C Programming

Notes

Swipe>>>

Introduction

C programming is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at the Bell Telephone Laboratories to develop the UNIX operating system. C is the most widely used computer language. It keeps fluctuating at number one scale of popularity along with Java programming language, which is also equally popular and most widely used among modern software programmers.

Facts about C

- C was invented to write an operating system called UNIX.
- C is a successor of B language which was introduced around the early 1970s.
- The language was formalized in 1988 by the American National Standard Institute (ANSI).
- The UNIX OS was totally written in C.
- Today C is the most widely used and popular System Programming Language.
- Most of the state-of-the-art software have been implemented using C.

Applications of C Programming

C was initially used for system development work, particularly the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as the code written in assembly language. Some examples of the use of C are -

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Databases
- Language Interpreters
- Utilities

C Programs

A C program can vary from 3 lines to millions of lines and it should be written into one or more text files with extension ".c"; for example, *hello.c*.

C - Program Structure

Before we study the basic building blocks of the C programming language, let us look at a bare minimum C program structure so that we can take it as a reference in the upcoming chapters.

A C program basically consists of the following parts –

- Preprocessor Commands
- Functions
- Variables
- Statements & Expressions
- Comments

Let us look at a simple code that would print the words "Hello World" −

```
#include <stdio.h>
int main()
{
  /* my first program in C */
  printf("Hello, World! \n");
    return 0;
}
```

Let us take a look at the various parts of the above program −

- The first line of the program *#include <stdio.h>* is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
- The next line *int main()* is the main function where the program execution begins.
- The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
- The next line *printf(...)* is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
- The next line return 0; terminates the main() function and returns the value 0.

C - Basic Syntax

You have seen the basic structure of a C program, so it will be easy to understand other basic building blocks of the C programming language.

❖ Tokens in C

A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following C statement consists of five tokens −

```
printf("Hello, World! \n");
```

The individual tokens are −

```
printf
(
  "Hello, World! \n";
)
```

Semicolons ;

In a C program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

Given below are two different statements −

```
printf("Hello, World! \n");
return 0;
```

Comments

Comments are like helping text in your C program and they are ignored by the compiler. They start with /* and terminate with the characters */ as shown below −

```
/* my first program in C */
```

You cannot have comments within comments and they do not occur within a string or character literals.

Identifiers

A C identifier is a name used to identify a variable, function, or any other user-defined item. An identifier starts with a letter A to Z, a to z, or an underscore '_' followed by zero or more letters, underscores, and digits (0 to 9).

C does not allow punctuation characters such as @, $, and % within identifiers. C is a case-sensitive programming language. Thus, *Manpower* and *manpower* are two different identifiers in C. Here are some examples of acceptable identifiers –

mohd zara abc move_name a_123
myname50 _temp j a23b9 retVal

Keywords

The following list shows the reserved words in C. These reserved words may not be used as constants or variables or any other identifier names.

auto	***else***	***long***	***switch***
break	***enum***	***register***	***typedef***
case	***extern***	***return***	***union***
char	***float***	***short***	***unsigned***
const	***for***	***signed***	***void***
continue	***goto***	***sizeof***	***volatile***
default	***if***	***static***	***while***
do	***int***	***struct***	***_Packed***
double			

Whitespace in C

A line containing only whitespace, possibly with a comment, is known as a blank line, and a C compiler totally ignores it.

Whitespace is the term used in C to describe blanks, tabs, newline characters and comments. Whitespace separates one part of a statement from another and enables the compiler to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the following statement – ***int age;***

C - Data Types

Data types in c refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

The types in C can be classified as follows –

Sl.No.	Types & Description
1	**Basic Types:** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2	**Enumerated types:** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
3	**The type void:** The type specifier *void* indicates that no value is available.
4	**Derived types:** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.

Integer Types

The following table provides the details of standard integer types with their storage sizes and value ranges –

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision –

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The void Type

The void type specifies that no value is available. It is used in three kinds of situations –

Sl.No.	Types & Description
1	**Function returns as void:** There are various functions in C which do not return any value or you can say they return void. A function with no return value has the return type as void. For example, void exit (int status);
2	**Function arguments as void:** There are various functions in C which do not accept any parameter. A function with no parameter can accept a void. For example, int rand(void);
3	**Pointers to void:** A pointer of type void * represents the address of an object, but not its type. For example, a memory allocation function void *malloc (size_t size); returns a pointer to void which can be casted to any data type.

C - Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because C is case-sensitive.

The following basic variable types –

Sl.No.	Type & Description
1	**Char:** Typically a single octet(one byte). It is an integer type.
2	**Int:** The most natural size of integer for the machine.

3	**Float:** A single-precision floating point value.
4	**Double:** A double-precision floating point value.
5	**Void:** Represents the absence of type.

C programming language also allows to define various other types of variables, which we will cover in subsequent chapters like Enumeration, Pointer, Array, Structure, Union, etc. For this chapter, let us study only basic variable types.

Variable Definition in C

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows – ***type variable_list;***

Here, type must be a valid C data type including char, w_char, int, float, double, bool, or any user-defined object; and variable_list may consist of one or more identifier names separated by commas. Some valid declarations are shown here –

```
int    i, j, k;
char   c, ch;
float  f, salary;
double d;
```

The line int i, j, k; declares and defines the variables i, j, and k; which instruct the compiler to create variables named i, j and k of type int.
Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows –

```
type variable_name = value;
```

Some examples are –

```
extern int d = 3, f = 5;    // declaration of d and f.
int d = 3, f = 5;           // definition and initializing d and f.
byte z = 22;                // definition and initializes z.
char x = 'x';               // the variable x has the value 'x'.
```

For definition without an initializer: variables with static storage duration are implicitly initialized with NULL (all bytes have the value 0); the initial value of all other variables are undefined.

Lvalues and Rvalues in C

There are two kinds of expressions in C –

- **lvalue** – Expressions that refer to a memory location are called "lvalue" expressions. An lvalue may appear as either the left-hand or right-hand side of an assignment.
- **rvalue** – The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right-hand side but not on the left-hand side of an assignment.

Variables are lvalues and so they may appear on the left-hand side of an assignment. Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side. Take a look at the following valid and invalid statements –

```
int g = 20; // valid statement
10 = 20; // invalid statement; would generate compile-time error
```

Constants

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called literals.
Constants can be of any of the basic data types like *an integer constant, a floating constant, a character constant, or a string literal*. There are enumeration constants as well.
Constants are treated just like regular variables except that their values cannot be modified after their definition.

❖ Integer Literals

An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, and nothing for decimal.
An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Here are some examples of integer literals –

```
212         /* Legal */
215u        /* Legal */
0xFeeL      /* Legal */
078         /* Illegal: 8 is not an octal digit */
032UU       /* Illegal: cannot repeat a suffix */
```

Following are other examples of various types of integer literals –

```
85          /* decimal */
0213        /* octal */
0x4b        /* hexadecimal */
30          /* int */
30u         /* unsigned int */
30l         /* long */
30ul        /* unsigned long */
```

❖ Floating-point Literals

A floating-point literal has an integer part, a decimal point, a fractional part, and an exponent part. You can represent floating point literals either in decimal form or exponential form.

❖ The register Storage Class

The register storage class is used to define local variables that should be stored in a register instead of RAM. This means that the variable has a maximum size equal to the register size (usually one word) and can't have the unary '&' operator applied to it (as it does not have a memory location).

```
{
  register int  miles;
}
```

The register should only be used for variables that require quick access such as counters. It should also be noted that defining 'register' does not mean that the variable will be stored in a register. It means that it MIGHT be stored in a register depending on hardware and implementation restrictions.

❖ The static Storage Class

The static storage class instructs the compiler to keep a local variable in existence during the life-time of the program instead of creating and destroying it each time it comes into and goes out of scope. Therefore, making local variables static allows them to maintain their values between function calls.

The static modifier may also be applied to global variables. When this is done, it causes that variable's scope to be restricted to the file in which it is declared.

In C programming, when static is used on a global variable, it causes only one copy of that member to be shared by all the objects of its class.

```
#include <stdio.h>
/* function declaration */
void func(void);
static int count = 5; /* global variable */
main() {
  while(count--) {
    func();
  }
  return 0;
}

/* function definition */
void func( void ) {
  static int i = 5; /* local static variable */
  i++;
  printf("i is %d and count is %d\n", i, count);
}
```