Systems Programming: Practical 2 — Compiling C programs

If CIS has not yet given you access to mira, you will not be able to do the first part of this this practical. Please see the Duo page for Systems Programming for instructions on how to register for mira access. Part A of this practical should be straightforward; it is critical that you can do these basic steps. Please ask demonstrators for help if you cannot get this working. Part C is optional.

When adopting any new language, getting a "Hello, World!" program working demonstrates you can enter, compile and run a whole, if simple, program.

A Familiarize yourself with the gcc development environment:

1. Log in to Linux timeshare service (mira) using X2Go. To do this on a Windows MDS computer, go to the appsanywhere at `https://appsanywhere.durham.ac.uk/` and launch "X2go-Mira r3." If using your own computer, you can download X2Go from `https://www.x2go.org`; set "Host" to "mira.dur.ac.uk" and "Session Type" to "MATE". Use your CIS username and password to login and click "Yes" if asked about trusting the host key.
2. Open a terminal window (click on the screen icon at the top, with tooltip "MATE Terminal").
3. Create a `main.c` file and type in the hello world example from the lecture. Use a text editor: if you haven't used Linux before, you might want to use `pico`/`nano`: `https://www.dur.ac.uk/resources/its/info/guides/17Pico.pdf`[1]
4. If using an editor in Linux proves too much for you, then it is possible to edit the source file from Windows (with Notepad or something like that) and then compile it from Linux - MDS Windows (J: drive) and MDS Linux share the same filespace for your home folder.
5. Compile your program with the commandline `gcc main.c`
6. Execute your program (which is called `a.out` by default) by running `./a.out`
7. Try looking at the different stages of the output by performing only parts of the compilation process up to
   (a) Preprocessing
   (b) Compilation
   (c) Assembly
   See the slides from Lecture 3 as to which flags do which and look at the output to see if it makes sense.

B A small program

In DNA strings, symbols "A" and "T" are complements of each other, as "C" and "G". Write a function which takes one side of the DNA and gets the complementary side. You may assume that the DNA strand is never empty.

Examples:

- ATTGC ⇒ TAACG
- GTAT ⇒ CATA

Start a file `dna.c` and fill in the appropriate code in the section marked TODO:

[1] In lectures, I will be using `vim`, which is a far more powerful text editor than `nano`. For a quick tutorial on `vim`, run `vimtutor` on the commandline. Lesson 1 of this tutorial should be enough to get you through the practical, but if you decide to use `vim`, it's worth trying the later lessons of the tutorial too.

```
#include <string.h>
#include <stdio.h>

char *dna_strand(const char *dna)
{
    int len = strlen(dna); // This is the length of your strand
    char *result = strdup(dna); // This allocates memory for your result
    // You can now write into result with result[i] = ...
    // You can read entries from dna with dna[i]
    // TODO


    return result;
}

int main(){
    printf("The complementary strand is: [%s]\n", dna_strand("GTAT"))
    return 0;
}
```

Make sure to compile, run and check the output. Test other inputs.

C Try a more complex program:

1. Write a bubblesort program in C. If this proves too tricky for you, use this one: `http://www.algorithmist.com/index.php/Bubble_sort.c`. You do not need to output the result.
2. Write a binary search program in C, using an array that has been sorted with bubble sort - again no need to output. If this is beyond you try `http://www.algorithmist.com/index.php/Binary_Search`
3. Compile and execute your program on some fixed array.
4. Try changing the compiler optimisation options `-O0` through to `-O3` and also `-Os` to optimise for speed and size.
5. Compare the file sizes (since this quite a simple program, you may find that the file sizes are the same).
6. Look at the compilation output before assembly (with `-S` option). Can you understand what is going on?

D Try to use CoPilot to create a complex problem:

1. Go to `https://copilot.microsoft.com/`, and login if needed
2. Try to write a prompt to be able to generate bubble sort in C and compare it to your implementation. Would the code work? Is it more efficient or elegant than your code? Be conscious that the code might use functions that you are unfamiliar with; they might be covered in later lectures, but it is always good to understand what the code is doing and not treat operations as black boxes.
3. Try compiling the code and running it to test if it would work in the same way as your code.