Introduction to C Programming

Introduction

In the previous chapter, we learned how to design algorithms and draw flowcharts. These help us plan and understand a solution step by step before we start coding. But planning is only the first part. To make our solution work on a computer, we must express these logical steps in a specific programming language such as Python, Java or C. Among these languages, C is very important. It is fast, powerful, and forms the base for many other programming languages. In this chapter, we will start learning C programming. We will learn how to write simple C programs, understand the main components of a C program, and explore the basic characters, numbers, and symbols used in C. This is our initial step into the world of computer programming.

Why Do We Need a Programming Language?

Natural languages often suffer from ambiguity, where a single sentence can have multiple interpretations. This is a significant challenge when communicating with computers, which require instructions that are precise and unambiguous. Unlike humans, computers cannot conclude meaning from unclear instructions. Programming languages address this by enforcing strict rules. These rules ensure that every instruction has exactly one intended meaning. Learning a programming language is a new form of communication. We can effectively command computers to perform specific tasks.

Understanding Programs and Programming

Before exploring C, let us start with core programming concepts. A program is a specific sequence of instructions directing a computer to perform tasks, such as calculations or data display. Since computers process only binary codes (0s and 1s), we use structured programming languages like C that enforce unambiguous sentence structure. Specialized tools called compilers then translate this human-readable code into executable machine language (binary codes). Programming languages exist on a spectrum: High-level languages prioritize human readability and abstraction, while low-level languages offer hardware control at the cost of complexity. C uniquely bridges these categories, providing both efficiency and expressive power.

History of C Language

The C Language was developed by computer scientist Dennis Ritchie at Bell Labs in 1972. The C programming language was created to overcome limitations found in earlier languages like B and BCPL. Ritchie aimed for a language that was simpler, faster and could access computer hardware directly. Despite its age, C remains widely used globally because it is relatively easy to learn, could run efficiently on many different computer systems and allowed programmers to write powerful and efficient code. Its core ideas heavily influenced later major computer languages. Today, C continues to be a fundamental language taught in schools and used in industry to build software systems.

Block diagram of a C program

Figure 6.1 shows the basic flow of a C program. It starts with the Preprocessor directives like *#include* followed by the main Function int *main()*. Then, variables are declared. Output is produced using *printf()*, and the program ends with a return statement, usually return 0;, indicating successful execution. First step to write a C program is to type it using an editor. We can use any available editor on our computer systems. Compiler like GCC can be used for compiling a C program.

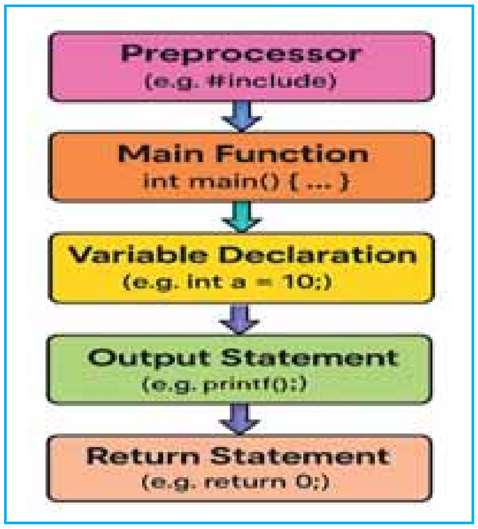


Figure 6.1 : Basic flow of a C Program

Step-by-Step: Create, Save, Open, and Run - hello.c

1. Open an editor
2. Type your code
3. Click on **File → Save As**
4. Name the file as: hello.c
5. Make sure to **save it with ".c" extension** (not .txt)

Following C program demonstrates how to print "Hello, World!" on screen.

```
/* C Program to print Hello World! */

#include <stdio.h>
int main()
{
      printf("Hello, World!\n");

      return 0;
}
Result:
Hello, World!
```

This is a basic program that beginners write to understand how C programming works. C program begins with the line **#include <stdio.h>,** which tells the computer to include the Standard Input/Output library. This allows us to use functions like *printf()* to display output on the screen. The second line, **int main()**, marks the starting point of the program. Following this, the opening curly brace { indicates the beginning of the main function's body where the program instructions will be written. Inside the body, the line **printf("Hello, World!\n");** is used to print the message “Hello, World!” on the screen and the \n in the string moves the cursor to the next line. The line **return 0;** tells the computer that the program has successfully finished its execution. Finally, the closing curly brace } marks the end of the main function. This simple program demonstrates the basic structure and functionality of a typical C program. Steps to write, save, compile and execute C program are given at the end of this chapter.

Character Set in C Language

In C programming, the character set refers to the collection of characters that are used to write programs. These characters include letters, digits, special symbols, white spaces and other control characters. Understanding the character set is essential because every program is made up of valid characters that form variables, functions, operators and other elements of the language. Table 6.1 shows the character set of C language.

Character Set Category	**Description**	**Examples**
1. Letters	Alphabets used to form identifiers (variables), keywords and other names in C. Both uppercase and lowercase letters are allowed while declaring identifiers.	• Uppercase letters (A–Z) • Lowercase letters (a–z) **Example:** To declare variable names. • int Age; char name;
2. Digits	Used to represent numeric values, array indices, etc.	• 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 **Example:** To define numeric constants, array index, etc. • int num = 45; • int num[5];
3. Special symbol / characters	Symbols that have a special meaning in C (used to specify operators, delimiters, structuring code and defining syntax).	• Arithmetic operators like +, -, *, / etc. • Address of operator (&) • Logical AND (&&), Logical OR (||), etc. • Curly braces to define code blocks { } • assignment operator (=) and • Many more like , = =, (), [], #, %, !, ^ etc.

Table 6.1 : Character Set in C

White Spaces

In C programming, white space characters are used to separate words, symbols and elements in the source code. They do not affect the execution of the program but are important for making the code readable and organized. Common white space characters include spaces, tabs and newlines. Table 6.2 describes different types of white space characters used in C.

White Space Character	Usage/Description
Space	Used to separate keywords, identifiers and operators (e.g., `int a = 10;`)
Tab (\t)	Used to indent code for better structure and readability
Newline (\n)	Moves the cursor to the next line; often used in output
Carriage Return (\r)	Returns the cursor to the beginning of the line
Form Feed (\f)	Used in printing to move to the next page (rarely used)

Table 6.2 : White Spaces

Keywords

In C programming, keywords are predefined, reserved words that have special meanings to the compiler. They are used to perform specific tasks and control the flow of the program. Keywords cannot be used as names for variables, functions or any other identifiers because they are part of the C syntax. C has a total 32 keywords. Table 6.3 shows some of the commonly used keywords.

Keyword	Meaning/Usage
int	Used to declare integer variables. Like 1, 500, 78
float	Used to declare floating-point variables. Like 784.5, 6.78
char	Used to declare character variables
return	Used to return a value from a function
if	Used for conditional branching
else	Used with 'if' for alternative execution
while	Used to create loops that run as long as a condition is true
for	Used to create loops with initialization, condition and increment/ decrement in one line

void	Specifies that a function returns no value
break	Terminates loops or switch statements

Table 6.3 : Keywords

Identifiers

In C programming, identifiers are the names used to identify variables, functions, arrays, structures and other user-defined elements. An identifier allows the programmer to refer to specific data or functionality in the code. Identifiers must follow specific rules in C.

1. an identifier must start with a letter (A–Z or a–z) or an underscore (_)
2. After the first character, digits (0–9), letters or underscores can be used
3. Keywords cannot be used as identifiers

As we discussed earlier, C is case-sensitive, so 'Value' and 'value' are different identifiers. Table 6.4 provides examples of valid and invalid identifiers:

Identifier	Valid/Invalid and Reason
totalMarks	Valid - starts with a letter and contains only letters
_value	Valid - starts with underscore
count1	Valid - letters and digits allowed after the first character
1value	Invalid - starts with a digit
float	Invalid - 'float' is a keyword
user-name	Invalid - hyphen is not allowed
Value	Valid - different from 'value' due to case sensitivity

Table 6.4 : Identifiers

Variables and Constants

Variables and constants are fundamental concepts used to store data within a program. A variable is a name assigned to a storage location where data is stored during the program's execution. The value of a variable may change during the execution of a program. We must declare a variable by specifying its data type (e.g., *int, float, char*) and assigning it a name before using it. For example,

int age = 25;

declares an integer variable named age and initializes it with the value 25. This value can be used in program later.

A constant is a fixed value that, once defined, cannot be altered by the program. Constants ensure that important values are not accidentally modified. A common way to define a constant is to use the *#define* pre-processor directive or the const keyword. For example,

#define PI 3.14159

declaration ensures the value of PI remains 3.14159 (constant) throughout the program. Variables and Constants allow programmers to manage and manipulate data effectively in a program.

Executing C Program in Ubuntu (Linux)

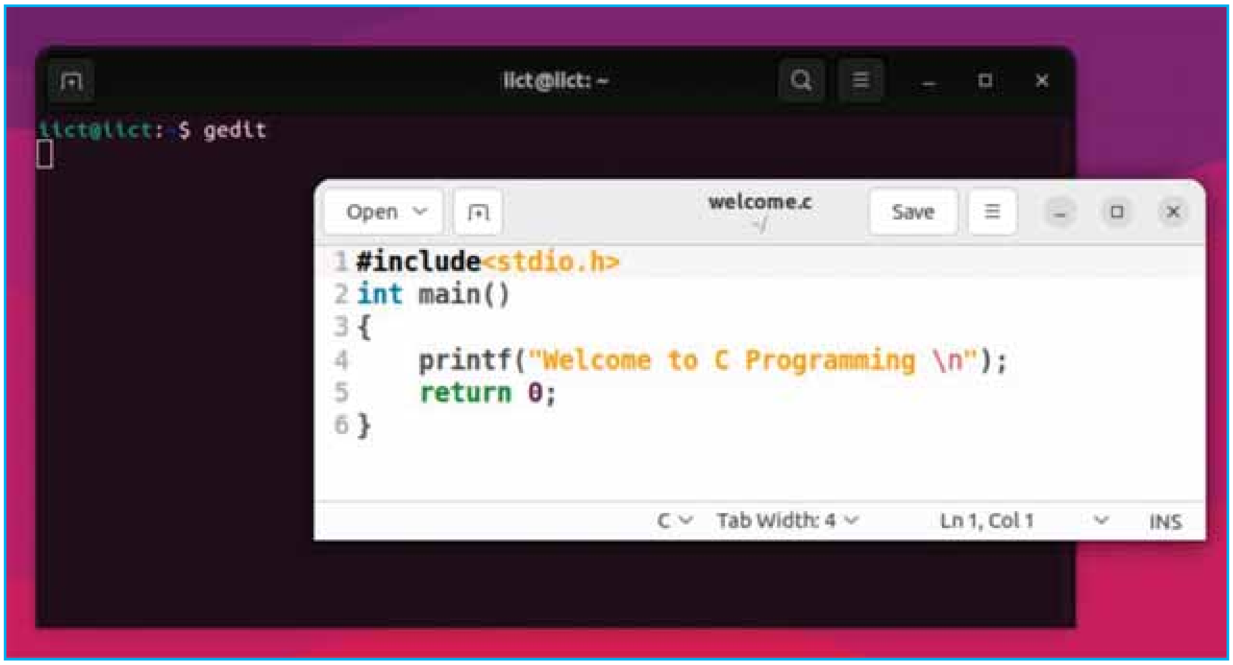
Let us understand the process of writing, compiling and executing a C program on Ubuntu operating system (OS). We can write and execute C programs using the Ubuntu OS terminal.

To run a C program in Ubuntu, follow these steps:

1. **Write the Program**

   Open any text editor like **gedit** as shown in figure 6.2 and write the following C code in **gedit** editor:

```
/* C program execution in Ubuntu OS */
#include <stdio.h>
int main()
{
   printf("Welcome to C Programming \n");
   return 0;
}
```



```
llct@llct:~$ gedit
```

```
#include<stdio.h>
int main()
{
    printf("Welcome to C Programming \n");
    return 0;
}
```

Figure 6.2 : Writing C Program in gedit Text Editor

2. **Save the Program**

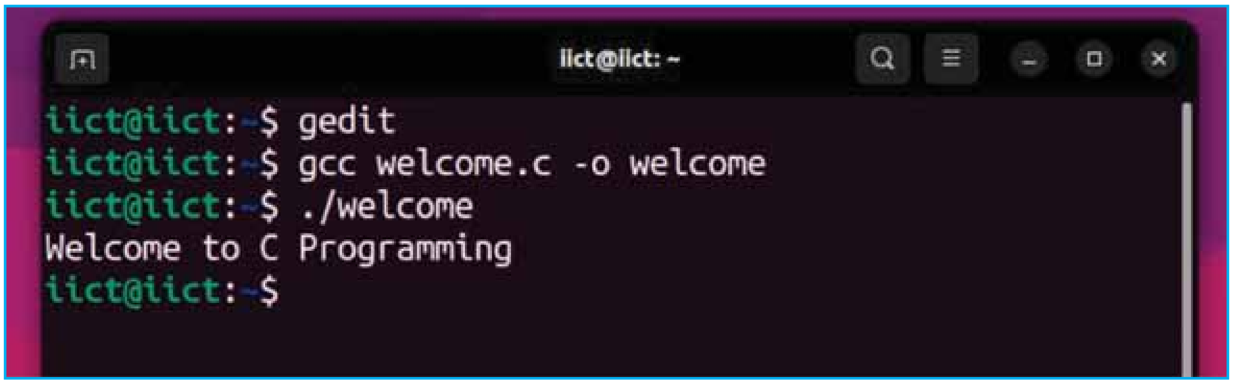
   Save the file with the name "**welcome.c**". It is crucial to ensure the file has a **".c"** extension to be recognized as a C source file.

3. **Compile the Code**

   We need GCC (GNU Compiler Collection) to compile the C Code. Most Ubuntu versions come with GCC which includes the C compiler. Launch your Ubuntu terminal and compile the program using the GCC command as shown below (line 2 of Figure 6.3):

   ```
   gcc welcome.c -o welcome
   ```

   This command compiles "**welcome.c**" and generates an executable file named "**welcome**".



```
iict@iict:~$ gedit
iict@iict:~$ gcc welcome.c -o welcome
iict@iict:~$ ./welcome
Welcome to C Programming
iict@iict:~$
```

Figure 6.3 : Compile and Execute C Program on Ubuntu Terminal

4. **Execute the Program**

   To run the compiled program, enter the command as shown below (line 3 of Figure 6.3):

   ```
   ./welcome
   ```

   Upon execution, you can see the following output on your terminal:

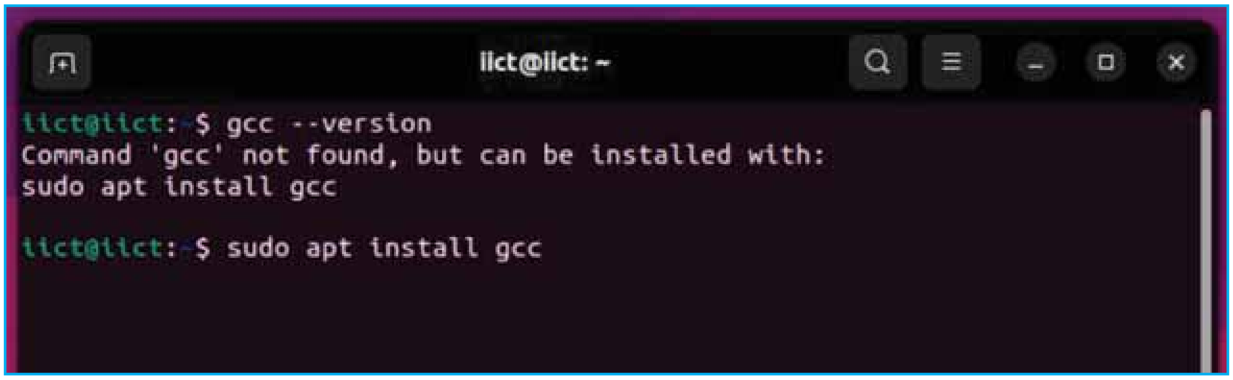
   ```
   Welcome to C Programming
   ```

By following these steps, you can successfully write, compile, and execute a C program on an Ubuntu operating system.

Consider following important points while executing C program on the Ubuntu (or any Linux-based) OS:

1. **GCC Installation:** By default, Ubuntu operating systems have GCC installed.

   - To check installation, run **gcc --version** command on your Ubuntu terminal as shown in Figure 6.4.
   - If not installed, run **sudo apt install gcc** command to install GCC in your computer as shown in Figure 6.4.



```
iict@iict:~$ gcc --version
Command 'gcc' not found, but can be installed with:
sudo apt install gcc

iict@iict:~$ sudo apt install gcc
```

Figure 6.4 : GCC Installation in Ubuntu

2. **File Extensions:** Save your program with a **.c** extension (e.g., welcome.c) so the compiler recognizes it as C code.
3. **Compilation Errors:** If there are syntax errors in your code, GCC will display error messages with line numbers. Carefully read them and fix the mistakes.
4. **Case Sensitivity:** Linux is case-sensitive. File names like **Welcome.c** and **welcome.c** are treated as different files.
5. **Good Practice:** Always use meaningful file names for source files (like welcome.c) and output files.

Summary

This chapter introduces the core concepts of C programming, a language known for its speed, efficiency and portability. We studied the importance of instructions in programming and how C serves as a bridge between human logic and machine execution. The basic structure of a C program is discussed, including character sets, keywords, and identifiers—essential elements for writing valid C code. Additionally, we cover the history of C, which was developed by Dennis Ritchie, and its continued relevance in modern software development. Key tools such as the GCC compiler and text editors are introduced for writing and running C programs. Practical examples are provided to demonstrate how to compile and execute C code on Linux systems.

EXERCISE

1. What is the use of the #include directive in a C program?
2. Why is the main() function important in C?
3. What does the return 0; statement signify in C programming?
4. What is the role of curly braces { } in a C program?
5. How do you display output in C programming?

6. Explain the history of C programming.

7. List out character set in C programming.

8. Explain Case Sensitivity in C programming with example.

9. List out the steps to run C program in Linux.

10. List out the steps to save C program file.

11. **State whether true or false.**

(1) The main() function is optional in a C program.

(2) Keywords like int and float can be used as variable names.

(3) The #include directive is used to include libraries in a program.

(4) C is not a case-sensitive programming language.

(5) printf() is used to display output in C.

12. **Fill-in the blanks.**

(1) The ……….. function marks the starting point of a C program.

(2) The ……….. directive includes a header file in a C program.

(3) The ……….. symbol is used to terminate a statement in C.

(4) ……….. is used to print text on the screen.

(5) Keywords cannot be used as ……….. in C programming.

13. **Multi-choice questions. Choose the most correct answer.**

(1) Which symbol is used to include a header file in C?

(a) $ (b) @ (c) # (d) %

(2) What is the output of printf("Hello \t World!");?

(a) Hello World? (b) Hello World

(c) Hello World! (d) Error

(3) Which keyword is used to declare an integer variable?

(a) int (b) float (c) char (d) include

(4) Which of the following is not a valid identifier in C?

(a) totalMarks (b) 1value (c) _value (d) value1

(5) What is the purpose of return 0; in C?

(a) Start the program

(b) End the main function

(c) Include header

(d) Print output

(6) Which of the following is a valid variable name?

(a) 123abc (b) _count (c) float (d) -value

(7) Which is used to declare real number?

(a) int (b) float

(c) char (d) None

(8) Which escape sequence is used for a new line?

(a) \t (b) \n (c) \ (d) \r

(9) Which of the following is used to define code blocks?

(a) () (b) [] (c) {} (d) <>

(10) Which of the following is a keyword in C?

(a) loop (b) int (c) print (d) name

Laboratory Exercises

1. Write a C program that prints your full name using the printf() function.
2. Write a C program to print "Hello, India!".
3. Write a C program to print a message in the following manner.

   Hello,

   World!