ARULMIGU PALANIANDAVR ARTS COLLEGE FOR WOMEN
(Autonomous)
(Re-Accredited with 'B++' Grade by NAAC 3rd Cycle)
Run by Arulmigu Dhandayuthapani Swamy Thirukoil, H.R & C.E Dept. Government of Tamil Nadu
A Government Aided College - Affiliated to Mother Teresa Women's University, Kodaikanal
CHINNAKALAYAMPUTHUR(PO), PALANI - 624615

DEPARTMENT OF COMPUTER SCIENCE

AND

COMPUTER APPLICATION

LEARNING RESOURCE

PROGRAMMING IN "C"

Introduction to C

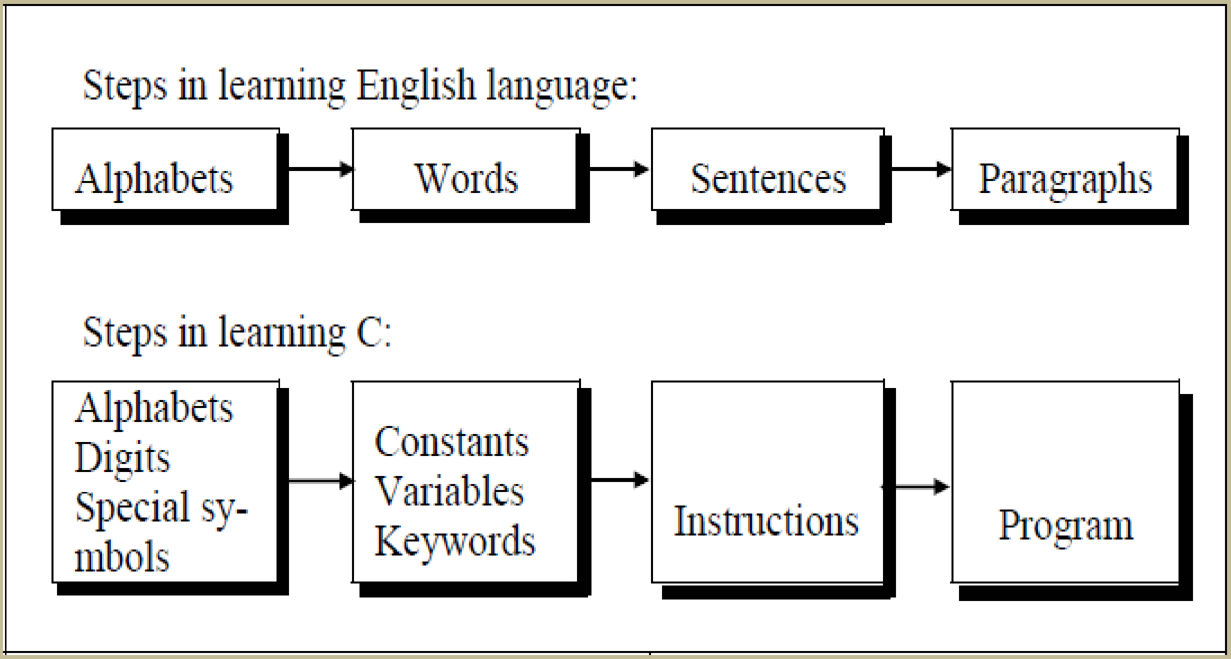
- C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie. In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL, etc

- ANSI C standard emerged in the early 1980s, this book was split into two titles: The original was still called *Programming in C*, and the title that covered ANSI C was called *Programming in ANSI C*.

- **Program**

  - A computer ***program*** is just a collection of the instructions necessary to solve a specific problem.

  - The basic operations of a computer system form what is known as the computer's ***instruction set.*** And the approach or method that is used to solve the problem is known as an ***algorithm***


Steps in learning English language:
Alphabets
Words
Sentences
Paragraphs
Steps in learning C:
Alphabets Digits Special sy-mbols
Constants Variables Keywords
Instructions
Program

Types of program

So for as programming language concern these are of two types.

- Low level language
- High level language

Low level language:

Low level languages are **machine level** and **assembly level language**. In machine level language computer only understand digital numbers i.e. in the form of 0 and 1

High level language:

These languages are machine independent, means it is portable. The language in this category is Pascal, Cobol, Fortran etc. High level languages are understood by the machine. So it need to translate by the translator into machine level

Structure of C Language program

1) Comment line

2) Preprocessor directive

3) Global variable declaration

4)main function( )

{

  Local variables;

 Statements;

 }

User defined function

}

}

Comment line

It indicates the purpose of the program. It is represented as

/*……………………………..*/

Comment line is used for increasing the readability of the program. It is useful in explaining the program and generally used for documentation. It is enclosed within the decimeters. Comment line can be single or multiple line but should not be nested. It can be anywhere in the program except inside string constant & character constant.

- **Preprocessor Directive:** #include<stdio.h> tells the compiler to include information about the standard input/output library. It is also used in symbolic constant such as #define PI

3.14(value). The stdio.h (standard input output header file) contains definition &declaration of system defined function such as printf( ), scanf( ), pow( ) etc. Generally printf() function used to display and scanf() function used to read value

- **Global Declaration:**

  This is the section where variable are declared globally so that it can be access by all the functions used in the program. And it is generally declared outside the function.

- **main()**

It is the user defined function and every function has one main() function from where actually program is started and it is encloses within the pair of curly braces.

The main( ) function can be anywhere in the program but in general practice it is placed in the first position.

Syntax : main()

{

……..

……..

……..

}

The main( ) function return value when it declared by data type as int main( )

{

return 0

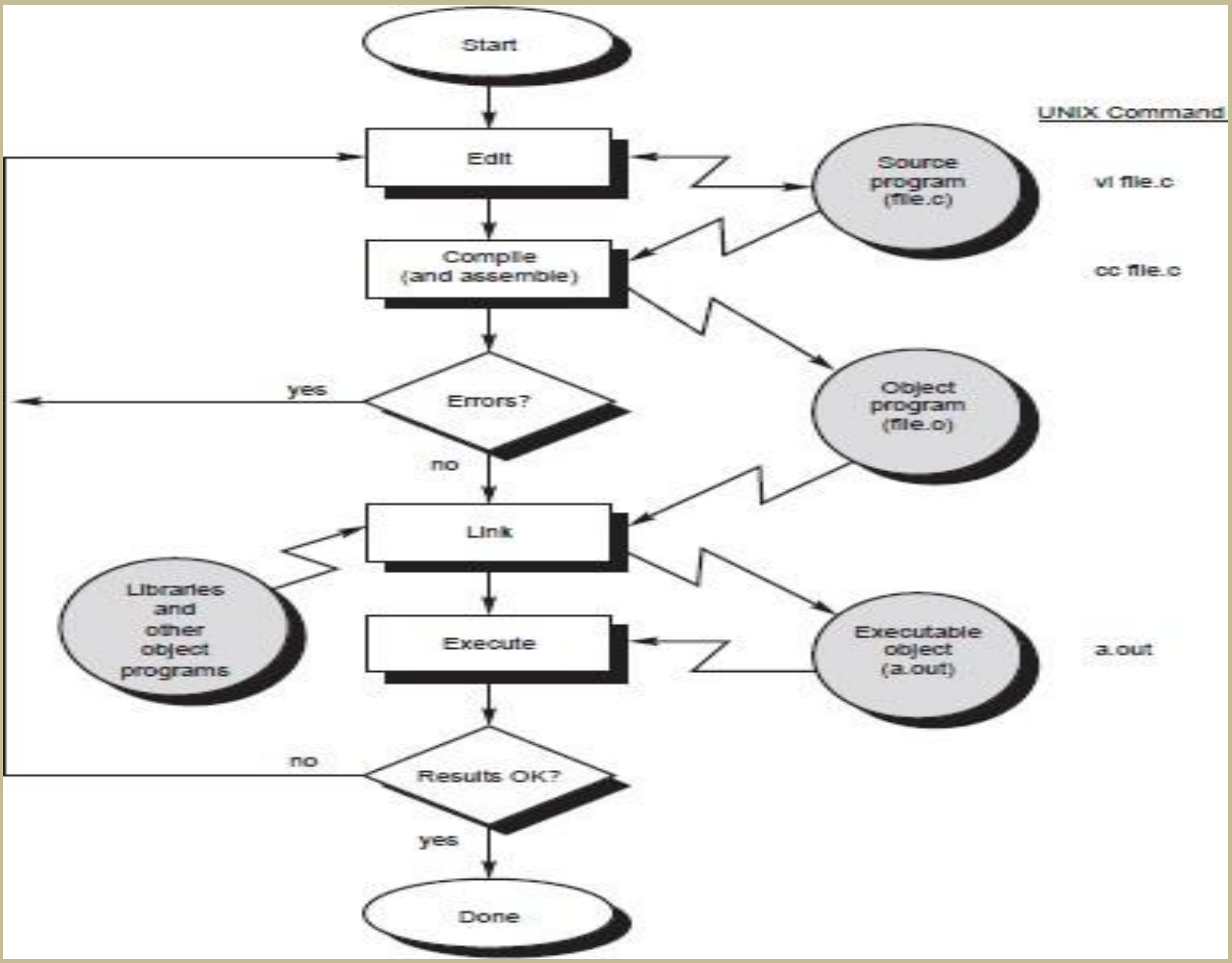
}

The main function does not return any value when void (means null/empty) as void main(void) or void main()

```
{
printf ("C language");
}
```

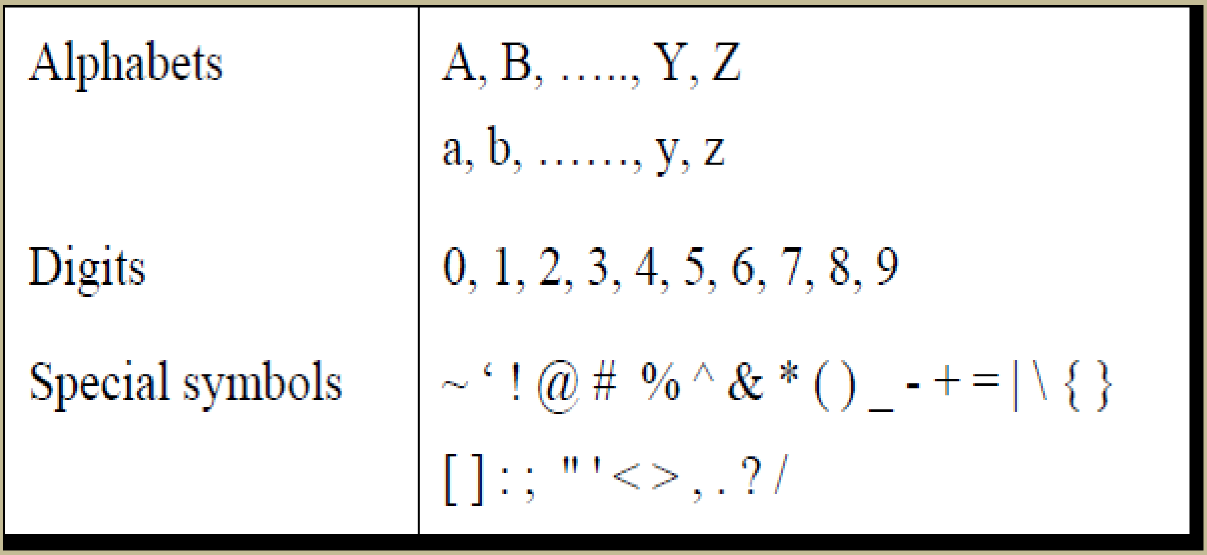
Output: C language

Steps for Compiling and executing the Programs



Character set

A character denotes any alphabet, digit or special symbol used to represent information. Valid alphabets, numbers and special symbols allowed in C are



Alphabets	A, B, ….., Y, Z a, b, ……, y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

Identifiers

- name should only consists of alphabets (both upper and lower case), digits and underscore (_) sign.
- first characters should be alphabet or underscore
- name should not be a keyword
- since C is a case sensitive, the upper case and lower case considered differently, for example code, Code, CODE etc. are different identifiers.

identifiers are generally given in some meaningful name such as value, net salary, age, data etc.

- **Keyword**

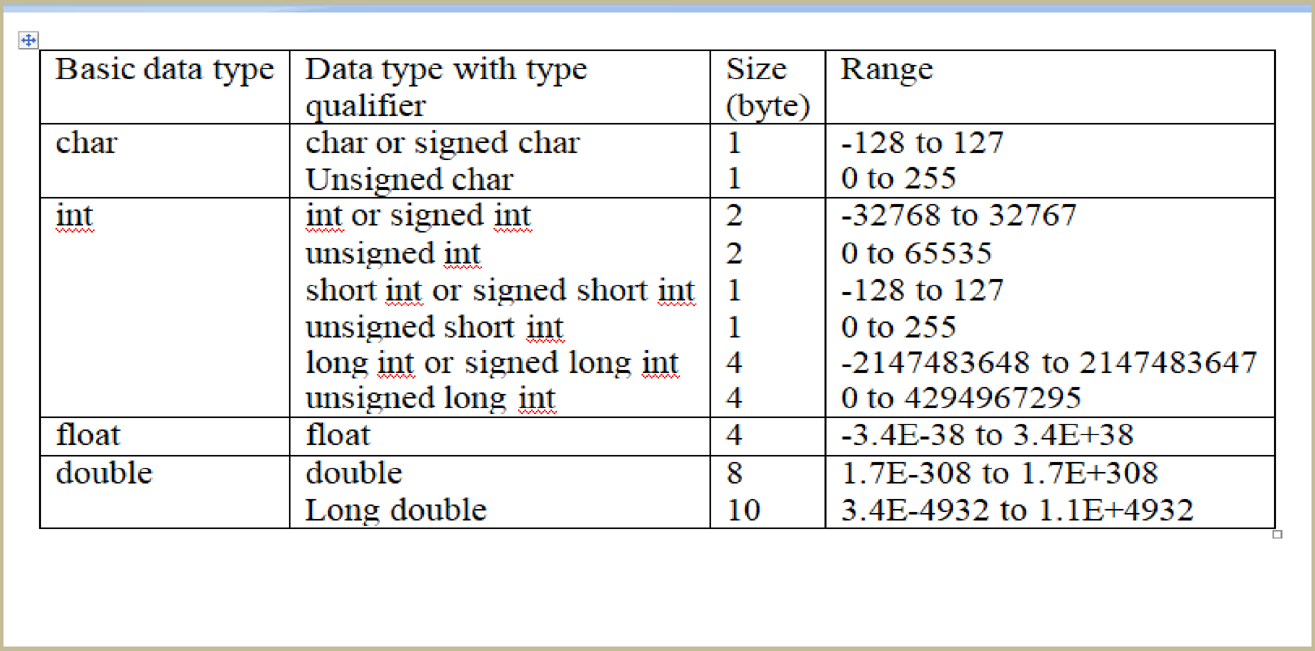
There are certain words reserved for doing specific task, these words are known as **reserved word** or **keywords**

- Some examples are **int, short, signed, unsigned, default, volatile, float, long, double, break, continue, typedef, static, do, for, union, return, while, do, extern, register, enum, case, goto, struct, char, auto, const etc.**

DATA TYPES

- **basic built-in data types: int, float, double, char**
- **Enumeration data type: enum**
- **Derived data type: pointer, array, structure, union**
- **Void data type: void**

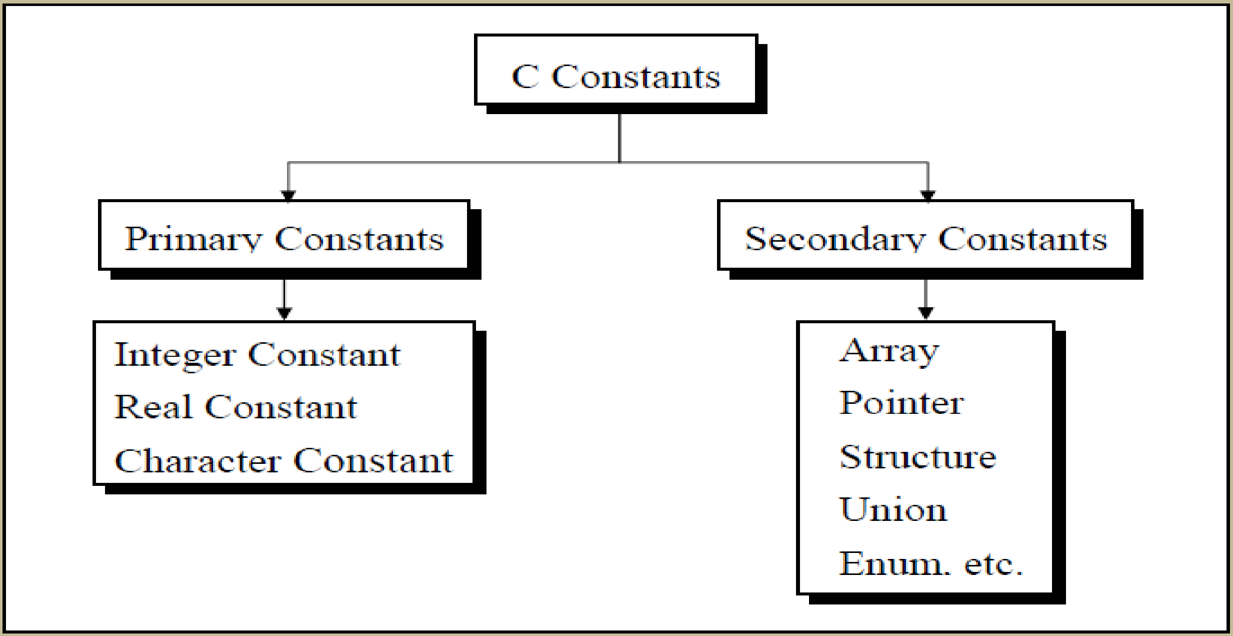
Basic data type



Basic data type	Data type with type qualifier	Size (byte)	Range
char	char or signed char Unsigned char	1 1	-128 to 127 0 to 255
int	int or signed int unsigned int short int or signed short int unsigned short int long int or signed long int unsigned long int	2 2 1 1 4 4	-32768 to 32767 0 to 65535 -128 to 127 0 to 255 -2147483648 to 2147483647 0 to 4294967295
float	float	4	-3.4E-38 to 3.4E+38
double	double Long double	8 10	1.7E-308 to 1.7E+308 3.4E-4932 to 1.1E+4932

Constants

- Constant is an any value that cannot be changed during program execution. In C, any number, single character, or character string is known as a *constant*
- C constants can be divided into two major categories**:**
- **Primary Constants**
- **Secondary Constants**



Variables

- Variable is a data name which is used to store some data value or symbolic names for storing program
- computations and results. The value of the variable can be change during the execution. The rule for naming the variables is same as the naming identifier. Before used in the program it must be declared. Declaration of variables specify its

name, data types and range of the value that variables can store depends upon its data types.

- Syntax: int a; char c; float f;

Variable initialization

- When we assign any initial value to variable during the declaration, is called initialization of variables. When variable is declared but contain undefined value then it is called garbage value. The variable is initialized with the assignment operator such as
- Data type variable name=constant;

Example:

```
 int a=20;
Or
 int a;
 a=20;
 statements
```

Expressions

- An expression is a combination of variables, constants, operators and function call. It can be arithmetic, logical and relational for
- example:-

```
int z= x+y      // arithmatic expression
```

a>b   //relational

a==b   // logical fun

c(a, b)   // function call

- Expressions consisting entirely of constant values are called *constant expressions*. So, the expression
- 121 + 17 - 110
- is a constant expression because each of the terms of the expression is a constant value. But if i were declared to be an integer variable, the expression
- 180 + 2 – j
- would not represent a constant expression.

Operator

- This is a symbol use to perform some operation on variables, operands or with the constant. Some operator required 2 operand to perform operation or Some required single operation.
- Several operators are there those are, arithmetic operator, assignment, increment , decrement, logical, conditional, comma, size of , bitwise and others.

Arithmatic Operator

This operator used for numeric calculation. These are of either Unary arithmetic operator, Binary arithmetic operator.

- Binary arithmetic operator on other hand required two operand and its operators are +(addition), -(subtraction),
- *(multiplication), /(division), %(modulus).

Assignment Operator

- A value can be stored in a variable with the use of assignment operator. The assignment operator(=) is used in assignment statement and assignment expression
- int x= y;
- int Sum=x+y+z;

Increment and Decrement

- The Unary operator ++, --, is used as increment and decrement which acts upon single operand. Increment operator increases the value of variable by one
- .Similarly decrement operator decrease the value of the variable by one. And these operator can only used with the variable, but cann't use with expression and constant as ++6 or ++(x+y+z).

Relational Operator

- It is use to compared value of two expressions depending on their relation. Expression that contain relational operator is called relational expression.
- Here the value is assign according to true or false value. a.(a>=b) || (b>20)
- b.(b>a) && (e>b)
- c. 0(b!=7)

Conditional Operator

It sometimes called as ternary operator. Since it required three expressions as operand and it is represented as (? , :).

- SYNTAX
- exp1 ? exp2 :exp3

- Here exp1 is first evaluated. It is true then value return will be exp2 . If false then exp3.

  EXAMPLE

  ```
  void main()

  {

  int a=10, b=2

  int s= (a>b) ? a:b; printf("value is:%d");

  }
  ```

  Output:

  ```
  Value is:10
  ```

Comma Operator

- Comma operator is use to permit different expression to be appear in a situation where only one expression would be used. All the expression are separator by comma and are evaluated from left to right.
- EXAMPLE
- `int i, j, k, l; for(i=1,j=2;i<=5;j<=10;i++;j++)`

Sizeof Operator

- Size of operator is a Unary operator, which gives size of operand in terms of byte that occupied in the memory. An operand may be variable, constant or data type qualifier.

EXAMPLE

```
main( )
```

```
{

int sum; float f;

printf( "%d%d" ,size of(f), size of (sum) );

printf("%d%d", size of(235 L), size of(A));

}
```

Bitwise Operator

- Bitwise operator permit programmer to access and manipulate of data at bit level.
- Various bitwise operator enlisted are one's complement (~)
- bitwise AND (&)
- bitwise OR (|)
- bitwise XOR (^)
- left shift (<<)
- right shift (>>)

Logical or Boolean Operator

Operator used with one or more operand and return either value zero (for false) or one (for true). The operand may be constant, variables or expressions. And the expression that combines two or more expressions is termed as logical expression. C has three logical operators :

Operator	**Meaning**
&&	AND
||	OR

- !   NOT

Looping statements:-

- it is a block of statement that performs set of instructions. In loops
- Repeating particular portion of the program either a specified number of time or until a particular no of condition is being satisfied.
- There are three types of loops in c
- **1.While loop**
- **2.do while loop**
- **3.for loop**
- **While loop**

  Syntax:-

```
while(condition)
{
Statement 1;
Statement 2;
}
Or
while(test condition)
Statement;
```

The test condition may be any expression .when we want to do something a fixed no of times but not known about the number of iteration, in a program then while loop is used.

Here first condition is checked if, it is true body of the loop is executed else, If condition is false control will be come out of loop.

EXAMPLE

```
/* wap to print 5 times welcome to C" */ #include<stdio.h>

void main()

{

int p=1; While(p<=5)

{

printf("Welcome to C\n"); P=p+1;

}

}
```

Output:

```
Welcome to C

Welcome to C

Welcome to C

 Welcome to C

 Welcome to C
```

do while loop

This (do while loop) statement is also used for looping. The body of this loop may contain single statement or block of statement. The syntax for writing this statement is:

Syntax:-

```
Do

{

Statement;

}

while(condition);
```

Example:-

```
 #include<stdio.h>

void main()

{

int X=4; do

{

Printf("%d",X); X=X+1;

}whie(X<=10);

Printf(" ");

}
```

 Output: 4 5 6 7 8 9 10

for loop

In a program, for loop is generally used when number of iteration are known in advance. The body of the loop can be single statement or multiple statements. Its syntax for writing is:

Syntax:-

```
for(exp1;exp2;exp3)

{

Statement;

}
```

EXAMPLE

```
 void main()
{
int i; for(i=1;i<10;i++)
{

Printf(" %d ", i);
}
 }
```

```
Output:-1 2  3     4 5 6 7 8 9
```

Break statement(break)

- Sometimes it becomes necessary to come out of the loop even before loop condition becomes false then break statement is used. Break statement is used inside loop and switch statements. It cause immediate exit from that loop in which it appears and it is generally written with condition. It is written with the keyword as **break.**

Example :

```
void main()

{

int j=0; for(;j<6;j++) if(j==4) break;

}
```

Output:

```
0 1 2 3
```

Continue statement (key word continue)

Continue statement is used for continuing next iteration of loop after skipping some statement of loop. When it encountered control au Example:- void main()

```
{

int n;

for(n=2; n<=9; n++)

{

if(n==4) continue; printf("%d", n);

}
}
Printf("out of loop");
}
```

Output: 2 3 5 6 7 8 9 out of loop

tomatically passes through the beginning of the loop.

If statement

- ▣ Statement execute set of command like when condition is true and its syntax is

```
If (condition)
Statement;
```

EXAMPLE

```
void main()
{
int n;
printf (" enter a number:"); scanf("%d",&n);
If (n>10)
Printf(" number is grater");
}
```

Output:Enter a number:12 Number is greater

if…..else... Statement

- ▣ it is bidirectional conditional control statement that contains one condition & two possible action. Condition may be true or false, where non-zero value regarded as true & zero value regarded as false. If condition are satisfy true, then a single or block of statement executed otherwise another single or block of statement is executed.

- Its syntax is:-
- if (condition)
- {
- Statement1; Statement2;
- }
-
- else
- {
- Statement1; Statement2;
- }

Example:-

```
/* To check a number is eve or odd */

void main()

{

int n;

printf (“enter a number:”); sacnf (“%d”, &n);

If (n%2==0)
```

printf ("even number");

else

}

printf("odd number");

Output: enter a number:121 odd number

Nesting of if …else

- When there are another if else statement in if-block or else-block, then it is called nesting of if-else statement.
- Syntax is :-
-
- if (condition)
- {
-

  If (condition) Statement1;
-

  else
- }
- statement2;
- Statement3;

If….elseLADDER

- In this type of nesting there is an if else statement in every else part except the last part. If condition is false control pass to block where condition is again checked with its if statement.
- Syntax is :-
- if (condition) Statement1;
- else if (condition) statement2;
- else if (condition) statement3;
- else
- statement4;

ARRAY

- Array is the collection of similar data types or collection of similar entity stored in contiguous memory location. Array of character is a string
- **ADVANTAGES**: array variable can store more than one value at a time where other variable can store one value at a time.
- Example:
- int arr[100];
- int mark[100];
- **DECLARATION OF AN ARRAY :**
- Its syntax is :
- Data type array name [size];

- int arr[100];
- int mark[100];
- int a[5]={10,20,30,100,5}

INITIALIZATION OF AN ARRAY:

- After declaration element of local array has garbage value. If it is global or static array then it will be automatically initialize with zero. An explicitly it can be initialize that
- Data type array name [size] = {value1, value2, value3…}
-
- Example:
- in ar[5]={20,60,90, 100,120}

ACCESSING OF ARRAY ELEMENT

/*Write a program to input values into an array and display them*/

```
#include<stdio.h>

int main()

{

int arr[5],i; for(i=0;i<5;i++)

{

printf("enter a value for arr[%d] \n",i); scanf("%d",&arr[i]);

}

printf("the array elements are: \n"); for (i=0;i<5;i++)
{
printf("%d\t",arr[i]);
}
return 0;
}
```

OUTPUT:

```
Enter a value for arr[0] = 12 Enter a value for arr[1] =45 Enter a value for arr[2]
=59 Enter a value for arr[3] =98 Enter a value for arr[4] =21
The array elements are 12 45 59 98 21
```

Single dimensional arrays and functions

```
/*program to pass array elements to a function*/ #include<stdio.h>
void main()
{
int arr[10],i;
printf("enter the array elements\n"); for(i=0;i<10;i++)
{
scanf("%d",&arr[i]); check(arr[i]);
}
}
void check(int num)
```

```
{
if(num%2=0)
{
printf(”%d is even \n”,num);
}
else
{
printf(”%d is odd \n”,num);
}
}
```

Two dimensional arrays

- Two dimensional array is known as matrix. The array declaration in both the array i.e.in single dimensional array single subscript is used and in two dimensional array two subscripts are is used.
- Its syntax is
- Data-type array name[row][column];
- Or we can say 2-d array is a collection of 1-D array placed one below the other.
- Total no. of elements in 2-D array is calculated as **row*column**
- Example:-
- int a[2][3];
- Total no of elements=row*column is 2*3 =6

- ▣ It means the matrix consist of 2 rows and 3 columns

String

- ▣ Array of character is called a string. It is always terminated by the NULL character. String is a one dimensional array of character.
- ▣ We can initialize the string as char name[]={‘j’,’o’,’h’,’n’,’\o’};

String library function

- ▣ There are several string library functions used to manipulate string and the prototypes for these functions are in header file “string.h”. Several string functions are

strlen()

- ▣ This function return the length of the string. i.e. the number of characters in the string excluding the terminating NULL character.
- ▣ It accepts a single argument which is pointer to the first character of the string. For example-

  strlen(“suresh”);

  It return the value 6.

Example:-

```
#include<stdio.h>

#include<string.h>

void main()

{

char str[50];
```

```
 print(”Enter a string:”);

gets(str);

printf(“Length of the string is %d\n”,strlen(str));

}
```

Output:

```
Enter a string: C in Depth Length of the string is 8
```

strcmp()

- This function is used to compare two strings. If the two string match, strcmp() return a value 0 otherwise it return a non-zero value. It compare the strings character by character and the comparison stops when the end of the string is reached or the corresponding characters in the two string are not same.
  - strcmp(s1,s2) return a value:
  - <0 when s1<s2
  - =0 when s1=s2
  - >0 when s1>s2

strcpy()

- This function is used to copying one string to another string. The function strcpy(str1,str2) copies str2 to str1 including the NULL character. Here str2 is the source string and str1 is the destination string.
- The old content of the destination string str1 are lost. The function returns a pointer to destination string str1.

Example:-

```
#include<stdio.h>
 #include<string.h>
void main()
{
char str1[10],str2[10];
printf(“Enter a string:”);
scanf(“%s”,str2);
strcpy(str1,str2);
printf(“First string:%s\t\tSecond string:%s\n”,str1,str2);
strcpy(str,”Delhi”);
strcpy(str2,”Bangalore”);
printf(“First string :%s\t\tSecond string:%s”,str1,str2);
```

strcat()

- This function is used to append a copy of a string at the end of the other string. If the first string is “”Purva” and second string is “Belmont” then after using this function the string becomes “PusvaBelmont”. The NULL character from str1 is moved and str2 is added at the end of str1. The 2nd string str2 remains unaffected. A pointer to the first string str1 is returned by the function

- Example:-

```
#include<stdio.h>

 #include<string.h>

 void main()

{

char str1[20],str[20]; printf("Enter two strings:"); gets(str1);

gets(str2); strcat(str1,str2);

printf("First string:%s\t second string:%s\n",str1,str2); strcat(str1,"-one");

printf("Now first string is %s\n",str1);

}
```

Output

```
Enter                two               strings:               data               Base
First string: database second string: database

Now first string is: database-one
```

FUNCTION

- A function is a self contained block of codes or sub programs with a set of statements that perform some specific task or coherent task when it is called.
- There are basically two types of function those are
    - **Library function**
    - **User defined function**
- **Here in system defined function description**:

- **Function definition** : predefined, precompiled, stored in the library
- **Function declaration** : In header file with or function prototype.
- **Function call** : By the programmer
- **User defined function**
- Syntax:-
- Return type name of function (type 1 arg 1, type2 arg2, type3 arg3) Return type function name argument list of the above syntax
- So when user gets his own function three thing he has to know, these are.
- **Function declaration**
- **Function definition**
- **Function call**
- **Function declaration:-**
- Function declaration is also known as function prototype. It inform the compiler about three thing, those are name of the function, number and type of argument received by the function and the type of value returned by the function.
- While declaring the name of the argument is optional and the function prototype always terminated by the semicolon.
- **Function definition:-**
- Function definition consists of the whole description and code of the function. It tells about what function is doing what are its inputs and what are its out put It consists of two parts function header and function body
- Syntax:-

- return type function(type 1 arg1, type2 arg2, type3 arg3) /*function header*/
- {
- Local variable declaration; Statement 1;
- Statement 2;
- Return value
- **Function Call**
- When the function get called by the calling function then that is called, function call. The compiler execute these functions when the semicolon is followed by the function name.
- Example:- function(arg1,arg2,arg3);
- The argument that are used inside the function call are called **actual argument**
- Ex:-
- int S=sum(a, b);   //actual arguments

Actual argument

- The arguments which are mentioned or used inside the function call is knows as actual argument and these are the original values and copy of these are actually sent to the called function
- It can be written as constant, expression or any function call like Function (x);
- Function (20, 30); Function (a*b, c*d); Function(2,3,sum(a, b));

Formal Arguments

- The arguments which are mentioned in function definition are called formal arguments or dummy arguments.

- These arguments are used to just hold the copied of the values that are sent by the calling function through the function call.
- These arguments are like other local variables which are created when the function call starts and destroyed when the function ends.
- The basic difference between the formal argument and the actual argument are

The formal argument are declared inside the parenthesis where as the local variable declared at the beginning of the function block.

- The **formal argument** are automatically initialized when the copy of actual arguments are passed while other local variable are assigned values through the statements.
- Order number and type of actual arguments in the function call should be match with the order number and type of the formal arguments.

Call by value and call by reference

- **Call by value**
- In the call by value copy of the actual argument is passed to the formal argument and the operation is done on formal argument.When the function is called by ‘call by value’ method, it doesn’t affect content of the actual argument.

Example:-

```
    main()

    {

int x,y; change(int,int);

    printf(“enter two values:\n”); scanf(“%d%d”,&x,&y); change(x ,y);
    printf(“value of x=%d and y=%d\n”,x ,y);

    }

    change(int a,int b);

    {

    int k; k=a; a=b; b=k;

    }
```

```
Output: enter two values: 12 23

Value of x=12 and y=23
```

Call by reference

- Instead of passing the value of variable, address or reference is passed and the function operate on address of the variable rather than value.

Example:-

```
void main()

{

int a,b;

change(int *,int*);

printf(“enter two values:\n”); scanf(“%d%d”,&a,&b);change(&a,&b);

printf(“after changing two value of a=%d and b=%d\n:”a,b);

}

change(int *a, int *b)
```

```
{
int k; k=*a;
*a=*b;
*b= k;
 printf("value in this function a=%d and b=%d\n",*a,*b);
}
```

```
Output: enter two values: 12 32
Value in this function a=32 and b=12
After changing two value of a=32 and b=12
```

- So here instead of passing value of the variable, directly passing address of the variables. Formal argument directly access the value and swapping is possible even after calling a function.

Local, Global and Static variable

- **Local variable:-**
- variables that are defined with in a body of function or block. The local variables can be used only in that function or block in which they are declared.

Same variables may be used in different functions such as function()

```
{
int a,b; function 1();
}
function2 ()
```

```
{
int a=0; b=20;
}
```

- **Global variable**:-
- The variables that are defined outside of the function is called global variable. All functions in the program can access and modify global variables. Global variables are automatically initialized at the time of initialization.

Example:

```
 #include<stdio.h>
 void function(void); void function1(void); void function2(void); int a, b=20;
void main()
{
 printf(“inside main a=%d,b=%d \n”,a,b); function();
function1();
function2();
}
function()
{
 Prinf(“inside function a=%d,b=%d\n”,a,b);
}
 function 1()
```

```
{
prinf(“inside function a=%d,b=%d\n”,a,b);
}
function 2()
{
 prinf(“inside function a=%d,b=%d\n”,a,);
}
```

- **Static variables**:
- static variables are declared by writing the key word static.
- -syntax:-
- static data type variable name; static int a;
- -the static variables initialized only once and it retain between the function call. If its variable is not initialized, then it is automatically initialized to zero.

Example:

```
void fun1(void); void fun2(void); void main()
{
 fun1();
fun2();
}
 void fun1()
```

```
{
int a=10, static int b=2; printf("a=%d, b=%d",a,b); a++;
b++;
}
Output:
a= 10
 b= 2
a=10
b= 3
```

Monolithic Programming

- The program which contains a single function for the large program is called monolithic program. In monolithic program not divided the program, it is huge long pieces of code that jump back and forth doing all the tasks like single thread of execution, the program requires. Problem arise in monolithic program is that, when the program **size** increases it leads inconvenience and difficult to maintain
- such as testing, debugging etc. Many disadvantages of monolithic programming are:
    - Difficult to check error on large programs size.
    - Difficult to maintain because of huge size.
    - Code can be specific to a particular problem. i.e. it cannot be reused.
- Many early languages (FORTRAN, COBOL, BASIC, C) required one huge workspace with labelled areas that may does specific tasks but are not isolated.

Modular Programming

- The process of subdividing a computer program into separate sub-programs such as functions and subroutines is called Modular programming. Modular programming sometimes also called as structured programming. It enables multiple programmers to divide up the large program and debug pieces of program independently and tested.

Storage Classes

- Storage class in c language is a specified which tells the compiler where and how to store variables, its initial value and scope of the variables in a program
- Syntax of declaring storage classes is:-
- storage class data type variable name;There are four types of storage classes and all are keywords:-
- There are four types of storage classes and all are keywords:-
- **1) Automatic (auto)**
- **2) Register (register)**
- **3) Static (static)**
- **4) External (extern)**

Examples

auto float x;

or

float x;

extern int x;

register char c;

static int y;

Compiler assume different storage class based on:-

- **Storage class:-** tells us about storage place(where variable would be stored).
- **Intial value:-**what would be the initial value of the variable.
- If initial value not assigned, then what value taken by uninitialized variable.
- **Scope of the variable:-**what would be the value of the variable of the program.
- **Life time :-** It is the time between the creation and distribution of a variable or how long would variable exists.

Types of storage

- **Automatic storage class**
- The keyword used to declare automatic storage class is auto. Its features:-
- **Storage**-memory location
- **Default initial value**:-unpredictable value or garbage value.
- **Scope**:-local to the block or function in which variable is defined.
- **Life time**:-Till the control remains within function or block in which it is defined. It terminates when function is released.
- The variable without any storage class specifier is called automatic variable.

Example:-

main( )

```
{
auto int i; printf("i=",i);
}
```

- **Register storage class**
- The keyword used to declare this storage class is register. The features are:-
- **Storage:-**CPU register.
- **Default initial value** :-garbage value
- **Scope :-**local to the function or block in which it is defined.
- **Life time :-**till controls remains within function or blocks in which it is defined.
- Register variable don't have memory address so we can't apply address operator on it. CPU register generally of 16 bits or 2 bytes. So we can apply storage classes only for integers, characters, pointer type.

Static storage class

- The keyword used to declare static storage class is static. Its feature are:-
- **Storage**:-memory location
- **Default initial value**:- zero
- **Scope :-** local to the block or function in which it is defined.
- **Life time:-** value of the variable persist or remain between different function call.

Example:-

```
main( )
{
```

```
reduce( );

reduce( );

reduce ( );

}

reduce( )

{

static int x=10; printf("%d",x); x++;

}

Output:-10,11,12
```

- **External storage classes**
- The keyword used for this class is extern. Features are:-
- **Storage:-** memory area **Default initial value:-**zero **Scope :-** global
- **Life time:-**as long as program execution remains it retains. Declaration does not create variables, only it refer that already been created at somewhere else. So, memory is not allocated at a time of declaration and the external variables are declared at outside of all the function.

Example:-

```
int i,j;

void main( )

{

printf( "i=%d",i ); receive( ); receive ( ); reduce( );
reduce( );

}

receive( )

{

i=i+2;

printf("on increase i=%d",i);

}

reduce( )

{

i=i-1;

printf("on reduce i=%d",i);

}
Output:-i=0,2,4,3,2.
```

POINTER

- A pointer is a variable that store memory address or that contains address of another variable where addresses are the location number always contains whole number. So, pointer contain always the whole number. It is called pointer because it points to a particular location in memory by storing address of that location.
- Syntax-
- **Data type *pointer name;**
- Here * before pointer indicate the compiler that variable declared as a pointer. e.g.

```
int *p1; //pointer to integer type
 float *p2; //pointer to float type
 char *p3; //pointer to character type
```

Example:

```
void main()
{
 int i=105;
 int *p; p=&i;
printf("value of i=%d",*p);
printf("value of i=%d",*/&i);
 printf("address of i=%d",&i);
printf("address of p=%d",p);
printf("address of p=%u",&p);
}
```

- **Pointer Arithmetic**
- Pointer arithmetic is different from ordinary arithmetic and it is perform relative to the data type(base type of a pointer).

Example:-

- If integer pointer contain address of 2000 on incrementing we get address of 2002 instead of 2001, because, size of the integer is of 2 bytes.

Ex:-

```
void main( )

{

static int a[ ]={20,30,105,82,97,72,66,102};

int *p,*p1;

P=&a[1];

P1=&a[6];

printf("%d",*p1-*p);

printf("%d",p1-p);

}
```

Arithmetic operation never perform on pointer are:

- **addition, multiplication and division of two pointer. multiplication between the pointer by any number. division of pointer by any number**
- **Pointer Comparison**
- Pointer variable can be compared when both variable, object of same data type and it is useful when both pointers variable points to element of same array.
- ==,!=,<=,<,>,>=, can be used with pointer. Equal and not equal operators used to compare two pointer should finding whether they contain same address or not and they will equal only if are null or contains address of same variable.

Ex:-

```
void main()
```

```
{
static int arr[]={20,25,15,27,105,96} int *x,*y;
x=&a[5];
y=&(a+5); if(x==y) printf("same"); else printf("not");
}
```

Pointer to pointer

- ▣ Addition of pointer variable stored in some other variable is called pointer to pointer variable.
- ▣ Or
- ▣ Pointer within another pointer is called pointer to pointer.
- ▣ Syntax:-
- ▣ Data type **p;
- ▣ int x=22;
- ▣ int *p=&x;
- ▣ int **p1=&p;
- ▣ P 2000
- ▣ X 1000
- ▣ 22
- ▣ printf("value of x=%d",x);
- ▣ printf("value of x=%d",*p);

- printf(“value of x=%d”,*&x);
- printf(“value of x=%d”,**p1);
- printf(“value of p=%u”,&p);
- printf(“address of p=%u”,p1);
- printf(“address of x=%u”,p);
- printf(“address of p1=%u”,&p1);
- printf(“value of p=%u”,p);
- printf(“value of p=%u”,&x);

Pointer vs array

Example :-

```
void main()

static char arr[]=”Rama”;

char*p=”Rama”;

printf(“%s%s”, arr, p);
```

- In the above example, at the first time printf( ), print the same value array and pointer.
- Here array arr, as **pointer to character** and **p act as a pointer to array of character .**When we are trying to increase the value of arr it would give the error because its known to compiler about an array and its base address which is always printed to base address is known as constant pointer and the base address of array which is not allowed by the compiler.
- printf(“size of (p)”,size of (ar)); size of (p)   2/4 bytes

- size of(ar)   5 byes

Structure

- It is the collection of dissimilar data types or heterogenous data typesgrouped together. It means the data types may or may not be of same type.

  Structure declaration-

  struct tagname

  {

  Data type member1; Data type member2; Data type member3;

  ....

  ....

  Data type member n;

  }

  Structure variable declaration;

  struct student

  {

  int age; char name[20];

  char   branch[20];
  }; struct student s;

Initialization of structure variable

- Like primary variables structure variables can also be initialized when they are declared. Structure templates can be defined locally or globally. If it is local it can be used within that function. If it is global it can be used by all other functions of the program.

- We can’t initialize structure members while defining the structure struct student
- {
- int age=20;
- char name[20]=”sona”;
- }s1;
- The above is **invalid.**
- A structure can be initialized as struct student
- {
- int age,roll; char name[20];
- } struct student s1={16,101,”sona”}; struct student s2={17,102,”rupa”};
- If initialiser is less than no.of structure variable, automatically rest values are taken as zero.

Accessing structure elements

- Dot operator is used to access the structure elements. Its associativety is from left to right.
- structure variable ;
- s1.name[];
- s1.roll;
- s1.age;

Example:

```
#include<stdio.h>
 #include<conio.h>
void main()
{
int roll, age;
 char branch;
}
s1,s2;
printf("\n enter roll, age, branch=");
scanf("%d %d %c", &s1.roll, &s1.age, &s1.branch); s2.roll=s1.roll;
printf(" students details=\n");
printf("%d %d %c", s1.roll, s1.age, s1.branch); printf("%d", s2.roll);
}
```

Size of structure

- Size of structure can be found out using sizeof() operator with structure variable name or tag name with keyword.
- sizeof(struct student); or sizeof(s1);
- sizeof(s2);
- Size of structure is different in different machines. So size of whole structure may not be equal to sum of size of its members.

Array of structures

- When database of any element is used in huge amount, we prefer Array of structures.
- Example:
- Suppose we want to maintain data base of 200 students, Array of structuresis used.

```
#include<stdio.h>

#include<string.h>

struct student

{

char name[30];

 char branch[25]; int roll;

};

void main()

{

struct student s[200]; int i;

s[i].roll=i+1;

printf("\nEnter information of students:");

for(i=0;i<200;i++)

{

printf("\nEnter the roll no:%d\n",s[i].roll);
printf("\nEnter the name:");
 scanf("%s",s[i].name);
printf("\nEnter the branch:");
scanf("%s",s[i].branch); printf("\n");
}
printf("\nDisplaying information of students:\n\n"); for(i=0;i<200;i++)
{
printf("\n\nInformation for roll no%d:\n",i+1);
printf("\nName:");
puts(s[i].name);
printf("\nBranch:"); puts(s[i].branch);
}
}
```

Array within structures

- struct student
- {
- char name[30];
- int roll,age,marks[5];
- }; struct student s[200];
- We can also initialize using same syntax as in array.

Nested structure

- When a structure is within another structure, it is called Nested structure. A structure variable can be a member of another structure and it is representedas

```
struct student

{

element 1;

element 2;

………

………

struct student1

{

member 1;

member 2;

}variable 1;

……….

………. element n;

}variable 2;

printf("\nEnter the roll no:%d\n",s[i].roll);

printf("\nEnter the name:");

 scanf("%s",s[i].name);
```

```
printf("\nEnter the branch:");

scanf("%s",s[i].branch); printf("\n");

}

printf("\nDisplaying information of students:\n\n"); for(i=0;i<200;i++)

{

printf("\n\nInformation for roll no%d:\n",i+1);

printf("\nName:");

puts(s[i].name);

printf("\nBranch:"); puts(s[i].branch);

}

}
```

Passing structure elements to function

- We can pass each element of the structure through function but passing individual element is difficult when number of structure element increases. To overcome this, we use to pass the whole structure through function instead of passing individual element.

```
#include<stdio.h>

#include<string.h>

void main()

{

struct student
{
char name[30];
char branch[25]; int roll;
}struct student s; printf("\n enter name=");
gets(s.name); printf("\nEnter roll:");
scanf("%d",&s.roll);
 printf("\nEnter branch:");
gets(s.branch);
 display(name,roll,branch);
}
display(char name, int roll, char branch)
{
printf("\n name=%s,\n roll=%d, \n branch=%s", s.name, s.roll. s.branch);
}
```

UNION

- **Union** is derived data type contains collection of different data type or dissimilar elements. All definition declaration of union variable and accessing member is similar to structure, but instead of keyword struct the keyword union is used, the main difference between union and structure is
- **Syntax of union:**
- union student

- {
- datatype member1; datatype member2;
- };
- Like structure variable, union variable can be declared with definition or separately such as
- union union name
- {
- Datatype member1;
- }var1;

Example:-

struct student

- {
- int i;
- char ch[10];
- };struct student s;
- Here datatype/member structure occupy 12 byte of location is memory, where as in the union side it occupy only 10 byte.

Nested of Union

- When one union is inside the another union it is called nested of union.

Example:-

union a

```
{
int i; int age;
};
union b
{
char name[10];
union a aa;
};
 union b bb;
```

- There can also be union inside structure or structure in union.

Example:-

```
 void main()
{
struct a
{
int i;
char ch[20];
};
struct b
{
```

```
int i;

char d[10];

};

union z

{

struct a a1; struct b b1;

};

 union z z1; z1.b1.j=20; z1.a1.i=10; z1.a1.ch[10]= " i";

z1.b1.d[0]="j ";

printf(" ");
```

Dynamic memory Allocation

- The process of allocating memory at the time of execution or at the runtime, is called dynamic memory location.
- Two types of problem may occur in static memory allocation.
- If number of values to be stored is less than the size of memory, there would be wastage of memory.
- If we would want to store more values by increase in size during the execution on assigned size then it fails
- **malloc ()**
- returns the pointer to the 1st byte and allocate memory, and its return type is void, which can be type cast such as:

- int *p=(datatype*)malloc(size)
- If memory location is successful, it returns the address of the memory chunk that was allocated and it returns null on unsuccessful and from the above declaration a pointer of type**(datatype)** and size in byte.
- And **datatype** pointer used to typecast the pointer returned by malloc and this typecasting is necessary since, malloc() by default returns a pointer to void.

Example

```
int*p=(int*)malloc(10);

/*calculate the average of mark*/ void main()

{

int n , avg,i,*p,sum=0;

printf("enter the no. of marks ”);

 scanf(“%d”,&n);

p=(int *)malloc(n*size(int));

 if(p==null)

printf(“not sufficient”);

exit();

}

for(i=0;i<n;i++)

scanf(“%d”,(p+i)); for(i=0;i<n;i++)

 Printf(“%d”,*(p+i));
```

sum=sum+*p; avg=sum/n;

printf("avg=%d",avg);

- **calloc()**
- Similar to malloc only difference is that calloc function use to allocate multiple block of memory .
- two arguments are there

1st argument specify number of blocks
2nd argument specify size of each block.

Example:-

int *p= (int*) calloc(5, 2);

int*p=(int *)calloc(5, size of (int));

realloc()

The function realloc use to change the size of the memory block and it alter the size of the memory block without loosing the old data, it is called reallocation of memory.

- It takes two argument such as; int *ptr=(int *)malloc(size);
- int*p=(int *)realloc(ptr, new size);
- The new size allocated may be larger or smaller

Example:

```
#include<stdio.h>

#include<alloc.h>

 void main()
```

```
int i,*p;

p=(int*)malloc(5*size of (int)); if(p==null)

{

printf(“space not available”); exit();

printf(“enter 5 integer”); for(i=0;i<5;i++)

{

scanf(“%d”,(p+i)); int*ptr=(int*)realloc(9*size of (int) ); if(ptr==null)

{

printf(“not available”); exit();

}

printf(“enter 4 more integer”); for(i=5;i<9;i++) scanf(“%d”,(p+i)); for(i=0;i<9;i++)

printf(“%d”,*(p+i));

}
```

- **free()**
- Function free() is used to release space allocated dynamically, the memory released by free() is made available to heap again. It can be used for further purpose.
- Syntax for free declaration . void(*ptr)
- **free(p)**

- When program is terminated, memory released automatically by the operating system. Even we don't free the memory, it doesn't give error, thus lead to memory leak.
- We can't free the memory, those didn't allocated.

Dynamic array

- Array is the example where memory is organized in contiguous way, in the dynamic memory allocation function used such as malloc(), calloc(), realloc() always made up of contiguous way and as usual we can access the element in two ways as:
- **Subscript notation Pointer notation**

Example:

```
#include<stdio.h>
 #include<alloc.h>
 void main()
{
printf("enter the no.of values");
scanf("%d",&n);
p=(int*)malloc(n*size of int);
If(p==null)
printf("not available memory");
 exit();
}
```

```
for(i=0;i<n;i++)

{

printf(“enter an integer”);

scanf(“%d”,&p[i]); for(i=0;i<n;i++)

{

printf(“%d”,p[i]);

}

}
```

FILE HANDLING:

File: the file is a permanent storage medium in which we can store the data permanently.

- **Types of file can be handled**
- we can handle three type of file as
- **sequential file**
- **random access file**
- **binary file**

File Operation

- **opening a file:**
- Before performing any type of operation, a file must be opened and for this fopen() function is used.
- **syntax:**

- file-pointer=fopen(“FILE NAME ”,”Mode of open”);
- **File-pointer:** The file pointer is a pointer variable which can be store the address of a special file that means it is based upon the file pointer a file gets opened.
- **Declaration of a file pointer:-**
- FILE* var;
- **Modes of open**
- The file can be open in three different ways as

Read mode ’ r’/rt Write mode ’w’/wt Appened Mode ’a’/at

- **Reading** a character from a file
- **getc()** is used to read a character into a file Syntax:
- character_variable=getc(file_ptr);
- **Writing** acharacter into a file
- **putc()** is used to write a character into a file
- **puts**(character-var,file-ptr);

CLOSING A FILE

- **fclose()** function close a file.
- fclose(file-ptr);
- **fcloseall ()** is used to close all the opened file at a time

File Operation

- The following file operation carried out the file
- (1)creation of a new file
- (3)writing a file
- (4)closing a file
- Before performing any type of operation we must have to open the file.c, language communicate with file using A new type called **file pointer**.
- **Operation with fopen()**
- File pointer=fopen("FILE NAME","mode of open");
- If **fopen()** unable to open a file then it will return **NULL** to the file-pointer.

Reading and writing a characters from/to a file

- **fgetc()** is used for reading a character from the file
- **Syntax**:
- character variable= fgetc(file pointer);
- **fputc()** is used to writing a character to a file
- **Syntax**:
- fputc(character,file_pointer);

EXAMPLE

```
/*Program to copy a file to another*/

#include<stdio.h>
```

```
void main()
{
FILE *fs,*fd; char ch;
If(fs=fopen(“scr.txt”,”r”)==0)
{
printf(“sorry….The source file cannot be opened”); return;
}
If(fd=fopen(“dest.txt”,”w”)==0)
{
printf(“Sorry…..The destination file cannot be opened”); fclose(fs);
return;
}
while(ch=fgets(fs)!=EOF) fputc(ch,fd);
fcloseall();
}
```

Reading and writing a string from/to a file

fgets() is used for reading a string from the file **Syntax**:

- **fgets**(string, length, file pointer);
- **fputs**() is used to writing a character to a file
- **Syntax:**

fputs(string,file_pointer);

EXAMPLE

```
 #include<string.h>

 #include<stdio.h>

void main(void)

{

FILE*stream;

char string[]=”This is a test”;

 char msg[20];

/*open a file for update*/ stream=fopen(“DUMMY.FIL”,”w+”);


/*write a string into the file*/ fwrite(string,strlen(string),1,stream);

/*seek to the start of the file*/ fseek(stream,0,SEEK_SET);

/*read a string from the file*/ fgets(msg,strlen(string)+1,stream);

/*display the string*/

 printf(“%s”,msg);

fclose(stream);

}
```

One mark questions:

1. What is the disadvantage of arrays in C?

a. Multiple other data structures can be implemented using arrays

b. The amount of memory to be allocated needs to be known beforehand.

c. Elements of array can be accessed in constant time

d. Elements are stored in contiguous memory blocks

Answer: b.The amount of memory to be allocated needs to be known beforehand.

2. How many bytes does "int = D" use?

a.1

b.0

c.10

d. 2 or 4

Answer: d. 2 or 4

3. Directives are translated by the__________

a. Pre-processor

b. Compiler

c. Linker

d. Editor

Answer: a. Pre-processor

4. What feature makes C++ so powerful?

a. Reusing the old code

b. Easy implementation

c. Easy memory management

d. All of the above

Answer: All of the above

5. What is the maximum number of characters that can be held in the string variable char address line [40]?

a. 39

b. 41

c. 40

d. 38

Answer: 39

6. Which of the following SLT template class is a container adaptorclass?

a. Deque

b. Vector

c. List

d. Stack

Answer: Stack

7. If addition had higher precedence than multiplication, then the value of the expression (1 + 2 * 3 + 4 * 5) would be which of the following?

a. 27

b. 105

c. 69

d. 47

Answer: b. **105**

Explanation: (1 + 2 * 3 + 4 * 5)

= (1 + 2) * (3 + 4) * 5

= 3 * 7 * 5

= 105

8. Each instance of a class has a different set of______

a. Attribute values

b. Class interfaces

c. Return types

d. Methods

Answer: Attribute values

9. What is the return type of the fopen() function in C?

a. An integer

b. pointer to a FILE object

c. Pointer to an integer

d. None of the above

Answer:b. pointer to a FILE object

10. How to find the length of an array in C?

a. sizeof(a)/sizeof(a[0])

b. sizeof(a)

c. sizeof(a)*sizeof(a[0])

d. None of the above

Answer: **a. sizeof(a)/sizeof(a[0])**

11. Which of the following is not a storage class specifier in C?

a. extern

b. volatile

c. typedef

d. static

Answer: volatile

2. Which of the following will occur if we call the free() function on aNULL pointer?

a. The program will execute normally

b. Compilation Error

c. Runtime error

d. undefined behaviour

Answer: The program will execute normally

13. Which of the following should be used to free memory from a pointer allocated using the “new” operator?

a. free()

b. realloc()

c. delete

d. None of the above

Answer: delete14. Which data structure is used to handle recursion in C?

a. Stack

b. Queue

c. Trees

d. Deque

Answer: Stack

15. Which of the below statements is incorrect in case of union?

a. Union is a user-defined data structure
b. All data share same memory
c. Union stores methods too
d. union keyword is used to initialize

Answer: c. Union stores methods too

e. 5 . Operators have precedence. Precedence determines which operator is
f. a) faster
g. b) takes less memory

h. c) evaluated first

i. d) takes no arguments

Ans: c

34. Integer Division results in

a) Rounding the fractional part

b) Truncating the fractional part

c) Floating value

d) An Error is generated

Ans: b

35. Which of the following is a ternary operator?

a) ?:

b) *

c) sizeof

d) ^

Ans: a

36. What will be the output of the expression 11 ^ 5?

a) 5

b) 6

c) 11

d) None of these

Ans: d

37. The type cast operator is

a) (type)

b) cast()

c) (;;)

d) // " "

Ans: a

38. Explicit type conversion is known as

a) Casting

b) Conversion

c) Disjunction

d) Separation

Ans: a

39.

5

33. Operators have precedence. Precedence determines which operator is

a) faster

b) takes less memory

c) evaluated first

d) takes no arguments

Ans: c

34. Integer Division results in

a) Rounding the fractional part

b) Truncating the fractional part

c) Floating value

d) An Error is generated

Ans: b

35. Which of the following is a ternary operator?

a) ?:

b) *

c) sizeof

d) ^

Ans: a

36. What will be the output of the expression 11 ^ 5?

a) 5

b) 6

c) 11

d) None of these

Ans: d

37. The type cast operator is

a) (type)

b) cast()

c) (;;)

d) // " "

Ans: a

38. Explicit type conversion is known as

a) Casting

b) Conversion

c) Disjunction

d) Separation

Ans: a

16.How is the 3rd element in an array accessed based on pointer notation?

a.*a + 3

b. *(*a+3)

c.*(a + 3)

d.& (a+3)

Answer: c.***(a + 3)**

17. Which of the following function is used to open a file in C++?

a.fgets

b.fseek

c.fclose

d.fopen

Answer: d.**fopen ()**

18. Which of the following are correct file opening modes in C?

a.w

b.r

c.rb

d.All of the above

Answer: **d.All of the above**

19. What is required in each C program?

a. The program does not require any function.

b. The program must have at least one function

c. Output data

d. Input data

Answer: b. **The program must have at least one function**

20In which of the following languages is function overloading notpossible?

a.C++

b.C

c.Python

d.Java

Answer: **C**

21. What is the output of this statement “printf(“%d”, (a++))”?

a. Error message

b. The value of (a + 1)

c. The current value of a

d. Garbage

Answer: c.**The current value of a**

22. Why is a macro used in place of a function?

a. It reduces code size

b. It reduces execution time

c. It increases code size

d. It increases execution time

Answer: It reduces code size

23. What is the 16-bit compiler allowable range for integer constants?

a. -3.4e38 to 3.4e38

b. -32767 to 32768

c. -32668 to 32667

d. -32768 to 32767

Answer: d. -32768 to 32767

24. What is a lint?

a. Iteractive debugger
b. Ccompiler
c. C Interpreter

d. Anlyzing tool

Answer: d. Analyzing tool

25. Ifp is an integer pointer with a value 1000, then what will the valueof p + 5 be?

a.1005

b.1020

c.1004

d.1010

Answer: d. **1020**

26. How to declare a double-pointer in C?

a. int **val

b. int *val

c. int **val

d. int &val

Answer:c. int **val

27. What is the size of the int data type (in bytes) in C?

a.4

b.8

c.2

d.1

Answer: a. **4**

28. In the C language, the constant is defined _______.

a. Before main

b. Anywhere, but starting on a new line.

c.After main

d.None of the these

Answer: b. **Anywhere, but starting on a new line**

29. How are String represented in memory in C?

a. An Array of characters

b. Linked list of characters

c. The object of some class

d. Some as other primitive data types

Answer: An Array of characters

30. Which of the following is an exit controlled loop?

a. While loop

b. For loop

c. do-while loop

d. None of the above

Answer: c. **do-while loop**

31. What is output of the below program?

```
int main()
{
 int i;
 for(i=0; i<5; ++i++)
 {
   printf("Hello");
 }
 return 0;
}
```

a. Hello is printed 5 times
b. Compilation Error
c. Hello is printed 2 times
d. Hello is printed 3 times

Answer: b. Compilation Error

32. Library function getch() belongs to which header file?

a. stdio.h
b. conio.h
c. stdlib.h
d. stdlibio.h

Answer: b. conio.h

33. What should be the output :

```
int main()
{
   int a = 10/3;
   printf("%d",a);

   return 0;
}
```

a. 3.33
b. 3.0
c. 3
d. 0

Answer: c.3

34.Which of the following best describes the ordering of destructor calls for stack-resident objects in a routine?

a. The first object created is the first object destroyed; last created is last destroyed.
b. The first object destroyed is the last object destroyed; last created is first destroyed.

c. Objects are destroyed in the order they appear in memory, the object with the lowest memory address is destroyed first.

d. The order is undefined and may vary from compiler to compiler.

Answer: b. The first object destroyed is the last object destroyed; last created is first destroyed.

35. What feature makes C++ so powerful?

a. Easy implementation

b. Reusing the old code

c. Easy memory management

d. All of the above

Answer: d.All of the above

36. The members of union can be accessed using ___.

a. Dot Operator (.)

b. And Operator (&)

c. Asterisk Operator (*)

d. Right Shift Operator (>)

Answer: a. Dot Operator (.)

37. What is a function in C?

a. User defined data type

b. Block of code which can be reused

c. Declaration syntax

d. None of these

Answer: b. Block of code which can be reused

38. A C program contains ___.

a. At least one function

b. No function

c. No value from command line
d. All of these

Answer: a. At least one function

39.A recursive function in C ___.

a. Call itself again and again
b. Loop over a parameter
c. Return multiple values
d. None of these

Answer: a. Call itself again and again

40. Multiple values of the same variable can be tested using ___.

a. switch
b. for
c. Function
d. All of these

Answer: a. switch

41. Without a break statement in switch what will happen?

e. All cases will work properly
f. Cases will fall through after matching the first check
g. Switch will throw error
h. All of these

Answer: b. Cases will fall through after matching the first check

42. When all cases are unmatched which case is matched in a switch statement?

a. Default case
b. First case

c. No case

d. None of these

Answer: a. Default case

43.What will be the output of the following C code?

```
#include <stdio.h>

int main(){

    char grade = 'B';

    switch (grade) {

    case 'A':

        printf("Excellent!\n");

    case 'B':

    case 'C':

        printf("Well done\n");

    case 'D':

        printf("You passed\n");

    case 'F':

        printf("Better try again\n");

        break;
    default:

        printf("Invalid grade\n");

    }

}
```

a. Well done
b. You passed
c. Better try again
d. All of these

Answer: d. All of these

44. Loops in C programming are used to ___.

a. Execute a statement based on a condition
b. Execute a block of code repeatedly
c. Create a variable
d. None of these

Answer: b. Execute a block of code repeatedly

45. Which of these is an exit-controlled loop?

a. for
b. if
c. do...while
d. while

Answer: c. do...while

46.Which statements are used to change the execution sequence?

a. Loop control statement
b. Function statement
c. Conditional statement
d. All of these

Answer: a. Loop control statement

47. What will happen if the loop condition will never become false?

a. Program will throw an error
b. Program will loop infinitely
c. Loop will not run
d. None of these

Answer: b. Program will loop infinitely

48. Which of the following is not an arithmetic expression?

a. x = 10
b. x /= 10
c. x %= 10
d. x != 10

Answer: d. x != 10

49.What will be the output of the following C code?

#include <stdio.h>

int main()

```
{
   int x = 20;
   x %= 3;
   printf("%d",x);
   return 0;
}
```

a. 2
b. 2.5
c. Error
d. Warning

Answer: a. 2

50. What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
   float x = 23.456;
   printf("%.2f",x);
   return 0;
}
```

a. 23.45600
b. 23.456
c. 23.45
d. 23.46

Answer: d. 23.46

51. For which type, the format specifier "%i" is used?

a. int
b. char
c. float
d. double

Answer: a. int

52. What is the difference between float and double in C?

a. Both are used for the same purpose
b. Double can store just double value as compare to float value
c. Double is an enhanced version of float and was introduced in C99
d. Double is more precise than float and can store 64 bits

Answer: d. Double is more precise than float and can store 64 bits

53.Which is the correct format specifier for double type value in C?

a. %d
b. %f
c. %lf
d. %LF

Answer: c. %lf

54. What is the correct syntax to declare a variable in C?

a. data_type variable_name;
b. data_type as variable_name;
c. variable_name data_type;
d. variable_name as data_type;

Answer: a. data_type variable_name;

55.What will be the output of the following C code?

```
      #include <stdio.h>
        int main(){
        int i, j;
          for (i = 2; i < 10; i++) {
             for (j = 2; j <= (i / j); j++)
                if (!(i % j))
                break;
        if (j > (i / j))
            printf("%d ", i);
     }
     return 0;
 }
```

a. 2 3 4 5 6 7 8 9
b. 3 5 7 9
c. 2 3 5 7
d. 2 3 5 7 11

Answer: c. 2 3 5 7

56. What is the keyword used to declare a C file pointer.?

a. file
b. FILE
c. FILEFP
d. filefp

Answer: b. FILE

57. What are the primary iterations in C programming?

a. While loop
b. Do while loop
c. For loop
d. All of the above

Answer: d. All of the above

58. What is scanf() in c programming?

a. The layout of an input string
b. Array
c. Output function
d. All of the above

Answer: a) Layout of an input string.

59. What is a function?

e. Looping code
f. Code that operates when called
g. Unknown variable
h. All of the above

Answer: b) Code that operates when called.

60. What of the following is true?

a. Variables are functions
b. Variable is a type of output command
c. Variables are used to store values
d. All of the above

Answer: c) Variables are used to store values.

61. What will be the output of the following code segment?

```
int x=24, y=39, z=45

z=x+y;

y=z-y;

x=z-y;

printf("\n%d%d%d",x,y,z);
```

a. 24 39 63
b. 39 24 63
c. 24 39 45

d. 39 24 45

Answer: b. 39 24 63

62. Which operators are used to compare the values of operands to produce logical value in C language?

e. Logical operator
f. Relational operator
g. Assignment operator
h. None of the above

Answer: b. Relational operator

63. while assigning a value to a variable, which operators are used to perform arithmetic operations?

i. Logical operator
j. Assignment operator
k. Increment operator
l. Conditional operator

Answer: b. Assignment operator

64. Which operators perform operations on data in binary level?

a. Logical operator
b. Bitwise operator
c. Additional operators
d. None of the above

Answer: b. Bitwise operator

65. An external variable is one

e. Which resides in the memory till the end of the program
f. Which is globally accessible by all functions

g. Which is declared outside the body of any function

h. All of the above

Answer: d. All of the above

66. The operator "&" is used for

a. Bitwise AND

b. Bitwise OR

c. Logical AND

d. Logical OR

Answer: a. Bitwise AND

67. Which are built-in data structures in C programming?

a. Arrays

b. Structures

c. Files

d. All of the above

Answer: d. All of the above

68. Which is the correct sequence statements that swaps values of two statements?

a. a=a+b; a=a-b; b=a-b;

b. a=a+b, b=a-b; a=a-b;

c. a=a-b; a=a+b; b=b-a;

d. None of these

Answer: b. a=a+b, b=a-b; a=a-b;

69. By default a real number is treated as a

a. float

b. double

c. long double
d. integer

Answer: a. float

70. In an assignment statement a=b; which of the following statement is true ?

a. The variable a and the variable b are same.
b. The value of b is assigned to variable a but if b changes later, it will not effect the value of variable a.
c. The value of b is assigned to variable a but if b changes later, it will effect the value of variable a
d. The value of variable a is assigned to variable b, and the value of variable b is asssigned to variable a.

Answer: b.The value of b is assigned to variable a but if b changes later, it will not effect the value of variable a.

71. Which of the following is responsible for conversion of C programs to machine language ?

a. Operating system
b. An editor
c. A compiler
d. An interpreter

Answer: c. A compiler

72. If a=8, b=3 and c=-5 are integers, then value of a*b/c is

a. -4
b. -2.8
c. +2.8
d. +3

Answer: a. -4

73. Which of the following is a valid string constant?

a. "programming"
b. "programming
c. 'programming
d. $programming$

Answer: a. "programming"

74. Which of the following cannot be used as identifiers?

a. Letters
b. Digits
c. Underscores
d. Spaces

Answer: d. Spaces

75. What is the return type of ftell function?

a. double
b. int
c. long
d. D float

Answer: c. long

76. The result of a relational operator is always

a. either True or False
b. is less than or is more than
c. is equal or less or more
d. All of these

Answer:a. either True or False

77. Maximum value of an unsigned integer is

a. 65535
b. 32767
c. -32767
d. -65535

Answer: a. 65535

78.What will happen if in a C program you assign a value to an array element whose subscript exceeds the size of array?

a. The element will be set to 0.
b. The compiler would report an error.
c. The program may crash if some important data gets overwritten.
d. The array size would appropriately grow.

Answer: a. The element will be set to 0.

79.C programs are converted into machine language with the help of

a. An Editor
b. A compiler
c. An operating system
d. None of the above

Answer:b. A compiler

80. A C variable cannot start with

a. An alphabet
b. A number

c. A special symbol other than underscore
d. both (b) and (c)

Answer: d. both (b) and (c)

81. Which of the following is allowed in a C Arithmetic instruction

a. []
b. {}
c. ()
d. None of the above

Answer: c. ()

82. Which of the following shows the correct hierarchy of arithmetic operations in C

a. / + * -
b. * - / +
c. + - / *
d. * / + -

Answer: d. * / + -

83. What is an array?

a. An array is a collection of variables that are of the dissimilar data type.
b. An array is a collection of variables that are of the same data type.
c. An array is not a collection of variables that are of the same data type.
d. None of the above.

Answer: b. An array is a collection of variables that are of the same data type.

84. What is right way to Initialization array?

a. int num[6] = { 2, 4, 12, 5, 45, 5 } ;

b. int n{} = { 2, 4, 12, 5, 45, 5 } ;
c. int n{6} = { 2, 4, 12 } ;
d. int n(6) = { 2, 4, 12, 5, 45, 5 } ;

Answer:a. int num[6] = { 2, 4, 12, 5, 45, 5 } ;

85. An array elements are always stored in _________ memory locations.

a. Sequential
b. Random
c. Sequential and Random
d. None of the above

Answer: a. Sequential

86. What is the right way to access value of structure variable book{ price, page }?

a. printf("%d%d", book.price, book.page);
b. printf("%d%d", price.book, page.book);
c. printf("%d%d", price::book, page::book);
d. printf("%d%d", price->book, page->book);

Answer: a. printf("%d%d", book.price, book.page);

87. perror( ) function used to ?

a. Work same as printf()
b. prints the error message specified by the compiler
c. prints the garbage value assigned by the compiler
d. None of the above

Answer: b. prints the error message specified by the compiler

88. Bitwise operators can operate upon?

a. double and chars

b. floats and doubles
c. int and float
d. int and char

Answer: d. int and char

89. What is C Tokens?

a. The smallest individual units of c program
b. The basic element recognized by the compiler
c. The largest individual units of program
d. A & B Both

Answer:d. A & B Both

90. What is Keywords?

a. Keywords have some predefine meanings and these meanings can be changed.
b. Keywords have some unknown meanings and these meanings cannot be changed.
c. Keywords have some predefine meanings and these meanings cannot be changed.
d. None of the above

Answer:c. Keywords have some predefine meanings and these meanings cannot be changed.

91. What is constant?

a. Constants have fixed values that do not change during the execution of a program
b. Constants have fixed values that change during the execution of a program
c. Constants have unknown values that may be change during the execution of a program
d. None of the above

Answer:a. Constants have fixed values that do not change during the execution of a program

92. Which is the right way to declare constant in C?

a. int constant var =10;
b. int const var = 10;
c. const int var = 10;
d. B & C Both

Answer:d. B & C Both

93. Which operators are known as Ternary Operator?

a. ::, ?
b. ?, :
c. ?, ;;
d. None of the above

Answer:b. ?, :

94. In switch statement, each case instance value must be _______?

a. Constant
b. Variable
c. Special Symbol
d. None of the above

Answer:a. Constant