt/seng265/lab-03/* .
```

howdy.c

This is a standard hello-world program. Compile it using the gcc command you learned in lecture, or try the one below.

```
gcc -Wall -std=c11 -o howdy howdy.c
```

Run the program and confirm that it works.

```
./howdy
```

pythag01.c, pythag02.c and pythag03.c

These are three different programs for calculating the length of the hypotenuse, of increasing complexity. Try building and running each, and see if you can follow the code. Each of these programs uses the **`math.h`** library, and so you'll need to add the **`-lm`** switch to the end of your gcc compile statement.

- **`pythag01.c`** uses values hard-coded into variables, and generates the output in **`main()`**.
- **`pythag02.c`** uses a function - values are passed to the **`pythag()`** function which returns the length of the hypotenuse.
- **`pythag03.c`** parses command line arguments - the lengths are provided on the command line after the name of the program.

`pythag03.c` has an error - it is trying to call the **`pythag()`** function, but that function is not included in the **`pythag03.c`** file. Can you add it to **`pythag03.c`**, so that the program can be compiled and run?

program01.c and program02.c

Copying functions from one file to another is awkward and error-prone. Instead, we often call a function from one file, and implement it in another. Try doing this with **`program01.c`** and **`program02.c`**, which together print the result of raising 10 to the fourth power.

- Note that **`program01.c`** calls the function **`expo()`**.
- Observe that **`program02.c`** implements the function **`expo()`**.

You'll need to specify both source code files as input on your **`gcc`** command line, and again use **`-lm`** to specify that the math library should be used.

q_factorial.c

This program will take a number from the command line, calculate and then print out its factorial. Complete the section for command line input checking and calculation of the factorial. Compile and run the program. What is the factorial of the following numbers?

- 4
- 13
- 14
- 17

Note that the factorial for 17 is negative! This is because the memory size allocated for **`int`** is too small for the correct answer.

Try changing the type of the appropriate variable to **`long int`**.

Compile and run the program - can you calculate the factorials of 17, or 30?

random_numbers.c

Compile and run this program, which produces 10 random integers between 1 and 10. What would you change in this program to generate 10 numbers between 1 and 50? What about 50 numbers between 1 and 10?

Optional: square_circle.c

Consider a square of size 2x2, with the origin at 0,0 in the centre of the square. Now imagine a circle of radius 1 drawn inside the square.

If we generate a large number of randomly-located points within the square, we'll find that the proportion of points that land within the circle is approximately pi/4.

Write a program called **`square_circle.c`** (base it on random_numbers.c if you like). Generate a large number of points (with x and y coordinates) within the square - between 1 and -1.

For each point, if (x * x + y * y) <= 1, then the point is on or inside the circle. Keep count of the number of points that fall within the circle, and divide this count by the number of points you generated overall, and multiply the result by 4. Is the number close to pi?

Add your work to your repo, and push it back

Pretend this work is an assignment, and that you need to push it back to us to be marked.

You've created a folder and edited a bunch of files. You could use **`git add`** on each one, but it's easier to take advantage of the fact that **`git add`** is recursive when called on a directory - it adds everything new in the directory, and all its subdirectories. Return to the root of your repository and use **`git add`** there:

```
git add .
```

Then write a git commit message, something like:

```
git commit -m "finished lab 3!"
```

Finally, push your changes back to the origin with git push:

```
git push
```

Present your results to your TA

Take a few minutes to present the results of your code changes and creation, and the git commit log to your TA. That is how you will be assessed in this lab.