UNIT I C PROGRAMMING

Structure of C Program, C Tokens: Variables, Data types, Constants,

Identifiers, key words and Operators, Expressions.

Control Structures: Conditional Statements (Simple if, if

-

else, Nested

-

if

-

Iterative

Statements

(

for,

While,

Do

else,

Switch).

-

While),

Jump

Statements (break, Continue).

**INTRODUCTION TO ‘C’ LANGUAGE:**

high

C

a

is

-

in

general

used

level

structured

oriented

programming

language,

-

purpose

programming, developed by

**Dennis Ritchie at AT&T Bell Labs**

, the USA between 1969 and

1973.

•

In

, the

1988

American National Standards Institute (ANSI)

had formalized the C la

nguage.

•

C was invented to write

UNIX

operating system.

•

C is a successor of 'Basic Combined Programming Language' (BCPL) called

B language.

•

Linux OS,

PHP, and

MySQL

are written in C.

•

C has been written in assembly language.

**Uses of C Programming**

**Language**

**)**

**Applications of C language**

**(**

In the beginning, C was used for developing system applications, e.g. :



[Database System](https://www.w3schools.in/dbms/intro/)

[s](https://www.w3schools.in/dbms/intro/)



Language Interpreters



[Compilers and Assembler](https://www.w3schools.in/what-is-compiler/)

[s](https://www.w3schools.in/what-is-compiler/)



[Operating System](https://www.w3schools.in/operating-system-tutorial/intro/)

[s](https://www.w3schools.in/operating-system-tutorial/intro/)



Network Drivers



Word Processors

**Features of C language**

•

It is a robust language with

**rich set of built**

**-**

**in functions**

and

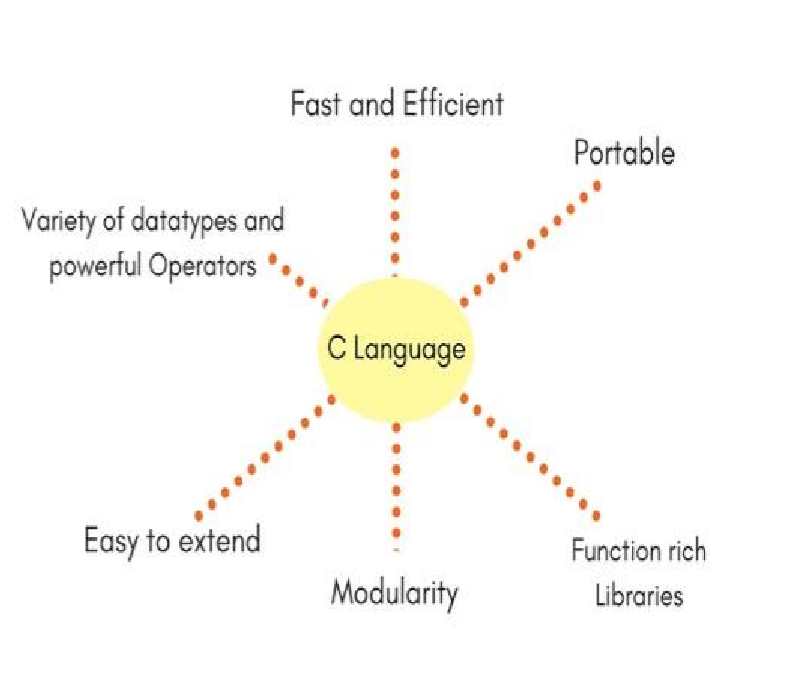
**operators**

that c

an be

used to write any complex program.

* The C compiler combines the capabilities of an assembly language with features of a high-level language.



* Programs Written in C are **efficient and fast**. This is due to its variety of data type and powerful operators.
* It is many time **faster than BASIC**.
* C is highly portable this means that programs **once written can be run on another machines** with little or no modification.
* Another important feature of C program, is its **ability to extend itself.**
* A C program is basically a **collection of functions** that are supported by C library. We can also create **our own function** and add it to C library.
* C language is the most widely used language in operating systems and **embedded system development today.**

**BASIC STRUCTURE OF C LANGUAGE:**

|  |
| --- |
| 1. Documentation section 2. Linking section 3. Definition section 4. Global declaration section 5. Main function section   {  Declaration section  Executable section  }   1. Sub program or function section |

The program written in C language follows this basic structure. The sequence of sections should be as they are in the basic structure. A C program should have one or more sections but the sequence of sections is to be followed.

1. **DOCUMENTATION SECTION :** comes first and is used to document the use of logic or reasonsin your program. It can be used to write the program's objective, developer and logic details.

The documentation is done in C language with /\* and \*/ . Whatever is written between these two **are called comments.**

1. **LINKING SECTION :** This section tells the compiler to link the certain occurrences of keywordsor functions in your program to the header files specified in this section.

|  |
| --- |
| e.g. #include <stdio.h> |

1. **DEFINITION SECTION :** It is used to declare some constants and assign them some

|  |
| --- |
| value.e.g. #define MAX 25 |

Here #define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

1. **GLOBAL DECLARATION SECTION :** Here the variables which are used through out theprogram (including main and other functions) are declared so as to make them global(i.e accessible to all parts of program)

|  |
| --- |
| e.g. int i; (before main()) |

1. **MAIN FUNCTION SECTION :** It tells the compiler where to start the execution

|  |
| --- |
| frommain()  { point from execution starts  } |

main function has two sections

* 1. declaration section : In this the variables and their data types are declared.
  2. Executable section : This has the part of program which actually performs the task we need.

1. **SUB PROGRAM OR FUNCTION SECTION :** This has all the sub programs or the functionswhich our program needs.

**SIMPLE ‘C’ PROGRAM:**

/\* simple program in c \*/ #include<stdio.h> main()

{

printf(“welcome to c programming”);

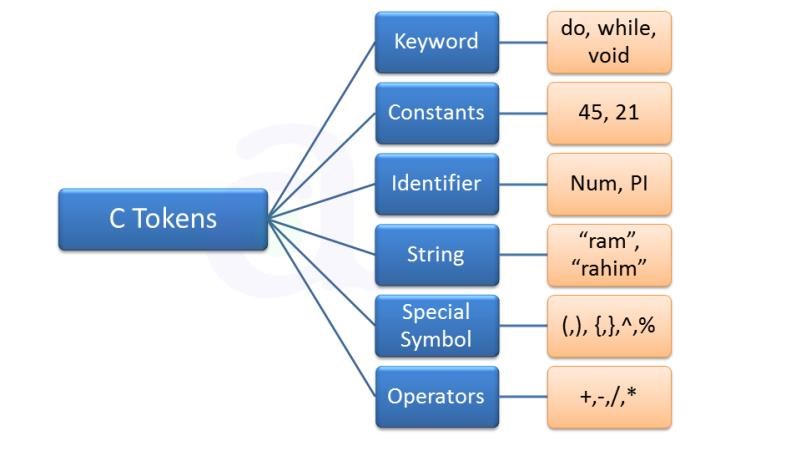
} /\* End of main \*/

**C-TOKENS :**

C program is basically a collection of many functions. But it has basic building blocks, these basic buildings blocks in C language which are constructed together to write a C program.

This basic building blocks are called Token.

✓ Each and every smallest individual unit in a C program is known as C tokens ✓ In C, Programming tokens are of six types. They are,



* In C Programming punctuation, individual words, characters etc are called tokens.
* Tokens are basic building blocks of C Programming

|  |  |
| --- | --- |
| Token | Meaning |
| Keyword | A variable is a meaningful name of data storage location in computer memory. When using a variable you refer to memory address of computer |
| Constant | Constants are expressions with a fixed value |
| Identifier | The term identifier is usually used for variable names |
| String | Sequence of characters |
| Special Symbol | Symbols other than the Alphabets and Digits and white-spaces |
| Operators | A symbol that represents a specific mathematical or non-mathematical action |

We have learnt different type of tokens in C. In the upcoming chapters we will learn C tokens, keywords and identifiers in more details.

**KEYWORDS :**

There are certain words, called keywords (reserved words) that have a predefined meaning in ‘C’ language. These keywords are only to be used for their intended purpose and not as identifiers.

The following table shows the standard ‘C’ keywords

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |  |  |  |  |

**IDENTIFIERS :**

Names of the variables and other program elements such as functions, array, etc, are known as identifiers.

There are few rules that govern the way variable are named (identifiers).

1. Identifiers can be named from the combination of **A-Z, a-z, 0-9, \_(**Underscore).
2. The **first alphabet** of the identifier should be either an alphabet or an underscore.

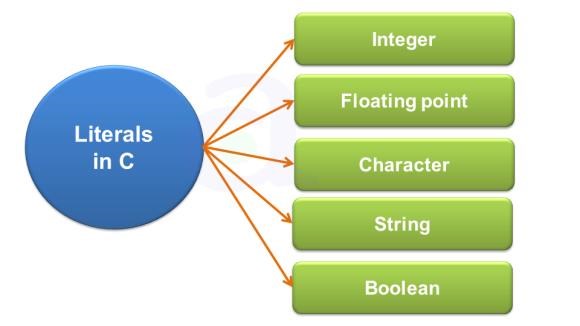
**digit are not allowed.**

1. It should not be a keyword.
2. Identifiers in C are case sensitive, foo and Foo are two different identifiers.

|  |
| --- |
| **Eg: name,ptr,sum** |

## Constants or Literals

Literals in C are sequence of characters (digits, letters and other characters) that represent constant values to be stored in variables. C specifies five major type of literals. They are below :



|  |  |
| --- | --- |
| **Constant** | **Type of Value Stored** |
| Integer Literals | Literals which stores integer value |
| Floating Literals | Literals which stores float value |
| Character Literals | Literals which stores character value |
| String Literals | Literals which stores string value |
| Boolean Literals | Literals which stores true or false |

**NUMERIC CONSTANTS:**

1. Example for an integer constant is 786,-127
2. Long constant is written with a terminal ‘l’or ‘L’,for example 1234567899L is a Long constant.
3. Unsigned constants are written with a terminal ‘u’ or ‘U’,and the suffix ‘ul’ and ‘UL’ indicates unsigned long. for example 123456789u is a Unsigned constant and 1234567891ul is an unsigned long constant.
4. **Floating point constants** contain a decimal point or an exponent or both. For Eg :

123.4,1e-2,1.4E-4,etc.The suffixes f or F indicate a float constant while the absence of f or F indicate the double, l or L indicate long double.

**CHARACTER CONSTANTS:**

A character constant is written as one character with in single quotes such as ‘a’. The value of a character constant is the numerical value of the character in the machines character set. **String constants** or string literal is a sequence of zero or more characters surrounded by a double quote. Example , “ I am a little boy”. quotes are not a part of the string.

To distinguish between a character constant and a string that contains a single character ***ex: ‘a’ is not same as “a”.***

‘a’ is an integer used to produce the numeric value of letter a in the machine character set,

while “a” is an array of characters containing one character and a ‘\0’ as a string in C

is an array of characters terminated by NULL.

**Enumeration constant** , it is a list of constant integer values.

Ex.: enum color { RED, Green, BLUE }

The first name in the enum has the value 0 and the next 1 and so on unless explicit values are specified.

If not all values specified , unspecified values continue the progression from the last specified value. For example

Enum months { JAN=1, FEB,MAR, …, DEC -

Where the value of FEB is 2 and MAR is 3 and so on.

# String

In C and C++, strings are nothing but an array of characters ended with a null character (‘\0’).This null character indicates the end of the string. Strings are always enclosed in double quotes. Whereas, a character is enclosed in single quotes in C and C++.

**Declarations for String:**

* char string[20] = {‘g’, ’e’, ‘e’, ‘k’, ‘s’, ‘f’, ‘o’, ‘r’, ‘g’, ’e’, ‘e’, ‘k’, ‘s’, ‘\0’};
* char string[20] = "atnyla";
* char string [] = "atnyla";

## Special symbol

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose.[] () {}, ; \* = #

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **<** |  | **.** |  |
| **,** | **>** | **\_** |
|  |  |  |  |  |
| **(** | **)** | **;** | **$** | **:** |
|  |  |  | **#** |  |
| **%** | **[** | **]** | **?** |
|  |  |  | **}** |  |
| **'** | **&** | **{** | **"** |
|  |  |  | **/** |  |
| **^** | **!** | **\*** | **|** |
|  |  |  | **+** |  |
| **-** | **\** | **~** |  |

• **Brackets[]:** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.

|  |  |  |
| --- | --- | --- |
| * **Parentheses():** These special symbols are used to indicate function calls and function parameters. * **Braces{}:** These opening and ending curly braces mark the start and end of a block of code containing more than one executable statement. * **semicolon (; ):** It is used to separate more than one statements like in for loop is separates initialization, condition, and increment. * **comma (,):** It is an operator that essentially invokes something called an initialization list. * **an asterisk (\*):** It is used for mutiplication. * **assignment operator (=):** It is used to assign values. * **preprocessor(#):** The preprocessor is a macro processor that is used automatically by the compiler to transform your program before actual compilation.   **Operators**  An operator is a symbol that takes one or more arguments and operates on them to produce a result. Operators are of many types and are considered in operator chapter There are many types of operators in C which are given below:   * Unary Operator, * Arithmetic Operator, * shift Operator, * Relational Operator, * Bitwise Operator, * Logical Operator, • Ternary Operator and * Assignment Operator.       **VARIABLE**  A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.  It is a way to represent memory location through symbol so that it can be easily identified. | | |
|  |  |  |

syntax to declare a variable:

**type variable\_list;**

The example of declaring the variable is given below:

1. **int** a;
2. **float** b;
3. **char** c; Here, a, b, c are variables. The int, float, char are the data types.

Rules for defining variables

* + A variable can have alphabets, digits, and underscore.
  + A variable name can start with the alphabet, and underscore only. It can't start with a digit. o No whitespace is allowed within the variable name. o A variable name must not be any reserved word or keyword, e.g. int, float, etc.

|  |  |
| --- | --- |
| **Valid variable names:** | **Invalid variable names:** |
| **int** a; **int** \_ab; **int** a30; | **int** 2hello; **int** a b; **int** **long**; |

**DATA TYPES :**

To represent different types of data in C program we need different data types. A data type is essential to identify the storage representation and the type of operations that can be performed on that data. C supports four different classes of data types namely

* 1. Basic Data types
  2. Derives data types
  3. User defined data types
  4. Pointer data types

**BASIC DATA TYPES:**

All arithmetic operations such as Addition , subtraction etc are possible on basic data types.

E.g.: int a,b; Char c;

**The following table shows the Storage size and Range of basic data types:**

**TYPE** **LENGTH** **RANGE**

|  |  |  |
| --- | --- | --- |
| Unsigned char | 8 bits | 0 to 255 |
| Char | 8 bits | -128 to 127 |
| Short int | 16 bits | -32768 to 32767 |
| Unsigned int | 32 bits | 0 to 4,294,967,295 |
| Int | 32 bits | -2,147,483,648 to 2,147,483,648 |
| Unsigned long | 32 bits | 0 to 4,294,967,295 |
| Enum | 16 bits | -2,147,483,648 to 2,147,483,648 |
| Long | 32 bits | -2,147,483,648 to 2,147,483,648 |
| Float | 32 bits | 3.4\*10E-38 to 3.4\*10E38 |
| Double | 64 bits | 1.7\*10E-308 to 1.7\*10E308 |
| Long double | 80 bits | 3.4\*10E-4932 to 1.1\*10E4932 |

**DERIVED DATA TYPES:**

Derived datatypes are used in ‘C’ to store a set of data values. Arrays and Structures are examples for derived data types.

|  |
| --- |
| Ex: int a[10];  Char name[20]; |

**USER DEFINED DATATYPES:**

C Provides a facility called **typedef** for creating new data type names defined by the user. For Example ,the declaration ,

|  |
| --- |
| **typedef int Integer;** |

makes the name Integer a synonym of int. Now the type Integer can be used in declarations ,casts,etc,like,

**Integer num1,num2;**

Which will be treated by the C compiler as the declaration of num1,num2as int variables. “typedef” ia more useful with structures and pointers.

**POINTER DATA TYPES:**

Pointer data type is necessary to store the address of a variable.

**INPUT AND OUTPUT STATEMENTS :**

**Input** means to provide the program with some data to be used in the program and **Output** means to display data on screen or write the data to a printer or a file.

C programming language provides many built-in functions to read any given input and to display data on screen when there is a need to output the result.

### scanf() and printf() functions

The standard input-output header file, named stdio.h contains the definition of the functions printf() and scanf(), which are used to display output on screen and to take input from user respectively.

|  |
| --- |
| #include<stdio.h> void main()  {  // defining a variable int i;  printf("Please enter a value...");  /\*  reading the value entered by the user  \*/  scanf("%d", &i);  /\*  displaying the number as output  \*/  printf( "\nYou entered: %d", i);  } |

You must be wondering what is the purpose of %d inside the scanf() or printf() functions. It is known as **format string** and this informs the scanf() function, what type of input to expect and in printf() it is used to give a heads up to the compiler, what type of output to expect.

|  |  |
| --- | --- |
| **Format String** | **Meaning** |
| **%d** | **Scan or print an integer as signed decimal number** |
| **%f** | **Scan or print a floating point number** |
| **%c** | **To scan or print a character** |
| **%s** | **To scan or print a character string. The scanning ends at whitespace.** |

We can also **limit the number of digits or characters** that can be input or output, by adding a number with the format string specifier, like "%1d" or "%3s", the first one means a single numeric digit and the second one means 3 characters, hence if you try to input 42, while scanf() has "%1d", it will take only 4 as input. Same is the case for output.

**NOTE :** printf() function returns the number of characters printed by it, and scanf() returns the number of characters read by it.

### getchar() & putchar() functions

* The getchar() function reads a character from the terminal and returns it as an integer.
* This function reads only single character at a time. You can use this method in a [loop](https://www.studytonight.com/c/loops-in-c.php) in case you want to read more than one character.
* The putchar() function displays the character passed to it on the screen and returns the same character. This function too displays only a single character at a time. In case you want to display more than one characters, use putchar() method in a loop.

|  |
| --- |
| #include <stdio.h> void main( )  { int c;  printf("Enter a character");  /\*  Take a character as input and store it in variable c  \*/  c = getchar();  /\*  display the character stored in variable c  \*/  putchar(c);  } |

When you will compile the above code, it will ask you to enter a value. When you will enter the value, it will display the value you have entered.

### gets() & puts() functions

The gets() function reads a line from **stdin**(standard input) into the buffer pointed to by str [pointer,](https://www.studytonight.com/pointers-in-c.php) until either a terminating newline or EOF (end of file) occurs. The puts() function writes the string str and a trailing newline to **stdout**.

|  |
| --- |
| #include<stdio.h>    void main()  {  /\* character array of length 100 \*/ |

str → This is the pointer to an array of chars where the C string is stored. (Ignore if you are not able to understand this now.)

|  |
| --- |
| char str[100];  printf("Enter a string"); gets( str ); puts( str ); getch();  } |

When you will compile the above code, it will ask you to enter a string. When you will enter the string, it will display the value you have entered.

**Difference between scanf() and gets()**

The main difference between these two functions is that scanf() stops reading characters when it encounters a space, but gets() reads space as character too.

If you enter name as **Study Tonight** using scanf() it will only read and store **Study** and will leave the part after space. But gets() function will read it completely.

# Programming Expression :

1. In programming, an expression is any legal combination of symbols that represents a value. [[Definition from Webopedia]](http://www.webopedia.com/TERM/E/expression.html)
2. C Programming Provides its own rules of Expression, whether it is legal expression or illegal expression. For example, in the C language x+5 is a legal expression.
3. Every expression consists of at least one operand and can have one or more operators.
4. Operands are values and Operators are symbols that represent particular actions. Valid C Programming Expression :

C Programming code gets compiled firstly before execution. In the different phases of compiler, c programming expression is checked for its validity.

|  |  |
| --- | --- |
| Expressions | Validity |
| a + b | Expression is valid since it contain + operator which is binary operator |
| + + a + b | Invalid Expression |

**OPERATORS :**

An operator is a symbol that tells the compiler to perform certain mathematical or logical manipulations. They form expressions.

C operators can be classified as

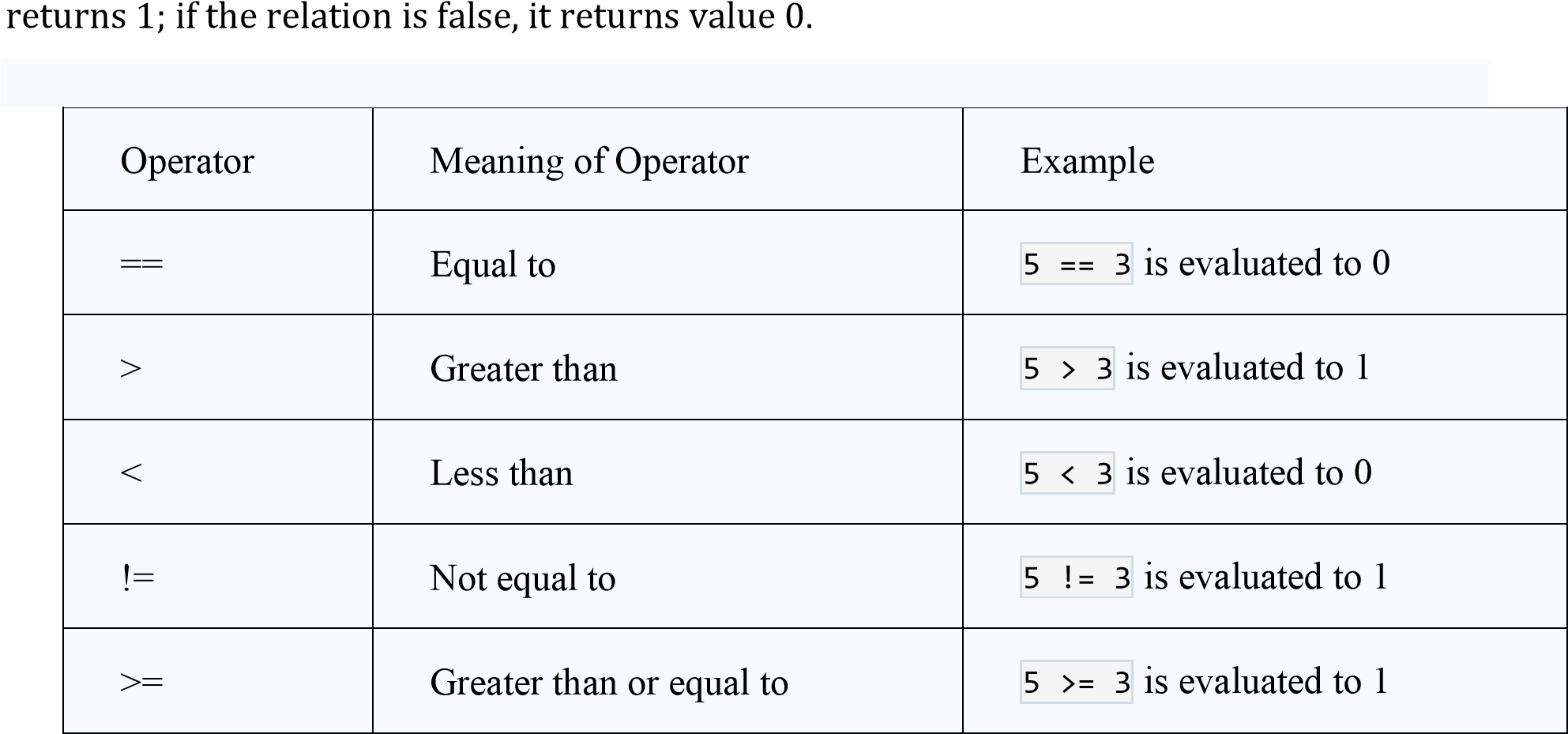
1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Increment or Decrement operators
6. Conditional operator
7. Bitwise operators
8. Special operators

1. **ARITHMETIC OPERATORS :** All basic arithmetic operators are present in C.

|  |  |
| --- | --- |
| operator | meaning |
| + | add |
| - | subtract |
| \* | multiplication |
| / | division |
| % | modulo division(remainder) |

An arithmetic operation involving only real operands(or integer operands) is called real arithmetic(or integer arithmetic). If a combination of arithmetic and real is called mixed mode arithmetic.

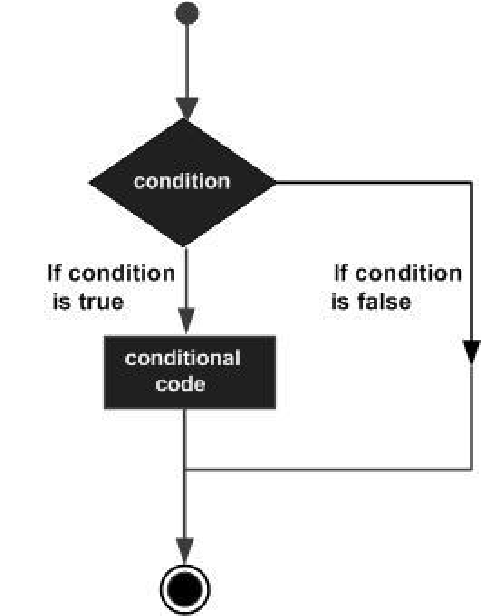
1. **RELATIONAL OPERATORS :**

A relational operator checks the relationship between two operands. If the relation is true, it 

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | | | |
| **3.** | | | | | Operator | | | | |  |  | | | | | |  | | |
| Meaning of Operator | Example | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | | Less than or equal to |  | | | | | |
| <= | | | | |  | 5 <= 3 | | is evaluated to 0 | | |
|  | | | | |  |  | | | | | |
| **LOGICAL OPERATORS :** An expression of this kind which combines two or more relationalexpressions is termed as a logical expressions or a compound relational expression. The operators and truth values are | | | | | | | | | | | |
| **4.** v | | | **Operator** | | | | | **Description** | | | | | | | **Example** | | |  | |
| && **(logical and)** | | | | | If both the operands are non-zero, then the condition becomes true. | | | | | | | (0 && 1) is false | | |
| || **(logical or)** | | | | | If any of the two operands are non-zero, then the condition becomes true. | | | | | | | (0 || 1) is true | | |
| ! **(logical not)** | | | | | Logical NOT Operator Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | | | | | | | !(0 && 1) is true | | |
| **ASSIGNMENT OPERATORS :** They are used to assign the result of an expression to a ariable.The assignment operator is '='. | | | | | | | | | | | | | | |
| **operat or** | | | **description** | | | | | | | | **example** | | | | | |
| = | | | assigns values from right side operands to left side operand | | | | | | | | A=B | | | | | |
| += | | | adds right operand to the left operand and assign the result to left | | | | | | | | A+=B is same as A=A+B | | | | | |
| -= | | | subtracts right operand from the left operand and assign the result to left operand | | | | | | | | A-=B is same as A=A-B | | | | | |
| \*= | | | mutiply left operand with the right operand and assign the result to left operand | | | | | | | | A\*=B is same as A=A\*B | | | | | |
| /= | | | divides left operand with the right operand and assign the result to left operand | | | | | | | | A/=B is same as A=A/B | | | | | |
| %= | | | calculate modulus using two operands and assign the result to left operand | | | | | | | | A%=B is same as A=A%B | | | | | |
| <<= | | | Left shift AND assignment operator. | | | | | | | | A <<= 2 is same as A = A<< 2 | | | | | |
|  | | | Right shift AND assignment operator. | | | | | | | | A >>= 2 is same as A = A >> 2 | | | | | |
| >>= | | |
| &= | | | Bitwise AND assignment operator. | | | | | | | | A &= 2 is same as A = A & 2 | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| **5.**    C | | | ^= | | | bitwise exclusive OR and assignment operator. | | | | | | A ^= 2 is same as A = A ^ 2 | | | | | |
| **INCREMENT AND DECREMENT OPERATORS :**  programming has two operators increment ++ and decrement -- to change the value of an | | | | | | | | | | | | | | |
|  | o perand (constant or variable) by 1.  I ncrement ++ increases the value by 1 whereas decrement -- decreases the value by 1.  These two operators are unary operators, meaning they only operate on a single operand. | | | | | | | | | | | | | |  | | |
| 1. **CONDITIONAL OPERATOR :** A ternary operator pair "?:" is available in C to construct conditional expressions of the form      |  | | --- | | **exp1 ? exp2 : exp3;** |     It work as if exp1 is true then exp2 else exp3     1. **BIT WISE OPERATORS :** C supports special operators known as bit wise operators for manipulation of data at bit level. They are not applied to float or double. | | | | | | | | | | | | | |
| **8.** | | **Operator** | | | | | **Description** | | | | | | | | | |
| & (bitwise and) | | | | | Bitwise AND operator give true result if both operands are true. otherwise, it gives a false result. | | | | | | | | | |
| | (bitwise or) | | | | | Bitwise OR operator give true result if any of the operands is true. | | | | | | | | | |
| ^ (bitwise XOR) | | | | | Bitwise Exclusive-OR Operator returns a true result if both the operands are different. otherwise, it returns a false result. | | | | | | | | | |
| ~(bitwise compliment) | | | | | Bitwise One's Complement Operator is unary Operator and it gives the result as an opposite bit. | | | | | | | | | |
| << (left shift) | | | | | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | | | | | | | | | |
| >> (right shift) | | | | | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | | | | | | | | | |
| **SPECIAL OPERATORS :** | | | | | | | | | | | | | | |

These operators which do not fit in any of the above classification are **,(comma), sizeof, Pointer operators(& and \*) and member selection operators (. and ->)**. The comma operator is used to link related expressions together.

**sizeof** is a unary operator that returns the size of data (constants, variables, array, structure, etc).

Control Structures or SELECTION STATEMENTS (DECISION MAKING):

Decision Making Statement in C. Decision making statement is depending on the condition block need to be executed or not which is decided by condition. If the condition is "true" statement block will be executed, if condition is "false" then statement block will not be executed.

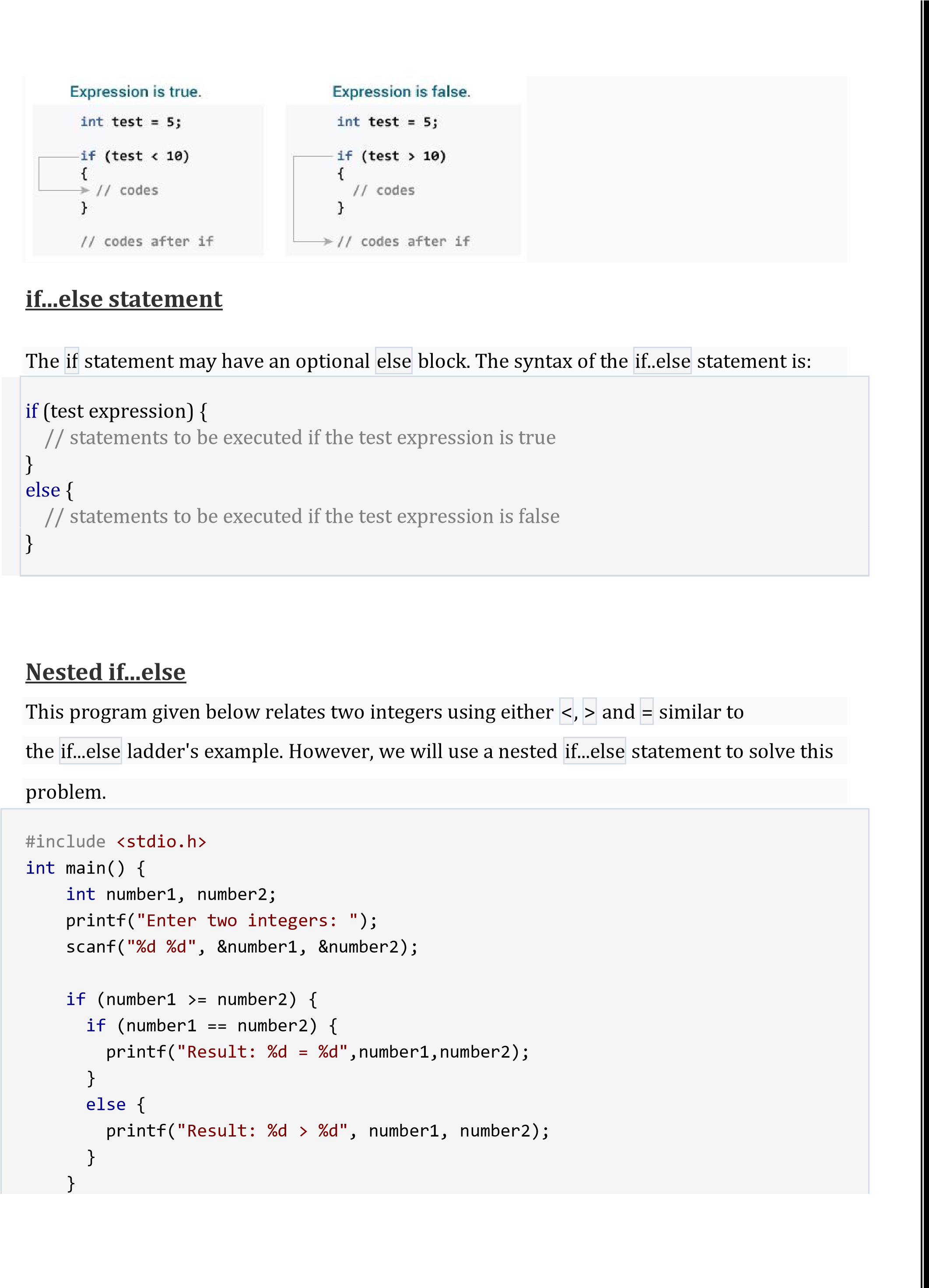
C programming language provides the following types of decision making statements.

|  |  |
| --- | --- |
|  | **Statement & Description** |
| **Sr.No.** |
| 1 | [if statement](https://www.tutorialspoint.com/cprogramming/if_statement_in_c.htm)  An **if statement** consists of a boolean expression followed by one or more statements. |
| 2 | [if...else statement](https://www.tutorialspoint.com/cprogramming/if_else_statement_in_c.htm)  An **if statement** can be followed by an optional **else statement**, which executes when the Boolean expression is false. |
| 3 | [nested if statements](https://www.tutorialspoint.com/cprogramming/nested_if_statements_in_c.htm)  You can use one **if** or **else if** statement inside another **if** or **else if** statement(s). |
| 4 | [switch statement](https://www.tutorialspoint.com/cprogramming/switch_statement_in_c.htm)  A **switch** statement allows a variable to be tested for equality against a list of values. |

|  |  |
| --- | --- |
| • | If the test expression is evaluated to true, statements inside the body of if are executed. |
| • | If the test expression is evaluated to false, statements inside the body of if are not executed. |

|  |
| --- |
| The if statement evaluates the test expression inside the parenthesis (). |

[**if statement**](https://www.tutorialspoint.com/cprogramming/if_statement_in_c.htm)



## switch statement

|  |
| --- |
| else { printf("Result: %d < %d",number1, number2);  } return 0;  } |

C switch statement is used when you have multiple possibilities for the if statement.

Switch case will allow you to choose from multiple options.

The basic format of the switch statement is: Syntax:

|  |
| --- |
| switch(variable)  { case 1:  //execute your code  break;  case n:  //execute your code  break;  default:  //execute your code break;  } |

Example: #include<stdio.h> main() { int a;

printf("Please enter a no between 1 and 5: "); scanf("%d",&a);

switch(a) { case 1: printf("You chose One"); break; case 2: printf("You chose Two"); break;

case 3: printf("You chose Three"); break; case 4: printf("You chose Four"); break; case 5: printf("You chose Five."); break; default : printf("Invalid Choice. Enter a no between 1 and 5"); break;

}

} o/p

Please enter a no between 1 and 5: **1**

**You chose One**

**Iterative Statements (for, While, Do-While)**

A loop statement allows us to execute a statement or group of statements multiple times. Given below is the general form of a loop statement in most of the programming languages −

|  |  |
| --- | --- |
|  | **Loop Type & Description** |
| **Sr.No.** |
| 1 | [while loop](https://www.tutorialspoint.com/cprogramming/c_while_loop.htm)  Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body. |
| 2 | [for loop](https://www.tutorialspoint.com/cprogramming/c_for_loop.htm)  Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. |
| 3 | [do...while loop](https://www.tutorialspoint.com/cprogramming/c_do_while_loop.htm) |

C programming language provides the following types of loops to handle looping requirements.

|  |
| --- |
| while(condition)  {  statement(s);  } |

[**while loop**](https://www.tutorialspoint.com/cprogramming/c_while_loop.htm)

|  |  |
| --- | --- |
|  | It is more like a while statement, except that it tests the condition at the end of the loop body. |

A **while** loop in C programming repeatedly executes a target statement as long as a given condition is true.

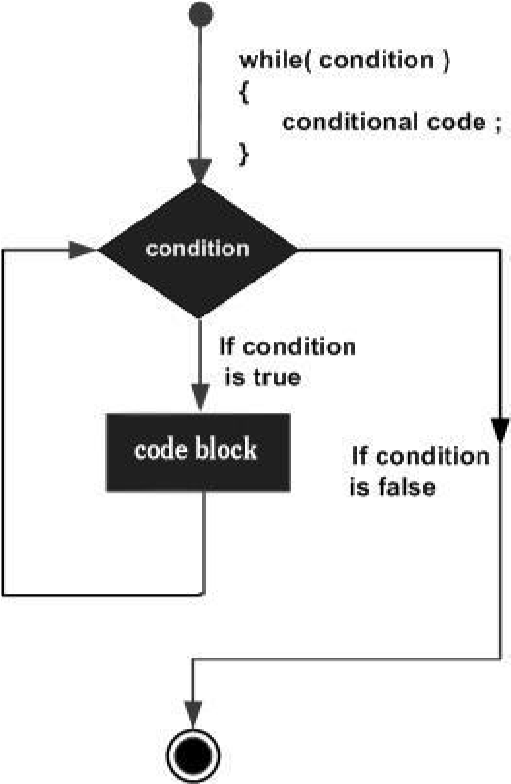
The syntax of a **while** loop in C programming language is −

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any nonzero value. The loop iterates while the condition is true.

When the condition becomes false, the program control passes to the line immediately following the loop.

|  |
| --- |
| #include <stdio.h>  int main ()  {  /\* local variable definition \*/ int a = 10;    /\* while loop execution \*/  while( a < 15 ) {  printf("value of a: %d\n", a); a++;  }  return 0;  } |

Here, the key point to note is that a while loop might not execute at all. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.



Example

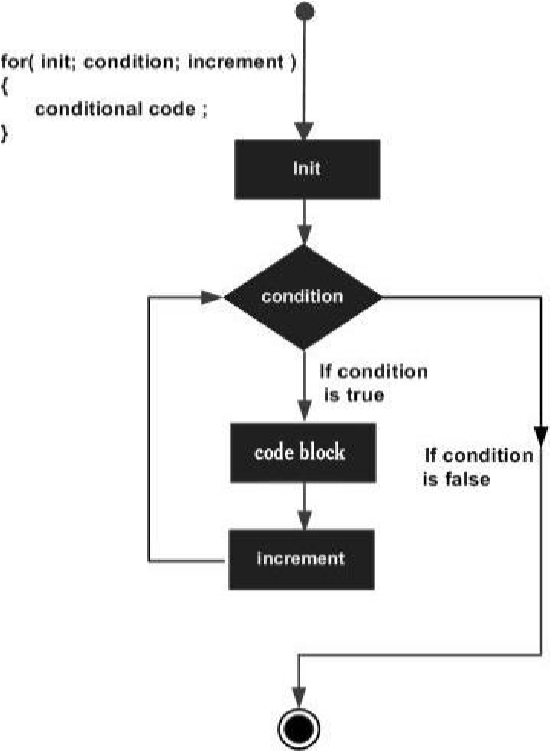
When the above code is compiled and executed, it produces the following result − value of a: 10 value of a: 11 value of a: 12 value of a: 13 value of a: 14

**For loop**

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

|  |
| --- |
| The syntax of a **for** loop in C programming language is − for ( init; condition; increment )  {  statement(s);  } |

Here is the flow of control in a 'for' loop −

* The **init** step is executed first, and only once. This step allows you to declare and initialize any loop control variables.
* Next, the **condition** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and the flow of control jumps to the next statement just after the 'for' loop.
* After the body of the 'for' loop executes, the flow of control jumps back up to the **increment** statement.
* The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again condition). After the condition becomes false, the 'for' loop terminates.

Flow Diagram

Example

|  |
| --- |
| #include <stdio.h> Int main () {  int a;  /\* for loop execution \*/ for( a = 10; a < 20; a = a + 1 ){ printf("value of a: %d\n", a);  }    return 0;  } |

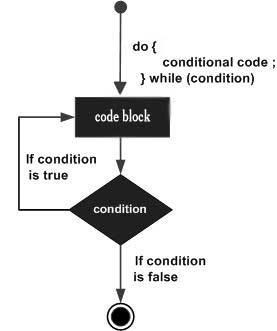
When the above code is compiled and executed, it produces the following result − value of a: 10 value of a: 11 value of a: 12 value of a: 13 value of a: 14 value of a: 15 value of a: 16 value of a: 17 value of a: 18 value of a: 19

**do While**

A **do...while** loop is similar to a while loop, except the fact that it is guaranteed to execute at least one time.

|  |
| --- |
| do  {  statement(s); } while( condition ); |

The syntax of a **do...while** loop in C programming language is −

Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop executes once before the condition is tested.

If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop executes again. This process repeats until the given condition becomes false.

|  |
| --- |
| #include <stdio.h>    int main () {    /\* local variable definition \*/ int a = 10;    /\* do loop execution \*/ do {  printf("value of a: %d\n", a); a = a + 1;  }while( a < 20 );    return 0;  } |

When the above code is compiled and executed, it produces the following result − value of a: 10 value of a: 11 value of a: 12 value of a: 13 value of a: 14 value of a: 15 value of a: 16 value of a: 17 value of a: 18 value of a: 19

**Loop Control Statements (break, Continue).**

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

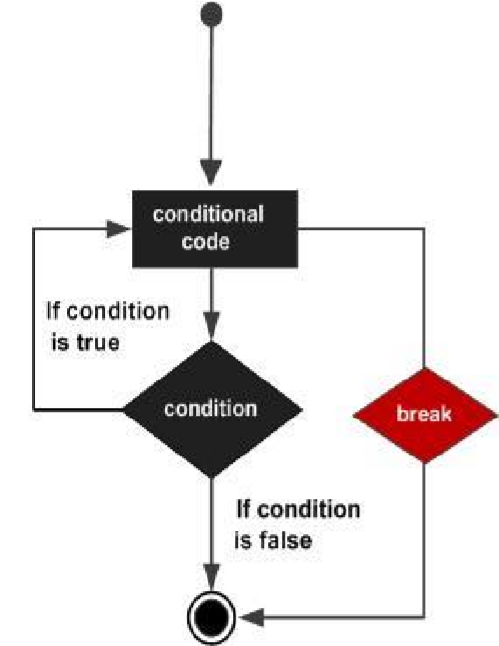
C supports the following control statements.

## Break statement

|  |  |
| --- | --- |
|  | **Control Statement & Description** |
| **Sr.No.** |
| 1 | [break statement](https://www.tutorialspoint.com/cprogramming/c_break_statement.htm)  Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch. |
| 2 | [continue statement](https://www.tutorialspoint.com/cprogramming/c_continue_statement.htm)  Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. |
| 3 | [goto statement](https://www.tutorialspoint.com/cprogramming/c_goto_statement.htm)  Transfers control to the labeled statement |

The **break** statement in C programming has the following two usages −

* When a **break** statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
* It can be used to terminate a case in the **switch** statement (covered in the next chapter).



The syntax for a **break** statement in C is as follows −

|  |
| --- |
| break; |
| #include <stdio.h>  int main () {  /\* local variable definition \*/ int a = 10;  /\* while loop execution \*/  while( a < 20 ) {  printf("value of a: %d\n", a); a++;    if( a > 15)  {  /\* terminate the loop using break statement \*/ break;  }  }  return 0;  } | | |

When the above code is compiled and executed, it produces the following result − value of a: 10 value of a: 11 value of a: 12

value of a: 13

value of a: 14

value of a: 15

**Continue**

**statement**

The

**continue**

statement in C programming works somewhat like the

**break**

statement.

Instead of forcing termination, it forces the next iteration of the loop to take place,

skipping any code in between.

For the

**for**

loop,

**continue**

statement

causes the conditional test and increment

portions of the loop to execute. For

the

**while**

and

**do...while**

loops,

**continue**

statement causes the program control to

pass to the conditional tests.

The syntax for a

**continue**

statement in C is as follows −

continue

;

#include

stdio.h

<

>

int

main

()

{

/\* local variable definition \*/

int

a

=

10

;

/\* do loop execution \*/

For(a=10;a<20;a++)

{

if

(

a

==

15

)

{

/\* skip the iteration \*/

continue

;

}

printf

(

"value of a:

%d

\

n"

,

a

)

;

}

return

0

;

}

When the above code is compiled and executed, it produces the following result −

value of a: 10

value of a: 11

value of a: 12

value of a: 13

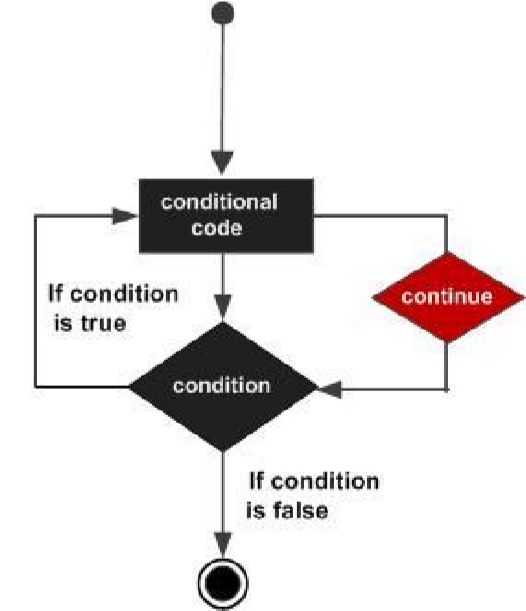
value of a: 14

value of a: 16

value of a: 17

value of a: 18

value of a: 19



# goto statement

A **goto** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.

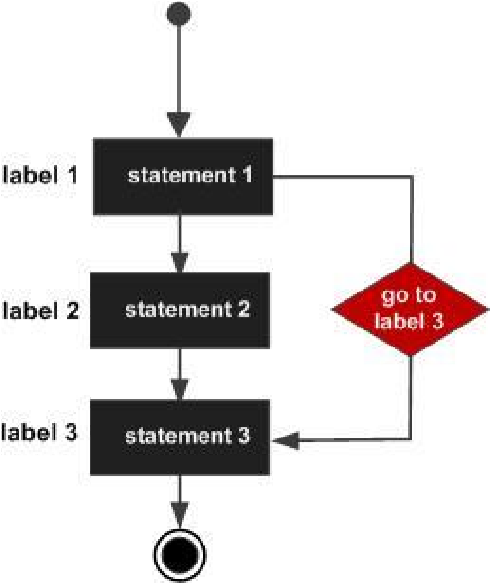
**NOTE** − Use of **goto** statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.

|  |
| --- |
| The syntax for a **goto** statement in C is as follows − goto label;  ..  . label:  statement; |

Syntax

|  |
| --- |
| /\* To read and print the number, if number is positive only\*/  #include <stdio.h> **int** main()  {  **int** number;    printf("Enter an integer number: "); scanf("%d",&number);  **if**(number<=0) goto end;  printf("Number is : %d", number);  end:  printf("Bye Bye !!!"); **return** 0;  } |

Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.



Output

First run:

Enter an integer number: 123 Number is : 123 Bye Bye !!!

Second run:

Enter an integer number: 0 Bye Bye !!