

DeepDive
Functions in C
&
Intro
Loops



Beta

@cl8mentor



Agenda

1. Understanding your first C program
2. C Program execution
3. C program Compilation process (PCAL)
4. Understanding Variables
5. Googling Function
6. Intro to loops

Your first C program

- C is a **compiled programming language**, like Go, Java, Swift or Rust.
- A compiled language generates a **binary file** that can be directly **executed** and distributed.
- C is **not** garbage collected. This means we have to **manage memory ourselves**. It's a complex task and one that requires a lot of attention to prevent bugs, but it is also what makes C **ideal** to write programs for embedded devices like Arduino.
- C does not hide the complexity and the capabilities of the machine underneath. You have a lot of power, once you know what you can do.




cont...

alx

```
#include <stdio.h>

int main(void) {
    printf("Hello, World!");
}
```

- we first import the **stdio** library: gives access to input/output functions.
 - **Why have libraries?** C is a very small language at its core, and anything that's not part of the core is provided by libraries. Some of those libraries are built by normal programmers, and made available for others to use. Some other libraries are built into the compiler e.g stdio and others.
 - **stdio** is the library that provides the **printf()** function.
 - This function is wrapped into a **main()** function. The **main()** function is the **entry point** of any C program.
- 




cont...

alx

```
#include <stdio.h>

int main(void) {
    printf("Hello, World!");
}
```

- What is a function? A function is a routine that takes one or more arguments, and returns a single value.
 - In the case of `main()`, the function gets no arguments, and returns an integer. We identify that using the `void` keyword for the argument, and the `int` keyword for the return value.
 - The function has a body, which is wrapped in curly braces. Inside the body we have all the code that the function needs to perform its operations.
 - The `printf()` function is written differently, as you can see. It has no return value defined, and we pass a string, wrapped in double quotes. We didn't specify the type of the argument.
 - That's because this is a function invocation. Somewhere, inside the `stdio` library, `printf` is defined as `int printf(const char *format, ...);`
- 

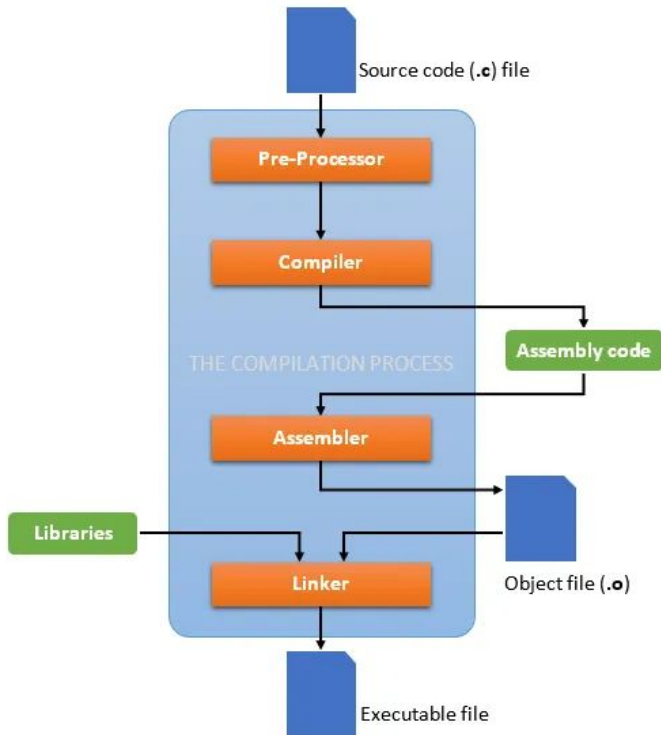
C program execution

```
#include <stdio.h>

int main(void) {
    printf("Hello, World!");
}
```

- The C prog. will be run by the operating system when the program is executed
- To run the program we must first **compile** it. Any Linux or macOS computer already comes with a C compiler built-in.
- A compiler is a special program that translates a programming language's source code into machine code, bytecode or another programming language.
- GCC is the most common C language compiler

C program compilation (PCAL)



Compilation is the translation of source code (the code we write) into object code (sequence of statements in machine language) by a compiler.

The compilation process has four different steps:

1. The preprocessing stage
2. The compiling stage
3. The assembling stage
4. The linking stage

Step 1: Preprocessing -E

- The preprocessor has several roles:
 - it gets rid of all the comments in the source file(s)
 - it includes the code of the header file(s), which is a file with extension `.h` which contains C function declarations and macro definitions
 - it replaces all of the macros by their values
- The output of this step will be stored in a file with a `.i` extension

Step 2: Compiling -S

- The compiler will take the preprocessed file and generate **IR** code (Intermediate Representation), so this will produce a **.s** file.
- Other compilers might produce assembly code at this step of compilation.

Step 3: Assembling -c

- The assembler takes the IR code and transforms it into object code, that is code in machine language (i.e. binary). This will produce a file ending in .o

Step 4: Linking

The linker creates the final executable, in binary, and can play two roles:

- linking all the source files together, that is all the other object codes in the project. For example, if I want to compile **main.c** with another file called **secondary.c** and make them into one single program, this is the step where the object code of **secondary.c** (that is **secondary.o**) will be linked to the **main.c** object code (**main.o**).
- linking function calls with their definitions. The linker knows where to look for the function definitions in the static libraries or dynamic libraries.

cont..

- By default, after this fourth and last step, that is when you type the whole `gcc main.c` command without any options, the compiler will create an executable program called **a.out**, that we can run by typing `./a.out` in the command line.
- We can also choose to create an executable program with the name we want, by adding the "-o" option to the gcc command, placed after the name of the file or files we are compiling

Variables



Variables

Variables are the **storage area** in a code that the program can easily manipulate. Every variable in C language has some specific type- that determines the layout and the size of the memory of the variable, the range of values that the memory can hold, and the set of operations that one can perform on that variable.

X and y

```
X = 100, y = 200
```

```
X = 120
```


Rules for Naming a Variable

1. The name of the variable must not begin with a digit.
2. A variable name can consist of digits, alphabets, and even special symbols such as an underscore (_).
3. A variable name must not have any keywords, for instance, float, int, etc.
4. There must be no spaces or blanks in the variable name.
5. The C language treats lowercase and uppercase very differently, as it is case sensitive. Usually, we keep the name of the variable in the lower case.

```
8name = "Charles";           //invalid
Name8 = "Charles";           //valid _name
First name = "Charles";       //invalid
```

Data Type of the Variable

We must assign a data type to all the variables that are present in the C language. These define the type of data that we can store in any variable. If we do not provide the variable with a data type, the C compiler will ultimately generate a syntax error or a compile-time error.

```
char height = 174;  
int height = 174;  
char a = 2;
```

Functions

google.com



Resources

1. [High and Low level languages](#)
2. [C - beginner's Handbook](#)
3. [Compiling C files with GCC](#)
4. [C Functions](#)
5. [Loops in C](#)

**See you at
the next
session!**

