Richard's C Programming Course

September—December 2017

Tips welcome in XRP:
richard43NZXStHcjJi2UB8LGDQGFLKNs

Overview of Course

You will learn the basics of:

- C programming
- Bash and UNIX systems
- Computational theory
- Basic Data structures and algorithms
- Threading
- Processes
- Networking

Course Details

- Recommended Text:
  - "The C Programming Language" second ed. by K&R
- Five (5) assignments
- Course discussion including during streaming
  - https://discord.gg/aTDJSkG
- Course thread:
  - https://www.sythe.org/threads/richards-c-programming-course-follow-along/
- Course youtube:
  - https://www.youtube.com/channel/UCqK2GLaI0VTZnK0SuJBTbzw
- Price: free, but XRP tips appreciated!
  - richard43NZXStHcjJi2UB8LGDQGFLKNs

Lectures & Assignments

- One lecture per week
  - Half slides, half coding
- Check your timezone:
  - 11am Monday GMT+12 (my time)
  - 6pm Sunday GMT-5
  - 3pm Sunday GMT-8
  - 11pm GMT-0
- One assignment every 2-3 weeks
  - Don't worry they'll be fun!

Who is Richard?

Richard Holland

- Degrees:
  - University of Queensland
  - B.IT (Software Design)
  - B.Sc (Physics)

- Commercial programmer for 13 years
  - Consulting
  - Manage programming related businesses
    - E.g. https://toastwallet.com

Coding Environment

- For this course we will be compiling programs using the GNU Compiler Collection
- Everything you see will be done under the following setup
- You need to download and install Virtual Box and Ubuntu desktop
- You need to create a new VM with Ubuntu
- Instructions on the thread:
  - https://www.sythe.org/threads/richards-c-programming-course-follow-along/

C Basics

- C programs are text files
  - You can write them with any text editor
  - We will use nano, and later vi

- C programs are compiled through three major steps:
  - The preprocessor
  - The compiler
  - The linker

C Syntax

- C programs are built of functions and always start at a function named main:

```
int main(int argc, char** argv) {
    return 0;
    /* Some comment */
}
```

- { } Curly braces denote a sequence of statements to be executed
- ( ) Parentheses denote an argument list
- ; Semi colon is used to end a statement
- /* anything inside these characters is a comment */

C Syntax continued

- The preprocessor takes directives starting with a hash character e.g.
  - #include <stdio.h>
  - #define PI 3.1415926
- These are not actually C code, but rather tell the preprocessor how to amalgamate the code you've written before passing to the compiler

You only learn by doing, so let's write some code!

Variables

- Should be defined at the top of your function
- Look like this:
  - `char x;`
  - `double myvar = 2.0;`
- You should initialize them
  - using the assignment operator =
- If you do not they will contain random data

Variable types

- Integer types:
  - `int`
  - `long`
- Floating point types (i.e. decimals)
  - `float`
  - `double`
- Character type
  - `char` (treated as a one byte integer)
- Null type
  - `void`

Arrays

- Variables sit in memory
- An array is a number of variables next to each other in memory
- This code creates an array of 10 integers

```
int x[10];
```

- We can access them like normal variables using the square brace notation:

```
x[5] = 1;
```

- Arrays are zero indexed

Strings

- In C, strings or text is represented as an array of characters with an extra 'null character' at the end:

```
char str[6] = "hello";
```

- In memory this looks like:

h	e	l	l	o	\0

- Each box in the array contains a character, which is actually a single byte integer 0-255
- The ASCII standard maps numbers to characters for display

Conditionals

- If statement

```
if (i > 5) {
    printf("greater\n");
} else {
    printf("not greater\n");
}
```

- Switch statement

```
switch(i) {
    case 1:
            printf("i is one\n");
            break;
    case 2:
            printf("i is two\n");
            break;
    default:
            printf("i is neither one nor two\n");
}
```

Loops

- While Loop

```
while (i > 5)
    printf("greater\n");
    i = i - 1;
}
```

- For loop

```
for (i = 10; i > 5; i = i - 1) {
    printf("greater\n");
}
```

- Do while loop

```
do {
    printf("greater\n");
    i = i - 1;
} while (i > 5);
```

Function calls

- Call a function by using the name of the function and passing arguments to it in parens:
  - `printf("testing\n");`
- Everything in C is pass-by-value
  - (more on this later)
- Ask yourself what this program does:

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

Basic input and output functions

- Printing to the screen (i.e. stdout):
  - `printf("%d", myint);`
- Reading from the keyboard (i.e. stdin)
  - `scanf("%d", &myint);`
- More on how this & operator works later, when we do pointers

Let’s do an example