**INDUSTRIAL TRAINING REPORT**

**C-PROGRAMMING ITS IMPORTANCE**

Submitted in partial fulfillment of requirement of the

Degree of

**BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE & ENGINEERING**



SUBMITTED BY SUBMITTED TO

Name: Yashvardhan Jadhav Prof. Ranjeet Osari

Er.no.: EN20CS301519

**Department of Computer Science & Engineering**

**Faculty of Engineering**

**MEDI-CAPS UNIVERSITY, INDORE- 453331**

**Aug-Dec 2023**

**Report Approval**

The project work **“C-Programming & its Importance”** is hereby approved as a creditable study of an engineering/computer application subject carried out and presented in a manner satisfactory to warrant its acceptance as prerequisite for the Degree for which it has been submitted.

It is to be understood that by this approval the undersigned do not endorse or approved any statement made, opinion expressed, or conclusion drawn there in; but approve the “Project Report” only for the purpose for which it has been submitted.

Internal Examiner

Name:

Designation

Affiliation

External Examiner

Name:

Designation

Affiliation

**DECLARATION**

I hereby declare that the In-house training entitled “C-Programming and its importance” submitted in partial fulfillment for the award of the degree of Bachelor of Technology in ‘Computer Science & Engineering’ completed under the supervision of Prof. Dr Arpit Neema, Computer Science & Engineering Department from 12 June 2023 to 12 July 2023. Further, I declare that the content of this Industrial Training, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for the award of any degree or diploma. Signature and name of student with date.

**Signature and Name of the student with date**

----------------------

**Certificate**

This is to certify that Mr. Priyesh Sharma has completed Industrial Training during the period from 12 June 2023 to 12 July 2023 in our Organization as a Partial Fulfillment of Degree of Bachelor of Technology in Computer Science & Engineering. He was trained in the field of Software Programming.

**Signature & Seal of Training Manager**

**------------------------------**

****

**Acknowledgement**

I would like to express my deepest gratitude to Honorable Chancellor, Shri R C Mittal, who has provided me with every facility to successfully carry out this Industrial Training, and my profound indebtedness to Prof. Dr. Dilip Kumar Patnaik, Vice Chancellor, Medi-Caps University, whose unfailing support and enthusiasm has always boosted up my morale. I also thank Prof. Dr. Pramod S. Nair, Dean, Faculty of Engineering, Medi-Caps University, for giving me a chance to work on this Industrial Training. I would also like to thank my Head of the Department Prof. Dr. Ratnesh Litoriya for his continuous encouragement for betterment of the Industrial Training.

I express my heartfelt gratitude to my Instructor and Guide Prof. Dr. Arpit Neema, Department of Computer Science & Engineering, MediCaps University, without whose continuous help and support, this Industrial Training would ever have reached to the completion.

It is their help and support, due to which we became able to complete the design and technical report.

Without their support this report would not have been possible.

**Name:** Yashvardhan Jadhav **Er. No.:** EN20CS301519

**B.Tech.** IV Year

**Department of Computer Science & Engineering**

**Faculty of Engineering**

**Medi-Caps University, Indore**

**Table of Content**

1.Introduction

* What is C
* First program of c

2.Data types & its types

3.Variable

4.Constants

5.Console I/O Functions

* Formatted I/O functions
* Unformatted I/O functions

6.Operators and its types

* Arithmetic operator
* Relational operator
* Logical operator
* Assignment operator
* Increment/ Decrement operator
* Conditional operator
* Bitwise operator

7. Conditional Statement

* If-else Statement
* Switch Statement

8. Types of loops

* While loop
* Do-While loop
* For loop

9. Jump Statement

* Break statement
* Return statement
* Goto statement
* Continue statement

10. Array

11. Strings

12. Functions

* User-defined functions
* Library functions

13.Pointers

14.File

* File I/O
* Types of files
* File operations
* Input/Output operations on file

## What is C?

C is a programming language developed at **AT & T‟s Bell Laboratories of USA in 1972**. It was designed and written by **Dennis Ritche**. **Dennis Ritchie** is known as the **founder of c language**.

It was developed to overcome the problems of previous languages such as B, BCPL etc. Initially, C language was developed to be used in UNIX operating system.

### General Structure of a C program:

/\* Documentation section \*/

/\* Link section \*/

/\* Definition section \*/

/\* Global declaration section \*/ main()

{

Declaration part

Executable part (statements)

}

/\* Sub-program section \*/

* 1. The documentation section is used for displaying any information about the program like the purpose of the program, name of the author, date and time written etc, and this section should be enclosed within comment lines. The statements in the documentation section are ignored by the compiler.
  2. The link section consists of the inclusion of header files.The definition section consist

s of macro definitions, defining constants etc,.

* 1. Anything declared in the global declaration section is accessible throughout the program, i.e. accessible to all the functions in the program.
  2. main() function is mandatory for any program and it includes two parts, the declaration part and the executable part.
  3. The last section, i.e. sub-program section is optional and used when we require including user defined functions in the program.

First C Program

Before starting the abcd of C language, you need to learn how to write, compile and run the first c program.

To write the first c program, open the C console and write the following code:

1. #include <stdio.h>
2. #include <conio.h>
3. **void** main(){
4. printf("Hello C Language");getch();}

**#include <stdio.h>** includes the **standard input output** library functions. The printf() function is defined in stdio.h .

**#include <conio.h>** includes the **console input output** library functions. The getch() function is defined in conio.h file.

**void main()** The **main() function is the entry point of every program** in c language. The void keyword specifies that it returns no value.

**printf()** The printf() function is **used to print data** on the console.

**getch()** The getch() function **asks for a single character**. Until you press any key, it blocks the screen.

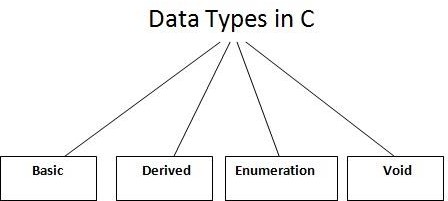
keywords**:** A keyword is a **reserved word.** All keywords have fixed meaning that means we cannot change. Keywords serve as basic building blocks for program statements. All keywords must be written in lowercase. A list of 32 keywords in c language is given below:

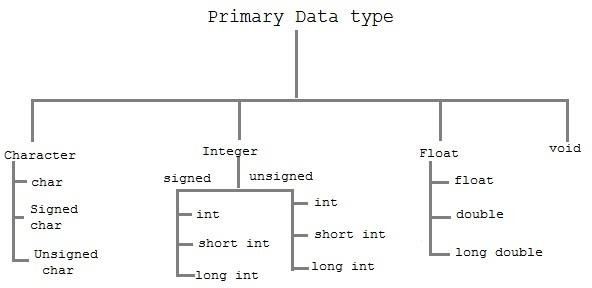
|  |  |  |  |
| --- | --- | --- | --- |
| **auto** | **break** | **case** | **char** |
| **const** | **continue** | **default** | **do** |
| **double** | **enum** | **else** | **extern** |
| **float** | **for** | **goto** | **if** |
| **int** | **long** | **return** | **register** |
| **signed** | **short**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

**Note: Keywords we cannot use it as a variable name, constant name etc.**

## Data Types/Types:

* To store data the program must reserve space which is done using datatype. A datatype is a keyword/predefined instruction used for allocating memory for data.





|  |  |
| --- | --- |
|  | **Data Types** |
| **Basic Data Type** | **int, char, float, double** |
| **Derived Data Type** | **array, pointer, structure, union** |
| **Enumeration Data Type** | **enum** |
| **Void Data Type** | **void** |

**Note: We call Basic or Primary data type.**

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals. The memory size of basic data types may change according to 32 or 64 bit operating system. Let‟s see the basic data types. Its size is given **according to 32 bit architecture**.

**Size and Ranges of Data Types with Type Qualifiers**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Size (bytes)** | **Range** | **Control String** |
| char or signed char | 1 | -128 to 127 | %c |
| unsigned char | 1 | 0 to 255 | %c |

|  |  |  |  |
| --- | --- | --- | --- |
| int or signed int | 2 | -32768 to 32767 | %d or %i |
| unsigned int | 2 | 0 to 65535 | %u |
| short int or signed short int | 1 | -128 to 127 | %d or %i |
| unsigned short int | 1 | 0 to 255 | %d or %i |
| long int or signed long int | 4 | -2147483648 to 2147483647 | %ld |
| unsigned long int | 4 | 0 to 4294967295 | %lu |
| float | 4 | 3.4E-38 to 3.4E+38 | %f or %g |
| double | 8 | 1.7E-308 to 1.7E+308 | %lf |
| long double | 10 | 3.4E-4932 to 1.1E+4932 | %Lf |

## Variables

A **variable** is a name of memory location. It is used to store data. Variables are changeable, we can change value of a variable during execution of a program. . It can be reused many times.

### Note: Variable are nothing but identifiers.

Rules to write variable names:

1. A variable name contains maximum of 30 characters/ Variable name must be upto 8 characters.
2. A variable name includes alphabets and numbers, but it must start with an alphabet.
3. It cannot accept any special characters, blank spaces except under score( \_ ).
4. It should not be a reserved word.

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

**Declaration of Variables :** A variable can be used to store a value of any data type. The declaration of variables must be done before they are used in the program. The general format for declaring a variable.

**Syntax :** data\_type variable-1,variable-2, , variable-n;

Variables are separated by commas and declaration statement ends with a semicolon.

**Assigning values to variables :** values can be assigned to variables using the assignment operator (=). The general format statement is :

we can also assign a value to a variable at the time of the variable is declared. The general format of declaring and assigning value to a variable is :

Types of Variables in C

There are many types of variables in c:

1. local variable
2. global variable
3. static variable

## Constants

Constants **refer** to fixed values that do not change during the execution of a program.

### Note: constants are also called literals.

### supports several kinds of constants.

***TYPES OF C CONSTANT:***

1. Integer constants
2. Real or Floating point constants
3. Character constants
4. String constants
5. Backslash character constants

**Formatted and Unformatted Console I/O Functions**.

**Input / Output (I/O) Functions :** In „C‟ language, two types of Input/Output functions are available, and all input and output operations are carried out through function calls. Several functions are available for input / output operations in „C‟. These functions are collectively known as the standard i/o library.

**Input:** In any programming language input means to feed some data into program. This can be given in the form of file or from command line.

**Output:** In any programming language output means to display some data on screen, printer or in any file.

### The Standard Files

C programming treats all the devices as files. So devices such as the display are addressed in the same way as files and the following three files are automatically opened when a program executes to provide access to the keyboard and screen.

|  |  |  |
| --- | --- | --- |
| **Standard File** | **File Pointer** | **Device** |
| Standard input | stdin | Keyboard |
| Standard output | stdout | Screen |
| Standard error | stderr | Your screen |

Input / Output functions are classified into two types

getch()

getche()

puts()

gets()

putchar()

getchar()

putc()

getc()

**Output**

**Input**

fprintf()

fscanf()

print()

scanf()

**Output**

**Input**

Unformated I/O Functions

Formated I/O Functions

**I / O Functions**

. **Formated I/O Functions :** formatted I/O functions operates on various types of data.

**1 : printf() :** output data or result of an operation can be displayed from the computer to a standard output device using the library function printf(). This function is used to print any combination of data.

**Syntax :** printf(“control string “, variable1, variable2, , variablen);

**Ex :** printf(“%d”,3977); // **Output**: 3977

### printf() statement another syntax :

**Syntax : printf(**“fomating string”);

**Formating string:**it prints all the character given in doublequotes (“ “)

**scanf() :** input data can be entered into the computer using the standard input „C‟ library function called scanf(). This function is used to enter any combination of input.

**Syntax :** scanf(“control string “,&var1, &var2, , &varn);

The scanf() function is used to read information from the standard input device (keyboard). Ex : scanf(“ %d “,&a);-> hello

Each variable name (argument) must be preceeded by an ampersand (&). The (&) symbol gives the meaning “address of “ the variable.

**Unformatted I/O functions:**

1. **Character I/O**

### String I/O

**a) character I/O:**

1. getchar(): Used to read a character from the standard input
2. putchar(): Used to display a character to standard output
3. getch() and getche(): these are used to take the any alpha numeric characters from the standard input

getche() read and display the character

getch() only read the single character but not display

1. putch(): Used to display any alpha numeric characters to standard output

### String I/O:

1. gets(): Used for accepting any string from the standard input(stdin) eg:gets()
2. puts(): Used to display a string or character array Eg:puts()
3. Cgets():read a string from the console eg; cgets(char \*st)

**OPERATORS AND EXPRESSIONS:**

**Operators :** An operator is a Symbol that performs an operation. An operators acts some variables are called operands to get the desired result.

Ex : a+b;

Where a,b are operands and + is the operator.

**Types of Operator :**

* 1. Arithmetic Operators.
  2. Relational Operators.
  3. Logical Operators.
  4. Assignment Operators. 5). Unary Operators.

1. Conditional Operators.
2. Special Operators.
3. Bitwise Operators.
4. Shift Operators.

### Arithmetic Operators

An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables).

C Program to demonstrate the working of arithmetic operators #include <stdio.h>

void main()

{

int a = 9,b = 4, c;

c = a+b;

printf("a+b = %d \n",c);

c = a-b;

printf("a-b = %d \n",c);

c = a\*b;

printf("a\*b = %d \n",c);

c=a/b;

printf("a/b = %d \n",c);

c=a%b;

printf("Remainder when a divided by b = %d \n",c);

}

#### Output

a+b = 13

a-b = 5

a\*b = 36

a/b = 2

Remainder when a divided by b=1

**Relational Operators**. A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Operands may be variables, constants or expressions. Relational operators are used in [decision making](https://www.programiz.com/c-programming/c-if-else-statement) and [loops](https://www.programiz.com/c-programming/c-for-loop).

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Meaning** | **Example** | **Return value** |
| < | is less than | 2<9 | 1 |
| < = | is less than or equal to | 2 < = 2 | 1 |
| > | is greater than | 2 > 9 | 0 |
| > = | is greater than or equal to | 3 > = 2 | 1 |
| = = | is equal to | 2 = = 3 | 0 |
| != | is not equal to | 2!=2 | 0 |

### Logical Operators.

These operators are used to combine the results of two or more conditions. An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in [decision making in C programming](https://www.programiz.com/c-programming/c-if-else-statement).

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Meaning** | **Example** | **Return value** |
| && | Logical AND | (9>2)&&(17>2) | 1 |
| || | Logical OR | (9>2) || (17 = = 7) | 1 |
| ! | Logical NOT | 29!=29 | 0 |

**Logical AND :** If any one condition false the complete condition becomes false.

#### Truth Table

|  |  |  |
| --- | --- | --- |
| **Op1** | **Op2** | **Op1 && Op2** |
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

**Logical OR :** If any one condition true the complete condition becomes true.

#### Truth Table

|  |  |  |
| --- | --- | --- |
| **Op1** | **Op2** | **Op1 // Op2** |
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

**Logical Not :** This operator reverses the value of the expression it operates on i.e, it makes a true expression false and false expression true.

|  |  |
| --- | --- |
| **Op1** | **Op1 !** |
| true | false |
| false | true |

**Assignment Operators.** Assignment operators are used to assign a value (or) an expression (or) a value of a variable to another variable.

**Syntax :** variable name=expression (or) value (or) variable Ex : x=10;

y=a+b; z=p;

#### Compound assignment operator:

„C‟ provides compound assignment operators to assign a value to variable in order to assign a new value to a variable after performing a specified operation.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Meaning** |
| + = | x + = y | x=x+y |
| - = | x - = y | x=x-y |
| \* = | x \* = y | x=x\*y |
| / = | x / = y | x=x/y |
| % = | x % = y | X=x%y |

### Increment and Decrement Operators /Unary Operators:

Unary operators are having higher priority than the other operators. **Unary operators**, meaning

#### they only operate on a single operand. Increment Operator in C Programming

1. Increment operator is used to increment the current value of variable by adding integer 1.
2. Increment operator can be applied to only variables.
3. Increment operator is denoted by ++.

We have two types of increment operator i.e Pre-Increment and Post-Increment Operator**. Pre-Increment**

Pre-increment operator is used to increment the value of variable before using in the expression. In the Pre-Increment value is first incremented and then used inside the expression.

#### b = ++y;

In this example suppose the value of variable „y‟ is 5 then value of variable „b‟ will be 6 because the value of „y‟ gets modified before using it in a expression**.**

#### Post-Increment

Post-increment operator is used to increment the value of variable as soon as after executing expression completely in which post increment is used. In the Post-Increment value is first used in a expression and then incremented.

b = x++;

In this example suppose the value of variable „x‟ is 5 then value of variable „b‟ will be 5 because old value of „x‟ is used

#### Note :

We cannot use increment operator on the constant values because increment operator operates on only variables. It increments the value of the variable by 1 and stores the incremented value back to the variable

#### b = ++5;

**or**

**b = 5++;**

The **syntax** of the operators is given below.

++<variable name> --<variable name>

<variable name>++ <variable name>--

The operator ++ adds 1 to the operand and – subtracts 1 from the operand. These operators in two forms : prefix (++x) and postfix(x++).

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| ++x | Pre increment |
| - -x | Pre decrement |
| x++ | Post increment |
| x-- | Post decrement |

Where

1. : ++x : Pre increment, first increment and then do the operation.
2. : - -x : Pre decrement, first decrements and then do the operation. 3 : x++ : Post increment, first do the operation and then increment. 4 : x- - : Post decrement, first do the operation and then decrement.

// C Program to demonstrate the working of increment and decrement operators #include <stdio.h>

int main()

{

int a = 10, b = 100;

float c = 10.5, d = 100.5; printf("++a = %d \n", ++a); printf("--b = %d \n", --b); printf("++c = %f \n", ++c); printf("--d = %f \n", --d); return 0;

}

#### Output

++a = 11

--b = 99

++c = 11.500000

++d = 99.500000

### Multiple increment operators inside printf

#include<stdio.h>

**void** main() {

**int** i = 1;

printf("%d %d %d", i, ++i, i++);

}

**Output : 3 3 1**

## Sequence of Printing Evaluating Expressions in PrintfPictorial representation

#### Explanation of program

I am sure you will get confused after viewing the above image and output of program.

* 1. Whenever more than one format specifiers (i.e %d) are directly or indirectly related with same variable (i,i++,++i) then we need to evaluate each individual expression from right to left.
  2. As shown in the above image evaluation sequence of expressions written inside printf will be – i++,++i,i
  3. After execution we need to replace the output of expression at appropriate place

|  |  |  |
| --- | --- | --- |
| No | Step | Explanation |
| 1 | Evaluate i++ | At the time of execution we will be using older value of i = 1 |
| 2 | Evaluate  ++i | At the time of execution we will be increment value already modified after step 1 i.e i = 3 |
| 2 | Evaluate i | At the time of execution we will be using value of i modified in step 2 |

**Postfix and Prefix Expression in Same Statement**

#include<stdio.h> #include<conio.h> void main() {

int i = 0, j = 0; j = i++ + ++i;

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

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

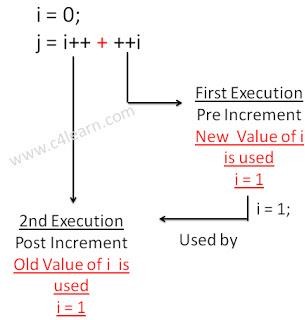
}

**Output :**

2

2

Explanation of Program



### Conditional Operator/ Ternary operator:

conditional operator checks the condition and executes the statement depending of the condition. A conditional operator is a ternary operator, that is, it works on 3 operands.

Conditional operator consist of two symbols.

1 : question mark (?). 2 : colon ( : ).

**Syntax** : condition ? exp1 : exp2;

It first evaluate the condition, if it is true (non-zero) then the “exp1” is evaluated, if the condition is false (zero) then the “exp2” is evaluated.

#include <stdio.h> int main(){

char February; int days;

printf("If this year is leap year, enter 1. If not enter any integer: "); scanf("%c",&February);

// If test condition (February == 'l') is true, days equal to 29.

// If test condition (February =='l') is false, days equal to 28. days = (February == '1') ? 29 : 28;

printf("Number of days in February = %d",days); return 0;

}

### Output

If this year is leap year, enter 1. If not enter any integer: 1 Number of days in February = 29

### Bitwise Operators:

Bitwise operators are used to manipulate the data at bit level. **It operates on integers only. It may not be applied to float.**In arithmetic-logic unit (which is within the CPU), mathematical operations like: addition, subtraction, multiplication and division are done in bit-level which makes processing faster and saves power. To perform bit-level operations in C programming, bitwise operators are used.

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise XOR |
| < < | Shift left |
| > > | Shift right |
| ~ | One‟s complement. |

### Bitwise AND operator &

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

Let us suppose the bitwise AND operation of two integers 12 and 25. 12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

00001100

& 00011001

00001000 = 8 (In decimal)

### Bitwise OR operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25 00001100

| 00011001

00011101 = 29 (In decimal) }

### Output

Output =29

### Bitwise XOR (exclusive OR) operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25 00001100

| 00011001

00010101 = 21 (In decimal)

### Bitwise complement operator ~

Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

35 = 00100011 (In Binary)

Bitwise complement Operation of 35.

~ 00100011

11011100 = 220 (In decimal)

#### Twist in bitwise complement operator in C Programming

The bitwise complement of 35 (~35) is -36 instead of 220, but why?

For any integer n, bitwise complement of n will be -(n+1). To understand this, you should have the knowledge of 2's complement.

#### 2's Complement

Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1. For example:

|  |  |  |
| --- | --- | --- |
| **Decimal** | **Binary** | **2's complement** |
| **0** | **00000000** | **-(11111111+1) = -00000000 = -0(decimal)** |
| **1** | **00000001** | **-(11111110+1) = -11111111 = -256(decimal)** |
| **12** | **00001100** | **-(11110011+1) = -11110100 = -244(decimal)** |
| **220** | **11011100** | **-(00100011+1) = -00100100 = -36(decimal)** |

#### Note: Overflow is ignored while computing 2's complement.

The bitwise complement of 35 is 220 (in decimal). The 2's complement of 220 is -36. Hence, the output is -36 instead of 220.

Bitwise complement of any number N is -(N+1). Here's how:

bitwise complement of N = ~N (represented in 2's complement form) 2'complement of ~N= -(~(~N)+1) = -(N+1)

#### There are two Bitwise shift operators in C programming:

* Right shift operator
* Left shift operator.

#### Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

#### Left Shift Operator

Left shift operator shifts all bits towards left by certain number of specified bits. It is denoted by

<<.

**If-else statement :** The if-else statement is an extension of the simple if statement. The general form is. The if...else statement executes some code if the test expression is true (nonzero) and some other code if the test expression is false (0).

**Syntax :** if (condition)

{

true statement;

}

else

{

false statement;

}

statement-x;

If the condition is true , then the true statement and statement-x will be executed and if the condition is false, then the false statement and statement-x is executed.

Or

If test expression is true, codes inside the body of if statement is executed and, codes inside the body of else statement is skipped.

If test expression is false, codes inside the body of else statement is executed and, codes inside the body of if statement is skipped.

#### Flowchart of if...else statement in C ProgrammingFlowchart

**Example:**

#### // Program to check whether an integer entered by the user is odd or even

#include <stdio.h> int main()

{

int number;

printf("Enter an integer: "); scanf("%d",&number);

// True if remainder is 0 if( number%2 == 0 )

printf("%d is an even integer.",number); else

printf("%d is an odd integer.",number); return 0;

}

#### Output

Enter an integer: 7 7 is an odd integer.

### Nested if-else statement

When a series of decisions are involved, we may have to use more than on if-else statement in nested form. If –else statements can also be nested inside another if block or else block or both.

**Switch statement :** when there are several options and we have to choose only one option from the available ones, we can use switch statement. Depending on the selected option, a particular task can be performed. A task represents one or more statements.

### Syntax:

switch(**expression**)

{

case value-1:

statement/block-1; break;

case value-2:

statement/block t-2; break;

case value-3:

statement/block -3; break;

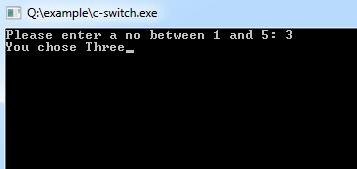
case value-4:

statement/block -4; break;

default:

default- statement/block t; break;

printf("You chose Two"); break;



### Why use loops in C language?

Suppose that you have to print table of 2, then you need to write 10 lines of code.By using the loop statement, you can do it by 2 or 3 lines of code only.

#### A looping process would include the following four steps.

1 : Initialization of a condition variable. 2 : Test the condition.

3 : Executing the body of the loop depending on the condition. 4 : Updating the condition variable.

5 : language provides three iterative/repetitive loops.

1. : while loop
2. : do-while loop 3 : for loop

**While Loop: Syntax :**

variable initialization ;

#### while (condition)

{

statements ;

variable increment or decrement ;

}

**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 )

**The while loop is an entry controlled loop statement, i.e means the condition is evaluated first** and it is true, then the body of the loop is executed. After executing the body of the loop, the condition is once again evaluated and if it is true, the body is executed once again, the process of repeated execution of the loop continues until the condition finally becomes false and the control is transferred out of the loop.

### Example : Program to print first 10 natural numbers

#include<stdio.h> #include<conio.h> void main( )

{

int x; x=1;

while(x<=10)

{

printf("%d\t", x); x++;

}

getch();

}

### Output

**1 2 3 4 5 6 7 8 9 10**

**do-while loop**

**Syntax :** variable initialization ;

**do**{

statements ;

variable increment or decrement ;

#### }while (condition);

The **do-while** loop is an **exit controlled loop statement** The body of the loop are executed first and then the condition is evaluated. If it is true, then the body of the loop is executed once again. The process of execution of body of the loop is continued until the condition finally becomes false and the control is transferred to the statement immediately after the loop. The statements are always executed at least once.

### For Loop:

* This is an **entry controlled looping** statement.
* In this loop structure, more than one variable can be initialized.
* One of the most important features of this loop is that the three actions can be taken at a time like variable initialization, condition checking and increment/decrement.
* The for loop can be more concise and flexible than that of while and do-while loops.

**Syntax :** for(initialization; condition; increment/decrement)

{

Statements;

}

**Example:** #include<stdio.h> #include<conio.h> void main( )

{

int x;

for(x=1; x<=10; x++)

{

printf("%d\t",x);

}

getch();

}

#### Output

1 2 3 4 5 6 7 8 9 10

**Jump Statements**

Jumping statements are used to transfer the program‟s control from one location to another, these are set of keywords which are responsible to transfer program‟s control within the same block or from one function to another**.**

## There are four jumping statements in C language:

* goto statement
* return statement
* break statement
* continue statement

**goto statement :** goto statement doesnot require any condition. This statement passes control anywhere in the program i.e, control is transferred to another part of the program without testing any condition.

**Syntax :** goto label;

. . . . .

. . . . .

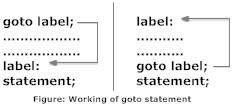
label:

statements; Inthissyntax, **label** isan identifier.

When, the control of program reaches to goto statement, the control of the program will jump to the **label:** and executes the code below it.

#### Or

The goto statement requires a label to identify the place to move the execution. A label is a valid variable/identifier name and must be ended with colon **( : )**



## ARRAYS

#### Using Arrays in C\

#### C supports a derived data type known as *array* that can be used to handle large amounts of data (multiple values) at a time.

#### Definition:

An array is a group (or collection) of same data types.

**Array** is a *collection* or *group* of elements (data). All the elements of array are *homogeneous* (similar). **It has contiguous memory location**.

An array is a data structured that can store a fixed size sequential collection of elements of same data type.

#### What‟s the need of an array?

Suppose you have to store marks of 50 students, one way to do this is allotting 50 variables. So it will be typical and hard to manage. For example we can not access the value of these variables with only 1 or 2 lines of code.

Another way to do this is array. By using array, we can access the elements easily. Only few lines of code is required to access the elements of array.

#### Where arrays are used

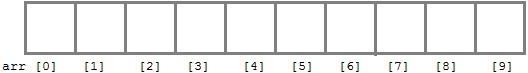
* to store list of Employee or Student names,
* to store marks of a students,
* or to store list of numbers or characters etc.

**Declaration of an Array**

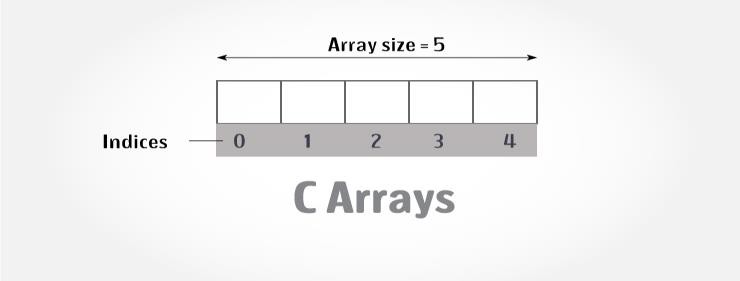
data-type variable-name[size/length of array];

#### For example:

int arr[10];



int arr[ 5];



Here **int** is the data type, **arr** is the name of the array and 10 is the size of array. It means array **arr** can only contain 10 elements of **int** type. **Index** of an array starts from 0 to size-1 i.e first element of **arr** array will be stored at arr[0] address and last element will occupy arr[9].

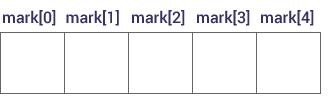
### How to access the elements of an array?

You can access elements of an array by **indices/index.** You can use array subscript (or index) to access any element stored in array. Subscript starts with 0, which means array\_name[0] would be used to access first element in an array.

In general array\_name[n-1] can be used to access nth element of an array. where n is any integer number.

Example float mark[5];

Suppose you declared an array mark as above. The first element is mark[0], second element is mark[1] and so on.



#### Few key notes:

* Arrays have 0 as the first index not 1. In this example, mark[0]
* If the size of an array is n, to access the last element, (n-1) index is used. In this example, mark[4]
* Suppose the starting address of mark[0] is 2120d. Then, the next address, a[1], will be 2124d, address of a[2] will be 2128d and so on. It's because the size of a float is 4 bytes.

#### Input data into array

As you can see, in above example that I have used „for loop‟ and „scanf statement‟ to enter data into array. You can use any loop for data input.

#### Code:

for (x=0; x<=19;x++)

{

printf("enter the integer number %d\n", x); scanf("%d", &num[x]);

}

#### Reading out data from an array

For example you want to read and display array elements, you can do it just by using any loop. Suppose array is mydata[20].

for (int i=0; i<20; i++)

{

printf("%d\n", mydata[x]);

}

**Exmaple** #include<stdio.h> #include<conio.h> void main()

{

int i;

int arr[]={2,3,4}; *//Compile time array initialization*

for(i=0 ; i<3 ; i++) { printf("%d\t",arr[i]);

}

getch();

}

#### Output

2 3 4

#### Exmaple

* 1. include <stdio.h>
  2. #include <conio.h>
  3. **void** main(){
  4. **int** i=0;
  5. **int** marks[5]={20,30,40,50,60};//declaration and initialization of array
  6. clrscr(); 7.

8. //traversal of array 9. **for**(i=0;i<5;i++){

10. printf("%d \n",marks[i]);

11. } 12.

13. getch();

14. }

#### utput

20

30

40

50

60

### STRINGS:

#### String Concepts

**String** is an *array of characters* that is terminated by \0 (null character). This null character indicates the end of the string. Strings are always enclosed by double quotes ( " " ). Whereas, character is enclosed by single quotes.

In „C‟ language the group of characters, digits, and symbols enclosed within double quotation ( " " ) marks are called as string otherwise a string is an array of characters and terminated by NULL character which is denoted by the escape sequence „\0‟.

#### C Strings

**Declaration of String**: C does not support string as a data type. However, it allows us to represent strings as character arrays. In C, a string variable is any valid C variable name and it is always declared as an array of characters.

The general form of declaration of a string variable is :

**Syntax:** char string\_name[size];

The size determines the number of characters in the string name.

**Note:** In declaration of string size must be required to mention otherwise it gives an error.

**Ex:** char str[]; // Invalid char str[0]; // Invalid char str[-1]; // Invalid char str[10]; // Valid char a[9]; //Valid

Using this declaration the compiler allocates 9 memory locations for the variable a ranging from 0 to 8.

0 1 2 3 4 5 6 7 8

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

Here, the string variable a can hold maximum of 9 characters including NULL(\0)

character.

#### Initializing Array string

**Syntax :** char string\_name[size]={“string” };

**Note:** In Initialization of the string if the specific number of character is not initialized it then rest of all character will be initialized with NULL.Z

char str[5]={'5','+','A'}; str[0]; ---> 5

str[1]; ---> +

str[2]; ---> A

str[3]; ---> NULL

str[4]; ---> NULL

Note: In initialization of the string we can not initialized more than size of string elements.

#### Ex:

char str[2]={'5','+','A','B'}; // Invalid

### FUNCTIONS:



#### User‐Defined Functions

**Definition:** A function is a block of code/group of statements/self contained block of statements/ basic building blocks in a program that performs a particular task. It is also known

as ***procedure*** or ***subroutine*** or **module**, in other programming languages.

To perform any task, we can create function. A function can be called many times. It provides *modularity* and code *reusability*.

#### Advantage of functions

1. **Code Reusability**

By creating functions in C, you can call it many times. So we don't need to write the same code again and again.

#### Code optimization

It makes the code optimized we don't need to write much code.

#### Easily to debug the program.

**Example:** Suppose, you have to check 3 numbers (781, 883 and 531) whether it is prime number or not. Without using function, you need to write the prime number logic 3 times. So, there is repetition of code.

But if you use functions, you need to write the logic only once and you can reuse it several times.

#### Types of Functions

There are two types of functions in C programming:

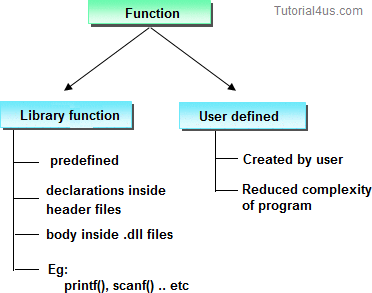
1. **Library Functions**: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc. You just need to include appropriate header files to use these functions. **These are already declared and defined in C libraries. oints to be Remembered**

System defined functions are declared in header files

System defined functions are implemented in .dll files. (DLL stands for Dynamic Link Library).

To use system defined functions the respective header file must be included.

1. **User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code. Depending upon the complexity and requirement of the program, you can create as many user-defined functions as you want.



#### ELEMENTS OF USER-DEFINED FUNCTINS :

In order to write an efficient user defined function, the programmer must familiar with the following three elements.

1 : Function Declaration. (Function Prototype). 2 : Function Call.

3 : Function Definition

#### Function Declaration. (Function Prototype).

A function declaration is the process of tells the compiler about a function name.

#### Syntax

return\_type function\_name(parameter/argument); return\_type function-name();

**Ex :** int add(int a,int b);

int add();

**Note:** At the time of function declaration function must be terminated with **;**. **Calling a function/function call**

When we call any function control goes to function body and execute entire code.

**Syntax** : function-name();

function-name(parameter/argument);

return value/ variable = function-name(parameter/argument);

|  |  |  |
| --- | --- | --- |
| **Ex** : | add(); | // function without parameter/argument |
|  | add(a,b); | // function with parameter/argument |
|  | c=fun(a,b); | // function with parameter/argument and return values |

#### Defining a function.

Defining of function is nothing but give body of function that means write logic inside function body.

#### Syntax

return\_ type function-name(parameter list) **// function header.**

{

declaration of variables;

body of function; **// Function body**

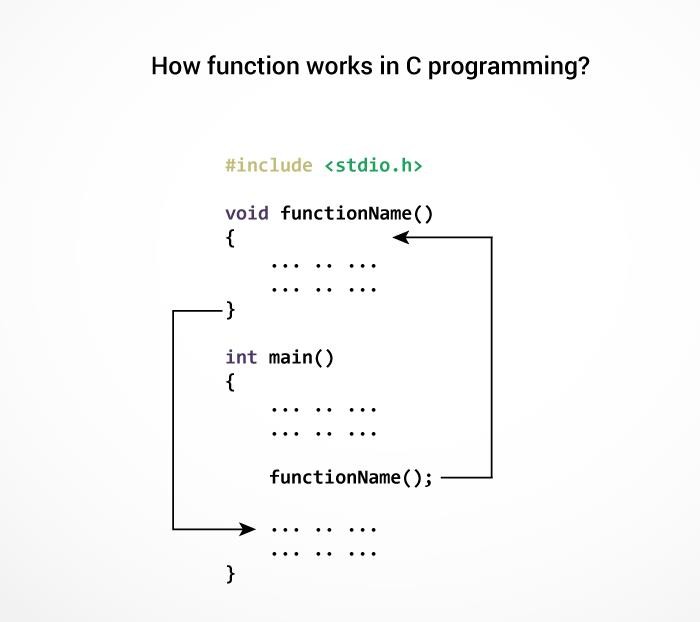
return statement; (expression or value) //**optional**

}

|  |  |  |  |
| --- | --- | --- | --- |
| Eg: int | add( int x, int y) |  | int add( int x, int y) |
| { |  |  | { |
|  | int z; | ( **or** ) | return ( x + y ); |
|  | z = x + y; |  | } |

return z;

}



#### The execution of a C program begins from the main() function.

#### : STANDARD I/O LIBRARY FUNCTIONS <STDIO.H>

|  |  |  |
| --- | --- | --- |
| **Functions** | **DataType** | **Purpose** |
| printf() | int | Send data items to the standared output device. |
| scanf() | int | Enter data items from the standard input device. |
| gets(s) | char | Enter string s from the standard input device. |
| getc(f) | int | Enter a string character from file f. |
| getchar() | int | Enter a single character from the standard input device. |
| putc(c,f) | int | Send a single character to file f. |
| puts(s) | int | Send string s to the standard output device. |
| putchar(c) | int | Send a single character to the standard output device. |
| fgetc(f) | int | Enter a single character from file f. |
| fgets(s,I,f) | char | Enter string s, containing I characters, from file f. |
| fprintf(f) | int | Send data items to file f. |
| fscanf(f) | int | Enter data items from file f. |
| fputc(c,f) | int | Send a single character to file f. |
| fputs(s,f) | int | Send string s to file f. |
| fread(s,il,i2,f) | int | Enter i2 data items, each of size i1 bytes, from file f. |
| fclose(f) | int | Close file f, return 0 if file is successfully closed. |

1. **: STANDARD LIBRARY FUNCTIONS <STDLIB.H>**

|  |  |  |
| --- | --- | --- |
| **Functions** | **DataType** | **Purpose** |
| abs(i) | int | Return the absolute value of i. |
| atof(s) | double | Convert string s to a double-precesion quantity. |
| calloc(u1,u2) | void\* | Allocate memory for an array having u1 elements, each of length u2 bytes. Return a pointer to the beginning of the allocated space. |
| exit(u) | void | Close all files and buffers, and terminate the program. |
| free(p) | void | Free a block of allocated memory whose beginning is indicated by p. |
| malloc(u) | void\* | Allocate u bytes of memory. |
| rand() | int | Return a random positive integer. |
| realloc(p,u) | void\* | Allocate u bytes of new memory to the pointer variable p, return a pointer to the beginning of the new memory space. |
| system(s) | int | Pass command string s to the operating system. |
| srand(u) | void | Initialize the random number generator. |

#### : STRING LIBRARY FUNCTIONS <STRING.H>

|  |  |  |
| --- | --- | --- |
| **Functions** | **DataType** | **Purpose** |
| strlen() |  | Finds length of string |
| strlwr() |  | Converts a string to lowercase |
| strupr() |  | Converts a string to uppercase |
| strcat() |  | Appends one string at the end of another |
| strcpy() |  | Copies a string into another |
| strcmp() |  | Compares two strings |
| strrev() |  | Reverses string |

### POINTERS:

#### Introduction

Definition:

Pointer is a variable that stores/hold address of another variable of same data type/ t is also known as locator or indicator that points to an address of a value. A pointer is a derived data type in C



#### Benefit of using pointers

* + Pointers are more efficient in handling Array and Structure.
  + Pointer allows references to function and thereby helps in passing of function as arguments to other function.
  + It reduces length and the program execution time.
  + It allows C to support dynamic memory management.

#### Declaration of Pointer

data\_type\* pointer\_variable\_name; int\* p;

**Note: void type pointer works** with all data types, but isn't used often.

#### Initialization of Pointer variable

**Pointer Initialization** is the process of assigning address of a variable to **pointer** variable.

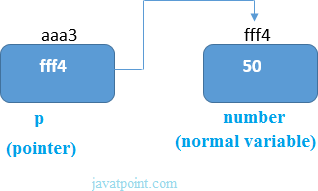
Pointer variable contains address of variable of same data type int a = 10 ;

int \*ptr ; *//pointer declaration* ptr = &a ; *//pointer initialization* or,int \*ptr = &a ; *//initialization and declaration together* **Note**:Pointer variable always points to same type of data. float a;

int \*ptr;

ptr = &a; //ERROR, type mismatch

Above statement defines, p as pointer variable of type int. **Pointer example**



As you can see in the above figure, pointer variable stores the address of number variable i.e. fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of \* (**indirection operator**), **we can print the value of pointer variable p.**

**Reference operator (&) and Dereference operator (\*)**

& is called reference operator. It gives you the address of a variable. There is another operator that gets you the value from the address, it is called a dereference operator (\*).

#### Symbols used in pointer

#### Dereferencing of Pointer

Once a pointer has been assigned the address of a variable. To access the value of variable, pointer is dereferenced, using the **indirection operator** \*.

int a,\*p; a = 10;

p = &a;

printf("%d",\*p); *//this will print the value of a.* printf("%d",\*&a); *//this will also print the value of a.* printf("%u",&a); *//this will print the address of a.* printf("%u",p); *//this will also print the address of a.* printf("%u",&p); *//this will also print the address of p.*

#### KEY POINTS TO REMEMBER ABOUT POINTERS IN C:

* + Normal variable stores the value whereas pointer variable stores the address of the variable.
  + The content of the C pointer always be a whole number i.e. address.
  + Always C pointer is initialized to null, i.e. int \*p = null.
  + The value of null pointer is 0.
  + & symbol is used to get the address of the variable.
  + \* symbol is used to get the value of the variable that the pointer is pointing to.
  + If a pointer in C is assigned to NULL, it means it is pointing to nothing.
  + Two pointers can be subtracted to know how many elements are available between these two pointers.
  + But, Pointer addition, multiplication, division are not allowed.
  + The size of any pointer is 2 byte (for 16 bit compiler).

#### Example:

#include <stdio.h> #include <conio.h> void main(){

int number=50; int \*p;

clrscr();

p=&number;//stores the address of number variable printf("Address of number variable is %x \n",&number); printf("Address of p variable is %x \n",p);

printf("Value of p variable is %d \n",\*p); getch();

}

#### Output

Address of number variable is fff4 Address of p variable is fff4 Value of p variable is 50

**Example:** #include <stdio.h> int main()

{

int \*ptr, q; q = 50;

/\* address of q is assigned to ptr \*/ ptr = &q;

/\* display q's value using ptr variable \*/ printf("%d", \*ptr);

return 0;

}

#### Output

50

**FILE**

* When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
* If you have to enter a large number of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.
* You can easily move your data from one computer to another without any changes.

#### File I/O:-

Sometimes it is necessary to store the data in a manner that can be later retrieved and displayed either in a part or in whole. This medium is usually a “file” on the disk. File I/O can be handled by using different functions.

1. **Formatted functions**:- The file input function fscanf( ) and the file output function fprintf( ) are called formatted file I/O functions.
2. **Unformatted functions:**- The input functions like getc( ), getw( ), and fread( ) are called unformatted file input functions and putc( ), putw( ), and fwrite( ) functions are unformatted file output functions. Each and every function is having its own syntax and meaning.

**File streams:-** Stream is either reading or writing of data. The streams are designed to allow the user to access the files efficiently. A stream is a file or physical device like key board, printer, monitor, etc., The FILE object uses these devices. When a C program is started, the operating system is responsible for opening three streams: standard input stream (**stdin**), standard output stream (**stdout**), standard error(**stderr**).Normally the stdin is connected to the keyboard, the stdout and stderr are connected to the monitor.

**Files**

File is a collection of bytes that is stored on secondary storage devices like Hard disk. OR

A **file** represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a ready made structure.

#### Note:

All files related function are available in **stdio.h** header file.

#### Types of Files

When dealing with files, there are two types of files you should know about:

* 1. Text files
  2. Binary files

#### Text files

Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.

When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.

They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

#### Binary files

Binary files are mostly the .bin files in your computer.

Instead of storing data in plain text, they store it in the binary form (0's and 1's).

They can hold higher amount of data, are not readable easily and provides a better security than text files.

### File Operations

In C, you can perform four major operations on the file, either text or binary:

* + Naming a file/Creation of new file
  + Opening an existing file
  + Reading data from file
  + Writing data into file
  + Closing a file

#### Steps for processing a file

* Declare a file pointer
* open a file using fopen() function
* Process the file using suitable file functions.
* close the file using fclose() function.

#### Declaration of a file

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and program.

#### Syntax

FILE **\*fp**;

#### Opening a file - for creation and edit

The fopen() function is used to create a new file or to open an existing file.

#### General Syntax :

fp = fopen("fileopen","mode")

**For Example:** fopen("E:\\cprogram\\newprogram.txt","w"); fopen("E:\\cprogram\\oldprogram.bin","rb"); **Closing a File**

The file (both text and binary) should be closed after reading/writing. Closing a file is performed using library function fclose(). fclose(fptr); //fptr is the file pointer associated with file to be closed. **File Opening Modes**

|  |  |
| --- | --- |
| **Mode** | **Description** |
| r | opens a text file in read mode |
| w | opens a text file in write mode |
| a | opens a text file in append mode |
| r+ | opens a text file in read and write mode |
| w+ | opens a text file in read and write mode |
| a+ | opens a text file in read and write mode |
| rb | opens a binary file in read mode |

|  |  |
| --- | --- |
| wb | opens a binary file in write mode |
| ab | opens a binary file in append mode |
| rb+ | opens a binary file in read and write mode |
| wb+ | opens a binary file in read and write mode |
| ab+ | opens a binary file in read and write mode |

#### Difference between Append and Write Mode

Write (w) mode and Append (a) mode, while opening a file are almost the same. Both are used to write in a file. In both the modes, new file is created if it doesn't exists already.

The only difference they have is, when you open a file in the write mode, the file is reset, resulting in deletion of any data already present in the file. While in append mode this will not happen. Append mode is used to append or add data to the existing data of file(if any). Hence, when you open a file in Append(a) mode, the cursor is positioned at the end of the present data in the file.

#### Formatted File I/O Functions Syntax of fprintf is

fprintf (fp, “control string”, list);

**Example:** fprintf(fp1, “%s %d”, name, age);

#### Syntax of fscanf is,

fscanf(fp, “control string”, list);

**Example**: fscanf(fp, “%s %d”, name, & age);

#### Note:

* fscanf is used to read list of items from a file
* fprintf is used to write a list of items to a file.

#### Note:

EOF – End of file (when EOF encountered the reading / writing should be terminated)

#### Example:

#include <stdio.h> main(){

**FILE** \*fp;

fp = fopen("file.txt", "w");//opening file

fprintf(fp, "Hello file by fprintf...\n");//writing data into file fclose(fp);//closing file

}

#### Example 1: Write to a text file using fprintf()

#include <stdio.h> int main()

{

int num;

FILE \*fptr;

fptr = fopen("C:\\program.txt","w"); if(fptr == NULL)

{

printf("Error!"); exit(1);

}

printf("Enter num: "); scanf("%d",&num);

fprintf(fptr,"%d",num); fclose(fptr);

return 0;

}

#### Example 2: Read from a text file using fscanf()

#include <stdio.h> int main()

{

int num;

FILE \*fptr;

if ((fptr = fopen("C:\\program.txt","r")) == NULL){ printf("Error! opening file");

// Program exits if the file pointer returns NULL. exit(1);

}

fscanf(fptr,"%d", &num); printf("Value of n=%d", num); fclose(fptr);

return 0;

}

**Input/Output Operation on files**

To perform Input/Output Operation on files we need below functions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Function** | **Operation** | **Syntax** |  |
| 1 | getc() | Read a character from a file | getc( fp) |  |
| 2 | putc() | Write a character in file | putc(c, fp) |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | fprintf() | To write set of data in file | fprintf(fp, "control string", list) |  |
| 4 | fscanf() | To read set of data from file. | fscanf(fp, "control string", list) |  |
| 5 | getw() | To read an integer from a file. | getw(fp) |  |
| 6 | putw() | To write an integer in file. | putw(integer, fp) |  |