THE UNIVERSITY OF SYDNEY

COMP2017 / COMP9017 Week 2 Tutorial

Introduction to C

Welcome to C!

In addition to bash scripts, you may have noticed a number of compiled files as you've been rummaging around. These can be identified as unreadable messes of poorly rendered symbols. You can see one by running **cat** on just about any file in */bin*, for example **cat /bin/grep**.

This mess of symbols is a series of assembly instructions that indicate a number of operations that a processor is to perform. Obviously writing this directly is a painful and time consuming exercise, so programming languages have been developed that 'compile' down to this assembly.

As is often the starting point for learning a new language, the syntax for hello world in C is as follows:

```c
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```

We can also see that the function is named 'main'. This is a special function name in C such that when the code is run, the 'main' function will be the first one to be called. The `printf` function is one of the print statements in C and it sends a text stream to standard out, allowing it to interface with the rest of linux.

While it is nice to have this sitting around in a file, in order to run this code we need to compile it. There are two major (one might even say competing) C compilers: gcc and clang. The name of the compiled file can be specified using the -o flag as follows.

```
gcc -o hello_world hello_world.c
```

or

```
clang -o hello_world hello_world.c
```

Then you can simply execute the compiled code.

```
./hello_world
```

Try writing, compiling and running hello world in c. What does the compiled file look like?

Question 1: Hello From C

You are to write a simple C program that will output `Hello World!` to the terminal.

```
$ ./hello
Hello World!
```

Question 2: Meet and Greet!

Like in previous semesters, you should be familiar with being able to query the user for data via `scanner.nextLine()` or `input()` and retrieve arguments from the command line.

Write a program that will ask for the user's name and output their name mixed with a command line argument given.

```
$ ./greet Yo!
What is your name? David
Yo! David
```

For help on using the functions necessary to complete the question, you can retrieve information about the following functions `scanf` and `fgets` by using `man 3 scanf` and `man 3 fgets`.

Question 3: Repeat

Write a program that reads characters from stdin and outputs them to stdout. For example:

```
$ ./repeat
Hello there
Hello there
^D
```

Question 4: Echo 419

Write a program that outputs the arguments that are given to it. For example:

```
$ ./echo Echo 419 to Cortana, come in
Echo 419 to Cortana, come in
```

C types

The C Programming language is a statically typed language, therefore any variable must have a type associated with it. You can declare variables with following types and more:

- `char and unsigned char`
- `int and unsigned int`
- `short and unsigned short`
- `long and unsigned long`
- `double`
- `float`

However, each type is defined by the cpu architecture, platform and compiler. When writing portable C code you may need to consider the differences of the primitive types between platform targets. C exposes `unsigned` types which can be prepended to integer types to eliminate the usage of the sign bit as part of the type encoding.

C provies access to memory allocations using array and pointer types. To declare an array in C you need to specify the size. C does support variable length arrays with the C99 standard however, but it typically considered bad practice.

```
int array_1[10];
int array_2[] = { 1, 2, 3, 4 };
int array_3[5] = {1, 2, 3}; // Will still be 5 elements
```

Your code should not depend on this feature your compiler will typically warn you if it has been misused. You will be able to dynamically allocate memory using functions such as malloc and navigating them through a pointer type.

Question 5: C Declarations and Initialisations

As a group, answer the following questions and discuss with your tutor about the following:

1. What are the differences between these declarations and initialisations?

```c
const char * ptr = "hello";
const char array[] = "hello";
const char array2[] = { 'h', 'e', 'l', 'l', 'o' };
const char array3[] = { 'h', 'e', 'l', 'l', 'o', '\0' };
const char array4[5] = { 'h', 'e', 'l', 'l', 'o' };
const char array5[6] = { 'h', 'e', 'l', 'l', 'o', 0 };
const char array6[20] = { 'h', 'e', 'l', 'l', 'o' };
const char array7[20] = { 0 };
const char array8[20] = "hello";
```

2. Given the code above, what does the following output?

```c
printf("%zu %zu\n", sizeof(ptr), sizeof(array));
printf("%zu %zu\n", sizeof(array2), sizeof(array3));

printf("%zu %zu\n", sizeof(*ptr), sizeof(&array));
printf("%zu %zu\n", sizeof(&array2), sizeof(&array3));
```

3. What does the following output, given that `sizeof(int)` is 4?

```c
int x[] = { 1, 2, 3 };
int * p1 = x;
int * p2 = x + 1;
printf("%zu %zu\n", sizeof(x[0]), sizeof(x));
printf("length: %zu", sizeof(x) / sizeof(x[0]));
printf("p1 value, p2 value: %d %d\n", *p1, *p2);
printf("p1 value with offset: %d\n", *(p1 + 1));
printf("p2 value with offset: %d\n", *(p2 - 1));
printf("p1 value plus scalar: %d\n", (*p1) + 2);
printf("p1 plus offset followed: %d\n", *(p1 + 2)
printf("p1 plus offset followed: %d\n", p1[2]);
```

Question 6: Array and Pointer equivalence

The array and pointer type holds an address as its value, a common operation on a array and pointer types are dereferencing operations (`*`) which allows retrieval of the value stored at the address.

We are able to retrieve the address from a value type (as well as array and pointer types) by using the address operator (`&`). Supplying an integer value to the address, you can navigate the array or pointer using integer arithmetic, referencing and derefering operations.

Given these pointer statements, can you provide an equivalent statement?

*p = _____

*(p+10) = _____

&r[20] = _____

&(g[0]) = _____

&*p = _____

p++ = _____

(&(r[5])[5]) = _____

What is the difference between pointers and arrays? Can we mix notation?

Functions

As a programmer you are able to modularise your code into functions. Similar to variables, functions have a type associated as part of the function signature.

The return type of a function requires a value of the same type to be passed back to the function caller. Only `void` functions do not return any value.

Format:

```
<return type> <function_name>([<parameter type> <parameter name>,]) {
  //Function body
}
```

Example:

```
int add_two(int a, int b) {
  return a + b;
}
```

Question 7: Swap

Implement the swap function, your function should swap the values of `a` and `b` respectively.

```
void swap(int a[], int b[]) {
    //
    // TODO
    //
}

int main(void) {
    int a = 2;
    int b = 3;
    swap(?, ?); //Specify the variables to swap
    printf("%d %d\n", a, b); // should print 3 2

    return 0;
}
```

Does your program execute differently if you change `swap` type parameters to `int*`?

Question 8: atoi

Implement your own atoi function that converts a given string to an integer.

```c
int atoi(const char s[]) {
    //
    // TODO
    //
}

int main(void) {
    printf("%d\n", atoi(""));
    printf("%d\n", atoi("0"));
    printf("%d\n", atoi("0123"));
    printf("%d\n", atoi("1234"));
    printf("%d\n", atoi("-1234"));

    return 0;
}
```

Question 9: Reverse

Write a program that reverses every line of input, you may assume none of the input lines are longer than 100 characters. Use the `fgets` function instead of `scanf` for this exercise because scanf does not distinguish between spaces, newlines and tab characters. Refer to the isspace function for more information.

Sample input:

```
abc 123
lorem ipsum
dolor sit amet
```

Sample output:

```
321 cba
muspi merol
tema tis rolod
```

Question 10: Quiz - 1

In the last 10 minutes of the tutorial, you will be required to undertake a quiz that will test your knowledge of the previous week's content. It will be a simple assessment that corresponds to 1% of your final grade. You can access the quiz on Canvas.