ONLINE

SUPPROT SERVICES

CERTIFICATE IN

INFORMATION TECHNOLOGY

Run & Managed by NASO

IGNOU SC-2281

Jakhepal-Ghasiwala Road, Sunam

For more information visit us at: nirmancampus.co.in

Call us at: 9815098210, 9256278000

PROGRAM AND PROGRAMMING

A program is a set of instructions. These instructions are arranged in a sequence to solve a problem. The process of writing a program is called **Programming**. These computer programs or software can be broadly classified into two categories:

- System Software, and
- Application Software

System software is used to control the functionality of the computer system while the Application programs are used to solve a particular problem of the user. There are many computer languages used for developing these software.

DEVELOPMENT OF AN EFFICIENT PROGRAM

Before developing a program, programmer must have the precise details of the problem to be solved. These include the inputs available and the outputs to be generated. During development, a programmer has to pass the several stages of software. These stages are:

Problem Definition: Before software development, a programmer must know the details of the problem to be solved. It takes considerable skill to determine the problems in the existing system. Under study system provides the details of the problem to be solved. The programmer studies them. He discusses them with the System Analyst or the user and tries to understand the requirements.

Algorithm: The algorithm is a sequence of finite steps to solve a given problem. The construction of the algorithm requires creative thinking. Before developing a system, a programmer first set out the algorithm to visualize possible alternatives.

Flow-Charts: Flow-Chart is the pictorial representation of the algorithm. By using the suitable data the programmer should check the validity of his logic. This is very important for understanding the problem logic.

Coding: The sequence of operations defined in flow-chart are converted into a High-Level programming language such as C, C++, and PASCAL etc. writing the instructions using some computer language is called coding. This coding file is known as source program.

Compilation: The source program is fed into the computer. But a computer does not understand this source file. The compiler converts it into a machine understandable form. This machine understandable form is known as object code. This conversion of source code into object code is known as compilation. During compilation, compiler scans the source program for syntax errors. If there are syntax errors in the program, compiler generates error messages. These errors must be corrected to generate the object code.

Testing Process: The compiler only detects errors in the syntax. It cannot find the errors in the logic of the program. It is the programmer's task to find these logical errors. Any faulty logic can only be detected by examining the output. Finding and correcting such errors in the program is known as **Debugging**. This entire process is called Testing.

ALGORITHM

There are many definitions of an algorithm. The following definition is appropriate in computing science.

"The term algorithm may be defined as a sequence of instructions written in such a way that if the instructions are executed in the specified sequence, the desired result will be obtained."

In simple words, we can say that it is a finite set of sequence of steps to solve a particular problem. It must terminate and should not repeat one or more instructions infinitely.

DEVELOPING AN ALGORITHM

The first step in developing an algorithm is to identify a problem. Once a problem is identified, it is necessary to develop a step-by-step method of solution to solve the problem using a computer.
In algorithms, steps are very important because all the instructions are listed in the order in which they are carried out. Algorithms can be written in ordinary language like English.
The salient points for writing algorithm are given below:

- The inputs and outputs should be carefully specified.
- The flow of program execution should be taken care.
- Subroutines and functions should be used wherever required.
- The instructions should not be unclear.
- Number of repetitions should be finite i.e. the procedure should reach to some result.
- Algorithm must produce output(s)

Following are some examples of algorithms:

Example1: Calculate and print multiplication of two numbers:

Algorithm

Step1: Read two numbers A and B.
Step2: Multiply A and B and store it in C.
Step3: Print C.
Step4: Stop.

Example2: Write an algorithm to decide whether roots of $ax^2+bx+c=0$ will be real or not,given $a\neq0$.

Algorithm

Setp1: Read values of A, B and C
Step2: Calculate $D=B^2 – 4AC$.
Step3: If D<0 then
Print – Roots are not real.
Else
Print – Roots are real.
Step4: Stop

Example3: Write an algorithm to find the sum of 1 to N natural positive integers.

Algorithm

Step1: Read value of N
Step2: SUM=0;
Step3: Repeat Step4 FOR I=1 to N
Step4: SUM=SUM+I
Step5: Print SUM
Step6: Stop

FLOW CHARTING

Flow chart is the pictorial representation of algorithm. Flow charts are not required, but they are helpful in better understanding of the program logic.
Flowcharts are of two types:

- System Flow Charts.
- Program Flow Charts.

System Flowcharts

A system flowchart describes the data flow and operations for a data processing system. These flowcharts show how the data processing is to be accomplished.

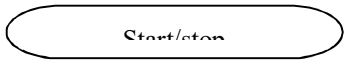
Program Flowcharts

A program flowchart describes the sequence of operations and decisions for a particular program. Program Flowcharts make use of standard conventions & symbols for various types of processing operations.

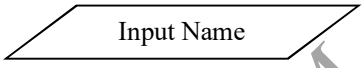
SYMBOLS USED IN FLOW CHARTS.

The following symbols are used for drawing flow charts:

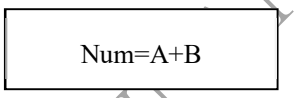
1. **The terminal Symbol**: Rectangles with rounded ends are used to indicate the beginning (START) and end (STOP)



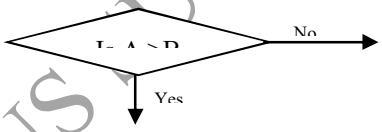
2. **The Input/ Output symbol**: Parallelograms are used to represent input/output operations.



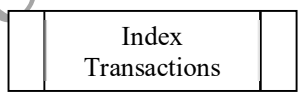
3. **The process/storage symbol**: Rectangles are used to indicate the processing operation such as storage or the arithmetic operations



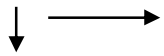
4. **The decision symbol**: Diamond shaped boxes are used for decision-making procedures. The arrows pointing out are labeled with Yes or No.



5. **Pre-defined process symbol**: Rectangle with width shown as double lines is used to represent a process which has been set out in detail somewhere else.



6. **Arrows:** Are used to show the flow of sequence of symbols.



7. **Notes**: Are used to add descriptions or notes or remarks

8. **Connector**: Exit to, Entry from another part of flow chart.

Advantages of Flow Charts:

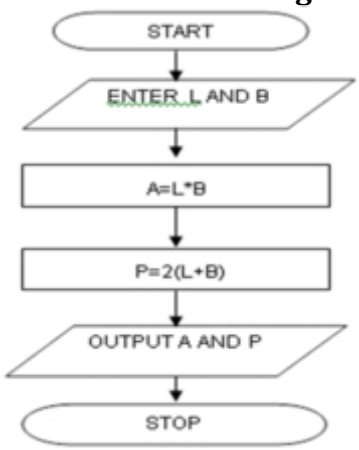
1. Flow charts help us to understand the logic of the problem.
2. With the help of flow-charts; logic drawn by one programmer is understood by other easily.
3. It is easier to detect a logical error in the flowchart than in the program listing.
4. Flow charts help a lot in program maintenance.

Disadvantages of Flow-Charts:

1. It becomes difficult to keep a flowchart neat and uncluttered if logic is complex.
2. It is difficult to change flow-chart without redrawing it, which is time consuming.
3. They sometimes become big and complex.

FLOWCHART EXAMPLES:

Example1: Given the length L and breadth B for a rectangle. Find the area A and perimeter P.



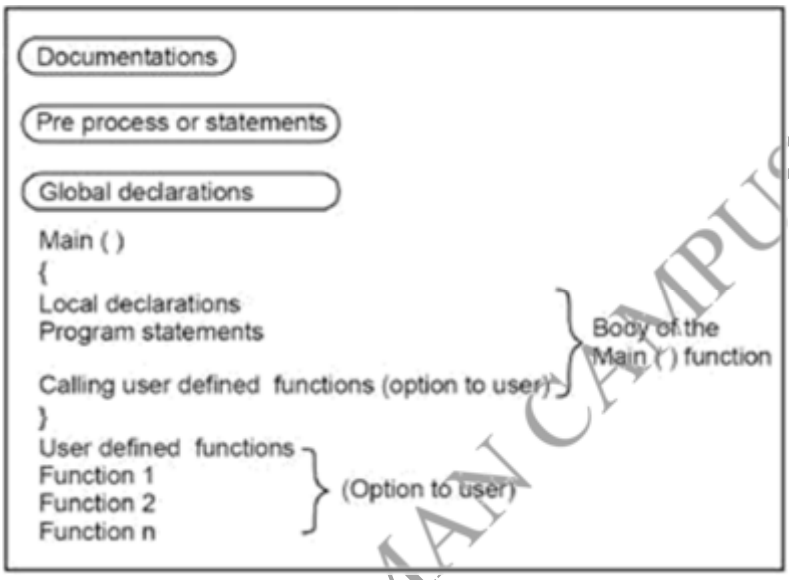
BASIC BUILDING BLOCKS OF C LANGUAGE

HISTORY OF C LANGUAGE

C language is very popular programming language. It was developed in the early 1970s. It was developed by Dennis M. Ritchie at Bell Labs (AT&T). It is developed from the BCPL (Basic Combined Programming Language). C Language is also called the Middle Level Programming language. It is because, it can be used to develop System and Application Programs.

STRUCTURE OF A C PROGRAM

The structure of a C program is a protocol (rules) to the programmer. These rules guide the programmer how to create the structure of the C program. The general basic structure of C program is shown in the figure below.



Documentations

The documentation section consists of a set of comment lines. It allows programmer to define some description of the program.

Preprocessor Statements

The preprocessor statement begins with # symbol. They are also called the **preprocessor directive**. These statements instruct the compiler to include header files or to define symbolic constants etc. before compiling. Some of the preprocessor statements are listed below.

```
# include <stdio.h>
# include <math.h>
# include <stdlib.h>
# include <CONIO.h>
```
header files

```
# define P L 3.1412.
# define TRVE 1
# define FALSE Ø
```
Symbolic constants

Global Declarations

Global declarations can eb variables or functions. They are declared before the main ( ) function. These global variables can be accessed by all the user defined functions in the program.

The main ( ) function

Execution of C program starts with main () function. No C program is executed without the main function. Every C program should contain only one main ( ) function.

Braces

The left braces after main() indicates the beginning of the main ( ) function and the right braces indicates the end of the main ( ) function.

Local Declarations

Variables declared inside main() are local declarations of main() functions. They can be used only within the main() function.

Program statements

These statements are the instructions of program. They are used to perform a specific task (operations). An instruction may contain an input-output statements, expression, control statements, simple assignment statements etc. Each executable statement should be terminated with semicolon.

User defined functions

These are subprograms. A subprogram is also a function. Subprograms are the logical grouping of statements to perform some specific task. These functions are written by the user. Therefore, they are called user defined functions.

CHARACTER SET:

The set of allowed characters in the language is called the character set. These character are used to form the vocabulary of the language, i.e. identifiers, keywords, constants, variable etc. Character set of C consists of:

- Letters – Alphabets in Upper and Lower Case, i.e. A - Z and a - z
- Digits – From 0 to 9
- Special characters: e.g. &, *, +, $, =, !, %, >, ?, (,), {, } etc.

TOKENS

Tokens are like the words and punctuation marks in natural language. These are the basic and the smallest units of C program. Similarly, in C these elements are keywords, constants, identifiers, operators, strings, and special Symbols. These are called tokens of c language. Following figure shows the six types of tokens present in the C language.

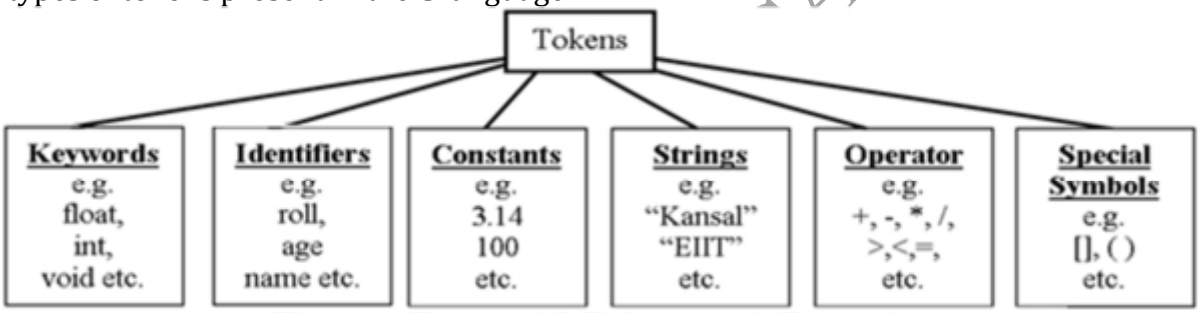


Figure – Types of C Tokens with Examples

Any C program consists of these tokens.

IDENTIFIERS

An Identifier is a name given to a programming element. This program element can be a variable, constant, function, array, structure etc. An identifier consists of a few letters, numbers and a special character (underscore). An identifier consists of maximum of 31 characters. C is a case sensitive language. It means identifiers in upper and lower case are distinct.

Rules for declaring identifiers:

Following are the rules for declaring identifiers:

1. The name of the identifier must begin with a character.
2. The name of identifier can contain maximum of 31 characters.
3. Identifier name cannot be a keyword.
4. Don't use white space in the name of identifier.
5. An identifier name can have letters, digits and valid special characters (underscore).

Following are some examples of valid identifier:

roll, a1, first_name, age, fathername etc.

Following names for identifiers are invalid:

1a ⟶ because first character must be a character
void ⟶ because it is a keyword
first name ⟶ because it contains white space
%age ⟶ because it contains a special character except than underscore

KEY WORDS/RESERVE WORDS

Those words which are predefined in the compiler of C language are called keywords. Keywords are also known as reserve words. We cannot change the meaning of these keywords. All the keywords must be written in the lower case characters only. When we use these keywords in the Turbo C Editor, they are displayed in the white color. Normally, a C compiler has 32 keywords. But new C Compilers have 37 keywords. Following are the C language keyword:

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

VARIABLES

The term 'variable' consists of two words vari (vary) + able. It means which can be changed. Thus, a variable allows us to change its value during runtime/execution time of the program. These are used to store the data temporarily in the main memory of computer. This data remain in memory during the execution of program.

To use variables in the program, they must be declared in program. For this, programmers have to give it a name and a data type. For deciding a variable name, following rules should be taken care.

1) The name of variable should not exceed than 31 characters.
2) A variable name can contain alphabets, digits or underscore.
3) The variable name must start with a Character.
4) Special Symbols except Underscore '_' are not allowed in variable name.
5) A keyword cannot be a variable name.
6) Uppercase and lowercase variable names are significant, i.e., roll, ROLL and Roll are the different variables.

Some examples of valid variable names are – a1, first_name, my_float_no, age, rollno etc.

Type Declaration/ Variable Declaration

C language is a strongly typed language. It means all the variables must be declared before using them. Variable declaration contains two things:

1. The data type of variable, and
2. The variable name

The syntax for declaring a variable is:

```
Data_typevariable_name;
```

Here, Data_type tells the type of data to be stored in the variable. The variable_name is any valid name for the variable.

For example: int age; float percent; char grade;

In the above examples, int, float and char are the data types and age, percent and grade are the valid variable names.

Storing/Assigning Value To A Variable

Assigning values to a variable means storing a value in the variable. For this we use assignment operator, e.g.

```
int a;                    char grade;
a = 20;                   grade = 'A';
```

In the above example, 20 value is assigned to an integer variable 'a' with the help of assignment operator. Here, '=' is the assignment operator.

Variable Initialization

It is an assignment to variable at declaration time. For example:

```
int age = 20;
float pi = 3.14;
```

CONSTANTS:

Those identifiers which do not allow to change their value during execution time of program are called constants. So, constants have fixed values. C supports several types of constants. We can classify the constants as follows-

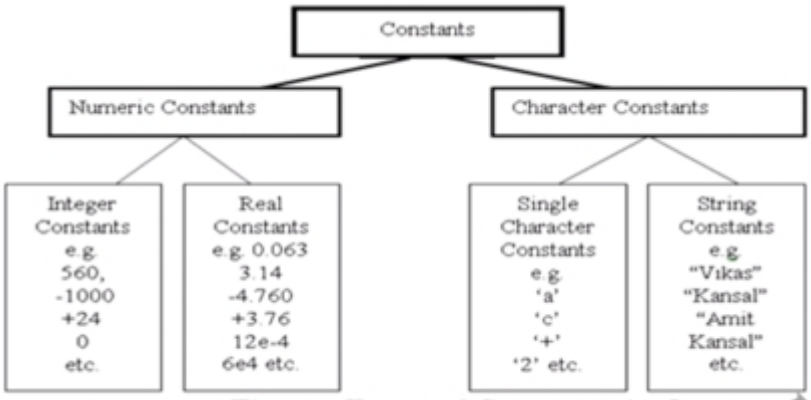


Figure: Types of Constants in C

Numeric Constants:

These constants consist of digits from 0 to 9. They can be with or without decimal points in them. These numbers may be preceded with +ve or –ve sign. There are two types of numeric constants:

(1) Integer Constants
(2) Real or Floating Point Constants

1. **Integer Constants:** It is a signed or unsigned whole number. C supports Integer constants in the form of decimal, octal and hexadecimal numbers. Examples of integer constants are- 550 (Decimal), 043 (Octal), 0xA3(Hexadecimal).
2. **Real Constants**: Real constants are also known as floating point constants. Any number with fractional part is called real or floating point constant. This number may be preceded by the +ve or –ve sign. A real constant can be written in decimal or exponential form. Examples of real constants are-

   Real constant in Decimal form is : 0.2569
   Real Constant in Exponential form is : +2.64e3

   The exponential form is used when value is too small or too large. In this form, number is divided in two parts-mantissa and exponent. Mantissa parts appears before 'e' and the exponent part is after 'e'.

Character Constants

There are two types of character constants:

(1) Single Character Constants, and
(2) String Constants.

1. **Single Character Constants:** These constants must be enclosed within single quote. It can either be a single alphabet, a single digit or a single special symbol. Example of character constants are- 'v', '2', '*' etc.
2. **String Constants**: The combination of characters is called a string. Any string may consist of alphabets, digits and symbols. The string must be enclosed within double quotes. Example of string constants are- "Param Kansal", "Phone no. 9501010979" etc.

Constant Declaration:

To define the constant we use **const keyword**. Its syntax is:

const<data type><constants name>;

For example;

const float pi=3.14;

The alternative method of declaring constants is **Symbolic Constants**. For this, '#define' preprocessor directive is used.

For Example:

#define PI 3.14
#define MAX 100

DATA TYPES:

Data type is the type of data to be stored in the main memory. C language is rich in data types. The Data types in C language can be classified as-

- Primitive Data Types, and
- Non-Primitive data types

Primary Or Primitive Data Types

These data types are predefined in the compiler of C. These data types are also known as *fundamental or built-in* data types. These data types are: int, float, char, double and void. All C compilers support these data types. We can classify these data types into Integer, Real and void types.

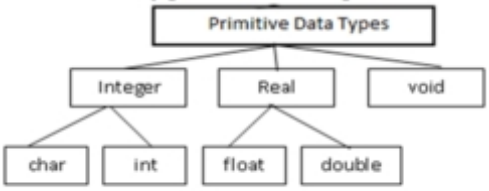


Figure: Primitive Data Types

Following table shows required memory size and the range of values of these data types.

Data Type	Size	Value Range	Range in Decimal	Description
char	1 Byte	-2^7 to 2^7-1	-128 to 127	Stores character value
int	2 Bytes	-2^{15} to $2^{15}-1$	-32768 to 32767	Stores integer value
float	4 Bytes	-2^{31} to $2^{31}-1$	$3.4x10^{-38}$ to $3.4x10^{+38}$	Stores fractional value
Double	8 Bytes	-2^{63} to $2^{63}-1$	$1.7x10^{-308}$ to $1.7x10^{+308}$	Stores fractional value

Table – Size and Range of Basic Data Types in C

char data type:

It is used to store the character data. It takes one byte memory to store value. Its value range is -2^7 to 2^7-1. The range in decimal is -128 to 127. Following program show how to use it:

```
void main( )
{
char ch='A';
printf("%d",ch);                    //shows 65 (ASCII code of A)
printf("%c",ch);                    //shows A
}
```

int data type:

It is used to store the integer data. It takes two bytes memory to store value. Its value range is -2^{15} to $2^{15}-1$. The range in decimal -32768 to 32767. Following program show how to use it:

```
void main( )
{
int a=65;
printf("%d",a);                     //shows 65
}
```

float data type:

It is used to store the single precision fractional data. It takes 4 bytes memory to store value. Its value range is -2^{31} to $2^{31}-1$. The range in decimal $3.4x10^{-38}$ to $3.4x10^{+38}$. Following program show its use:

```
void main( )
{
float a=6.5;
printf("%f",a);                     //shows 6.500000
}
```

double data type:

It is used to store the double precision fractional data. It takes 8 bytes memory to store value. Its value range is -2^{63} to $2^{63}-1$. The range in decimal $1.7x10^{-308}$ to $1.7x10^{+308}$. Following program show how to use it:

```
void main( )
{
double a=6.5;
printf("%lf",a);                    //shows 6.500000
}
```

Void type

The void type has empty or null value. This data type is commonly used with functions. Those functions which do not return any value, have void type.

NON-PRIMITIVE/SECONDARY DATA TYPES

Those data types which are not inbuilt in C, are called non-primitive data types. These data types are: derived, user-defined, and pointers.

Derived Data Type

Those data types which are derived from the basic data types are called derived data types. Arrays, Structure and Union are the derived data types.

User-Defined Data Types

These data types are defined by the user. The 'enum' and 'typedef' are used for this purpose.

Pointers

Pointer is a powerful feature of C language. Pointers are used to store the memory address of a variable.

OPERATORS AND THEIR TYPES

Operators are the symbols used to perform specific operations. For example, + operator is used to perform addition operation, > is used to compare values etc. These operators perform their operation on the operands. **Operands** are the identifier on which operation is performed. Based on the operand, operators can be classified as:

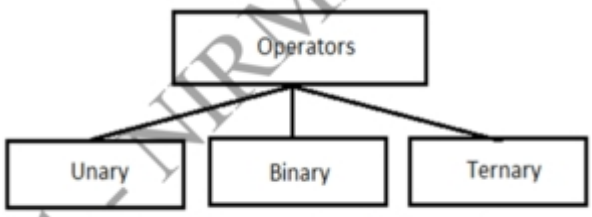


Fig: Types of Operators based on Operands

Unary Operators:

Those operators which require **one operand** to perform their operation are called unary operators. Examples of these operators are: ++, --, !etc.

For example: a++, --b

Binary Operators:

Those operators which require **two operands** to perform their operation are called binary operators. Examples of these operators are: +, -, *, %, &&, > etc. Most of the operators in C are Binary operators.

For example: a+b, a>b

Ternary Operators:

Those operators which require three operands to perform their operation are called Ternary operators. This operator is also called Conditional Operator. Example of this operator is: __?___:___ This is the only ternary operator in C language.

For Example: big = a>b **?**a **:** b ;

Operators can also be categorized according to their operations. They can be categorized as below.

1.) Arithmetic Operators.
2.) Comparison Operators
3.) Logical/Boolean Operators
4.) Assignment Operators
5.) Increment/Decrement Operators
6.) Bitwise Operators
7.) Conditional Operators
8.) Additional Operator

ARITHMETIC OPERATORS:

These operators are used for arithmetic operations. These are five operators. These are +, -, *, / and %. All these are binary operators. Following table shows the operations of these operators:

Name	Operator	Description	Example	
Addition	+	Used to perform Addition of numbers.	2+4 => 6	2.0+4.0 =>6.0
Subtract	-	Used to perform subtraction or used as any unary minus.	6-2 => 4	6.0-4.0=>2.0
Multiply	*	Used to perform multiplication of numbers.	7*2 => 14	7.0*2.0 =>14.0
Division	/	Used to perform division of numbers.	5/2 => 2 Integer division	5.0/2.0 =>2.5 Real Division
Modulus	%	Used to get remainder value after division of numbers.	7%4=>3	5.0%2.0 Not allowed

RELATIONAL OPERATORS:

These are also called **comparison operators**. These operators are used for comparing values. These are 6 operators. These are >, <, >=, <=, == and !=. All these are binary operators. Following table shows the operations of these operators:

Name	Operator	Description	Example	Result
Equal to	= =	Used to check whether two values are equal	4= =5 5= = 5	False True
Not Equal to	!=	Used to check whether two values are not equal	4! =5 4! =4	True False
Greater then	>	Used to check whether first value is greater than second	4>5 5>4	False True
Less then	<	Used to check whether first value is lass then second	4<5 5<4	True False
Greater than or equal to	>=	Used to check whether first value is greater than or equal to second value	5>=5 6>=8	True False
Less than or equal to	<=	Used to check whether first value is lesser than or equal to second value	4<=5 4<=2	True False

LOGICAL OPERATORS

These are also called **Boolean Operators**. These operators are used to form compound relational expressions. These operators are also used to compare values. These are 3 operators. These are && (AND), || (OR) and ! (NOT). AND and OR are binary operators and NOT is unary. Following table shows the operations of these operators:

Name	Operators	Description	Associatively	Example	Result
AND	&&	Return true if both operands are true otherwise returns false	Left to Right	3>5 && 4>5 3>5 && 4<5 3<5 && 4>5 3<5 && 4<5	False False False True
OR	||	Return true if at least one operand s are true	Left to Right	3>5 || 4>5 3>5 || 4<5 3<5 || 4>5 3<5 || 4<5	False True True True
NOT	!	Return true if operand is false & vice – versa	Right to Left	!(3<5) !(3>5)	False True

ASSIGNMENT OPERATORS

These Operators are used to assign/store values in a variable. The symbol of assignment operator is '='. Consider the following examples which show how to use assignment operator in C programs:

```
a = - 2;          // assigns –ve value (-2) to the variable.
b = 5;            // assigns value (5) to the variable.
c = a + b;        // assigns the result of expression to the variable.
a = a + 10;       // self-assignment of a variable.
```

Assignment operators can also be used as shorthand operators. Shorthand assignment operators are useful in self-assignment statements. Following table shows the examples of shorthand operators used in C:

Shorthand Operator	Example for Shorthand Assignment	Equivalent Self-Assignment
+=	a+ =2	a = a + 2
- =	a- =2	a = a – 2
*=	a* =2	a = a * 2
/=	a / =2	a = a / 2
%=	a%=2	a = a % 2

Table – List of Shorthand Assignment Operands

INCREMENT AND DECREMENT OPERATOR

These are unary operators. They require only one operand. In C, '++' is the *increment* operator and '--' is the *decrement* operator. Increment operator adds one to the current value while the decrement operator decreases one to the current value.

Consider the following example:

```
int a = 5, b = 6;
a ++ ;          //a becomes 6
b – –;          //becomes 5
```

These operators can be classified into two categories. These categories are named as:

- Prefix Increment/Decrement operator. (++a/--a)
- Postfix Increment/Decrement operator. (a++/a--)

CONDITIONAL OPERATOR

It is the only ternary operator used in c language. It requires three operands to perform its operation. This operator is used to carry out conditional operations. It can be used in place of if – else statement. The syntax for conditional operator is:

Condition?Expression1 for True Condition:Expression2 for False Condition

Example of Ternary Operator:

```
int x, y, big;
x = 100;
y = 15;
big = x>y ?x : y;
```

EXPRESSION

An expression is the valid combination of operators and operands. Here, operands may be a variable or it can be a constant. Consider the following example:

```
c = a+b;
a = c + 10;
```

Here c, a,b, and 10 are the operands and + and = are the operators. Expressions can be categorized according to the operators used in the expressions:

1. Arithmetic Expressions
2. Relational Expressions
3. Logical Expressions
4. Mixed mode Expressions

1. **Arithmetic Expressions**

When Arithmetic operators are used in an expression, itis called as *Arithmetic Expression*. This expressions return numeric value. For Example:

c=a+b c=a*b etc.

2. **Relational Expressions**

When relational operators are used in an expression, it is called as *Relational Expressions*. Relational Expression returns either True (non zero value) or False (zero value). For example:

a > b a = = b 5 > = 2 etc.

3. **Logical expressions**

When logical operators along with relational operators are used in the expression, then the expression is termed as Logical Expression. Logical Expression also returns either True or False value. For example:

(a > b)&&(a > c) (a > b) || (b > c) ! (a > b) etc.

4. **Mixed mode expressions**

When an expression contains more than one type of operator, it is called mixed mode expression. For example:

(a+b) >= (c*d)

HIERARCHY OR PRECEDENCE OF OPERATORS IN EXPRESSIONS

An expression may contain more than one type of operators. In such situation, which operator is evaluated first?, is decided by the hierarchy of operators. The sequence of operators in which they are applied on the operands in an expression is called the *Precedence of Operators*.

Consider the following examples which illustrates how expressions are evaluated using operators precedence–

```
a = 5 * 4 / 4 + 8 – 9 / 3;      (* is evaluated)
a = 20 / 4 + 8 – 9 / 3;         (/ is evaluated)
a = 5 + 8 – 9 / 3;              (/ is evaluated)
a = 5 + 8 - 3;                  (+ is evaluated)
a = 13 - 3;                     (- is evaluated)
a = 10;
```

TYPE CONVERSION

When value of one type is converted into other type, it is called *Type Conversion.* There are two types of type conversions.

(1.) Automatic Conversion or Implicit Conversion
(2.) Casting Value or Explicit Conversion.

Automatic/Implicit Conversion:

This type of conversion is automatic. For this type of conversion, we use assignment (=) operator. It is also called implicit conversion. This type of conversion is used when lower data type operand is converted into higher data type. There is not loss of information in this type of conversion.

Consider the following example for automatic conversion:

```
int m = 15;
float n = m;
```

Casting a value or Explicit Conversion

This is forceful conversion. For this type of conversion, we use caste operator. It is also called explicit conversion. There may or may not be any loss of information in this type of conversion. This type of conversion is used when higher data type operand is converted into lower data type.

The syntax for this type of casting is:

(Desired data type) Expression

For example:

```
float m;
int n = 7;
m = (float)n/2;
```

CONTROL STATEMENTS:

Flow of execution of program is sequential by nature. To control this execution flow, we use control statement. So, we can say that those statements which are used to control the execution flow in a program are called *Control Statements*. These control statements can be categorized into three divisions:

1. Conditional / Branching / Decision-Making Control Statements.
2. Looping or Repetitive Control Statements.
3. Unconditional Control Statements.

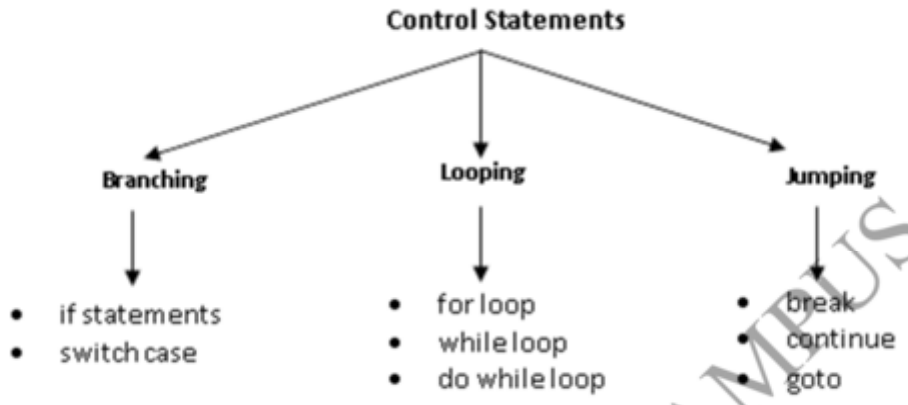


Fig: Classification of Control Statements

BRANCHING CONTROL STATEMENTS

These statements are used for decision making purpose. These are also called ***conditional*** **or** ***selection*** **or** ***decision – making*** control statements. C language supports the following Branching Control Statements:

i. The 'if else' statements.
ii. The 'switch case' statement.

If else statements:

If else statements can be used in the four different ways:

- Only if statement
- If and else statement
- else if ladder statement
- nested if else statement

If statement:

It is a branching statement. It is used for decision making purpose. If the given condition is true, it will run the statement otherwise do nothing.

Consider the following example and its syntax:

Syntax:

```
if (Conditional_Expression)
Statement;
        Or
if(Conditional_Expression)
{
    Statements;
}
```

Example:

```
#include<stdio.h>
void main( )
{
int a, b;
a=10;
b=10;
if (a==b)
    printf("Both numbers are equal");
}
```

If and else statement:

It is a branching statement. It is used for decision making purpose. If the given condition is true it will run the statements1 otherwise it will run the statements2, as shown below in the example:

Syntax:

```
if (Conditional_Expression)
{
        Statements1;
}
else
{
        Statements2;
}
```

Example:

```
void main( )
{
int a, b;
a=10;
b=20;
if (a==b)
      printf("Both numbers are equal");
else
      printf("Numbers are not equal");
}
```

Else if statement:

It is another form of if statement. It is also used for decision making purpose. It is used when we have to test multiple conditions. It is a chain of many if else statements. It executes the first true condition. If no condition is true, it will execute the last else statement. Consider the following syntax and example:

Syntax:

```
if(Conditional_Expression1)
        Statement1;
else if ((Conditional_Expression2)
      Statement2;
    else if ----
    -
       -
       -
         else if ((Conditional_Expression_n)
           Statement n;
```

Example:

```
void main()
{
float per;
per=65;
if(per>=75)
printf("Grade - A");
else if(per>=60)
printf("Grade - B");
else if(per>=40)
      printf("Grade - C");
            else
            printf("Grade - D");
}
```

Nested if else statement:

It is another form of branching statement. It is also used for decision making purpose. It is used to test multiple conditions. When one 'if else' statement is used within another if-else statement, it is called Nested if else statement. Consider the following syntax and example:

Syntax:

```
if(Conditional_Expression)
{
        if(Conditonal_Expression1)
                Statement;
        else
                Statements;
}
else
{
        if(Conditonal_Expression2)
                Statements;
        else
                Statements;
}
```

Example:

```
void main()
{
int a=5, b=8, c=6;
      if(a>b)
      {
            if(a>c)
            printf("A is largest");
            else
            printf("C is largest");
      }
      else
      {
            if(b>c)
            printf("B is largest");
            else
            printf("C is largest");
      }
}
```

Switch case:

It is another branching statement. It is also used for the decision making purpose. It is like else if statement. It is used to test multiple conditions. It is used when we have to test a variable or expression against a limited set of constant values. It is used to test integer type operands only. Consider the following syntax and example:

```
Syntax:
switch (expression or variable)
{
        case value1:
                Statements1;
                break;
        case value2:
                Statements2;
                break;
        .
        .
        case value_n:
                Statements_n;
                break;
        default:
                Statements;
}
```

```
Example:
void main()
{
char ch;
ch='i';
  switch(ch)
  {
  case 'a':
       printf("a is a vowel");break;
  case 'e':
       printf("e is a vowel");break;
  case 'i':
       printf("i is a vowel");break;
  case 'o':
       printf("o is a vowel");break;
  case 'u':
       printf("u is a vowel");break;
  default:
       printf("Not a vowel");
  }
}
```

In switch case, expression or variable is matched with each case value. When a match is found, the corresponding block of case is executed. If not case matches with the variable's value, default statement will be executed.

LOOPING STATEMENTS:

These statements are also called **iterative statements**. These are used for repetitions. C provides three types of iterative or looping control structures:

a. The 'for' loop.
b. The 'while' loop.
c. The 'do while' loop.

Any looping control statement, in general, would consist of the following components:

- **Initialization** – It is the starting value of counter variable
- **Test condition** – it is the end value of the counter variable.
- **Iteration** – Increases or decreases the value of counter variable.
- **Body of loop** – These are statements to be executed repeatedly.

Looping control structures may be classified as -

A. Entry - Controlled loop (Pre-Test Loop), and
B. Exit - Controlled loop (Post-Test Loop).

ENTRY CONTROLLED LOOPS:

In the entry-controlled loop, the control conditions are tested before the body of loop. The 'for' and 'while' loops are the entry-controlled loops

For loop:

It is a looping control statement. It is used for repeating statements. It is entry controlled loop in which control conditions are tested before the body of loop.

```
Syntax:

for (initialization; test-condition; iteration)
{
        Body of the loop;
}
```

```
Example:
void main()
{
int i;
      for(i=1; i<=10; i++)
      {
      printf("\n%d", i);
      }
}
```

While loop:

It is a looping control statement. It is used for repeating statements. It is entry controlled loop. In this loop condition is tested before the statements (body) of loop.

Syntax:

```
Initialization;
while (condition)
{
        Statements;
        Iteration;
}
```

Example:

```
void main()
{
int i;
      i=1;                        //initialization
      while(i<=10)                //test condition
      {
      printf("\n%d",i);           //statement
      i++;                        //iteration
      }
}
```

EXIT CONTROLLED LOOP:

In the exit-controlled loop, the test condition is performed after the body of the loop. Therefore, this loop must execute at least once, even if the condition is false initially. Do while loop is an example of exit controlled loop.

Do while:

It is a looping control statement. It is also used for repeating statements. It is an exit controlled loop. In this loop, condition is tested after the body of loop. So this loop must execute at least once.

Syntax:

```
Initialization;
do
{
        Statements;
        Iteration;
} while (condition);
```

Example:

```
void main()
{
int i;
      i=11;
      do
      {
            printf("\n%d",i);
            i++;
      }while(i<=10);
}
```

NESTING OF LOOPS

When a loop is used within another loop, It is termed as Nesting of Loops. We can put any loop within another loop. For example, we can put a 'for' loop in another 'for' loop or a 'while' loop etc. Consider the following syntax and example:

Syntax:

```
for(initialization; condition; iteration)
{
    for(initialization; condition; iteration)
    {
    Statements;
    }
    Statements;
}
```

(Outer Loop / Inner Loop)

Example:

```
void main()
{
int i,j;
  for(i=1;i<=10;i++)
  {
     for(j=1;j<=5;j++)
     {
     printf("\t%d",i*j);
     }
  printf("\n");
  }
}
```

COMPARISON OF WHILE AND DO-WHILE LOOP

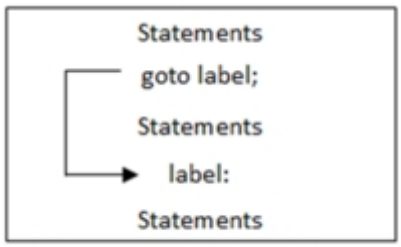
The comparison between 'while' and 'do while' statements is as follow:

1. In 'while' loop, the condition is tested before execution of the body of loop. But, in 'do while' loop, the condition is tested after execution of the body of the loop.
2. The 'do while' loop must execute at least once even if the condition is false initially. But the 'while' loop may not execute if the condition is not satisfied initially.
3. While loop is entry controlled loop and do while loop is exit controlled loop.
4. The 'do while' loop is followed by the semicolon (;) but in the 'while' loop, it is not given.

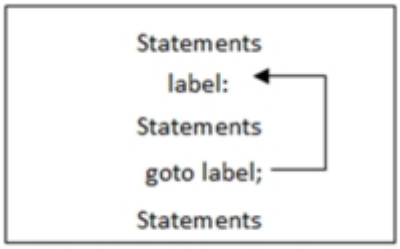
JUMPING STATEMENTS

These statements are used to jump the execution control from one to other part of the program. These are also known as **skipping statements**. These statements are goto, continue and break:

The 'goto' statement: It is a jumping statement. It transfers the control at the defined location. Location is defined by the label. It can use either the forward jump or backward jump, as shown in the figure.



Syntax: Forward Jumping



Syntax: Backward Jumping

Consider the following example:

```
Example:
void main()
{
int i=1;
//label for goto statement
start:
printf("%d",i);
i++;
if(i<=5)
goto start;
}
```

Output:

1
2
3
4
5

The 'break' statement: It is a jumping statement. It takes the control out of the control statement in which it is used. It is widely used in switch statement.

Following example shows the use of break statement in a loop.

```
Example:
void main()
{
int i;
   for(i=1;i<=10;i++)
   {
      if(i==5){
         break;
      }
   printf("\n%d",i);
   }
}
```

Output:

1
2
3
4

The 'continue' statement: It is a jumping statement. It takes the control to the next iteration in the loop. It is used in loops.

Following program shows how to use 'continue' statement

```
Example:
void main()
{
int i;
   for(i=1;i<=7;i++)
   {
      if(i==5)
      {
         continue;
      }
   printf("\n%d",i);
   }
}
```

Output:

1
2
3
4
6
7

FUNCTIONS:

A function is a subprogram. It is also called routine or procedure. It is a logical grouping of instructions in a program. Functions are used to break a complex problem into a small parts or modules. They can be used multiple times, but defined only once. Every executable program must have a main( ) function. The main function is an entry point of execution of the program. The main( ) function invokes the other functions to perform various tasks.

ADVANTAGES OF FUNCTIONS:

Using functions have a number of advantages. The main advantages of using a function are:

- It becomes easy to solve a complex program by dividing it into small functions.
- It becomes easy to debug the program.
- It is easy to maintain and modify a function.
- It can be called any number of times but defined only once.
- Small functions are self-documenting.
- It facilitates top-down modular programming approach as shown in the following figure.

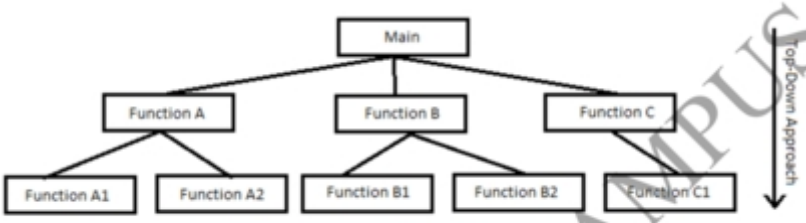


Figure: Functions

TYPES OF FUNCTIONS:

Functions can be classified in two categories:

1. **Library/Standard Functions**: Those functions, which are in-built or pre-defined in the C language, are called standard or library functions. All these functions are organized in header files. So to use any predefined functions, we must include the corresponding header file in which it is used. For example: clrscr( ), printf(), scanf(), sqrt(), strcpy() etc. are the predefined functions.
2. **User Defined Functions**: Those functions, which are defined by the user, are called user defined functions. User can define his own functions to fulfill his requirements. Function main( ) is the best example of user defined function.

FUNCTION DEFINITION:

Function definition defines the working of the function. It has a function type, name, parenthesis with zero of more arguments, and a body. The general format of the function definition is given below:

```
functiontype functionname (arguments)
{
        body of the function
        return statement;
}
```

In the above syntax, functiontype shows the type of value it would return to calling function. A function may or may not return a value to the calling function:

- Those functions which do not return a value are of **void type.** For example: clrscr( ) function does not return a value.
- Those functions which return a value to its calling function are called **return type function**. A function may return int, float, char etc. value. For example, sqrt(9) function returns the square root of the given number.

Arguments to the function are optional. It means there may be zero or more arguments. These arguments pass information to the function. These arguments are used in the body of the function.

Thus, user defined functions can be defined in four different ways. These are:

1. void type without arguments
2. void type with arguments
3. function returning values without argument
4. function returning values with argument

FUNCTION PROTOTYPE/DECLARATION:

A function prototype is a declaration. It declares a function and its type. Whenever the function is called, it is checked with its declaration/prototype. Function is declared when a function is called before its definition.

FUNCTION CALL:

To use a function, it must be called. When we call function, we use functionname with actual arguments. Consider the following syntaxes:

```
functionname(values);          //used to call a void type function
var=functionname(values);      //used to call a return type function
```

Following program example shows the function declaration, definition and call

```
void sum(int a, int b); // function prototype or declaration
void main()
{
sum(5,7);   //function call
}
void sum(int a, int b) //function definition
{
int  s=a+b;
printf("%d",s);
}
```

(Actual Arguments: 5, 7 — Formal Arguments: a, b)

ACTUAL AND FORMAL ARGUMENTS:

Arguments to the function are optional. It means there may be zero or more arguments. These arguments pass information to the function. These arguments are used in the body of the function. These arguments are of two types:

- Formal arguments
- Actual arguments/Parameters

```
void sum(int a, int b); // function prototype or declaration
void main()
{
sum(5,7);   //function call
}
void sum(int a, int b) //function definition
{
int  s=a+b;
printf("%d",s);
}
```

(Actual Arguments: 5, 7 — Formal Arguments: a, b)

Formal Arguments:

These arguments are passed in the function definition. They are also called as dummy arguments.

Actual Parameters:

These arguments are passed during function-call. These arguments may be variables or values.

Following program shows these types of arguments:

SCOPE RULES:

A scope is an area of the program where a defined variable can be used. Beyond that area, variable cannot be accessed. Declaration place of a variable decides the scope of a variable. There are two types of scopes for variables in C: Local Scope and Global Scope

Local Scope:

Those variables which are declared in the body of the function are of local scope. They can be used only within the function. They cannot be used outside the function body. Formal arguments also have the local scope.

Global Scope:

Those variables which are declared outside all functions are of global scope. A global variable can be used by any function in the entire program.

Consider the following program. It shows the local and global variables.

```
#include <stdio.h>
int g;          // global variable declaration
void main ()
{
  int a;        //local variable declaration
  a = 10;
  g = a *a;
   printf ("%d", g);
 }
```

PASSING PARAMETERS TO FUNCTIONS:

Parameters can be passed to function in two ways - by value and by reference, which are called 'call by value' and 'call by reference' respectively.

DIFFERENTIATE BETWEEN CALL BY VALUE AND CALL BY REFERENCE:

Call By Value

1. The formal and actual parameters are two different variables.
2. The formal parameters must not be preceded by address operator (&).
3. When the value of the formal parameter is changed, the corresponding value of actual parameter is not changed automatically.
4. Actual Parameter is read only.
5. Actual Parameter may be a constant, a variable, or an expression.
6. int x;

```
void swap(int x, int y)  //function definition
{
   int temp;
   temp = x;
   x = y;
   y = temp;
}
void main( )
{
   int a=5, b=7;
   swap(a, b);              //function call
   printf("\n%d",a);
   printf("\n%d",b);
}
```

Call By Reference

1. The formal and actual parameters are same, though their names may be different.
2. The formal parameters must be preceded by address operator (&)
3. The value of formal parameter automatically changes when the value of actual parameter is changed.
4. Actual parameter is read – write.
5. Actual Parameters must be a variable.
6. int &x;

```
void  swap(int *x, int *y) //function definition
{
    int temp;
    temp=*x;
   *x=*y;
   *y=temp;
}
void main( )
{
   int a=5, b=7;
   swap(&a, &b);     //function call
   printf("\n%d",a);
   printf("\n%d",b);
}
```

RECURSION/RECURSIVE FUNCTIONS:

A function which calls itself again and again is known as recursive function. Recursive functions are very useful while working with data structures like linked lists, trees etc. Consider the following example. It calculates the factorial of the given number using recursive function.

```
long fact(long n);                  //function declaration
void main()
{
int x, n;
n=4;
   x = fact(n);                     //function call
printf("\n%d" , x);
}
long fact(long int n)               //recursive function definition
{
int f=1;
          if(n!=0)
            f = n * fact(n-1);
return f;
}
```

INPUT AND OUTPUT:

C provides many functions for standard input and output. These functions are defined in a header file. This header file is stdio.h (standard input output header file). To use these input/output functions, we have to include header file in our program:#include<stdio.h>

Here, #include is the preprocessor directive and <stdio.h> is the header file.

C supports two types of I/O functions:

- Formatted functions
- Un-formatted functions

FORMATTED FUNCTIONS:

The printf() and scanf() are the formatted functions. We can format the input and output using these functions. Using these functions we can input and output any type of data in the program. These functions use formatting codes for input and output. Commonly used format codes are given in the following table:

Format Code	Description of the format code
%d	It is used for int type
%ld	It is used for long type
%f	It is used for float type
%lf	It is used for double type
%c	It is used for char type
%s	It is used for string type

Table: Format codes

The printf() function:

The printf() function is used for outputting any type of data on Screen/Console. This function uses format codes, like %d, %f, %c, %s, etc. to format different data types. Use of format code depends on the data type of the variable to be displayed.

The general syntax for the printf ( ) function is:

printf("Message…….."); OR

printf ("Format codes", arg1, arg2, ……);

Following examples show how to use printf ( ) function using different Format strings –

Example 1: printf ("Vikas Kansal"); //displays Vikas Kansal

Example 2: int a=5;printf ("%d", a); //displays 5

To format output, we use the field-width and the format-flags in the printf() function. The general syntax for this is as below:

printf ("%<format flag><field width>format code", argument);

The commonly used format flags are:

Format Flag	Example	Description
–(minus sign)	"%–d"	It is used to force the value of variable to be left aligned. By default, value of variable to be displayed is aligned right.
0 (zero)	"%05d"	It is used with Numeric type variables, i.e. int& float. By using this flag, it is possible to pad the displayed value with zeros in the leading blanks.

Table – Commonly used Format Flags

Consider the Following Examples:

The scanf() function:

The scanf() function is used for inputting any type of data during runtime. This function uses format codes, like %d, %f, %c, %s, etc. to format different data types. Use of format code depends on the data type of the variable to be input.

The general format for scanf( ) function is:

scanf("Format codes",&arg1, &arg2, …….);

Example for scanf() function is:

```
int a, b;
float c;
scanf("%d %f %d", &a, &c, &b);
```

The ampersand '&' symbol before each variable name is an operator. It specifies the address of the variable in the main memory.

UNFORMATTED INPUT/OUTPUT FUNCTIONS

Unformatted I/O functions are used for string and character type data. Following table shows the various I/O unformatted functions used in C:

Unformatted I/O functions	Description
getch( ), getche( ), getchar( )	Used for single character input
putch( ), putchar( )	Used for single character output
gets( )	Used for string input
puts( )	Used for string output

Table – Unformatted Input/Output Functions in C

CHARACTER I/O FUNCTIONS:

Character input and output functions are defined in the conio.h (console input output) header file. So, to use these functions, we include conio.h file in the program.

The getch( ) function: The getch ( ) function accepts a single character from the keyboard. But it does not display it on the screen. After typing the character, there is no need to press the 'Enter' key. Its syntax is:

char_variable=getch( );

The getche( ) function: This function is also used to read a single character from the keyboard at the console. It displays the character entered by the user at the console. The character 'e' in the getch**e** ( ) function means 'echo' (displays). Its syntax is:

char_variable = getche ( );

The getchar ( ) function: This function is also used to read a single character from the keyboard at the console. This function accepts the character data and waits for the 'Enter' key.

char_variable = getchar ( );

Following table shows the comparisons between the getch ( ), getche ( ) and getchar ( ) functions:

Name of the Function	Echo Character	'Enter'Key
getch ( )	No	No
getche ( )	Yes	No
getchar ( )	Yes	Yes

The putch ( ) and putchar ( ) functions are used for single character output. The syntax for using these functions is as follows:

putch (char_variable); putchar (char_variable);

Consider the following program:

```
void main()
{
char ch;
printf("\nEnter any character ");
ch=getchar();
printf("\nEntered character is ");
putchar(ch);
}
```

STRING INPUT/OUTPUT FUNCTIONS

The gets() and puts() are the unformatted input/output functions. These functions are used for the string type data. These functions are defined in the stdio.h file. So, we have to include this header file. The gets( ) function is used for unformatted string input. Using this function, we can input a string including spaces. The syntax for using this function is as follows:

gets(string_variable);

The puts ( ) function is used to display the unformatted string at the console. The syntax for using this function is as follows: puts (string_variable);

Following program shows the use of these functions:

```
void main()
{
char ch[20];
printf("\nEnter your name");
gets(ch);
printf("\nHello Mr./Miss/Mrs. ");
puts(ch);
}
```

Section B

STORAGE CLASSES:

Storage classes are used to define the scope, life and location of a variable in the program. In C, four storage classes are used to define scope, life and location of variable. These are:

1. Automatic Variables
2. Register Variables
3. Static Variables
4. External Variables

These classes are used during the declaration time of variable. We can say that storage classes define how the memory reference is carried out for a variable.

Automatic Variables:

These variables are also called Internal or local variables. The keyword **auto** is used to define this storage class. If we do not define any storage class, the default is auto.

Scope: Its scope is limited. They can be used only within the function in which they are declared.

Life: Their life time is small. They remain in the memory till the function execution.

Location: These variables are stored in the main memory (RAM).

Static Variables:

Static variables are defined within a function. The keyword **static** is used to define this storage class.

Scope: Its scope is limited. They can be used only within the function in which they are declared.

Life: Their life time is long. They remain in the memory till the program execution.

Location: These variables are stored in the main memory (RAM).

Register Variables:

Automatic variables are stored in the main memory (RAM). Accessing a memory location takes time. So, to speed up the processing, some variables can also be stored in the registers of CPU. Those variables which are used frequently in the program, they can be stored in the registers. The keyword **register** is used to define this storage class..

Scope: Its scope is limited. They can be used only within the function in which they are declared.

Life: Their life time is small. They remain in the memory till the function execution.

Location: These variables are stored in the registers of the CPU.

External Variable:

These variables are declared outside any function. They can be used anywhere throughout the program. The keyword **extern** is used to define this storage class.

Following programs show how to use these storage classes with their output:

Example for auto:

```
void count()
{
auto int a=0;
a++;
printf("%d",a);
}
void main()
{
count();
count();
}
```

Output

```
1
```

Example for static:

```
void count()
{
static int a=0;
a++;
printf("%d",a);
}
void main()
{
count();
count();
}
```

Output

```
1
```

Example for register:

```
void count()
{
register int a=0;
a++;
printf("%d",a);
}
void main()
{
count();
count();
}
```

Output

```
1
```

Example for extern:

```
void main()

{

extern int a;

printf("%d",a);

}

int a=5;
```

Output

```
5
```

ARRAY:

An array is a homogeneous collection of values. It means an array can store more than one value of same data type at a time. It is an ordered and finite set of elements. Memory allocation to array is static. We cannot change its length during runtime. A contiguous block of memory is allotted to array. They are of two types;

1. One Dimensional Array
2. Multidimensional Array

ONE DIM ARRAY:

Those arrays which require one index/subscript to access its value is called one-dim array. These arrays are also called **Vectors or Linear Array**. These arrays have one dimension only –a single row or a single column. Smallest index of array is known as Lower Bound (LB). In C, it is always 0. Largest index of array is known as Upper Bound (UB). It is always equal to **length** – 1. Consider the following figure of one-dimensional array.

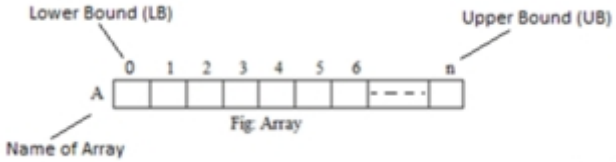


Fig. Array

Declaration of One Dim Array:

Before using arrays, we have to declare them. Following syntax is used to declare one dim array:

```
datatype arrayname[length];
```

For example:

```
int a[10];
```

Here, int is the data type of the array, a is the name of array, and 10 is the length of array. This array can store 10 integer values. Length of array must be constant.

One Dim Array Initialization:

Like the variables, arrays can also be initialized. It has the following syntax.

```
Datatype arrayname[length]={val1, val2, val3,…….};
```

For example:

```
int a[5]={5, 4, 8, 19, 10};
```

READING VALUES INTO ONE DIM ARRAY:

We can store values into one dimensional array during design time and run time. To store values during design time, commonly we initialize the array. Storing values into one dimensional array during runtime is called reading values into array. For this, we use the loop to input all values of array. In the loop, we use input functions to read the values one by one. Consider the following example:

```
int a[5];
for(i=0;i<5;i++)
{
   scanf("%d", &a[i]);
}
```

In the above code, we use scanf() function to read values of one dimensional array one by one using the for loop.

DISPLAYING ONE DIM ARRAY CONTENTS:

To display all the contents of array, we use loops. Output functions are used to display the contents in a loop. Consider the following example:

```
for(i=0;i<5;i++)
{
   printf("%d", a[i]);
}
```

In the above code, we use printf() function to display the contents of array one by one using the for loop.

Following program shows how to work with the one dim array.

```
//Program to sort the one dimensional array values
void main()
{
int a[5]={45,34,18,7,90};   //array initialization
int i, j, temp;
        //nested loop to sort the one dimensional array
        for( i = 0; i < 4 ; i++ )
        {
                for( j = 0; j < 4 ; j++ )
                {
                        //if current value is greater than next value then swap
                        if(a[ j ] > a [ j + 1] )
                        {
                        temp = a[j];
                        a[j] = a[j+1];
                        a[j+1] = temp;
                        }
                }
        }
        //loop to display sorted array values
        printf("\nSorted array is");
        for(i =0;i<5;i++)
        {
                printf("\n%d", a[i] );
        }
}
```

MULTIDIMENSIONAL ARRAY:

Those arrays which require more than one index/subscript to access its valueare called multi-dimensional array. They can be two-dimensional or three-dimensional. In two-dim arrays, we use two index values and in three dim arrays we use three subscripts.

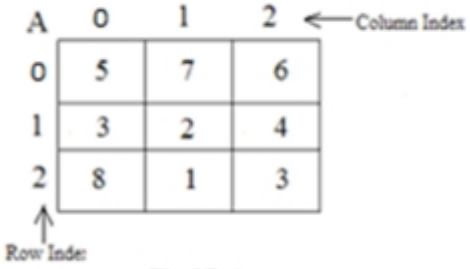


Fig: 2D Array

Fig: 3D Array

TWO DIMENSIONAL ARRAYS:

Those arrays which require two index/subscripts to access its value are called two-dimensional array. They are also called **matrix or tabular array**. These arrays have two dimensions – rows and columns. Row index and column index begins from 0. Before using arrays, we have to declare them in C.

Two Dim Array Declaration:

Following syntax is used to declare one dimensional array:

Datatype arrayname[**rows**] [**columns**];

For example:

int a[3][2];

Here, int is the data type of the array, a - is the name of the array, 3 is the numbers of rows in the array, 2 is the number of columns in the array. This array can store 6 values.

Two Dim Array Initialization:

Like the variables, arrays can also be initialized. It has the following syntax.

Datatype arrayname[rows] [columns]={ {val11, val12, …. }, {val21, val22,……}, …. };

For example:

int a[3][2]={ {5,2},{6,4}, {8,9} };

READING VALUES INTO 2D ARRAY:

We can store values into array during design time and run time. To store values during design time, commonly we initialize the array. Storing values into array during runtime is called reading values into array. For this, we use nested loop to input all values of 2D array. In the loop, we use input functions to read the values one by one. Consider the following example:

```
int a[5];
for(i=0;i<5;i++)
{
        for(i=0;i<5;i++)
        {
        scanf("%d", &a[i]);
        }
}
```

In the above code, we use scanf() function to read values of 2D array one by one using the nested for loop.

DISPLAYING 2D ARRAY CONTENTS:

To display all the contents of 2D array, we use nested loops. Output functions are used to display the contents in a nested loop. Consider the following example:

```
for(i=0;i<5;i++)
{
        for(i=0;i<5;i++)
        {
        printf("\t%d", a[i]);
        }
printf("\n");
}
```

In the above code, we use printf() function to display the contents of 2D array one by one using the nested for loop. It shows the contents in the matrix form.

Following program shows how to use 2D array:

```
//Program to multiply two 2D Arrays/Matrix
void main()
{
        int a[2][2]={{2,1}, {3,4}};   //array initialization
        int b[2][2]={{2,1}, {4,3}};   //array initialization
        int c[2][2]={{0,0}, {0,0}}, i, j, k;
        //nested loops to multiply two matrices
        for (i=0;i <2;i++)  {
                for (j=0;j<2;j++)   {
                        for(k=0;k<2;k++)
                        c[i][j]=c[i][j]+a[i][k]*b[k][j];
                }
        }
        //loop to show 2D array in matrix form
        printf("Multiplication of matrix: ");
        for (i=0;i<=2;i++)  {
                for (j=0;j<=2;j++)  {
                printf("%d\t",c[i][j]);
                }
                printf("\n");
        }
}
```

STRING:

It is a combination of one or more characters. It is enclosed in double quotes. They are terminated by null character (\0) in c. To store string in c, we use char array. In C, char arrays are considered as a whole during input output. It means we need not to use loops for input output a string. We can perform many operations on string using many built-in functions. These functions are stored in the **string.h** header file. Functions strcpy( ), strrev( ), strcat( ), strcmp( ), strlwr( ), strupr( ), strlen( ) are commonly used on strings.

String declaration:

For strings, we use char arrays in c. So to store string, we declare character array. its syntax is:

```
char string_variable[length];
```

For Example:

```
char str[20];
```

Here, string variable str can hold the maximum of 20 characters in it.

String initialization:

Like other arrays, strings can also be initialized. It can be initialized in many ways. Consider the following examples:

```
char name[10]= "Kansal";
char name[10]={'K', 'a', 'n',' s', 'a', 'l'};
char name[10]={'K', 'a', 'n', 's', 'a', 'l', '\0'};
char name[ ]= "Kansal";
char name[ ]={'K', 'a', 'n', 's', 'a', 'l'};
```

We can use any one method to initialize a string or char array.

STRING I/O:

For string input output, we can use the **scanf() and printf()** functions respectively. In these, functions, we use %s format code for string input and output. Consider the following examples:

```
char name[20]";
scanf("%s",name);     //for string input
printf ("%s", name);  //for string output
```

When we use scanf() for string input, we do not use & operator. But this function can store only a single word string. We cannot store multiple word string using scanf( ) function. For this, we use **gets( ) function**. This function is capable to store multiple word string in C. Consider the following example:

```
gets(name);
```

Here if we store "New Delhi", then it can store this value. But if we use scanf( ) function, then it will store only the word "New" in the string variable.

Similarly, we can also use **puts( ) function** to display strings in C.

STRING HANDLING FUNCTIONS

There are many predefined functions which are used to manipulate strings, such as strlen( ), strupr( ), strlwr( ), strcmp( ), strcpy( )etc. These functions are defined in the string.h header file. So we must include this file in our program to use string handling functions. Some commonly used string functions are:

- strlen( ) – string length function
- strcpy( ) – string copy function
- strcmp( ) – string compare function
- strrev( ) – string reverse function
- strcat( ) – string concatenation function
- strupr( ) – string upper function
- strlwr( ) – string lower function

The strlen( ) function:

Function strlen stands for string length. This function is used to count the total number of characters in the given string. Following code shows how to use this function:

```
strlen("punjab");      //it returns 6 as string length
```

The strcpy( ) function:

Function strcpy stands for string copy. This function is used to copy the one string value into string variable. Following code shows how to use this function.

strcpy(str, "Punjab");

It copy the string "Punjab" into the string variable str.

The strrev( ) function:

Function strrev stands for string reverse. This function is used to reverse the given string value. Following code shows how to use this function:

strrev("Punjab");

It reverses the given string to **bajnuP**.

The strcat( ) function:

Function strrev stands for string concatenation. This function is used to combines the two strings. It appends 2nd string at the end of first string. Following code shows how to use this function. Let the two strings are:

```
char str1[10]="Sunam";
char str2[10]="City";
strcat(str2, str1);
```

It concatenates the second string into first string - **CitySunam**

The strupr( ) function:

Function strupr stands for string upper. This function is used to convert the given string into Upper case (i.e. in capital letter). Following code shows how to use this function:

strupr("Punjab");

It converts the given string into **PUNJAB.**

The strlwr( ) function:

Function strlwr stands for string lower. This function is used to convert the given string into lower case (i.e. in small letter). Following code shows how to use this function:

strlwr("PUNJAB");

It converts the given string into punjab.

The strcmp( ) function:

Function strcmp stands for string compare. The strcmp( ) function is used to compare two strings. It returns one of the three possible values: zero, negative, or positive. For example:

If 1st string is equal to 2nd string, It returns 0

e.g. strcmp("punjab","punjab") returns 0

If 1st string is less than 2nd string, It returns negative value

e.g. strcmp("india","punjab") return –ve value

If 1st string is greater than 2nd string, It returns +ve value

e.g. strcmp("punjab","india") return +ve value

Following program shows how to use strings in the program. This Program checks whether the given string is palindrome or not.

```
#include<string.h>
void main()
{
        char str1[10]="madam";          //Initializing string variable
        char str2[10];                  //declaring string
        strcpy(str2,str1);              //copying str1 into str2
        strrev(str2);                   //reverses string str2
        if (strcmp(str1,str2)==0)       //compare if two strings are same or not
        printf("string is palindrome");
        else
        printf("string is not palindrome");
}
```

TABLE OF STRINGS:

Table of string is also called 2D character array or one dimensional string array. It is used to store more than one string value. For this, we have to declare the two dimensional character arrays. Each row of 2D char array is used to store a string value. Consider the following diagram to show the concept of table of strings.

D	e	l	h	I	\0	
P	u	n	j	a	b	\0
G	o	a	\0			
B	i	h	a	r	\0	

Fig: Table of strings

String Array/2D character array Declaration:

Table of strings is an array of strings. To create the table of strings, we have to declare the 2D array of characters as follows:

char states[4][7];

This array can store four strings. We can store the name of four states in this array. Each string can be of seven characters.

String Initialization:

Like other arrays, we can also initialize the string array. Consider the following example:

char states[4][7]={"Delhi", "Punjab", "Goa", "Bihar"};

Accessing Table of strings:

To access the table of strings, we have to use a single loop. Consider the following example which shows all the strings:

```
for(i=0;i<4;i++)
{
printf("\n%s", states[i]);
}
```

STRUCTURE:

Structure is another derived data types in C. Unlike array, a structure can hold data items of different types under a common name. Thus, we can define structure as:

"**Structure is a collection of heterogeneous data items**"

In many computer languages, like Pascal, this type of structure is referred as Record. Structures are used to organize complex data in a more meaningful way.

HOW TO DECLARE/DEFINE A STRUCTURE

To define a structure, we have to use the 'struct' keyword. The syntax for defining a structure is as follows:

```
struct structure-name {
        Data-type     data_item1;
        Data-type     data_item2
                .
        Data-type     data_item_n;
};
```

To access or use a structure, we have to declare its variables. Using these variables, we can access the data items of the structure with the help of dot/period '**.**' operator. This operator is also termed as **'member selection operator'**.

```
#include<stdio.h>
struct student              //Defining structure
{
    introll_no;             //Data Items of strucure
    char name[20];
    float percentage;
};
void main()
{
   struct student s1;       //Creating varibale of strucure
   printf("Enter roll no, name and percentage marks ");
   scanf("%d%s%f",&s1.roll_no,s1.name,&s1.percentage);   //Accessing strucure members
   printf("\nRoll No %d Name %s obtained %4.2f \% marks",s1.roll_no,s1.name,s1.percentage);
}
```

UNION

Like a structure, it is also a logical grouping of heterogeneous data items. The main difference between them is that in union, only one data member is active in memory at a time while in case of structure all the data members are active in memory at a time. To define a union in C, following syntax is used:

```
union union_name
{
        data-type dataitem1;
        data-type dataitem2;
        -
        -
        Data-type dataitem_n;
};
```

Here, 'union' is a keyword used to define a union, union_name is a valid identifier representing the name of union and data-types are the types of data items of a union.

```
union student {          //defining union
int roll;
char name[20];
};
void main() {
union student s1;        //creating variable of structure
printf("\nEnter roll number ");
scanf("%d",&s1.roll);    //accessing union members
printf("\nRoll no of the student is %d",s1.roll);
printf("\nEnter name of the student ");
scanf("%s",s1.name);
printf("\nName of the student is %s",s1.name);
}
```

COMPARISION BETWEEN ARRAYS AND STRUCTURES IN C

Both the arrays and structures are structured data types. But they differ in many ways. The differences between them are given below:

Arrays	Structures
1. An array is a collection of related data elements of same type.	1. Structure can have elements of different types
2. An array is a derived data type	2. A structure is a programmer-defined data type
3. Any array behaves like a built-in data types. All we have to do is to declare an array variable and use it.	3. But in the case of structure, first we have to design and declare a data structure before the variable of that type are declared and used.
4. Array allocates static memory	4. Structures allocate dynamic memory
5. Array uses index / subscript for accessing elements of the array	5. Structures use (.) operator for accessing the member of a structure

COMPARISION BETWEEN STRUCTURES AND UNIONS IN C

Structure	Union
1.The keyword struct is used to define a structure	1. The keyword union is used to define a union.
2. The size of structure is equal to the sum of sizes of its members.	2. The size of union is equal to the size of largest member.
3. Each member within a structure is assigned unique storage area of location.	3. Memory allocated is shared by individual members of union.
4 Altering the value of a member will not affect other members of the structure.	4. Altering the value of any of the member will alter other member values.
5. Individual member can be accessed at a time	5. Only one member can be accessed at a time.
6. Several members of a structure can initialize at once.	6. Only the first member of a union can be initialized.

7. All data members of a structure remain active in main memory at a time during program execution.	7. Only one data member is active at a time in main memory during program execution
8. It is used to declare a compounded data type. For example: it groups data members related to a person or an item, such as student, employee etc.	8. It is useful in cases where user selects any one data member only for the application, e.g. for preparing a list of student's names only.

POINTER

'Pointer' is the one of the most powerful feature of C & C++. Pointers are like the variables, which can hold the memory address of another variable. This memory address can be used to access the values of variables. Thus, pointers can access values directly from RAM. Consider the following example to understand the pointers:

Let a variable of integer type and assume that it is represented in the main memory as shown in the diagram below:

```
int a;
a = 25;
```

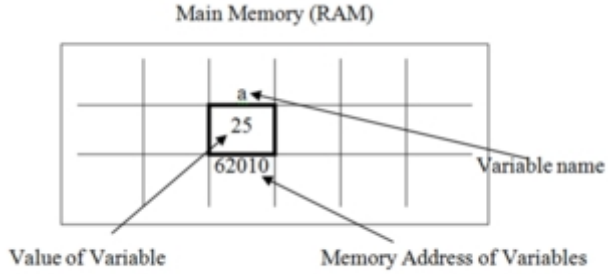


Fig: Representation of Variable in RAM with its Address Value

In the above diagram, 'a' is an integer type variable, which holds a value '25'. Let, the memory address for this variable is 62010. Thus, we can say that the value '25' can be accessed either by the variable's name 'a' or its memory address '62010'.

The pointer variables can be used to store the address of these variables. Consider the following example:

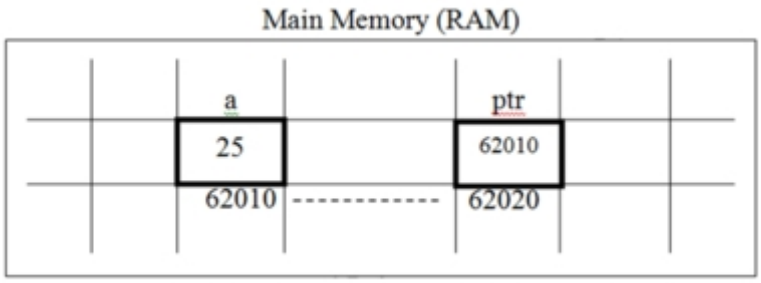


Fig: Representation of a Variable and a pointer variable in RAM

In the above diagram, ptr is a pointer variable. It holds the address of variable 'a'.

POINTER VARIABLE DECLARATION

The general syntax for declaring pointer variable is:

```
datatype *ptr;
```

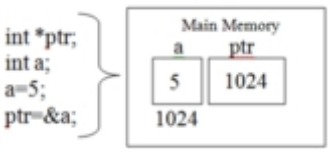
In pointer declaration, '*' sign is used to declare pointer variable. A pointer can point to the variable having same data type as that of a pointer variable. Consider the following example:

```
int * ptr;
```

The above example declares a pointer variable 'ptr' of int type. So, this pointer variable can point to only int type variables.

POINTER VARIABLE INITIALIZATION

To initialize pointer variable, we have to store the memory address of a variable in it. To do so, we use assignment operator. For storing the memory address of variable, we use the address operator (&) as shown below:



In the above example, address of variable of 'a' (1024) is stored in the pointer variable 'ptr'.

ACCESSING VALUES USING POINTERS OR POINTER OPERATORS:

To access values using pointers, we use pointer operators. Pointer operators are: indirection operator and address operator. Indirection operator is * and address operators is &. Indirection operator is used to access the value of a variable using pointer. Symbols & is used to get the address value of the variable. Consider the following example:

```
int *ptr, a=5;
ptr=&a;
printf("%d",*ptr);          // Accessing Values Using Pointers
```

In the above example, & operator is used to store the address of the variable. The indirection operator us used to show the value of variable using pointer variable.

POINTERS AND ARRAY:

Like variables, arrays can also be used using pointers. To access array through pointer, we have to store the memory address of first location of array. Consider the following example:

```
int a[5]={5,3,8,9,6};
int *ptr;
ptr=&a[0];    //stores the memory address of first location of array
or
ptr=a;        // it also stores the memory address of first location of array
```

After storing the memory address of array, we have to use the loop to access all the array values using pointer. Consider the following code:

```
for(i=0;i<5;i++)
{
        printf("\n%d",*(ptr+i));
}
```

This loop shows all the values of the loop through pointer.

FILES:

Files are used to store data permanently. For this, FILE data structure is used in c. For this data structure stdio.h header file must be included in the program. To open a file, we use fopen() function. Similarly, to close a file, we use fclose() function. Consider the following block of code. This code shows how to open and close file using file pointer.

```
#include<stdio.h>
void main()
{
    FILE *fptr;
    fptr=fopen("abc","r");
    -
    -
    fclose(fptr);
}
```

OPENING FILES

To actually open a file, we call 'fopen ( )' function. The 'fopen ( )' function accepts a **file name** (as a string) and a **mode value** indicates whether we want to open the file for reading or writing purpose. The mode variable is also a string. The fopen( ) function returns a pointer if the file can be opened. If the file cannot be opened, a NULL value is returned.

To open the file 'read.txt' for reading we might call the fopen ( ) functions as:

```
fptr = fopen ("read.txt", "r");
```

The mode string "**r**" indicates reading. Mode "**w**" indicates writing, so we could open 'write.txt' for output like this:

```
fptr = fopen("write.txt", "w");
```

The other values for the mode string are less frequently used. The third major mode is "**a**" for append. If we use "w" to write to a file, which already exists, its old contents will be discarded. Following table shows all the possible modes of opening a file:

Mode Value	Description of opening mode
"r"	Open an existing file for reading purpose only.
"w"	Open a new file for writing only. If a file with the specified filename is already exists, it will be overwritten.
"a"	Open an existing file for appending. Appending means adding new information at the end of the file.
"r+"	Open an existing file for both reading and writing purpose.
"w+"	Open a new file for both reading and writing.
"a+"	Open an existing file for both reading and appending.
"rb"	opens an existing binary file for reading
"wb"	Creates a binary file for writing.
"ab"	Opens an existing binary file for appending.
"r+b" or "rb+"	Opens an existing binary file for reading or writing.
"w+b" or "wb+"	Creates a binary file for reading or writing.
"a+b" or "ab+"	Opens or creates a binary file for appending.

Table - List of file - opening modes in C

CLOSING FILES

Although we can open multiple files, but there is a limit to how many files we can have open at once. If we open the files beyond the limit, the standard I/O library could run out of the resources it uses to keep track of open files. Closing a file simply involves calling fclose( ) function with the file pointer as its argument:

fclose (fptr);

Types of Files

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

1. Text files
2. Binary files

1. 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.

2. 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.
C provides a number of functions that helps to perform basic file operations. Following are the functions,

Sequential File Access in C

The simplest way that C programming information is stored in a file is *sequentially,* one byte after the other. The file contains one long stream of data.
File access in C is simply another form of I/O.

Function	description
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file

fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

Random Access To File

There is no need to read each record sequentially, if we want to access a particular record. C supports these functions for random access file processing.

fseek(), ftell() and rewind() functions

- **fseek()** - It is used to move the reading control to different positions using fseek function.
- **ftell()** - It tells the byte location of current position of cursor in file pointer.
- **rewind()** - It moves the control to beginning of the file.