CAB202 Tutorial 1
C Programming Introduction

Your C Environment must be set up before you do this tutorial; follow the instructions on the Blackboard site in the folder "For your first Tutorial". The tutorial begins with a detailed outline of how to run a simple 'Hello World' program, and a second program using the Zombie Development Kit (ZDK). There are then three sections of exercises. The first section includes exercises that your tutor can help you with. The second section includes assessable exercises that should be submitted via the Automatic Marking System (AMS). The third section includes challenge exercises that are not assessable. This tutorial counts for 3% of your grade.

Overview

The C programming language is designed to map efficiently to the machine code of the embedded microcontroller you will be using later in the semester, but provides the capabilities of a high level algorithmic language. By the completion of this prac, you will be able to create a basic C program that can utilise both control structures and types. This will first involve creating the source code, compiling, and executing a basic C program. You will then use this process, and make use of control structures and typed variables, to created coded solutions to some basic exercises.

Introduction to C programming ('Hello World')

The general purpose of programming is to create machine code that carries out a set of desired tasks. This machine code (a binary or executable file) is essentially a sequence of 1's and 0's used to control the CPU (central processing unit) of a computer. To create a piece of machine code, there are three distinct steps:

1. Creation of C source code
2. Compilation of the code
3. Execution of the code

1. Creating source code

To start creating a C program, or in fact for any program, you first must write some form of source code. To create C source code, a file is created with the ***.c** file extension. The file extension is used to tell the system how the information in the file is expected to be structured. For example, a 'hello world' program might exist in a source file called:

```
hello_world.c
```

Once this file is created, it must be edited to include the source code. This can be done in any program that allows you to edit basic text files (i.e. text editors like Notepad, Notepad++, Sublime Text, Gedit, TextEdit, etc).

A C program requires one essential component to run, a **main** function. While functions will be introduced later, think of the **main** function as an entry point. When running your program, this function provides the entry point into your code (open **'{'** bracket) and exit point when it is completed (the **return** statement or close **'}'** bracket). If the following code were compiled and run for example, the computer would enter your code and then return after having done nothing:

```
int main() {
     return 0;
}
```

This is technically a working program that could be compiled and run. However, this program doesn't do anything. Let's change that by writing **Hello World!** to the screen (the console window). To write any text to the console, the function called **printf()** is used. This function writes whatever **String** is supplied in the first argument. In C, **Strings** can be specified by enclosing quotation marks. For example, the follow code would print **Hello World!** to the console:

```
printf("Hello World!");
```

The **printf()** function also allows much more complicated behaviours. The following examples show only a few of the capabilities of the **printf()** function. A more complete list can be found at http://www.cplusplus.com/reference/cstdio/printf/ .

1. Print a string and then move to the next line ('**\n**' character does this):

```
printf("Hello World!\n");
```

2. Print the value of a **integer** type variable called 'countVar':

```
printf("The count is %d", countVar);
```

3. Print the value of a **float** type variable called 'piVar':

```
printf("The value of pi is %f", piVar);
```

4. Print the values of two **integer** type variables called 'firstVar' and 'secondVar':

```
printf("The values are %d and %d", firstVar, secondVar);
```

This function is easily called with the syntax outlined above, however to use this function we first need to include the library in which it resides. To include a library we use the **#include < >** command at the start of the source code file (before the **main** function). To use the **printf()** function, the **stdio.h** library (standard input and output library) must be included. Consequently, the following code would compile to make a complete program that writes **Hello World!** to the console:

```
#include <stdio.h>

int main() {
     printf("Hello World!");

     return 0;
}
```

This is now a complete C program. Now, before the program can be run, we need to compile the program and turn it into a language the computer can understand.

2. Compiling the source code

A computer does not understand a program in its high level form, such as C, C++ or Python source code. The code must first be changed it into a binary file that the computer can read and execute. This is done through the process known as code compilation. There are several different compilers out there for the many programming languages, but we will use the **gcc** compiler which came out of the GNU project that started in 1984.

The compiler is a very powerful tool that can compile programs using source files (***.c**), header files (***.h**), or shared libraries (like **stdio.h**). Specifically, the compilation process converts the textual source code version of a program into the machine code (a binary or executable file). This machine code is the sequence of 1's and 0's used to control the central processing unit (CPU) of a computer.

To compile a simple program with **gcc** open a console/terminal window and navigate to the directory where the program source code (**hello_world.c** in this example) is located. This is done with the **cd** command. Then run the **gcc** compiler on the source code file to produce a **hello_world** executable. For example, the commands for compiling source code that is located in **/home/you/hello_world/** is as follows:

```
cd /home/you/hello_world/
gcc hello_world.c –o hello_world
```

The second command is telling the computer to use the **gcc** compiler to compile the **hello_world.c** file into a **hello_world** executable file. The –o flag is telling **gcc** where to store the output of the compilation process. If there were no errors, in either your C program, or in the compilation process then you should end up with a **hello_world.exe** or **hello_world** binary file in the same directory as the **hello_world.c** file (you can check this by using the **ls** command after compiling).

3. Executing the compiled code

To run this executable, you have to execute it from the command window. When you do this, the computer enters the binary version of your **main** function, runs what is inside of it, and then returns when it reaches the end of the function. The execution of a binary file is:

```
./hello_world
```

Introduction to the ZDK

We will be using a support library called the ZDK to create game-like programs. The ZDK used in the first half of the semester is based on a library called ncurses. You don't need to know the details of ncurses, but you do need to include it in your C Environment setup, and use a few extra lines of code in your programs. You need to include cab202_graphics.h, initialise the window, and restore normal terminal behaviour after the program:

```
#include <cab202_graphics.h>

int main(int argc, char *argv[]) {
     setup_screen();
```

```
        // your code goes here

        cleanup_screen();
}
```

In the middle is where you put your code. If you want to draw a ball at the top left of the screen, you would add:

```
        draw_char( 0, 0, ‘O’ );
        show_screen();
```

To compile a program that uses the ZDK, you need to add further arguments to the command line for gcc,

```
cd /home/you/hello_world/
gcc hello_world.c –o hello_world \
    -Ifolder_where_zdk_is_installed \
    -Lfolder_where_zdk_is_installed \
    -lzdk01 -lncurses
./hello_world
```

Note that if all you include is the code above, you won’t actually see anything happen, because as soon as the ball is drawn the window is closed and normal terminal behaviour is returned. There are a variety of different methods you can use to see what you have drawn, the simplest being to add a never-ending while loop into your program:

```
        while (1) {}
```

If you have included a never-ending while loop in your ZDK program, you can stop the program using Ctrl-c.

At this point, you now have run through the entire process of creating, compiling and running code. Now using this knowledge, and the lecture notes from the first week, complete the following set of exercises. Submit solutions to the second page of exercises to the AMS.

Non-Assessable Exercises

NOTE: These exercises are not assessable, and the tutor can help you with them. Assessable exercises are in the following section.

1. Create a ‘Hello World’ program

Using the information discussed above, create the code for a program that prints ‘Hello World!’

2. Create a ZDK ‘Hello World’ program

Using the information discussed above, create the code for a program that places a ball at the top left hand corner of the screen using the ZDK.

3. Create a ZDK program that places several balls at different locations in the window

Create a program that moves the curser to the following locations and draws a ball:

(Row 10, column 8), (Row 20, column 6) (Row 7, Column 20), (Row 15, Column 5)

Use the `draw_char()` function. You can get more information about `draw_char()` by examining the Topic 1 Toolbox.

4. Adapt the zombie program

Change the zombie program to be a vampire program by:

- changing the Z for zombies to V for vampires
- changing the @ for player to *
- changing the message when you are caught from “You are zombie food!” to “You are vampire food!”

5. Create a program that draws a vertical line on the screen

Create a program that draws a vertical line on the left side of the screen. The line should be constructed using “|” symbols, and should be 5 characters in length. The line should be located at the top of the window and 2 characters in from the left side of the window (i.e. 2 spaces to the left of the “|”).

Exercises

Complete the assessed exercises via the AMS by following the links provided on the CAB202 Blackboard page.

Challenge Exercises

6. Modify ZombieDash

During the first lecture, you were asked to think about changes that could be made to ZombieDash to make it a more interesting game. Choose one of these and implement it.

7. Create Snake, Tetris, Pong, …

Choose another game and start to implement the basics of that game, or create your own game. Start by working out how to draw all of the shapes that you need for that game.