**UNIT 1**

**History of C**

* C is one of the high-level programming languages developed by Dennis Ritchie.
* C was originally developed for UNIX operating system to beat the issues of previous languages such as B, BCPL, etc.
* The UNIX operating system development started in the year 1969, and its code was rewritten in C in the year 1972.
* The UNIX operating system development started in the year 1969, and its code was rewritten in C in the year 1972.
* In 1985, Windows 1.0 was released. Even though Windows source code isn’t publicly available on the market, it’s been stated that its kernel is mostly written in C.
* In 1991, Linux kernel development started, and it’s additionally written in C.
* After a year it was released under the GNU license and was used as part of the GNU Operating System.
* The GNU operating system was started using C and Lisp programming languages. So, many of its components are written in C.
* In 1977, Oracle database development started, and in 1983 its code was rewritten from assembly to C. It became one in all the foremost widespread databases within the world.
* Now a days C is exclusively used for building OS, application packages and customized software. Because of its power and efficiency, it has gained more popularity.
* C is increasingly used by system programmers, application developers and researchers for a spread of programming tasks.

Importance of C Language

* C is called as a robust language, which has so many built-in functions and operations, which can be used to write any complex program.
* Generally, we use to call C as a middle level language. Because, the ‘C’ compiler combines the capabilities of an assembly language with the features of a high-level language. Therefore, it is best for writing both system software and business packages.
* ‘C’ Programs are efficient and fast.
* C is highly portable, that is, ‘C’ programs written on one computer can be run on another with little (or) no modification.
* ‘C’ language is best for structured programming, where the user can think of a problem in terms of function modules (or) blocks.
* It has the ability to extend itself.

**Basic Structure of C Programs**

The basic structure of a C program is divided into 6 parts which makes it easy to read, modify, document, and understand in a particular format. C program must follow the below mentioned outline in order to successfully compile and execute. Debugging is easier in a well-structured C program.

## Sections of the C Program

1. Documentation
2. Preprocessor Section
3. Definition
4. Global Declaration
5. Main() Function
6. Sub Programs

### 1. Documentation

This section consists of the description of the program, the name of the program, and the creation date and time of the program. It is specified at the start of the program in the form of comments. Documentation can be represented as:

// description, name of the program, programmer name, date, time etc.

**or**

/\*

description, name of the program, programmer name, date, time etc.

\*/

Anything written as comments will be treated as documentation of the program and this will not interfere with the given code. Basically, it gives an overview to the reader of the program.

### 2. Preprocessor Section

All the header files of the program will be declared in the [preprocessor](https://www.geeksforgeeks.org/cc-preprocessors/) section of the program. Header files help us to access other’s improved code into our code. A copy of these multiple files is inserted into our program before the process of compilation.

**Example:**

#include<stdio.h>

#include<math.h>

### 3. Definition

Preprocessors are the programs that process our source code before the process of compilation. There are multiple steps which are involved in the writing and execution of the program. Preprocessor directives start with the ‘#’ symbol. The #define preprocessor is used to create a constant throughout the program. Whenever this name is encountered by the compiler, it is replaced by the actual piece of defined code.

**Example:**

#define long longll

### 4. Global Declaration

The global declaration section contains global variables, function declaration, and static variables. Variables and functions which are declared in this scope can be used anywhere in the program.

**Example:**

intnum = 18;

### 5. Main() Function

Every C program must have a main function. The main() function of the program is written in this section. Operations like declaration and execution are performed inside the curly braces of the main program. The return type of the main() function can be int as well as void too. void() main tells the compiler that the program will not return any value. The intmain() tells the compiler that the program will return an integer value.

**Example:**

void main()

**or**

int main()

### 6. Sub Programs

User-defined functions are called in this section of the program. The control of the program is shifted to the called function whenever they are called from the main or outside the main() function. These are specified as per the requirements of the programmer.

**Example:**

**int sum(int x, int y)**

**{**

**returnx+y;**

**}**

**Programming Style**

Overview

 **Programming style** is a set of rules or guidelines used when writing the source code for a computer program. Following a particular programming style will help programmers read and understand source code conforming to the style, and help to avoid introducing errors

1. Documentation
2. Vertical Alignment
3. Comments
4. Indentation
5. Meaningful Identifier Names Consistently Typed
6. Appropriate use of Typedef

**Executing a C program**

1. Type the program in C editor and save with.
2. Press Alt + F9 to compile the program.
3. If there are errors, correct the errors and recompile the program.
4. If there are no errors, then press Ctrl + F9 to execute/run the program.
5. Press Alt + F5 to open User Screen and check the result.

**Constant**

C Constant is the most fundamental and essential part of the C programming language. Constants in C are the fixed values used in a program, and their value remains the same during the entire program execution.

* Constants are also called literals.
* Constants can be any of the [data types](https://www.w3schools.in/c-programming/data-types).
* It is considered best practice to define constants using only **upper-case** names.
* #include<stdio.h>
* voidmain()
* {
* constint SIDE =10;
* int area;
* area = SIDE\*SIDE;
* printf("The area of the square with side: %d is: %d sq. units", SIDE, area);
* }

**Variables**

Variables are memory locations(storage areas) in the C programming language.

The primary purpose of variables is to store data in memory for later use. Unlike [constants](https://www.w3schools.in/c-programming/constants) which do not change during the program execution, the value of a variable may change during execution. If you declare a variable in C, that means you are asking the operating system to reserve a piece of memory with that variable name.

typevariable\_name,variable\_name,variable\_name;

Variable Declaration and initialization

int width, height=5;

char letter='A';

float age, area;

double d;

/\* actual initialization \*/

width=10;

age=26.5;

**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

|  |  |
| --- | --- |
| **Sr.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 specifies *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. |

The array types and structure types are referred collectively as the aggregate types. The type of a function specifies the type of the function's return value. We will see the basic types in the following section, where as other types will be covered in the upcoming chapters.

**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 |

**Character set**

A character set is **an encoding system to let computers know how to recognize Character, including letters, numbers, punctuation marks, and whitespace**

A character set can also be called **a coded character set, a code set, a code page, or an encoding**. Examples of character sets include International EBCDIC, Latin 1, and Unicode. Character sets are chosen on the basis of the letters and symbols required.

**C Tokens**

Each word and punctuation mark in a C program is a token. C tokens are the smallest building block or smallest unit of a C program. This tutorial describes C Tokens.

The C compiler breaks a program into the smallest possible units and proceeds to the various stages of the compilation, which is called a token.

C Supports Six Types of Tokens:

1. [Identifiers](https://www.w3schools.in/c-programming/identifiers)
2. [Keywords](https://www.w3schools.in/c-programming/keywords)
3. [Constants](https://www.w3schools.in/c-programming/constants)
4. [Strings](https://www.w3schools.in/c-programming/strings)
5. [Operators](https://www.w3schools.in/c-programming/operators)
6. Special Symbols
7. [Identifiers](https://www.w3schools.in/c-programming/identifiers)
8. [Identifiers](https://www.w3schools.in/c-programming/identifiers)

**Identifiers**

C Identifiers are names given to different entities such as constants, variables, structures, functions, etc. This tutorial describes C Identifiers.

int amount;

doubletotalbalance;

**Keywords**

C Keywords must be in your information because you cannot use them as a variable name. This tutorial describes various Keywords available in C.

intmain()

{

float a, b;

printf("Showing how keywords are used.");

return0;

}

**Constants**

C Constants are like a variable, except their value never changes during execution once defined. This tutorial describes C Constants.

const type constant\_name;

#include<stdio.h>

voidmain()

{

constint SIDE =10;

int area;

area= SIDE\*SIDE;

printf("The area of the square with side: %d is: %d sq. units", SIDE, area);

}

**Constants types in c**

Constants are categorized into two basic types, each of which has subtypes/categories. These are:

Primary Constants

1. Numeric Constants
	* Integer Constants
	* Real Constants
2. Character Constants
	* Single Character Constants
	* String Constants
	* Backslash Character Constants

### Integer Constant

It refers to a sequence of digits. Integers are of three types viz:

1. Decimal Integer
2. Octal Integer
3. Hexadecimal Integer

Example:

15,-265,0,99818,+25,045,0X6

### Real constant

The numbers containing fractional parts like 99.25 are called real or floating points constant.

### Single Character Constants

It simply contains a single character enclosed within ' and ' (a pair of the single quote). It is to be noted that the character '**8**' is not the same as **8**. Character constants have specific integer values known as ASCII (American Standard Code for Information Interchange).

Example:

'X','5',';'

### String Constants

These are a sequence of characters enclosed in double quotes, and they may include letters, digits, special characters, and blank spaces. Note that "**G**" and '**G**' are different - because "G" represents a string as it is enclosed within a pair of double quotes, whereas 'G' means a single character.

Example:

"Hello!", "2015", "2+1"

### Backslash character constants

C supports some character constants having a backslash in front of it. The lists of backslash characters have a specific meaning known to the compiler. They are also termed "Escape Sequences".

Example:

\t is used to give a tab \n is used to give a new line

|  |  |
| --- | --- |
| Constants | Meaning |
| \a | beep sound |
| \b | backspace |
| \f | form feed |
| \n | new line |
| \r | carriage return |
| \t | horizontal tab |
| \v | vertical tab |
| \' | single quote |
| \" | double quote |
| \\ | backslash |
| \0 | null |

**Declaration of variables**

Variables are the most essential part of any programming language. we need to calculate the area of a rectangle. To make this arithmetic calculation, we need to store the length and width of the rectangle. To store the length and width of the rectangle, we need to allocate some space in a memory location for the data, and the name given to that memory location is called **Variable**.

### General syntax for declaring a variable

In variable declarations, we can declare variables in two ways:

* **Declaration of variable without initializing any value to it**

**data\_typevariable\_name;**

**Eg:-** char Final\_Grade; // Final\_Grade is a variable of type char, and no value is assigned to it.

* **Declaration of variable with initializing some value to it**

**data\_typevariable\_name = val;**

**Eg:-** **int age = 22; //** age is a variable of type int and holds the value **22**.

Here, **data\_type** specifies the type of variable like int, char, etc.

**variable\_name** specifies the name of the variable. **val** is the value for which we are initializing the variable.

**1.Create a C program to find the size of int, float, double and char data types.**

#include<stdio.h>

intmain() {

intintType;

floatfloatType;

doubledoubleType;

charcharType;

// sizeof evaluates the size of a variable

printf("Size of int: %zu bytes\n", sizeof(intType));

printf("Size of float: %zu bytes\n", sizeof(floatType));

printf("Size of double: %zu bytes\n", sizeof(doubleType));

printf("Size of char: %zu byte\n", sizeof(charType));

return0;

}

**Problem 2 Write a C Program to find out quotient and remainder and print result**

#include <stdio.h>

intmain() {

int dividend, divisor, quotient, remainder;

printf("Enter dividend: ");

scanf("%d", &dividend);

printf("Enter divisor: ");

scanf("%d", &divisor);

// Computes quotient

quotient = dividend / divisor;

// Computes remainder

remainder = dividend % divisor;

printf("Quotient = %d\n", quotient);

printf("Remainder = %d", remainder);

return0;

}

**OUTPUT**

Enter dividend: 25

Enter divisor: 4

Quotient = 6

Remainder = 1