CS402: Seminar 1
A C and C++ Primer

In this session we are going to take a brief look at C and C++. The two languages are quite similar in terms of syntax, and are often grouped together when discussing various programming languages. We will start by looking at basic C programs, and then cover some of the object-oriented features of C++.

1 Hello World

First things first, lets get up and running with the ubiquitous "Hello world!" program. Open up your favourite text editor (maybe *vim* or *emacs*), and type in the following program, saving it as *hello.c*

```
#include <stdio.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");

  return 0;
}
```


To compile this program, open up a terminal window and change to the directory containing the program. Then use the GCC compiler like so:

```
gcc -o helloworld hello.c
```

You will notice a few differences between C and Java, but overall the structure of the programs is mostly the same. One thing to notice is that you need to put special characters like the line break (`\n`) at the end of strings you want to print.

Run this program with:

```
./helloworld
```

2 A Few More Programs

This next program calculates factorials, demonstrating the use of functions and for loops in C. In C, it is important that you define your functions before you use them. You can do this just by keeping them further up the file, or you can use *prototype definitions* that just state the return type, name, and arguments of the function. The prototype of the `factorial` function would look like this:

```
int factorial(int n);
```

Below is the factorial program:

```
#include <stdio.h>

int factorial(int n) {
  int i = 0;
  int fac = 1;

  for(i = 1; i <= n; i++) {
    fac = fac * i;
  }

  return fac;
}

int main(int argc, char* argv[]) {
  int fac_five = factorial(5);

  printf("5! is %d\n", fac_five);
}
```


Task 1. Write a program that uses the formula $°C = \frac{5}{9}(°F - 32)$ to print a table of Farenheit temperatures, from 0 to 300 in steps of 20°, with the corresponding temperatures in Celsius. The example output would be something like:

```
0 -17
20 -6
40 4
...
300 148
```


3 A Few Pointers

Pointers are the main difference between C and Java. What you really need to know is that pointers *point* to some location in memory. The simple program that follows introduces pointers.

```
#include <stdio.h>

int main(int argc, char* argv[]) {
  int* pntr;
  int my_int = 6;

  // Make pntr point to the location of my_int
  pntr = &my_int;

  // Write some print statements to check the values
  printf("The value of my_int is %d,  the value of pntr is %p\n",
      my_int, pntr);

  // What is the value that pntr is pointing at?
  printf("Pntr is pointing at %d\n", *pntr);

  // We dereference the pointer with the *
  // to get at the value it's pointing at
```

```
  return 0;
}
```

When using pointers, adding a `*` to the beginning of the variable lets us get at the value it contains. So if we have:

```
int a = 50;
int* a_pntr = &a;

printf("Pointer is %x, the value is %d\n", a_pntr, *a_pntr);
```

This first uses the `&` to set `a_pntr` to the address of `a`, and then uses `*` to get at the *value* of `a_pntr` in order to print it out.

Task 2 To check your understanding of pointers, try writing a `swap` function here that will swap the values of variables by using pointers. How can we pass pointers to `value_a` and `value_b` into this function?

```
#include <stdio.h>
void swap(int* a, int* b) {
  /* insert code here */
}

main(int argc, char* argv[]) {
  int value_a = 4;
  int value_b = 5;

  /* use swap here, then print the values */
}
```


4 Allocating Memory

In C, we can declare arrays almost the same way we do in Java, using the square braces, and some number of elements, like so:

```
int[] my_array = {10, 11, 12, 14, 15};

printf("The first element is %d\n", my_array[0]);

/* Creating an array with space for 10 integers */
int x[10];
```

The problem with arrays in C is that we can't declare a dynamically sized array just using the square braces. So if we want to create an array of a given size, passed in by the user for example, we need to use the `malloc` function to *allocate* the memory for the array.
Malloc works with pointers, rather than the `int[]` style array declaration. This is because the call to `malloc` will return a pointer to the first element. In order to create an array, we do something like this:

```
int* my_malloc_array;

/* 10 could be any number, or an integer variable */
my_malloc_array = (int *) malloc(sizeof(int) * 10);
```

When you are done with memory, it must be deallocated so that it can be reused by the operating system. We deallocate memory using the `free` method (continuing the above example):

```
free(my_malloc_array);
```

Task 3 Take a look at the following program. It reads in a series of integers from the user, then prints them back out in reverse order. The problem is that `SIZE` is hardcoded as 5. Your task is to modify this program in order to read a dynamic number of integers. First ask the user how many ints they want to enter, then read them in, then print them back out.


```
#include <stdlib.h>
#include <stdio.h>
#define SIZE 5

int main() {
    int arr[SIZE];
    int temp;
    int i = 0;
    while (scanf("%d", &temp) > 0) {
        arr[i] = temp;
        i++;
    }
    /* i now points _past_ the last element. */
    /* Move i back to point at the last element */
    i--;        /* Point to last */
    for (; i>=0; i--) {
        printf("%d\n", arr[i]);
    }
    return 0;
}
```


The `scanf` function reads an integer from the command line into some variable, `temp` in this example. Note that you need to use the `&` to pass the address of the variable to read the variable into.

5 Classes

Classes are the first exclusively C++ feature we will be looking at. Classes provide a way of grouping functions and data, giving logical structure to our programs. A C++ class is declared in a header file, like so:

```
class Circle {
    public:
        Circle(double r);
        double getArea();
```

```
        double getCircumference();
    private:
        double radius;
        static const double PI;
};
```

The basic class declaration consists of a number of variable and function definitions, qualified with under either the `public` or `private` access modifiers. The `Circle` method is a constructor, so doesn't require a return type.The most important thing to note about C++ classes is the `;` after the closing brace. Type this code into a file, and save it as `Circle.h`.
We implement the methods defined in the class declaration in an "implementation" file:

```
#include "Circle.h"

const double Circle::PI = 3.1416;

Circle::Circle(double r):
    radius(r)
{
}

double Circle::getArea()
{
    return PI*radius*radius;
}

double Circle::getCircumference()
{
    return 2*radius*PI;
}
```


The implementation includes the header file in which we defined the class, and then provides definitions for all the functions we declared. Note the static variable `PI` is also defined in this file. The syntax after the constructor is known as an initializer list, as is a way of assinging values to the instance variables of the class. Save this file as `Circle.C`.

5.1 Using Classes

Let's try using our new `Circle` class with a simple example.

```
#include <iostream>

#include "Circle.h"

int main(int argc, const char* argv[])
{
    Circle c1(4.0);
    std::cout << "Circumference of c1 is : " << c1.getCircumference
        () << std::endl;

    Circle* c2 = new Circle(2.5);
    std::cout << "Area of c2 is : " << c2->getArea() << std::endl;
```

```
    delete c2;
}
```

We start by including our header file which contains the definition of the `Circle` class. We create the circle `c1` using the constructor, and call one of the class methods using the `.` operator. When we create the second circle, `c2`, we use the `new` operator, which allocates memory and creates the object, and returns a pointer to the object. This is important because when an object is created on the heap in this way, the pointer can be passed around and the same object can be used by multiple functions. When calling functions on an object pointer, we use the `->` syntax. This deferences the pointer and calls the correct method. To delete the object when we are done, we use the `delete` keyword[1]. The first circle was created on the stack, so will be destroyed automatically when it goes out of scope. Save this file as `main.C`, and let's compile this example:

```
g++ -o circle_test Circle.C main.C
```

Try running the example to ensure you get the results you expect.

6 Debugging

It's pretty likely that during the course of the assignment you will run into some bugs in your program. Finding bugs in C and C++ programs can be hard, mostly because of the notorious *segfault*. We will look at a few ways to help debug your program:

1. Compile your code with the `-g` flag, this will enable debugging symbols in your code.
2. Load your program with *gdb*, the GNU Debugger, like so: `gdb <your executable>`

3. Inside gdb, type `run` to run your program.
4. If your program segfaults, you should see some helpful information like line number and function name.

6.1 What else can I do with gdb?

There are a few nifty things in gdb that will make your life easier.

- `print <variable>` will print out the value of a variable. If you try and print something and see something like `$1 = (int *) 0x0`, then it probably means the variable (or pointer) has not been initialised.
- Placing breakpoints: inside gdb, `break <file>:<line>` will allow you to set a break point at a particular line (just before your program crashes, for example). Gdb will halt the program at the specified line, and will allow you to examine the values of variables, as well as set new breakpoints.

[1] `new` and `delete` are the C++ equivalents of `malloc` and `free`.

- Backtraces provide information on the functions that have been called when the program crashes. Try typing in `bt` to gdb when your program crashes. You should see information on the function it was in, as well as the path of functions leading up to the crash.

For more information on gdb, use the `help` command, or do some research online. Hopefully this basic guide contains enough to keep you going with this assignment.

7 Extra Resources

The main reference for C programming is Kernighan and Ritchie's book, The C Programming Language [1]. It is well worth getting hold of, both for it's excellent C content, and the fantastic example of technical writing.

C++ is a rich languange with a lot of depth, as such, there are a huge number of C++ reference books. We like The C++ Programming Language [2] and Programming: Principles and Practice Using C++ [3], two books written by Bjarne Stroustrup, the creator of C++. If you already have some C++ experience, then Scott Meyer's Effective C++ [4] is a fantastic book of tips and tricks for improving your code.

References

[1] Brian Kernighan & Dennis Ritchie, *The C Programming Language*, Prentice Hall Professional Technical Reference, 2nd Edition, 1988.

[2] Bjarne Stroustrup, *The C++ Programming Language*, Addison-Wesley Professional, 3rd Edition, 2000.

[3] Bjarne Stroustrup, *Programming: Principles and Practice Using C++*, Addison-Wesley Professional, 1st Edition, 2008.

[4] Scott Meyers, *Effective C++: 55 Specific Ways to Improve Your Programs and Designs*, Addison-Wesley Professional, 3rd Edition, 2005.