C Lectures

1.Introduction to C

Salih MSA

May 15, 2022

Abstract

By the end of this presentation, you would've understood the following:

- Why to learn C
- The structure of C programs
- How to run C programs
- The purpose and basic usage of a C compiler

Why learn C?

- C is highly efficient - used in operating systems, embedding systems, ML/AI, back-end for other languages
- C is simplistic - has concise set of keywords to express computation. Two benefits: less time spent learning specifically about the quirks of the C language and how to program in it, whilst allowing you to appreciate the motivations and features in other programming languages to ease more complex computation
- C gives a birds-eye view into workings of a computer - helps visualise the inner workings of computer systems (such as how programs are organised at a fundamental level in memory, CPU manipulation, etc.)

You get the point, I like C and it's a good language, certainly as a starting point.

How to build your first program

Structure - Theory

In C, source code are stored in files with the extension '.c' (e.g. my_first_program.c). Code to be accessed in other files are stored in files with the extension '.h' [1](e.g. my_library.h)
Each *statement* in C, regardless of the file it's located in, ends with a semicolon.
One of the .c files needs to contain a block of code called 'main', which the program calls first to run the program. [2]

[1]This is an over-simplification: .h are typically limited to the function prototypes, and are linked with the actual code (typically written in .c files) later. But .h files can be used for storing whatever you want despite any drawbacks so I'll live with the equivocation for now

[2]This CAN be changed, certainly Microsoft Visual Stdio let's you do so. But seriously, everyone likes main

How to build your first program

Structure - Example

Create a source code file called ‘my_first_program.c’ and put the following code inside of it:

```c
/* HI, im a comment
 * The program doesn't recognise me as anything valid
 * won't affect your code */

int main(void) /* there's different ways of writing this
                * this is the easiest */
{
     return 0 ; /* statement, therefore semicolon */
}
```

How to build your first program

Compiling - Theory

Compiling is the “transformation from source code into machine code"[3]. In laments terms, English code (or any other language) means nothing to your processing units - it needs to be translated into ones and zeros as is what they typically deal with. Luckily, all that YOU need to do is find one of hundreds of programs to do it for you. The main one we will be using is called ‘gcc’[4]and it is arguably the most popular variant amongst C compilers.
By default, C compilers will create an executable called ‘a.out’ to then be run.

[3]`https://www.cs.utah.edu/~germain/PPS/Topics/C_Language/compiling_C_programs.html`

[4]`https://gcc.gnu.org/`

How to build your first program

Compiling - Example

Open a terminal window and navigate to the same directory the code was before. The first of the following commands compiles and creates the executable, the second actually runs the program:

```
# compiles, creates executable called a.out
# using standard C
gcc --std=c89 my_first_program.c

# *nix command to run a.out file
./a.out
#  ( ./ means the executable is in this directory)
```

Note that there won't be output of any significance as all we've done is tell a program to exit as it's first instruction. So let's make something more interesting!

How to build your first *useful* program

Pre-processor commands

To issue instructions to the C compiler, several directives have been created. Each directive begins with a '#', with each telling the compiler to do certain things:

  - #include insert-file-here −*copiesthecodefromthespecifiedfile*#*defineinsert* − *name* − *hereinsert* − *val* − *here* − *createsalabel*(*calledmacros*)*whichcanbereferencedanywheretoautoma* *thisincludeslinesofcode*

- #undef insert-name-here - deletes the knowledge of the label defined above
- #ifdef / #ifndef - conditional statements where actions are done based on presence of labels (if defined, if not defined)

There's more commands, but these commands, especially the first one, are most common. For now, just know you'll need to use the first one to be able to do anything useful.

How to build your first *useful* program

More about #include

There's a number of useful 'libraries' which can be used. To name a few:

- stdio.h - contains input/output functions (such as printf (printing), reading from line (fgets))
- stdlib.h - contains various general-use functions and datatypes [5]
- math.h - contains math functions [6]
- string.h - contains string manipulation functions

[5] Technically it doesn't contain datatypes, rather aliases to datatypes (known as typedefs). And some of the 'functions' are macros. But again, not really detail needed for now

[6] This is an oddball in terms of C libraries. Unlike the other C headers, you have to manually link the actual code at compilation stage. Linking is a subject in it's own right but essentially you just add '-l m' to the gcc command

How to build your first *useful* program

Writing the useful program

In the following code, we include the IO library and call the classic print functions before exiting. We'll get more into calling functions and stuff later but one must go through the rite of printing "Hello World" before doing anything else.
Write the following code in a file called 'my_first_useful_program.c':

```c
#include <stdio.h>
/* library of input / output functions */

int main(void)
{
    printf("Hello World!") ;
    /* call print function imported */

    return 0 ;
}
```

How to build your first *useful* program

Running the useful program

Compile and run using the following commands [7]:

```
# compiles, creates executable called a.out
# using standard C
gcc --std=c89 my_first_useful_program.c

# *nix command to run a.out file
./a.out
#  ( ./ means the executable is in this directory)
```

You should see “Hello World!" (minus the quotation marks) printed to the screen. Congratulations, you’re now a C programmer.

[7] Note that the only difference in the building-and-running commands from before was the change in the filename - please remember this as I won’t be including such commands unless we’re doing tricky compiler stuff again (which we will later on).

How to build your first *useful* program

Your turn!

Make a copy of ‘my_first_useful_program.c’ called ‘my_second_useful_program.c’, and change the text to print your name.