# C LANGUAGE

# COMPENDIUM

WRITTEN BY

ANRICH TAIT

MARCH 2023

## Contents

# 1 INTRODUCTION

This document serves as both study notes and an easily accesible reference document for the C programming language. Please note that if I have shared this document with you please do not distribute it. I put a lot of effort into this and did not share it lightly. If you have read "The C Programming Language by Brian Kernighan and Dennis Richie" you will recognise the flow of this document, that is due to me writing the majority of these notes while studying that book. If you are just starting out with C I highly recommend you read that first. It is probably the best book on C out there.

## 1.1 How to use this document:

Throughout the document there are code blocks. These are easily identifiable bits of code that act as a examples for the topic that was explained. All code blocks are written in a way that they are directly applicable to C (you can copy paste).

## 1.2 Example code block:

```
#include <iostream>
using namespace std;

int main() {
    cout << "This is an example!"
    return 0;
    }

Output:
This is an example!
```

## 2 WHAT IS C?

C is a general-purpose programming language. The language has been called a "system programming language" but is useful for writing compilers, operating systems and major programs with wide application.

The fundamental types in C are characters, integers and floating-point numbers (more on these later). In addition there is a hierarchy of derived data types created with pointers, arrays, structures and unions. Expressions are formed with operators and operands; any expression, including an assignment or a function call can be a statement.

C provides the fundamental control-flow constructions required for well-structured programs: statement grouping, decision making (if-else), selecting one of a set of possible cases (switch), looping with the termination test at the top (while, for) or at the bottom (do), and early loop exit (break).

Functions may return values of basic types, structures, unions or pointers. Any function may be called recursively. Local variables are typically "automatic" or created anew with each invocation. Function definitions may not be nested but variables may be declared in a block structured fashion. Functions in a C program may exist in seperate source files that are compiled separately. Variables may be internal to a function, external but known only with a single source file or visible to the entire program.

C is a low-level language. This means that it deals with a computer's hardware components and constraints. Their prime function is to operate, manage and manipulate the computing hardware and components. Programs and applications written in low-level languages are directly executable on the computing hardware without any interpretation or translation. Another good example of low level languages is the dreaded assembly language.

C does not provide operations to deal directly with composite objects such as character strings, sets, lists or arrays. There are no operations to manipulate an entire array or string, although structures may be copied as a unit. C does not define any storage allocation facility other than static definition and the stack discipline provided by the local variables of functions. There is also no heap or garbage collection.

# 3 THE BASICS

This section will cover the basics of C, like I said if this is your first time programming I would recommend reading Brian Kernighan's book (in the Bibliography). For me this section helped mainly to remind myself of foundational concepts when I started trying to be too complicated. Remember that good code doesn't have to be complex, it has to work.

## 3.1 Everyones-first-program.c

No matter what language you are learning your first program will always be the simple task of printing "hello, world".

The task for first timers can be a pretty big hurdle. These are the steps I followed (bear in mind I am using arch linux and vim)

1. Create a new C file: touch hello-world.c
2. Open the file with vim: vim hello-world.c
3. Enter your code: See below.
4. Save the file.
5. Compile in the terminal with: gcc hello-world.c
6. Check the output for errors.
7. Fix errors.
8. Run the program with: ./a.out

```c
#include <stdio.h>

main() {
    printf("hello, world\n";
}
```

No matter the size of a C program it will always consist of a couple things. Namely, functions and variables. A function contains statements that specify the computing operations to be done. and variables store values used computation.

The above example uses the function "main". You can give a function any name but "main" is special. "main" is the function that the program will execute first. Therefore every program must have a "main" function somewhere inside it. "main" usually calls other functions to help perform it's job (some that you wrote and others provided by the libraries listed at the top of the program.

In this program the library "<stdio.h>" was included. This tells the compiler to include information about the standard input and output library. One way to cumminicate data between functions is by calling for the function to provide a list of values called "arguments". The parentheses after the function name surround the argument list.

```
#include <stdio.h> // include information about standard libray
main ()           // defines a function named main
{                 // statements of main are enclosed in braces
printf("hello, world")
}
```

In the above example "main" is defined to be a function that expects no arguments (this is indicated by the empty list "()" ). The statements of a function are enclosed in braces . The function main contains only one statement.

```
printf("hello, world")
```

Like mentioned before main calls the library function printf. This is used to output ("printf") the characters "hello, world".

The characters "hello, world" are called a character string or string constant, this will be explained further later in the document.

Keep in mind that the compiler will insert a new line. You need to provide it with the forward brace and the n character. Below the previous program is written without a newline character to demonstrate that the same output can be produced in a different format.

```
#include <stdio.h>

main () {
   printf("hello, ");
   printf("world");
   printf("\n")   //notice the \n character at the end.
}
```

The above code will result with an identical output to the previous example. This is because the C compiler does not recognize white space. Unlike languages like Python, white space is only used to make code more readable.

We create white space in output using things called escape sequences. More on this later.

## 4 Bibliography and other interesting links:

- The C Programming Language (2nd Edition) by Brian Kernighan, Dennis M. Ritchie
- For more information on what a low-level language is: techopedia.com/definition/3933/low-level-language