

EXERCISE 1

Content:

1. What is C Language?
2. Code, Compile & Run
3. My First C Program
4. Compilation process in c

I. What is C Language

The C Language is developed by Dennis Ritchie for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.

C programming is considered as the base for other programming languages, that is why it is known as mother language.

It can be defined by the following 5 ways:

1. Mother language

C language is considered as the mother language of all the modern programming languages because most of the compilers, JVMs, Kernels, etc. are written in C language, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.

It provides the core concepts like the **array**, **strings**, **functions**, **file handling**, etc. that are being used in many languages like C++, Java, C#, etc.

2. System programming language

A system programming language is used to create system software. C language is a system programming language because it can be used to do low-level programming (for example driver and kernel). It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.

It can't be used for internet programming like Java, .Net, PHP, etc.

3. Procedure-oriented programming language

A procedure is known as a function, method, routine, subroutine, etc. A procedural language specifies a series of steps for the program to solve the problem.

A procedural language breaks the program into functions, data structures, etc.

C is a procedural language. In C, variables and function prototypes must be declared before being used.

4. Structured programming language

A structured programming language is a subset of the procedural language. Structure means to break a program into parts or blocks so that it may be easy to understand.

In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

5. Mid-level programming language

C is considered as a middle-level language because it supports the feature of both low-level and high-level languages. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).

A Low-level language is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.

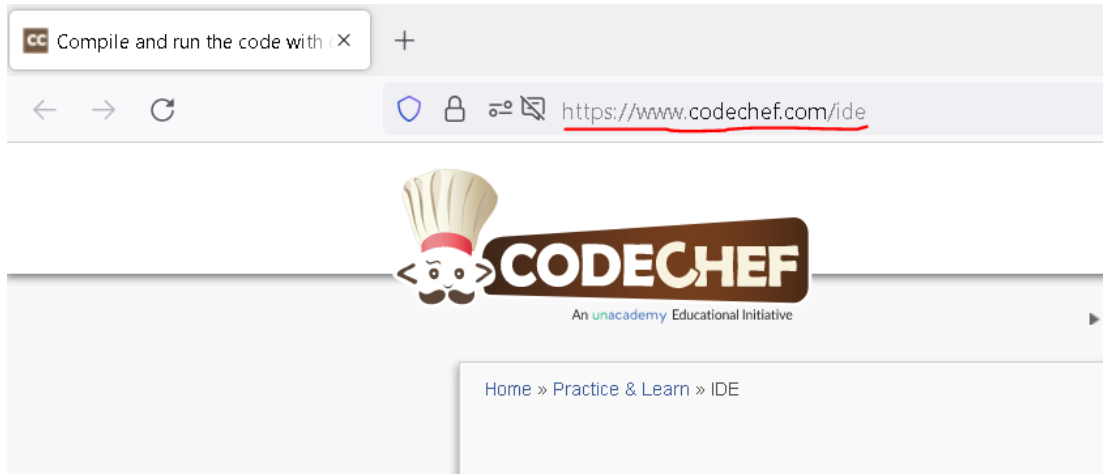
A High-Level language is not specific to one machine, i.e., machine independent. It is easy to understand.

II. Code, Compile & Run

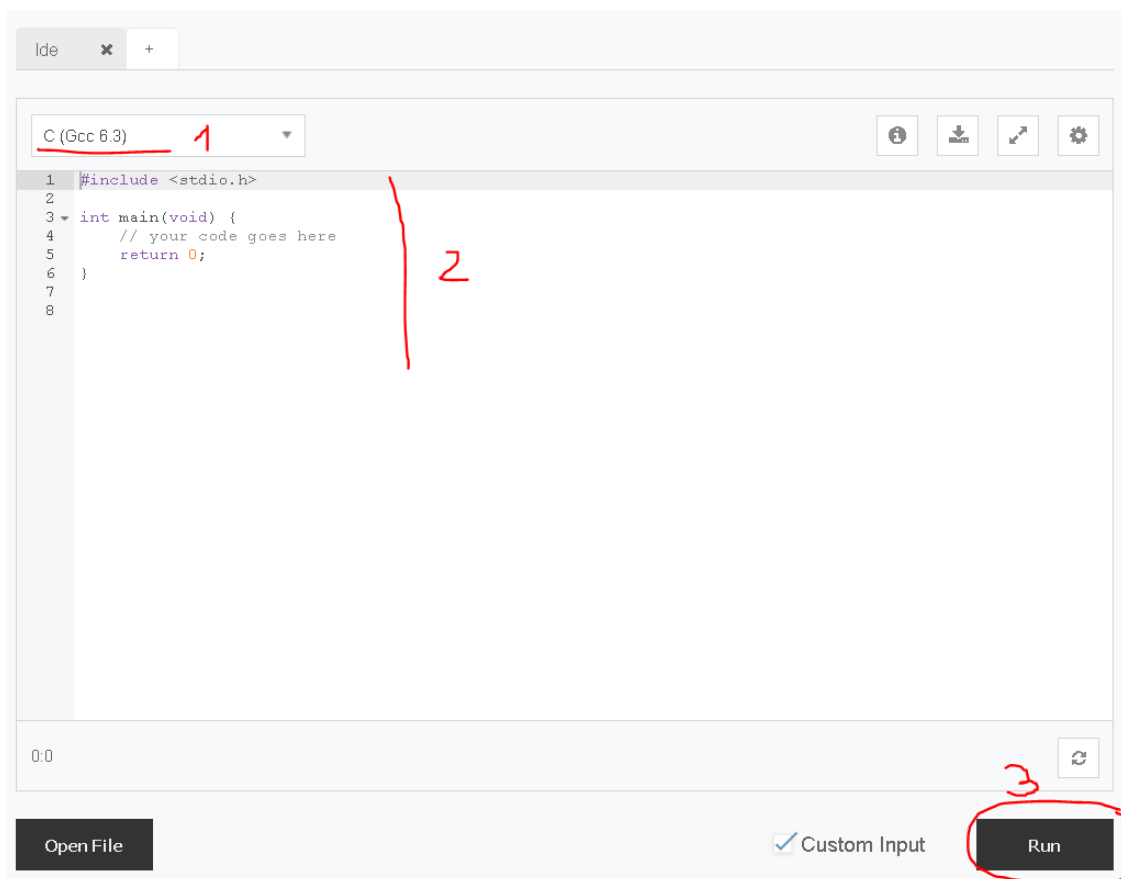
For the purpose of our labs we will use the CodeChef online IDE to compile & run our code. This online compiler supports multiple programming languages like Python, C++, C, Kotlin, NodeJS, and many more, but we will use it for C programming language.

You need a web browser. It can be Firefox, Chrome or the browser you prefer.

Step 1: Open your browser (for example Firefox) and load the URL <https://www.codechef.com/ide>



Step 2: In IDE you have to FIRST to **select from the drop-down menu C** language (1), SECOND you can **type your code** in typing area (2) and THIRD when you are ready with your program you can **compile and run it using RUN** (3) button, as it is shown on the picture below:



III. My First C Program

Before starting the abcd of C language, you need to learn how to write, compile and run the first c program.

Step 1: To write the first c program, open the CodeChef IDE (see part II) and write the following code:

```
1  #include <stdio.h>
2
3  int main(void) {
4      // your code goes here
5      printf("Hello World! This is my first program in C.");
6      return 0;
7  }
```

#include <stdio.h> includes the standard input output library functions. The **printf()** function is defined in **stdio.h**.

int main() The main() function is the entry point of every program in c language.

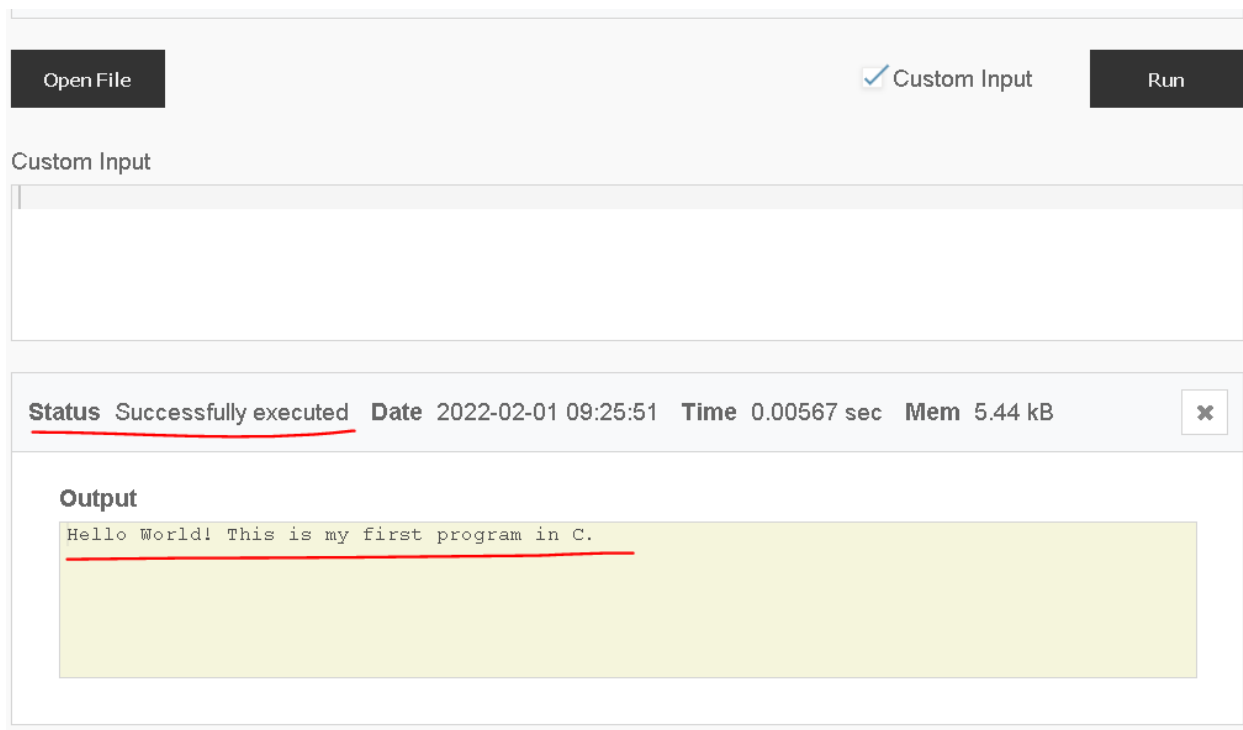
printf() The printf() function is used to print data on the console.

return 0 The return 0 statement, returns execution status to the OS. The 0 value is used for successful execution and 1 for unsuccessful execution.

Step 2: Compile and run the c program by **RUN** button:



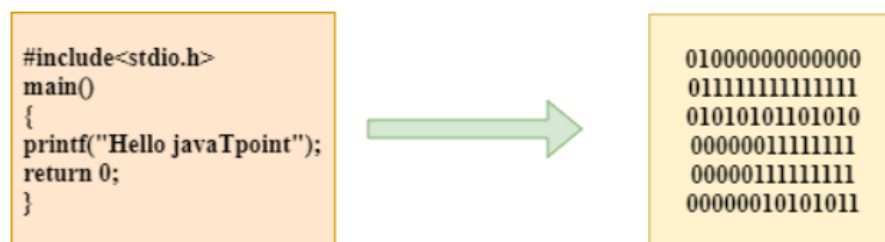
Step 3: You will see the following output on the screen in section **Output:**



The screenshot shows a web-based IDE interface. At the top, there are three buttons: 'Open File', 'Custom Input' (which is checked), and 'Run'. Below these is a 'Custom Input' text area. The main area displays the execution status: 'Status Successfully executed', 'Date 2022-02-01 09:25:51', 'Time 0.00567 sec', and 'Mem 5.44 kB'. Below the status bar, the 'Output' section shows the text 'Hello World! This is my first program in C.' on a yellow background.

IV. Compilation process in c

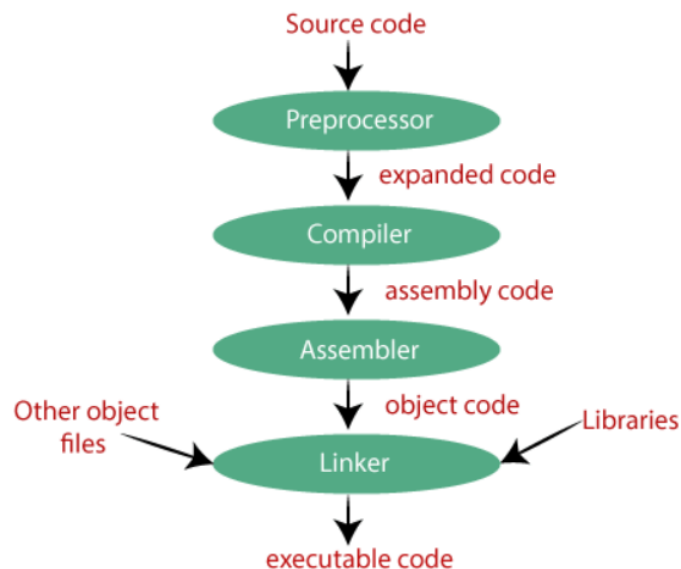
What is a compilation? The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.



The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.

The preprocessor takes the source code as an input, and it removes all the comments from the source code. The preprocessor takes the preprocessor directive and interprets it. For example, if `<stdio.h>`, the directive is available in the program, then the preprocessor interprets the directive and replace this directive with the content of the 'stdio.h' file.

Preprocessor, Compiler, Assembler and Linker are the phases through which our program passes before being transformed into an executable form.



Preprocessor

The source code is the code which is written in a text editor and the source code file is given an extension ".c". This source code is first passed to the preprocessor, and then the preprocessor expands this code. After expanding the code, the expanded code is passed to the compiler.

Compiler

The code which is expanded by the preprocessor is passed to the compiler. The compiler converts this code into assembly code. Or we can say that the C compiler converts the pre-processed code into assembly code.

Assembler

The assembly code is converted into object code by using an assembler. The name of the object file generated by the assembler is the same as the source file. The extension of the object file in DOS is '.obj,' and in UNIX, the extension is 'o'. If the name of the source file is 'hello.c', then the name of the object file would be 'hello.obj'.

Linker

Mainly, all the programs written in C use library functions. These library functions are pre-compiled, and the object code of these library files is stored with '.lib' (or '.a') extension. The main working of the linker is to combine the

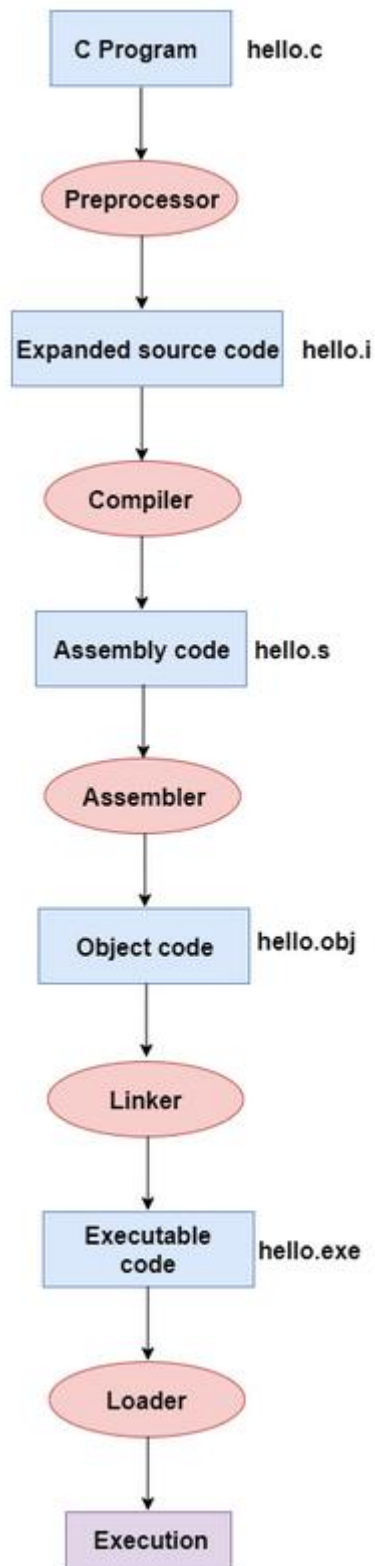
object code of library files with the object code of our program. Sometimes the situation arises when our program refers to the functions defined in other files; then linker plays a very important role in this. It links the object code of these files to our program. Therefore, we conclude that the job of the linker is to link the object code of our program with the object code of the library files and other files. The output of the linker is the executable file. The name of the executable file is the same as the source file but differs only in their extensions. In DOS, the extension of the executable file is '.exe', and in UNIX, the executable file can be named as 'a.out'. For example, if we are using printf() function in a program, then the linker adds its associated code in an output file.

Let's understand through an example.

hello.c

```
#include <stdio.h>
int main()
{
    printf("Hello world!");
    return 0;
}
```

Now, we will create a flow diagram of the above program:



In the above flow diagram, the following steps are taken to execute a program:

- Firstly, the input file, i.e., `hello.c`, is passed to the preprocessor, and the preprocessor converts the source code into expanded source code. The extension of the expanded source code would be `hello.i`.
- The expanded source code is passed to the compiler, and the compiler converts this expanded source code into assembly code. The extension of the assembly code would be `hello.s`.
- This assembly code is then sent to the assembler, which converts the assembly code into object code.
- After the creation of an object code, the linker creates the executable file. The loader will then load the executable file for the execution.