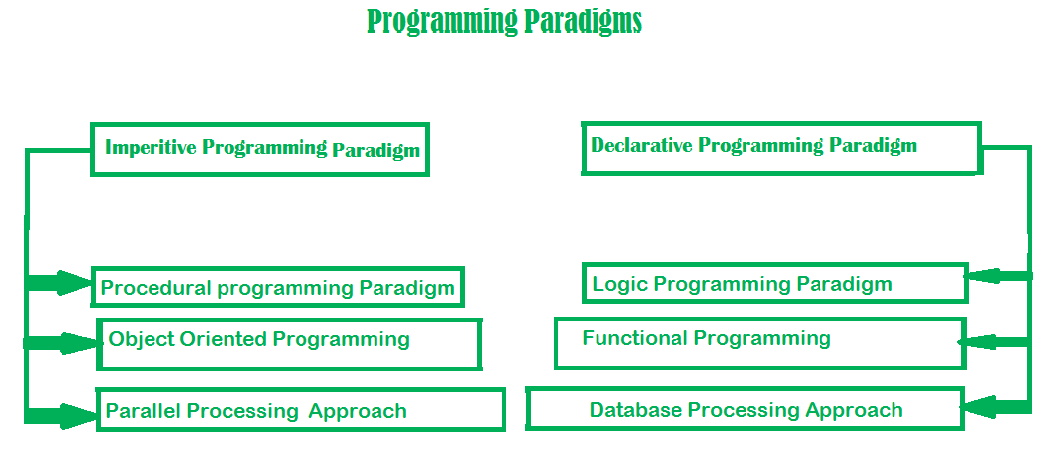
**UNIT I BASICS OF C PROGRAMMING**

**Introduction to programming paradigms – Applications of C Language - Structure of C program - C programming: Data Types - Constants – Enumeration Constants - Keywords – Operators: Precedence and Associativity - Expressions - Input/Output statements, Assignment statements – Decision making statements – Switch statement - Looping statements – Preprocessor directives - Compilation process**

**1.1 Introduction to programming paradigms**

**Paradigm** can also be termed as the method to solve some problem or do some tasks. Programming paradigm is an approach to solve problem using some programming language or also we can say it is a method to solve a problem using tools and techniques that are available to us following some approach. There are lots for programming language that are known but all of them need to follow some strategy when they are implemented and this methodology/strategy is paradigms. Apart from varieties of programming language there are lots of paradigms to fulfill each and every demand. They are discussed below:



**1. Imperative programming paradigm:** It is one of the oldest programming paradigm. It features close relation to machine architecture. It is based on Von Neumann architecture. It works by changing the program state through assignment statements. It performs step by step task by changing state. The main focus is on how to achieve the goal. The paradigm consist of several statements and after execution of all the result is stored.

**Advantages**

1. Very simple to implement
2. It contains loops, variables etc.

**Disadvantage:**

1. Complex problem cannot be solved
2. Less efficient and less productive
3. Parallel programming is not possible

**Examples of Imperative programming paradigm:** C, Fortran, Basic

Imperative programming is divided into three broad categories: Procedural, OOP and parallel processing. These paradigms are as follows:

**Procedural programming paradigm –** This paradigm emphasizes on procedure in terms of under lying machine model. There is no difference in between procedural and imperative approach. It has the ability to reuse the code and it was boon at that time when it was in use because of its reusability.

**Examples of Procedural programming paradigm:**C, C++, java, Pascal

**Object oriented programming –**The program is written as a collection of classes and object which are meant for communication. The smallest and basic entity is object and all kind of computation is performed on the objects only. More emphasis is on data rather procedure. It can handle almost all kind of real-life problems which are today in scenario.

**Advantages:**

* Data security
* Inheritance
* Code reusability
* Flexible and abstraction is also present

**Examples of Object-Oriented programming paradigm:** Simula, Java, C++, Objective-C, Visual Basic .NET

**Parallel processing approach –** Parallel processing is the processing of program instructions by dividing them among multiple processors. A parallel processing system posses many numbers of processor with the objective of running a program in less time by dividing them. This approach seems to be like divide and conquer. Examples are NESL (one of the oldest one) and C/C++ also supports because of some library function.

**2. Declarative programming paradigm:** It is divided as Logic, Functional, Database. In computer science the *declarative programming* is a style of building programs that expresses logic of computation without talking about its control flow. It often considers programs as theories of some logic.It may simplify writing parallel programs. The focus is on what needs to be done rather how it should be done basically emphasize on what code is actually doing. It just declares the result we want rather how it has be produced. This is the only difference between imperative (how to do) and declarative (what to do) programming paradigms. Getting into deeper we would see logic, functional and database.

**Logic programming paradigms –** It can be termed as abstract model of computation. It would solve logical problems like puzzles, series etc. In logic programming we have a knowledge base which we know before and along with the question and knowledge base which is given to machine, it produces result. In normal programming languages, such concept of knowledge base is not available but while using the concept of artificial intelligence, machine learning we have some models like Perception model which is using the same mechanism.   
In logical programming the main emphasize is on knowledge base and the problem. The execution of the program is very much like proof of mathematical statement, e.g., Prolog

**Functional programming paradigms –** The functional programming paradigms has its roots in mathematics and it is language independent. The key principle of this paradigms is the execution of series of mathematical functions. The central model for the abstraction is the function which are meant for some specific computation and not the data structure. Data are loosely coupled to functions.The function hide their implementation. Function can be replaced with their values without changing the meaning of the program. Some of the languages like perl, javascript mostly uses this paradigm.

**Examples of Functional programming paradigm: JavaScript**,**Haskell**, **Scala** ,**Erlang,** **Robert Virding, Lisp,ML**

**Database/Data driven programming approach –** This programming methodology is based on data and its movement. Program statements are defined by data rather than hard-coding a series of steps. A database program is the heart of a business information system and provides file creation, data entry, update, query and reporting functions. There are several programming languages that are developed mostly for database application. For example SQL. It is applied to streams of structured data, for filtering, transforming, aggregating (such as computing statistics), or calling other programs. So, it has its own wide application.

**1.2 Applications of C**

Use of the C programming language is not limited to the development of operating systems and applications. It is also used in GUI development, IDE development, etc.

1. Operating Systems:-

What is better than writing your own operating system? And yes, with the help of the C programming language, you can write your own operating system. Windows Kernel, Linux Kernel and Apple’s OS X kernel are mostly written in C.

2. GUI:-

It stands for Graphical User Interface. The C programming language also helps in developing popular adobe softwares like Photoshop, Premier Pro, Illustrator etc.

3. Embedded Systems:-

In daily life, we use different embedded systems like coffee machines, microwaves, climate control systems etc. These all are mostly programmed in C.

4. Database:-

The C programming language helps in developing the popular database management system, MySQL.

5. Ease of Computation:-

C provides faster computation in programs. The implementation of algorithms and data structures is swift in C. With the help of C, you can perform high degree calculations such as MATLAB, Mathematica etc.

6. Gaming:-

C programming is relatively faster than Java or Python. It has been used in various gaming applications and graphics. C programming language also helps in creating many popular childhood games like Tic-Tac-Toe, The Snake game etc.

7. Development of New languages:-

Due to the fast execution and simplicity, many languages like Java, C++, Python, PHP, PERL, JavaScript, etc were influenced by the development of C. In Python, C is used for building standard libraries. The syntax and control structures of PERL, PHP and C++ are based upon the C programming language.

8. Google:-

In the Google open source community, the projects are being handled by C/C++. And C/C++ also helped in developing google file system and chromium browser.

9. Assemblers:-

Mainly used to translate Assembly language to Machine language. C also helped in

developing GNU assembler.

10. Text Editors:-

C also helped in creating various text editors like Vim, Gedit etc.

11. Drivers:-

Another application of C is to write driver softwares like Keyboard driver, Network driver, mouse driver etc.

12. Interpreters:-

With the help of C programming language, you can create language interpreters. C helped in developing different programming language interpreters like Python and MATLAB interpreters etc.

13. Network Devices:-

Another application of C is to design network devices.

14. Compiler Design:-

C also helped in designing several popular compilers like Clang C, MINGW, Apple C etc. This is one of the most popular uses of C language.

**1.3 Structure of the C Program**

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.

## Sections of the C Program

There are 6 basic sections responsible for the proper execution of a program. Sections are mentioned below:

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.

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 pi 3.14

### **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:**

int num = 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 int main() 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)

{

return x+y;

}

## Structure of C Program with example

**Example:**Below C program to find the sum of 2 numbers:

* C

|  |
| --- |
| // Documentation  /\*\*   \* file: sum.c   \* author: you   \* description: program to find sum.   \*/    // Link  #include <stdio.h>    // Definition  #define X 20    // Global Declaration  **int** sum(**int** y);    // Main() Function  **int** main(**void**)  {  **int** y = 55;  **printf**("Sum: %d", sum(y));  **return** 0;  }    // Subprogram  **int** sum(**int** y)  {  **return** y + X;  } |

**Output**

Sum: 75

## **Explanation of the above Program**

Below is the explanation of the above program. With a description explaining the program’s meaning and use.

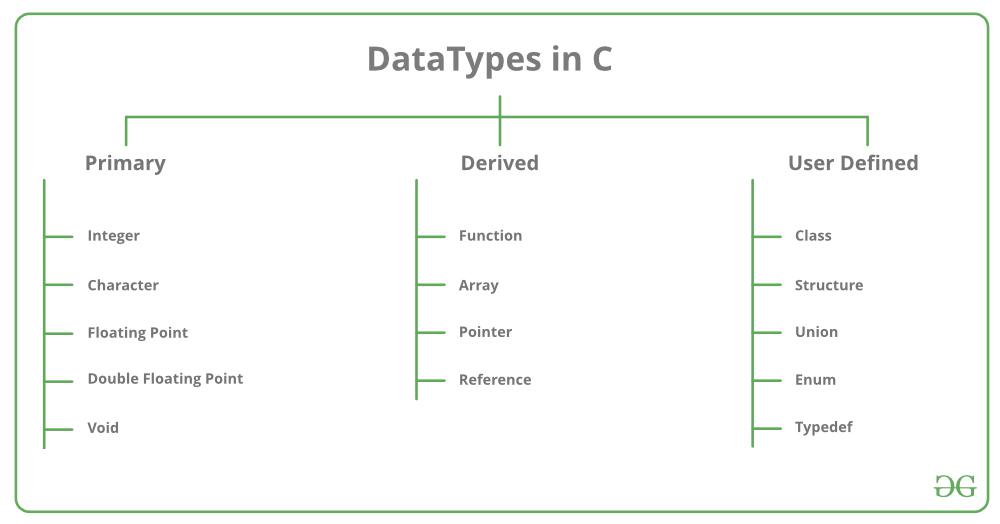
| **Sections** | **Description** |
| --- | --- |
| **/\*\*                      \* file: sum.c \* author: you \* description: program to find sum. \*/** | It is the comment section and is part of the description section of the code. |
| **#include<stdio.h>** | Header file which is used for standard input-output. This is the preprocessor section. |
| **#define X 20** | This is the definition section. It allows the use of constant X in the code. |
| **int sum(int y)** | This is the Global declaration section includes the function declaration that can be used anywhere in the program. |
| **int main()** | main() is the first function that is executed in the C program. |
| **{…}** | These curly braces mark the beginning and end of the main function. |
| **printf(“Sum: %d”, sum(y));** | printf() function is used to print the sum on the screen. |
| **return 0;** | We have used int as the return type so we have to return 0 which states that the given program is free from the error and it can be exited successfully. |
| **int sum(int y)  {  return y + X; }** | This is the subprogram section. It includes the user-defined functions that are called in the main() function. |

# 1.4 Data Types in C

Each variable in C has an associated data type. It specifies the type of data that the variable can store like integer, character, floating, double, etc. Each data type requires different amounts of memory and has some specific operations which can be performed over it. The data type is a collection of data with values having fixed values, meaning as well as its characteristics.

**The data types in C can be classified as follows:**

| **Types** | **Description** |
| --- | --- |
| **Primitive Data Types** | Primitive data types are the most basic data types that are used for representing simple values such as integers, float, characters, etc. |
| **User Defined Data Types** | The user-defined data types are defined by the user himself. |
| **Derived Types** | The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types. |



Different data types also have different ranges up to which they can store numbers. These ranges may vary from compiler to compiler. Below is a list of ranges along with the memory requirement and format specifiers on the ***32-bit GCC compiler***.

| **Data Type** | **Size (bytes)** | **Range** | **Format Specifier** |
| --- | --- | --- | --- |
| **short int** | 2 | -32,768 to 32,767 | %hd |
| **unsigned short int** | 2 | 0 to 65,535 | %hu |
| **unsigned int** | 4 | 0 to 4,294,967,295 | %u |
| **int** | 4 | -2,147,483,648 to 2,147,483,647 | %d |
| **long int** | 4 | -2,147,483,648 to 2,147,483,647 | %ld |
| **unsigned long int** | 4 | 0 to 4,294,967,295 | %lu |
| **long long int** | 8 | -(2^63) to (2^63)-1 | %lld |
| **unsigned long long int** | 8 | 0 to 18,446,744,073,709,551,615 | %llu |
| **signed char** | 1 | -128 to 127 | %c |
| **unsigned char** | 1 | 0 to 255 | %c |
| **float** | 4 | 1.2E-38 to 3.4E+38 | %f |
| **double** | 8 | 1.7E-308 to 1.7E+308 | %lf |
| **long double** | 16 | 3.4E-4932 to 1.1E+4932 | %Lf |

## **Integer Data Type**

The integer datatype in C is used to store the integer numbers(any number including positive, negative and zero without decimal part). Octal values, hexadecimal values, and decimal values can be stored in int data type in C.

* **Range:** -2,147,483,648 to 2,147,483,647
* **Size:** 4 bytes
* **Format Specifier:** %d

### **Syntax of Integer**

We use[**int keyword**](https://www.geeksforgeeks.org/int-1-sign-bit-31-data-bits-keyword-in-c/)to declare the integer variable:

**int** var\_name;

**Example of int**

|  |
| --- |
| // C program to print Integer data types.  #include <stdio.h>    **int** main()  {      // Integer value with positive data.  **int** a = 9;       // integer value with negative data.  **int** b = -9;       // U or u is Used for Unsigned int in C.  **int** c = 89U;       // L or l is used for long int in C.  **long** **int** d = 99998L;  **printf**("Integer value with positive data: %d\n", a);  **printf**("Integer value with negative data: %d\n", b);  **printf**("Integer value with an unsigned int data: %u\n",             c);  **printf**("Integer value with an long int data: %ld", d);  **return** 0;  } |

**Output**

Integer value with positive data: 9

Integer value with negative data: -9

Integer value with an unsigned int data: 89

Integer value with an long int data: 99998

## **Character Data Type**

Character data type allows its variable to store only a single character. The size of the character is 1 byte. It is the most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

* **Range:**(-128 to 127) or (0 to 255)
* **Size:** 1 byte
* **Format Specifier:** %c

### **Syntax of char**

The **char keyword** is used to declare the variable of character type:

**char** var\_name;  
**Example of char**

* C

|  |
| --- |
| // C program to print Integer data types.  #include <stdio.h>    **int** main()  {  **char** a = 'a';  **char** c;  **printf**("Value of a: %c\n", a);       a++;  **printf**("Value of a after increment is: %c\n", a);       // c is assigned ASCII values      // which corresponds to the      // character 'c'      // a-->97 b-->98 c-->99      // here c will be printed      c = 99;  **printf**("Value of c: %c", c);  **return** 0;  } |

**Output**

Value of a: a

Value of a after increment is: b

Value of c: c

## **Float Data Type**

In C programming [float data type](https://www.geeksforgeeks.org/c-float-and-double/) is used to store floating-point values. Float in C is used to store decimal and exponential values. It is used to store decimal numbers (numbers with floating point values) with single precision.

* **Range:**1.2E-38 to 3.4E+38
* **Size:** 4 bytes
* **Format Specifier:** %f

### **Syntax of float**

The **float keyword**is used to declare the variable as a floating point:

**float** var\_name;  
Example of Float

|  |
| --- |
| // C Program to demonstrate use  // of Floating types  #include <stdio.h>  **int** main()  {  **float** a = 9.0f;  **float** b = 2.5f;       // 2x10^-4  **float** c = 2E-4f;  **printf**("%f\n", a);  **printf**("%f\n", b);  **printf**("%f", c);  **return** 0;  } |

**Output**

9.000000

2.500000

0.000200

## **Double Data Type**

A [Double data type](https://www.geeksforgeeks.org/c-float-and-double/) in C is used to store decimal numbers (numbers with floating point values) with double precision. It is used to define numeric values which hold numbers with decimal values in C.

The double data type is basically a precision sort of data type that is capable of holding 64 bits of decimal numbers or floating points. Since double has more precision as compared to that float then it is much more obvious that it occupies twice the memory occupied by the floating-point type. It can easily accommodate about 16 to 17 digits after or before a decimal point.

* **Range:** 1.7E-308 to 1.7E+308
* **Size:** 8 bytes
* **Format Specifier:** %lf

### **Syntax of Double**

The variable can be declared as double precision floating point using the **double keyword:**

**double** var\_name;

### **Example of Double**

|  |
| --- |
| // C Program to demonstrate  // use of double data type  #include <stdio.h>  **int** main()  {  **double** a = 123123123.00;  **double** b = 12.293123;  **double** c = 2312312312.123123;  **printf**("%lf\n", a);  **printf**("%lf\n", b);  **printf**("%lf", c);  **return** 0;  } |

**Output**

123123123.000000

12.293123

2312312312.123123

**1.5 Constant in C**

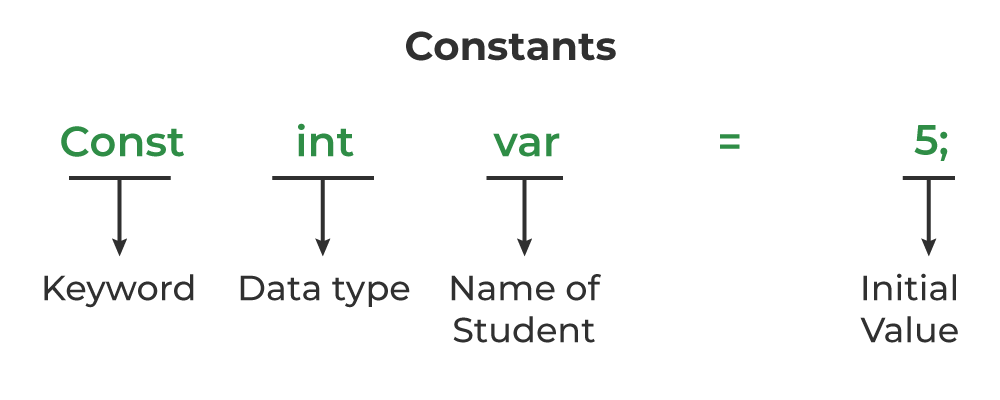
The constants in C are the read-only variables whose values cannot be modified once they are declared in the C program. The type of constant can be an integer constant, a floating pointer constant, a string constant, or a character constant. In C language, the **const**keyword is used to define the constants.

## **How to define a constant in C?**

We define a constant in C language using the **const**keyword. Also known as a const type qualifier, the const keyword is placed at the start of the variable declaration to declare that variable as a constant.

### **Syntax to Define Constant**

const *data\_type var\_name* = *value*;



## Example of Constants in C

|  |
| --- |
| // C program to illustrate constant variable definition  #include <stdio.h>  **int** main()  {       // defining integer constant using const keyword  **const** **int** int\_const = 25;       // defining character constant using const keyword  **const** **char** char\_const = 'A';       // defining float constant using const keyword  **const** **float** float\_const = 15.66;  **printf**("Printing value of Integer Constant: %d\n",             int\_const);  **printf**("Printing value of Character Constant: %c\n",             char\_const);  **printf**("Printing value of Float Constant: %f",             float\_const);  **return** 0;  } |

**Output**

Printing value of Integer Constant: 25

Printing value of Character Constant: A

Printing value of Float Constant: 15.660000

One thing to note here is that we have to**initialize the constant variables at declaration**. Otherwise, the variable will store some garbage value and we won’t be able to change it. The following image describes examples of incorrect and correct variable definitions.

**Types of Constants in C**

The type of the constant is the same as the data type of the variables. Following is the list of the types of constants

* Integer Constant
* Character Constant
* Floating Point Constant
* Double Precision Floating Point Constant
* Array Constant
* Structure Constant

We just have to add the const keyword at the start of the variable declaration.

## **Properties of Constant in C**

The important properties of constant variables in C defined using the const keyword are as follows:

### 1. Initialization with Declaration

We can only initialize the constant variable in C at the time of its declaration. Otherwise, it will store the garbage value.

### 2. Immutability

The constant variables in c are immutable after its definition, i.e., they can be initialized only once in the whole program. After that, we cannot modify the value stored inside that variable.

* C

|  |
| --- |
| // C Program to demonstrate the behaviour of constant  // variable  #include <stdio.h>  **int** main()  {      // declaring a constant variable  **const** **int** var;      // initializing constant variable var after declaration      var = 20;    **printf**("Value of var: %d", var);  **return** 0;  } |

**Output**

In function 'main':

10:9: error: assignment of read-only variable 'var'

10 | var = 20;

| ^

## **Defining Constant using #define Preprocessor**

We can also define a constant in C using #define preprocessor. The constant defined using #define are macros that behaves like a constant. These constants are not handled by the compiler, they are handled by the preprocessor and are replaced by their value before complication.

### **Syntax of Constant in C using #define**

#define *const\_name* *value*

### Example

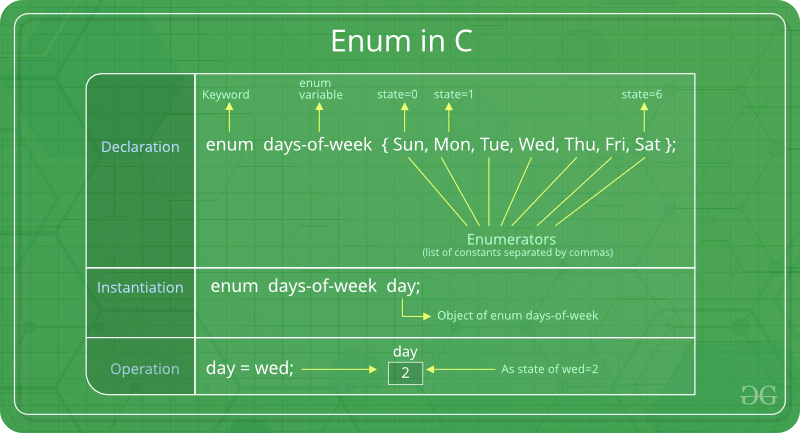
|  |
| --- |
| // C Program to define a constant using #define  #include <stdio.h>  #define pi 3.14  **int** main()  {  **printf**("The value of pi: %.2f", pi);  **return** 0;  } |

**Output**

The value of pi: 3.14

# 1.6 Enumeration (or enum) in C

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.



// In both of the below cases, "day" is

// defined as the variable of type week.

enum week{Mon, Tue, Wed};

enum week day;

// **Or**

enum week{Mon, Tue, Wed}day;

 C

|  |
| --- |
| // An example program to demonstrate working of enum in C  #include<stdio.h>  **enum** week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};    **int** main()  {  **enum** week day;      day = Wed;  **printf**("%d",day);  **return** 0;  } |

Output: 

2

In the above example, we declared “day” as the variable and the value of “Wed” is allocated to day, which is 2. So as a result, 2 is printed.  
Another example of enumeration is: 

* C

|  |
| --- |
| // Another example program to demonstrate working  // of enum in C  #include<stdio.h>    **enum** year{Jan, Feb, Mar, Apr, May, Jun, Jul,            Aug, Sep, Oct, Nov, Dec};    **int** main()  {  **int** i;  **for** (i=Jan; i<=Dec; i++)  **printf**("%d ", i);    **return** 0;  } |

Output: 

0 1 2 3 4 5 6 7 8 9 10 11

In this example, the for loop will run from i = 0 to i = 11, as initially the value of i is Jan which is 0 and the value of Dec is 11.   
**Interesting facts about initialization of enum.**   
**1.** Two enum names can have same value. For example, in the following C program both ‘Failed’ and ‘Freezed’ have same value 0. 

* C

|  |
| --- |
| #include <stdio.h>  **enum** State {Working = 1, Failed = 0, Freezed = 0};    **int** main()  {  **printf**("%d, %d, %d", Working, Failed, Freezed);  **return** 0;  } |

Output:

1, 0, 0

**2.** If we do not explicitly assign values to enum names, the compiler by default assigns values starting from 0. For example, in the following C program, sunday gets value 0, monday gets 1, and so on. 

* C

|  |
| --- |
| #include <stdio.h>  **enum** day {sunday, monday, tuesday, wednesday, thursday, friday, saturday};    **int** main()  {  **enum** day d = thursday;  **printf**("The day number stored in d is %d", d);  **return** 0;  } |

Output:

The day number stored in d is 4

**3.** We can assign values to some name in any order. All unassigned names get value as value of previous name plus one. 

* C

|  |
| --- |
| #include <stdio.h>  **enum** day {sunday = 1, monday, tuesday = 5,            wednesday, thursday = 10, friday, saturday};    **int** main()  {  **printf**("%d %d %d %d %d %d %d", sunday, monday, tuesday,              wednesday, thursday, friday, saturday);  **return** 0;  } |

Output:

1 2 5 6 10 11 12

**4.** The value assigned to enum names must be some integral constant, i.e., the value must be in range from minimum possible integer value to maximum possible integer value.  
**5.** All enum constants must be unique in their scope. For example, the following program fails in compilation. 

* C

|  |
| --- |
| **enum** state  {working, failed};  **enum** result {failed, passed};    **int** main()  { **return** 0; } |

Output:

Compile Error: 'failed' has a previous declaration as 'state failed'

**Keywords in C**

In C Programming language, there are many rules so to avoid different types of errors. One of such rule is not able to declare variable names with auto, long, etc. This is all because these are keywords. Let us check all keywords in C language

## **What are Keywords?**

Keywords are predefined or reserved words that have special meanings to the compiler. These are part of the syntax and cannot be used as identifiers in the program. A list of keywords in C or reserved words in the C programming language are mentioned below:

| auto | break | case | Char | const | continue | default | do |
| --- | --- | --- | --- | --- | --- | --- | --- |
| double | else | enum | Extern | float | for | goto | if |
| int | long | register | Return | short | signed | sizeof | static |
| struct | switch | typedef | Union | unsigned | void | volatile | while |

# 1.7 Operators in C

C Operators are symbols that represent operations to be performed on one or more operands. C provides a wide range of operators, which can be classified into different categories based on their functionality. Operators are used for performing operations on variables and values.

## **What are Operators in C?**

Operators can be defined as the symbols that help us to perform specific mathematical, relational, bitwise, conditional, or logical computations on operands. In other words, we can say that an operator operates the operands. For example, ‘+’ is an operator used for addition, as shown below:

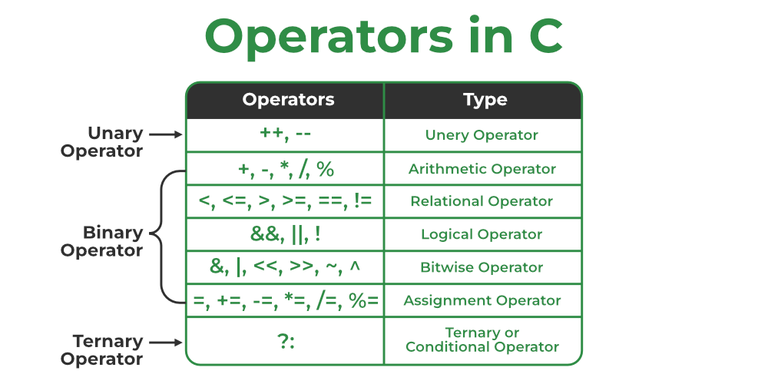
c = a + b;

Here, ‘+’ is the operator known as the addition operator, and ‘a’ and ‘b’ are operands.

## **Types of Operators in C**

C has many built-in operators and can be classified into 6 types:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Other Operators



## **1. Arithmetic Operations in C**

These operators are used to perform arithmetic/mathematical operations on operands. Examples: (+, -, \*, /, %,++,–). Arithmetic operators are of two types:

### **a) Unary Operators**:

Operators that operate or work with a single operand are unary operators. For example: Increment(++) and Decrement(–) Operators

### **b) Binary Operators**:

Operators that operate or work with two operands are binary operators. For example: Addition(+), Subtraction(-), multiplication(\*), Division(/) operators

## **2. Relational Operators in C**

These are used for the comparison of the values of two operands. For example, checking if one operand is equal to the other operand or not, whether an operand is greater than the other operand or not, etc. Some of the relational operators are (==, >= , <= )

## **3. Logical Operator in C**

Logical Operators are used to combining two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a Boolean value either **true** or **false**.

For example, the **logical AND** represented as the **‘&&’ operator in C** returns true when both the conditions under consideration are satisfied. Otherwise, it returns false. Therefore, a && b returns true when both a and b are true (i.e. non-zero)

## **4. Bitwise Operators in C**

The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing. For example, the **bitwise AND** operator represented as **‘&’ in C** takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1(True).

## **5. Assignment Operators in C**

Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and the right side operand of the assignment operator is a value. The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.

Different types of assignment operators are shown below:

### **a) “=”**

This is the simplest assignment operator. This operator is used to assign the value on the right to the variable on the left.   
**Example:**

a = 10;

b = 20;

ch = 'y';

### **b) “+=”**

This operator is the combination of the ‘+’ and ‘=’ operators. This operator first adds the current value of the variable on left to the value on the right and then assigns the result to the variable on the left.   
**Example:**

(a += b) can be written as (a = a + b)

If initially value stored in a is 5. Then (a += 6) = 11.

### **c) “-=”**

This operator is a combination of ‘-‘ and ‘=’ operators. This operator first subtracts the value on the right from the current value of the variable on left and then assigns the result to the variable on the left.   
**Example:**

(a -= b) can be written as (a = a - b)

If initially value stored in a is 8. Then (a -= 6) = 2.

### **d) “\*=”**

This operator is a combination of the ‘\*’ and ‘=’ operators. This operator first multiplies the current value of the variable on left to the value on the right and then assigns the result to the variable on the left.   
**Example:**

(a \*= b) can be written as (a = a \* b)

If initially, the value stored in a is 5. Then (a \*= 6) = 30.

### **e) “/=”**

This operator is a combination of the ‘/’ and ‘=’ operators. This operator first divides the current value of the variable on left by the value on the right and then assigns the result to the variable on the left.   
**Example:**

(a /= b) can be written as (a = a / b)

If initially, the value stored in a is 6. Then (a /= 2) = 3.

## **6. Other Operators**

Apart from the above operators, there are some other operators available in C used to perform some specific tasks. Some of them are discussed here:

### **i. sizeof operator**

* sizeof is much used in the C programming language.
* It is a compile-time unary operator which can be used to compute the size of its operand.
* The result of sizeof is of the unsigned integral type which is usually denoted by size\_t.
* Basically, the sizeof the operator is used to compute the size of the variable.

### **ii. Comma Operator**

* The comma operator (represented by the token) is a binary operator that evaluates its first operand and discards the result, it then evaluates the second operand and returns this value (and type).
* The comma operator has the lowest precedence of any C operator.
* Comma acts as both operator and separator.

### **iii. Conditional Operator**

* The conditional operator is of the form ***Expression1? Expression2: Expression3***
* Here, Expression1 is the condition to be evaluated. If the condition(Expression1) is *True* then we will execute and return the result of Expression2 otherwise if the condition(Expression1) is *false* then we will execute and return the result of Expression3.
* We may replace the use of if..else statements with conditional operators.

## **C Operators with Example**

* C

|  |
| --- |
| // C Program to Demonstrate the working concept of  // Operators  #include <stdio.h>    **int** main()  {    **int** a = 10, b = 5;      // Arithmetic operators  **printf**("Following are the Arithmetic operators in C\n");  **printf**("The value of a + b is %d\n", a + b);  **printf**("The value of a - b is %d\n", a - b);    **printf**("The value of a \* b is %d\n", a \* b);  **printf**("The value of a / b is %d\n", a / b);  **printf**("The value of a % b is %d\n", a % b);      // First print (a) and then increment it      // by 1  **printf**("The value of a++ is %d\n", a++);        // First print (a+1) and then decrease it      // by 1  **printf**("The value of a-- is %d\n", a--);        // Increment (a) by (a+1) and then print  **printf**("The value of ++a is %d\n", ++a);        // Decrement (a+1) by (a) and then print  **printf**("The value of --a is %d\n", --a);        // Assignment Operators --> used to assign values to      // variables int a =3, b=9; char d='d';        // Comparison operators      // Output of all these comparison operators will be (1)      // if it is true and (0) if it is false  **printf**(          "\nFollowing are the comparison operators in C\n");  **printf**("The value of a == b is %d\n", (a == b));  **printf**("The value of a != b is %d\n", (a != b));  **printf**("The value of a >= b is %d\n", (a >= b));  **printf**("The value of a <= b is %d\n", (a <= b));  **printf**("The value of a > b is %d\n", (a > b));  **printf**("The value of a < b is %d\n", (a < b));        // Logical operators  **printf**("\nFollowing are the logical operators in C\n");  **printf**("The value of this logical and operator ((a==b) "             "&& (a<b)) is:%d\n",             ((a == b) && (a < b)));  **printf**("The value of this logical or operator ((a==b) "             "|| (a<b)) is:%d\n",             ((a == b) || (a < b)));  **printf**("The value of this logical not operator "             "(!(a==b)) is:%d\n",             (!(a == b)));    **return** 0;  } |

**Output**

Following are the Arithmetic operators in C

The value of a + b is 15

The value of a - b is 5

The value of a \* b is 50

The value of a / b is 2

The value of a % b is 0

The value of a++ is 10

The value of a-- is 11

The value of ++a is 11

The value of --a is 10

Following are the comparison operators in C

The value of a == b is 0

The value of a != b is 1

The value of a >= b is 1

The value of a <= b is 0

The value of a > b is 1

The value of a < b is 0

Following are the logical operators in C

The value of this logical and operator ((a==b) && (a<b)) is:0

The value of this logical or operator ((a==b) || (a<b)) is:0

The value of this logical not operator (!(a==b)) is:1

## Time and Space Complexity

**Time Complexity:** O(1)

**Auxiliary Space:** O(1)

## **1.8 Precedence of Operators in C and Associativity**

The below table describes the precedence order and associativity of operators in C. The precedence of the operator decreases from top to bottom.

| **Precedence** | **Operator** | **Description** | **Associativity** |
| --- | --- | --- | --- |
| 1 | **()** | Parentheses (function call) | left-to-right |
| **[]** | Brackets (array subscript) | left-to-right |
| **.** | Member selection via object name | left-to-right |
| **->** | Member selection via a pointer | left-to-right |
| **a++/a–** | Postfix increment/decrement (a is a variable) | left-to-right |
| 2 | **++a/–a** | Prefix increment/decrement (a is a variable) | right-to-left |
| **+/-** | Unary plus/minus | right-to-left |
| **!~** | Logical negation/bitwise complement | right-to-left |
| **(type)** | Cast (convert value to temporary value of type) | right-to-left |
| **\*** | Dereference | right-to-left |
| **&** | Address (of operand) | right-to-left |
| **sizeof** | Determine size in bytes on this implementation | right-to-left |
| 3 | **\*,/,%** | Multiplication/division/modulus | left-to-right |
| 4 | **+/-** | Addition/subtraction | left-to-right |
| 5 | **<< , >>** | Bitwise shift left, Bitwise shift right | left-to-right |
| 6 | **< , <=** | Relational less than/less than or equal to | left-to-right |
| **> , >=** | Relational greater than/greater than or equal to | left-to-right |
| 7 | **== , !=** | Relational is equal to/is not equal to | left-to-right |
| 8 | **&** | Bitwise AND | left-to-right |
| 9 | **^** | Bitwise exclusive OR | left-to-right |
| 10 | **|** | Bitwise inclusive OR | left-to-right |
| 11 | **&&** | Logical AND | left-to-right |
| 12 | **||** | Logical OR | left-to-right |
| 13 | **?:** | Ternary conditional | right-to-left |
| 14 | **=** | Assignment | right-to-left |
| **+= , -=** | Addition/subtraction assignment | right-to-left |
| **\*= , /=** | Multiplication/division assignment | right-to-left |
| **%= , &=** | Modulus/bitwise AND assignment | right-to-left |
| **^= , |=** | Bitwise exclusive/inclusive OR assignment | right-to-left |
| **<>=** | Bitwise shift left/right assignment | right-to-left |
| 15 | **,** | expression separator | left-to-right |

# 1.9 C Expressions

An expression is a formula in which operands are linked to each other by the use of operators to compute a value. An operand can be a function reference, a variable, an array element or a constant.

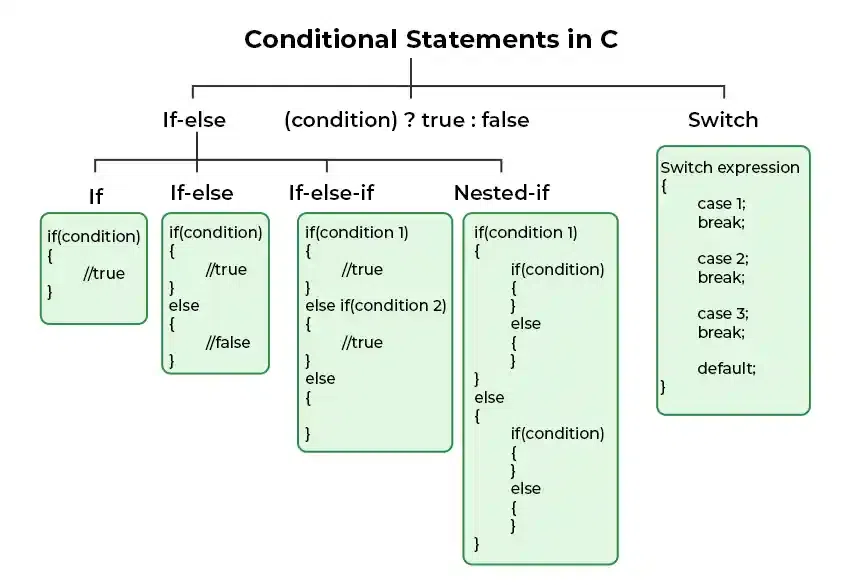
Let's understand through an example.

6\*2/ (2+1 \* 2/3 + 6) + 8 \* (8/4)

|  |  |
| --- | --- |
| **Evaluation of expression** | **Description of each operation** |
| 6\*2/( 2+1 \* 2/3 +6) +8 \* (8/4) | An expression is given. |
| 6\*2/(2+2/3 + 6) + 8 \* (8/4) | 2 is multiplied by 1, giving value 2. |
| 6\*2/(2+0+6) + 8 \* (8/4) | 2 is divided by 3, giving value 0. |
| 6\*2/ 8+ 8 \* (8/4) | 2 is added to 6, giving value 8. |
| 6\*2/8 + 8 \* 2 | 8 is divided by 4, giving value 2. |
| 12/8 +8 \* 2 | 6 is multiplied by 2, giving value 12. |
| 1 + 8 \* 2 | 12 is divided by 8, giving value 1. |
| 1 + 16 | 8 is multiplied by 2, giving value 16. |
| 17 | 1 is added to 16, giving value 17. |

# 1.10 Decision Making statement in C

The **conditional statements** (also known as decision control structures) such as if, if else, switch, etc. are used for decision-making purposes in C programs.



**1. if in C**

 It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statements is executed otherwise not.

### Syntax of if Statement

**if**(*condition*)

{

*// Statements to execute if*

*// condition is true*

}

**Example 1**

#include<stdio.h>

**int** main(){

**int** number=0;

printf("Enter a number:");

scanf("%d",&number);

**if**(number%2==0){

printf("%d is even number",number);

}

**return** 0;

}

**If Else statement**

If the test expression is evaluated to true,

* statements inside the body of if are executed.
* statements inside the body of else are skipped from execution.

If the test expression is evaluated to false,

* statements inside the body of else are executed
* statements inside the body of if are skipped from execution.

**Example 1**

// C program to illustrate If statement

#include <stdio.h>

**int** main()

{

**int** i = 10;

**if** (i > 15) {

**printf**("10 is greater than 15");

    }

**printf**("I am Not in if");

}

**Output**

I am Not in if

**Example 2**

**#include <stdio.h>**

int main() {

int num;

printf("Enter an integer: ");

scanf("%d", &num);

// true if num is perfectly divisible by 2

if(num % 2 == 0)

printf("%d is even.", num);

else

printf("%d is odd.", num);

return 0;

}

**Output**

Enter an integer: -7

-7 is odd.

## **if...else Ladder**

The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.

The if...else ladder allows you to check between multiple test expressions and execute different statements.

### Syntax of if...else Ladder

if (test expression1) {

// statement(s)

}

else if(test expression2) {

// statement(s)

}

else if (test expression3) {

// statement(s)

}

.

.

else {

// statement(s)

}

### Example 3: C if...else Ladder

// Program to relate two integers using =, > or < symbol

#include <stdio.h>

int main() {

int number1, number2;

printf("Enter two integers: ");

scanf("%d %d", &number1, &number2);

//checks if the two integers are equal.

if(number1 == number2) {

printf("Result: %d = %d",number1,number2);

}

//checks if number1 is greater than number2.

else if (number1 > number2) {

printf("Result: %d > %d", number1, number2);

}

//checks if both test expressions are false

else {

printf("Result: %d < %d",number1, number2);

}

return 0;

}

[Run Code](https://www.programiz.com/c-programming/online-compiler)

**Output**

Enter two integers: 12

23

Result: 12 < 23

## **Nested if...else**

It is possible to include an if...else statement inside the body of another if...else statement.

### Example 4: Nested if...else

This program given below relates two integers using either <, > and = similar to the if...else ladder's example. However, we will use a nested if...else statement to solve this problem.

#include <stdio.h>

int main() {

int number1, number2;

printf("Enter two integers: ");

scanf("%d %d", &number1, &number2);

if (number1 >= number2) {

if (number1 == number2) {

printf("Result: %d = %d",number1,number2);

}

else {

printf("Result: %d > %d", number1, number2);

}

}

else {

printf("Result: %d < %d",number1, number2);

}

return 0;

}

# Switch Statement in C

# Switch case statement evaluates a given expression and based on the evaluated value(matching a certain condition), it executes the statements associated with it. Basically, it is used to perform different actions based on different conditions(cases).

## Syntax of switch Statement in C

**switch(expression)**

**{**

**case** value1**:** statement\_1;

**break;**

**case** value2**:** statement\_2;

**break;**

.

.

.

**case** value\_n**:** statement\_n;

**break;**

**default:** default\_statement;

}

**Example**

**// C program to Demonstrate returning of day based numeric**

**// value**

**#include <stdio.h>**

**int main()**

**{**

**// switch variable**

**int var = 1;**

**// switch statement**

**switch (var) {**

**case 1:**

**printf("Case 1 is Matched.");**

**break;**

**case 2:**

**printf("Case 2 is Matched.");**

**break;**

**case 3:**

**printf("Case 3 is Matched.");**

**break;**

**default:**

**printf("Default case is Matched.");**

**break;**

**}**

**return 0;**

**}**

**Output**

Case 1 is Matched.

## **How switch Statement Work?**

The working of the switch statement in C is as follows:

1. **Step 1:** The switch variable is evaluated.
2. **Step 2:** The evaluated value is matched against all the present cases.
3. **Step 3A:** If the matching case value is found, the associated code is executed.
4. **Step 3B:** If the matching code is not found, then the default case is executed if present.
5. **Step 4A:** If the break keyword is present in the case, then program control breaks out of the switch statement.
6. **Step 4B:** If the break keyword is not present, then all the cases after the matching case are executed.
7. **Step 5:** Statements after the switch statement are executed.

**1.11 LOOPING STATEMENTS:**

A sequence of statements is executed until a specified condition is true.

This sequence of statements to be executed is kept inside the curly braces { } known as

the Loop body.

After every execution of loop body, condition is verified, and if it is found to be true the

loop body is executed again. When the condition check returns false, the loop body is

not executed.

There are 3 type of Loops in C language

1. for loop

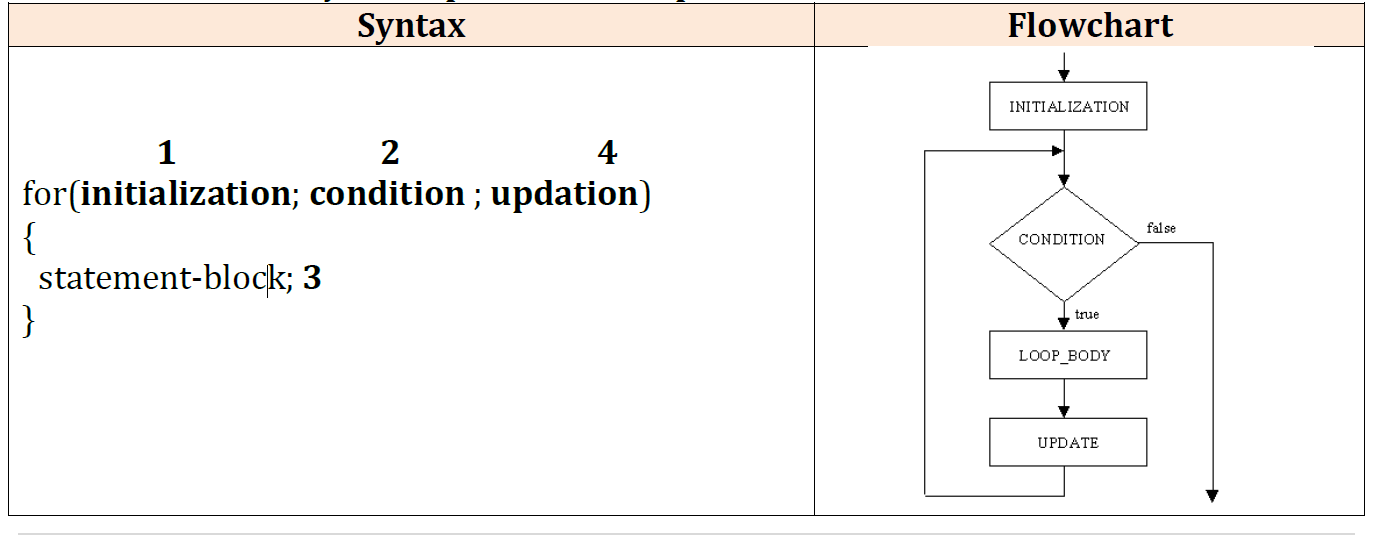
2. while loop

3. do-while loop

**for loop**

for loop is used to execute a set of statements repeatedly until a particular condition is

satisfied. We can say it an open-ended loop.



In for loop we have exactly two semicolons, one after initialization and second after condition.

In this loop we can have more than one initialization or increment/decrement, separated using comma operator. for loop can have only one condition.

**Example: sum of n numbers**

#include <stdio.h>

#include<conio.h>

void main()

{

int n, sum = 0, i ;

clrscr();

printf("Enter the number\n");

scanf("%d", &n);

for (i = 1; i <= n; i++)

sum = sum + i;

printf("Sum of entered integers =%d",sum);

getch();

}

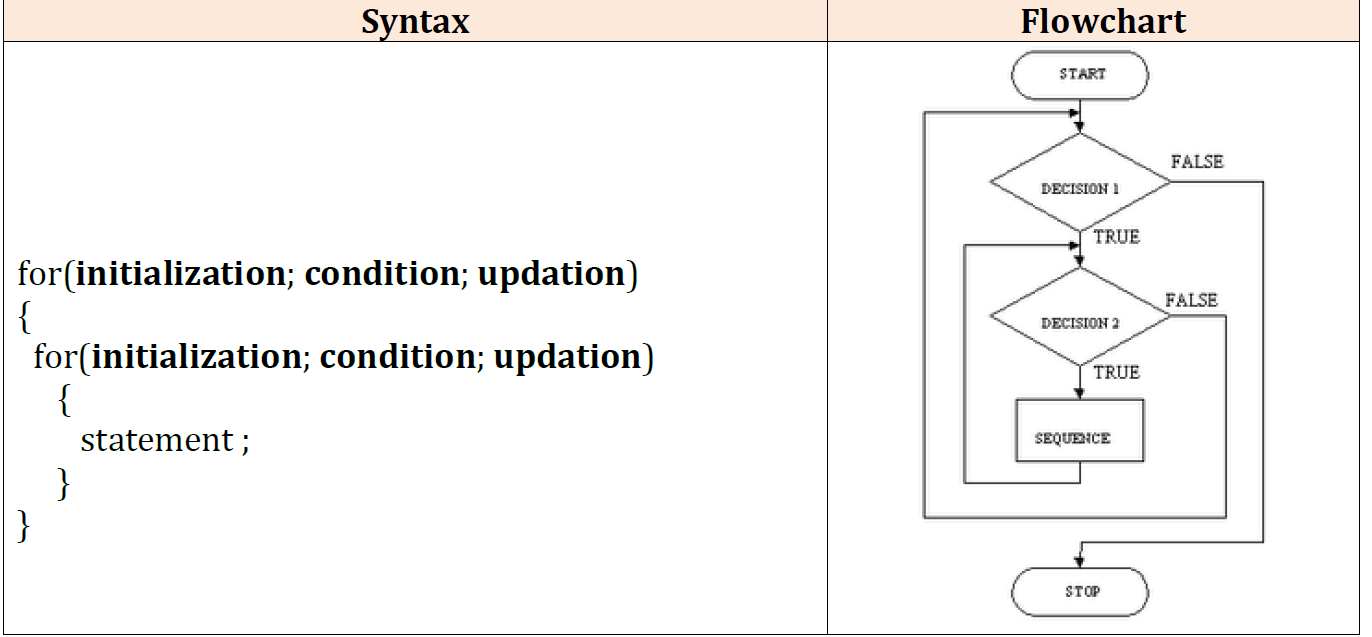
Output:

Enter the number: 5

Sum of entered integers 15

**Nested for loop:**

We can also have nested for loops, i.e one for loop inside another for loop.



**print half Pyramid of numbers**

#include<stdio.h>

#include<conio.h>

void main( )

{

int i,j;

clrscr();

for(i=1;i<5;i++)

{

printf("\n");

for(j=i;j>0;j--)

{ printf("%d",j);

}

}

getch();

}

output:

1

21

321

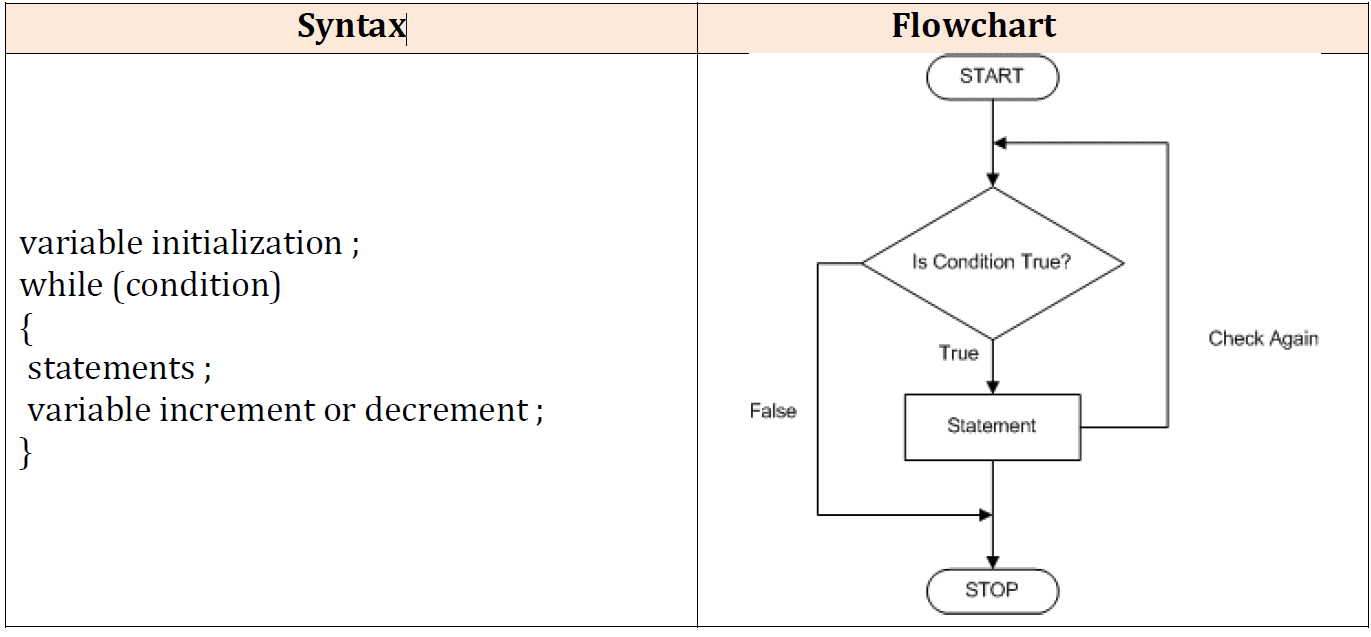
4321

54321

**while loop:**

while loop can be addressed as an entry control loop. It is completed in 3 steps.

* Variable initialization. ( e.g int x=0; )
* condition (e.g while( x<=10) )
* Variable increment or decrement ( x++ or x-- or x=x+2 )



**Example: Reverse the given number**

include <stdio.h>

#include<conio.h>

void main()

{

int n, sum = 0;

printf("Enter a number to reverse\n");

scanf("%d", &n);

while (n != 0)

{

a = n %10;

sum = sum\*10+ a;

n = n/10;

}

printf("Reverse of entered number=%d􀇳,sum􀈌;

getch();

}

Output:

Enter a number to reverse

123

Reverse of entered number=

321

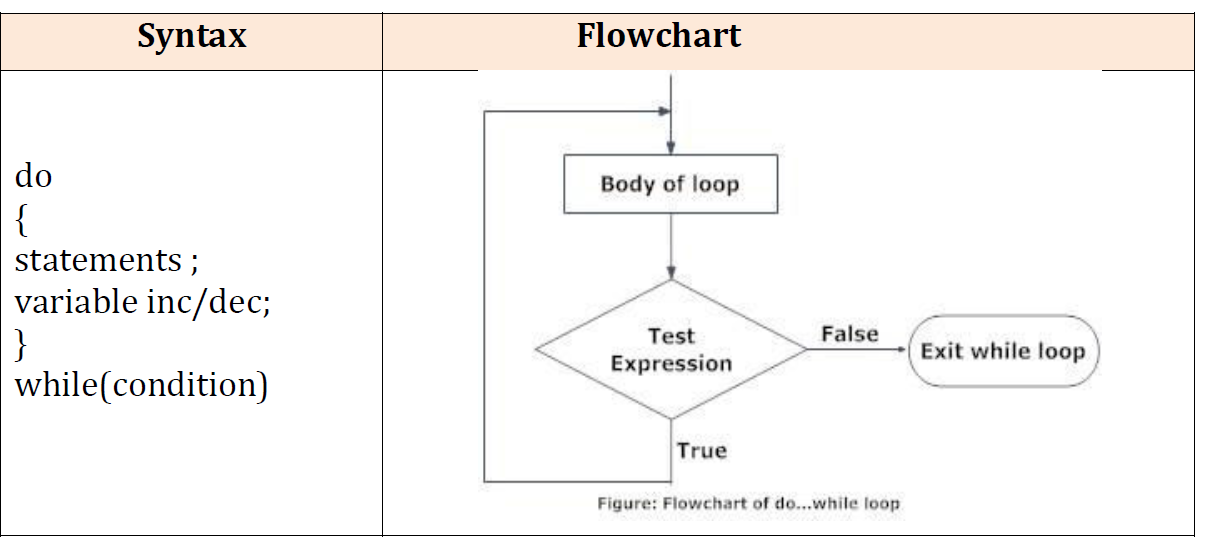
**do while loop**

In some situations it is necessary to execute body of the loop before testing the

condition. Such situations can be handled with the help of do-while loop. do statement

evaluates the body of the loop first and at the end, the condition is checked using while

statement. General format of do-while loop is,



**Example: print first ten multiple of 5**

#include<stdio.h>

#include<conio.h>

void main()

{

int a=5,i=1;

clrscr();

do

{

printf("%d\t",a\*i);

i++;

}

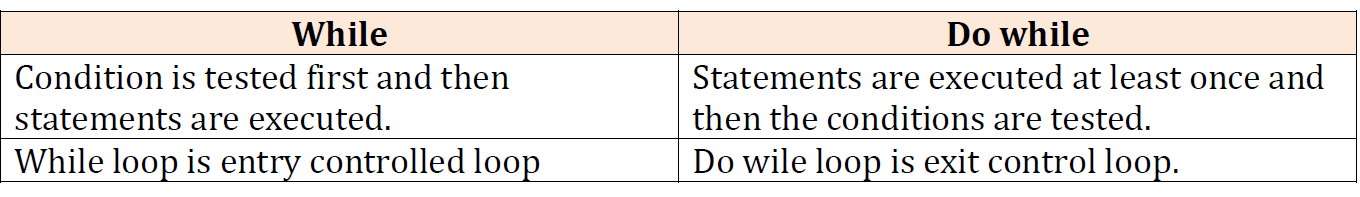
while(i <= 10);

getch();

}

output

5 10 15 20 25 30 35 40 45 50



**1.12 Unconditional Branching or Jump Statements:**

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to

leave the loop as soon as certain condition becomes true, that is called jumping out of

loop.

C language allows jumping from one statement to another within a loop as well as

jumping out of the loop.

1. break

2. continue

3. goto

**1) break statement**

When break statement is encountered inside a loop, the loop is immediately exited and

the program continues with the statement immediately following the loop.

while(condition check)

{

statement -1;

statement -2;

if (some condition)

{

break; // Jumps out of loop, no matter how many cycles are left, Loop is exited.

}

statement-3;

statement-4;

}

**Example: C Program to demonstrate break statement with for loop**

#include <stdio.h>

int main()

{

// using break inside for loop to terminate after 2 iteration

printf("break in for loop\n");

for (int i = 1; i < 5; i++) {

if (i == 3)

{

break;

}

else

{

printf("%d ", i);

}

return 0;

}

Output: break in for loop

1 2

**2) goto statement:**

The goto statement is used for altering the normal sequence of program

execution by transferring control to some other part of the program. The goto statement transfers control to a label. The given label must reside in the

same function.



Example:

#include <stdio.h>

int main()

{

int sum=0;

for(int i = 0; i<=10; i++){

sum = sum+i;

if(i==5)

{

goto addition;

}

}

addition:

printf("%d", sum);

return 0;

}

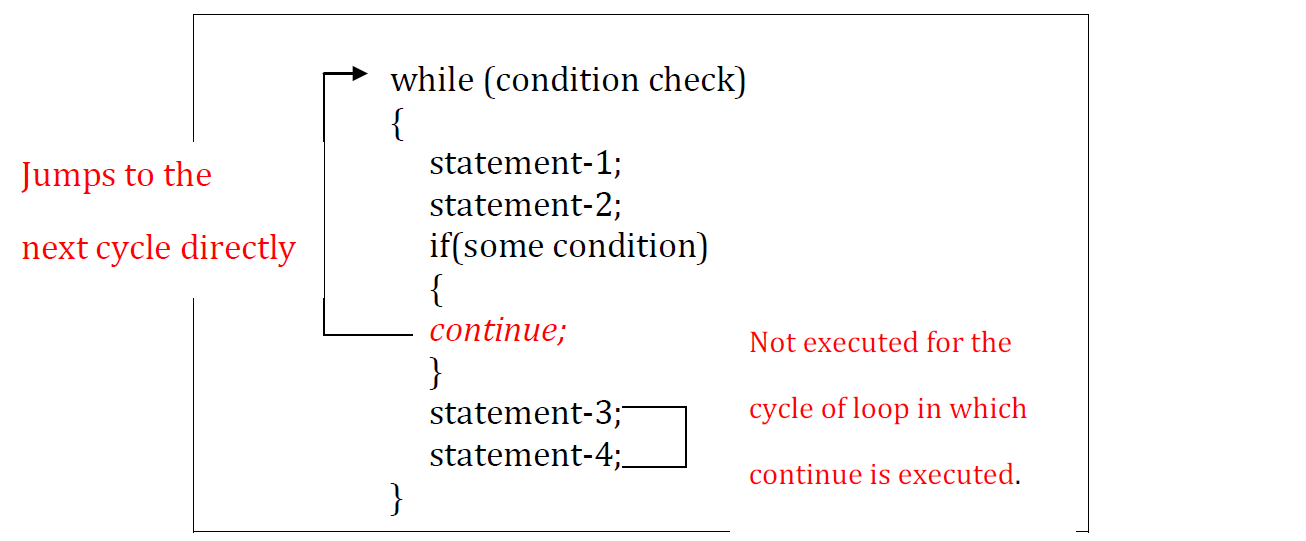
Output: 15

**3)continue statement**

It causes the control to go directly to the test-condition and then continue the loop

process. On encountering continue, cursor leave the current cycle of loop, and starts

with the next cycle.



**// C program to explain the use of continue statement with for loop**

#include <stdio.h>

int main()

{

**// for loop to print 1 to 8**

for (int i = 1; i <= 8; i++)

{

if (i == 4) //// when i = 4, the iteration will be skipped and for will not be printed

{

continue;

}

printf("%d ", i);

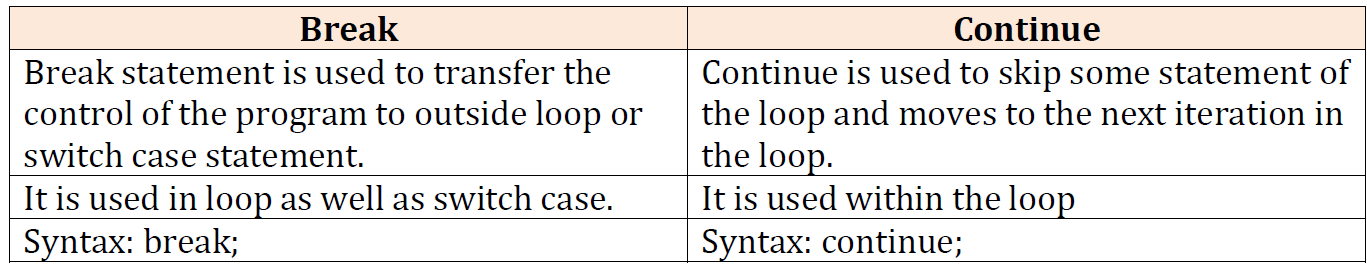
}

return 0;

}

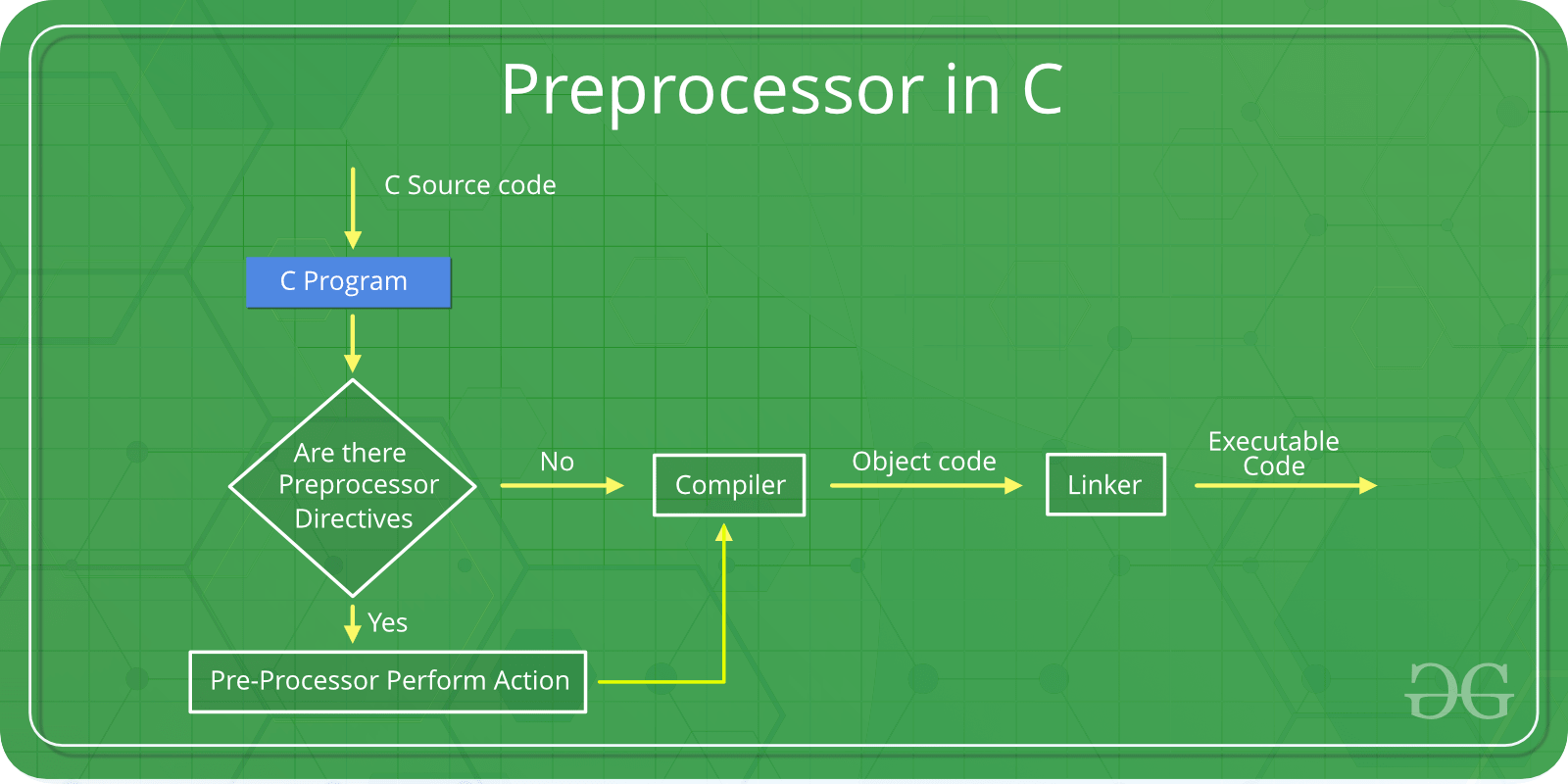
1 2 3 5 6 7 8

1 2 3 5 6 7 8



**1.13 Preprocessor directives in C:**

Preprocessors are programs that process the source code before compilation. A number of steps are involved between writing a program and executing a program in C.



Preprocessor programs provide preprocessor directives that tell the compiler to preprocess the source code before compiling. All of these preprocessor directives begin with a ‘#’ (hash) symbol.

The ‘#’ symbol indicates that whatever statement starts with a ‘#’ will go to the preprocessor program to get executed. These preprocessor directives cab be placed anywhere in our program.

Examples of some preprocessor directives are: #include, #define, #ifndef, etc.

**Note**: Remember that the **#** symbol only provides a path to the preprocessor, and a command such as include is processed by the preprocessor program. For example, #include will include the code or content of the specified file in your program.

**The following table lists all the preprocessor directives in C/C++:**

| **Preprocessor Directives** | **Description** |
| --- | --- |
| **#define** | Used to define a macro |
| **#undef** | Used to undefine a macro |
| **#include** | Used to include a file in the source code program |
| **#ifdef** | Used to include a section of code if a certain macro is defined by #define |
| **#ifndef** | Used to include a section of code if a certain macro is not defined by #define |
| **#if** | Check for the specified condition |
| **#else** | Alternate code that executes when #if fails |
| **#endif** | Used to mark the end of #if, #ifdef, and #ifndef |

**Types of C/C++ Preprocessors:**

There are 3 Main Types of Preprocessor Directives:

* Macros
* File Inclusion
* Conditional Compilation

**1. Macros**

In C/C++, Macros are pieces of code in a program that is given some name. Whenever this name is encountered by the compiler, the compiler replaces the name with the actual piece of code. The ‘#define’ directive is used to define a macro.

**Syntax of Macro Definition**

#define token value

where after preprocessing, the token will be expanded to its value in the program.

Example: C Program to illustrate the macro

#include <stdio.h>

#define LIMIT 5

int main()

{

for (int i = 0; i < LIMIT; i++) {

printf("%d \n", i);

}

return 0;

}

Output: 0 1 2 3 4

when the compiler executes the word LIMIT, it replaces it with 5. The word ‘LIMIT’ in the macro definition is called a macro template and ‘5’ is macro expansion.

**2.File Inclusion**

This type of preprocessor directive tells the compiler to include a file in the source code program. The #include preprocessor directive is used to include the header files in the C/C++ program.

The standard header files contain definitions of pre-defined functions like printf(), scanf(), etc. These files must be included to work with these functions. Different functions are declared in different header files.

For example, standard I/O functions are in the ‘iostream’ file whereas functions that perform string operations are in the ‘string’ file.

**Syntax**

#include <file\_name>

**3. Conditional Compilation**

Conditional Compilation in C/C++ directives is a type of directive that helps to compile a specific portion of the program or to skip the compilation of some specific part of the program based on some conditions. There are the following preprocessor directives that are used to insert conditional code:

#if Directive

#ifdef Directive

#ifndef Directive

#else Directive

#elif Directive

#endif Directive

Example:

#ifndef state

#define state “Tamilnadu”

#endif

**1.14 Compilation process in C:**

* 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.

**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.

**Linker**

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.The output of the linker is the executable file.

