Lab assignment 1: Writing and compiling code

Dr Martin Porcheron

You have two weeks to complete this lab assignment—you must have it marked/signed off by a lab helper in a timetabled lab session. You must get this assignment marked by **4th February 2022** but I recommend you try and complete this as soon as possible. This assignment is worth 5 marks.

This lab assignment involves getting set up with writing and compiling C code on the command line using GCC. You may have to refer to lecture material or material on the web including the C standard library to complete this assignment. The main outcome should be a familiarity with the process of structuring your program and compiling the different files of your solution.

I recommend you keep all your lab assignments in a new directory. If working on the lab machines, create this within your home directory for CSC371. Keep each lab assignment in a separate directory.

I have given instructions in this lab assignment for compiling code by command. I strongly encourage you to practice doing this, but you are also free to use an IDE if you wish. Using an IDE though, is your choice and you are responsible for configuring it.

Note: Do not copy and paste code from the PDF as it will contain non-standard characters and will not compile.

Task 1: Hello, World (1 mark)

In this exercise, you will have to write and compile a single-file program:

1. Create a file *task1.c* and populate it with the "Hello, World!" program below:

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
        printf("Hello, World!\n");
        return 0;
}
```

2. From the command line compile your program with the GCC compiler using:

```
$ gcc task1.c -o task1
```

Expected output

If your run your program using the command:

```
$ ./task1
```

...you should get the output:

```
Hello, World!
```

Task 2: Fibonacci number (2 marks)

In this exercise, you will have to create two implementation files and one header file. This is to get practice at compiling multiple files.

1. Create a file *fibonacci.h* and populate it with the function prototype below. You must include appropriate header guards (see L2 for information on how to add header guards).

   ```
   int fibonacci(int n);
   ```

   This function takes a *signed* `integer` n and returns a signed `integer`.

2. Create a file *fibonacci.c* that implements this function. You should first include *fibonacci.h* at the top of this file. You will then need to implement the `fibonacci` function using the following rules:

   - $F_0 <= 0$ (i.e. `n = 0`)
   - $F_1 = 1$ (i.e. `n = 1`)
   - $F_n = F_{(n-1)} + F_{(n-2)}$ (i.e. `n = fibonacci(n-1) + fibonacci(n-2)`)

   In other words, when n is less than or equal to `0`, return `0`, when `f` equals `1` return `1`, otherwise return a call to `fibonacci(n-1) + fibonacci(n-2)`.

3. Create a file *task2.c* that has a `main` function that contains exactly the code below. Do NOT modify the contents of this `main` function in your solution.

   ```
   int main(int argc, char* argv[]) {
           int result = fibonacci(FIBONACCI_NUM);
           printf("Fibonacci of %d = %d\n", FIBONACCI_NUM, result);
           return 0;
   }
   ```

   To make this code work, you must include two directives at the top of *task2.c* to include *fibonacci.h* and define a macro/constant `FIBONACCI_NUM`. Search the web for how to do this.

4. Compile your program using:

   ```
   $ gcc -c task2.c -o task2.o
   $ gcc -c fibonacci.c -o fibonacci.o
   $ gcc task2.o fibonacci.o -o task2
   ```

Expected output

If your run your program using the command:

```
$ ./task2
```

...and `FIBONACCI_NUM` is `10`, you should get the output:

```
Fibonacci of 10 = 55
```

Task 3: Program arguments (2 marks)

In this exercise, you will extend your solution to task 2.

1. Copy your *task2.c* file and give your new file the name *task3.c*.
2. Modify the code in the `main` function to retrieve the first command line argument passed into the program. Decode this string to be an integer, and pass it to the `printf` function and to the `fibonacci` function.
   - Two variables are passed into the `main` function. The first gives the count of the number of program arguments (and as such is often called `argc`), the second is an array of values (often called `argv`).
   - Every program will have at least one program argument, which will be the name of the program called. Therefore, if you are expecting a single value passed by command line to your program, `argc` must be equal to 2
   - You should make sure your code is safe—don't try to retrieve an array value if it doesn't exist. In future lab assignments, you'll be expected to do error handling without being prompted.
   - To get the value of the second `argv` value, use `argv[1]` (the array syntax is similar to Java, and starts counting at `0`)
   - To convert a string to an `integer` in C, use the `atoi` function from `stdlib.h`, e.g.:

     ```
     int num = atoi(argv[1]);
     ```

   Avoid using `scanf` as it is insecure. `scanf_s` is overkill and inefficient for something this simple.
3. Clean up your *task3.c* file, removing any unneeded code (i.e. the previous directive you included!)
4. Compile your program using the commands below (you may also need to recompile *fibonacci.c* if you deleted *fibonacci.o*):

   ```
   $ gcc -c task3.c -o task3.o
   $ gcc task3.o fibonacci.o -o task3
   ```

Expected output

If your run your program using the command:

```
$ ./task3 10
```

...you should get the output:

```
Fibonacci of 10 = 55
```