Loop Control Statements:-

Loop:

A loop is defined as a block of statements which are repeatedly executed for certain number of times.

When to use Loops:-

⇒ If you want to perform the same operation repeatedly number of times, then loops will be used.

Steps in Loops:-

1. Loop variable: The variable used in the loop.
2. Initialization: It is the first step in which the starting con) initial value is assigned to the loop variable.
3. Test condition: Test condition dermines how many times the loop will execute.
4. Increment/Decrement: Updating the loop variable.

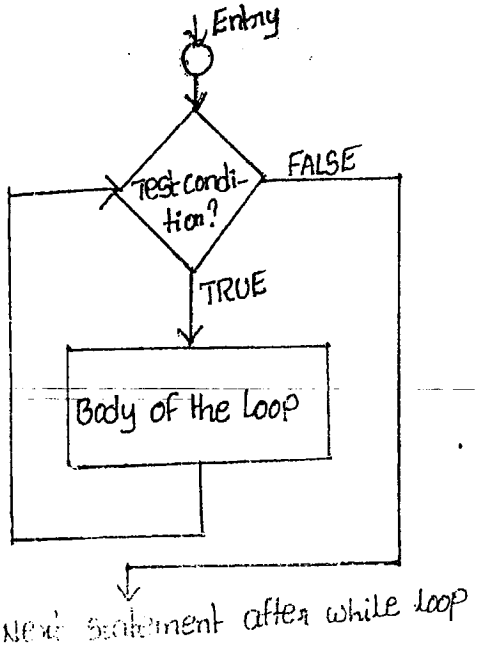
⇒ The C language supports 3 loop control statements:

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

While Loop:-

⇒ The basic format of while loop is:

```
while (test_condition)
{
    body of the loop
}
```



⇒ The while is an entry-controlled loop.

WORKING:-

⇒ The test-condition is evaluated first and if the condition is true, then the body of the loop is executed.

⇒ After execution of the body of the loop, the test-condition is once again evaluated and if it is true, the body of the loop is executed once again.

⇒ This process of repeated execution of the body continues till the test-condition becomes false.

⇒ If the test-condition becomes false, then the control is transferred out of the loop and the program execution continues with the statement immediately after the loop.

Example program:-

⇒ Write a program to display the message "welcome" 10 times.

```
/* Program to display the message "welcome" 10 times using while loop */
main()
{
   int i;
   i=1;
   while(i<=10)
   {
     printf("\n welcome");
     i=i+1;
   }
   getch();
}
```

OUTPUT:-

```
welcome
welcome
welcome
welcome
welcome
welcome
welcome
welcome
welcome
welcome
```

Explanation:-

In the above program, i is initialized to 1. The while loop checks the condition for i<=10. Since the condition is true, then the loop will be executed. ie. it displays the message "welcome" and increments i value by 1. Now i value becomes 2. Again it will check the condition, the condition is true and the body of the loop gets executed. This process continues till the condition becomes false.

② Write a program to find factorial of a given number using while loop.

```
/* Factorial of a given number */
main()
{
    int n, fact=1;
    clrscr();
    printf("Enter the number:");
    scanf("%d", &n);
    while(n>=1)
    {
        fact=fact*n;
        n--;
    }
    printf("\n Factorial of given number is: %d", fact);
    getch();
}
```

Sample Output:-

Enter the number: 4

Factorial of given number is: 24

Explanation:-

1st iteration:	2nd iteration	3rd iteration	4th iteration
n=4	n=3	n=2	n=1
Condition 4>=1 is true	3>=1 is true	2>=1 is true	1>=1 is true
∴ fact=1*4=4	∴ fact=4*3	∴ fact=12*2	∴ fact=24*1
n=n-1=3	n=3-1=2	=24	=24
		n=2-1=1	n=1-1=0

5th iteration:

n=0

0>=1 condition is false.

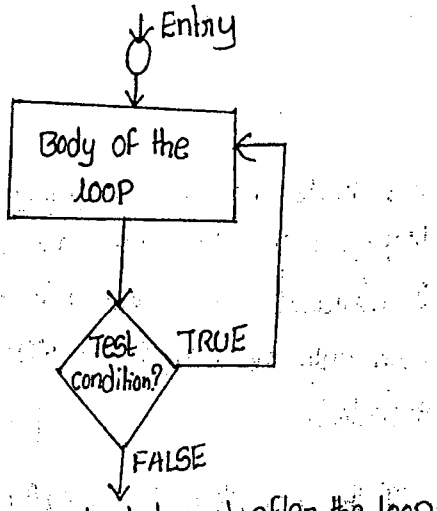
So control is transferred out of the loop.

⇒ Finally, it displays the value of fact i.e. 24

2. do-while loop:-

⇒ The basic format of do-while loop is:

```
do
{
  body of the loop;
}while(test-condition);
```



WORKING OF do-while loop:-

⇒ In do-while loop, First the body of the loop is executed once. After that, it checks the condition at the end of the loop. If the condition is true, the body of the loop will be executed once again. This process continues as long as the condition is true.

When the condition becomes false, the loop will be terminated and the control goes to the statement immediately after the loop.

⇒ The do-while loop is an exit-controlled loop because test-condition is evaluated at the end of the loop.

Example Program:-

/* Program to display first 10 numbers using do-while loop */

```
main()
{
  int i;
  i=1;
  clrscr();
  do
  {
    printf("\n %d", i);
    i++;
  }while(i<=10);
getch();
}
```

OUTPUT:-

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

Difference between while and do-while:-

while	do-while
1. The while loop is an entry-controlled loop. i.e. in while loop the condition is evaluated first and if it is true then only the body of the loop will be executed.	1. The do-while loop is an exit-controlled loop. i.e. in do-while loop first the body of the loop is executed, after that at the end of loop the condition is evaluated.

2. If the condition is initially false, the while loop will not be executed atleast once.
Ex:
`i=10;`
`while(i<=10)`
`{`
`printf("welcome");`
`}`
→ Here, the while loop will not be executed since the condition is false
output: None
	2. The do-while loop will execute atleast once even if the condition is initially false.
	Ex:
	`i=10;`
	`do`
	`{`
	`printf("welcome");`
	`}while(i<=0)`
	→ Here, the do-while loop is executed one time even if the condition is false
	output: welcome

3. Syntax:
`while(condition)`
`{`
`statements;`
`}`
Note: while should not be end with semicolon.
	3. Syntax:
	`do`
	`{`
	`statements;`
	`}while(condition);`
	Note: The do-while loop should terminate with semicolon.

3. for Loop:

⇒ The for loop is another entry-controlled loop.

⇒ The general form of for loop is:

```
for( initialization ; test-condition ; increment/decrement)
{
        body of the loop;
}
```

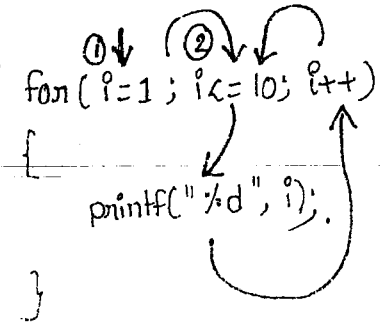
→ i.e updating loop variable.

WORKING:-

⇒ The execution of the for loop is as follows:

1. Initialization of the loop variable is done first using assignment statements such as i=10, count=0, j=0 etc. Initialization part is executed only once.
2. The test-condition is any relational expression such as i<10 that determines when to exit from the loop. The for loop continues to execute as long as the test condition is true. when the condition becomes false the loop is terminated and the execution continues with the statement after the loop.
3. When the body of the loop is executed, the control is transferred back to the for statement. The loop variable is updated and once again the test condition is checked. If the condition is true, the body of the loop is again executed. This process continues till the condition becomes false.

EX:



Example program:-

/* Program to display first 10 numbers in descending order */

```
main()
{
   int i;
   clrscr();
   for(i=10; i>=1; i--)
   {
      printf("\n %d", i);
   }
   getch();
}
```

OUTPUT:
10
9
8
7
6
5
4
3
2
1

Explanation:-

⇒ The value of i is initialized to 10 when the for loop execution starts.

⇒ Next the test condition i>=1 is evaluated. Initially the condition is true since i=10, 10>=1 is satisfied. So, the for loop will execute.

⇒ Upon executing the printf statement Compiler sends control back to the for loop where i value is decremented by 1. After decrementing the i value, the test condition is once again checked. If condition is true, the body is executed. This process continues till i value becomes 0.

Example program:-

/* Program to display all the even numbers from 1 to 10 */

```
main()
{  int i;
   for(i=2; i<=10; i=i+2)
   {
      printf("\n %d", i);
   }
}
```

OUTPUT:
2
4
6
8

JUMPS IN LOOPS:-

⇒ Loop performs a set of operations repeatedly until some condition becomes false.

⇒ Sometimes it is necessary to skip a part of the loop or exit the loop as soon as a certain condition occurs.

⇒ For example, consider you want to search for a particular name in a list of 100 names. The searching process should be terminated as soon as the desired name is found.

⇒ C supports the following jump statements:

1. break
2. continue
3. goto

1. break:-

⇒ The break statement is used to terminate the loop.

⇒ When a break statement is encountered inside a loop, the loop is immediately terminated and the program execution continues with the statement after the loop. i.e. the break statement skips from the loop in which it is defined.

⇒ The break statement is used with conditional if statement.

Ex:
```
while(condition)
{
------
------
if(condition)
    break;
------
------
}
------
```
Exit from loop

```
do
{
------
------
------
if(condition)
    break;
------
------
}while(condition);
------
```
Exit from loop

```
for(i=1;i<=10;i++)
{
------
------
------
if(condition)
    break;
------
------
}
------
```
Exit from loop

⇒ We can also use break statements inside nested for loops.

⇒ When break statement is used in nested loops, the control would terminate only from the loop in which it is defined.

Ex:

```
for(------)
{
  -------
  for(-----)
  {
    -----
    -----
    if(condition)
      break;
    ------
  }
  ------
  ------
}
```

Exit from inner most loop

```
while(condition)
{
  ------
  -----
  while(condition)
  {
    ------
    if(condition)
      break;
    ----
    ----
  }
  ------
  ------
}
```

Exit from inner loop

Example PROGRAMS:-

① Write a program to exit loop when you encounter 5.

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

Output:
1
2
3
4

→ In the above program the loop will be terminated as soon as i becomes 5.

② Write a program to display the message 10 times using infinite loop.

```
main()
{
   int i=1;
   for( ; ;)
   {
      printf("\n Hello");
      if(i==10)
       break;
       i++;
   }
   getch();
}
```

→ In the above program the loop will be terminated as soon as i value becomes 10.

2. Continue statement:-

⇒ The continue statement is opposite to the break statement.

⇒ The continue statement is used to continue with the next iteration.

⇒ When continue statement is used inside a loop, it tells the compiler "skips the following statements and continue with the next iteration".

Ex:

```
while(condition)
{
   -------
   -------
   if(condition)
     continue;
   -------
   -------
}
```

continue with the next iteration

```
for(i=1; i<10; i++)
{
   -------
   -------
   -------
   if(condition)
     continue;
   -------
   -------
}
```

Example programs:-

⇒ Display all the numbers from 1 to n which are not divisible by 4.

```
main()
{
    int i,n;
    clrscr();
    printf("Enter n value:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        if(i%4==0)
          continue;
          printf("\n%d",i);
    }
    getch();
}
```

→ In the above program, if the number is divisible by 4 then it will not display that number, it will continue with the next iteration.

⇒ Write a program to display all the even numbers from 1 to 10 using continue statement.

```
main()
{
   int i;
   for(i=1;i<=10;i++)
   {
      if(i%2!=0)
         continue;
         printf("%d",i);
   }
}
```

Difference between break and continue:-

Break	Continue
1. Exits from current loop	1. Loop takes next iteration.
2. Control passes to next statement after the loop.	2. Control passes at the beginning of the loop.
3. Terminates the program.	3. Never terminates the program.

3. goto statement:-

⇒ We can also use goto statement to jump within a loop and to exit from the loop.

```
while(condition)
{
    ------
    ------
    if(error)
      goto stop;
    ------
    ------
    ------
    if(condition)
      goto abc;
    ------
    ------
    abc:
    ------
    ------
}
stop:
------
------
```

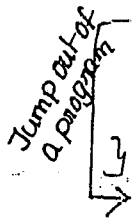
Jump within the loop

Exit from the loop.

JUMPING OUT OF THE PROGRAM

⇒ We can jump out of a program by using the library function exit().

Ex:
```
main()
{
    -----
    -----
    -----
    if(condition)
        exit(0);
    -----
    -----
}
```



⇒ The exit() function takes an integer value as its argument.

Normally 0 → indicates normal termination

non zero → indicates termination due to some error.

⇒ The use of exit() function requires the inclusion of header file <stdlib.h>.

Additional Features of for loops:-

The for loop has several additional features.

1. More than one variable can be initialized at a time in the for statement.

Ex:
```
for(i=10,j=0; i<10; i++)
{
    ----
}
```

→ Here the initialization section has two parts i=10 and j=0 which are seperated by comma.

2. Like in the initialization section, the increment/decrement section may also have more than one part.

Ex:
```
for(i=10,j=1; i<10; i--,j++)
{
    ----
}
```

3. Further more, the test condition may have any compound relation (→ more than one condition)

Ex:
```
for(i=10,j=1; i>=1 && j<=10; i--,j++)
{
    printf("\n %d \t %d", i,j);
}
```

Example program:-

```
/* Program to display numbers from 1 to 10 in ascending & descending order */
main()
{
    int i,j;
    clrscr();
    for(i=1,j=10; i<=10 && j>=1 ; i++,j--)
    {
        printf("\n %d \t %d", i,j);
    }
    getch();
}
```

OUTPUT:

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

4. Another Feature of for loop is that one or more sections can be omitted if necessary.

Ex:
```
i=1;
for( ; i<=10; i++)
{
  printf("\n %d", i);
}
```
⇒ Here, the initialization section is omitted in the for loop. The initialization has been done before the for loop.

Ex:
```
for(i=1; i<=10; )
{
  printf("\n %d", i);
  i=i+1;
}
```
⇒ Here, the increment section is omitted in the for loop. The increment section has been done inside the for loop.

Ex:
```
i=1;
for( ; i<=10 ; )
{
  printf("\n %d", i);
  i=i+1;
}
```
⇒ Here both initialization and increment sections are omitted in the for loop.

Note: Even if the sections are not present, the semicolons seperating the sections must remain.

5. Infinite loops:

⇒ If the test condition is not present, the for loop forms an infinite loop.

Ex:
```
for(i=1; ; i++)
{
  ------
}
```
⇒ Here there is no test condition. It leads to an infinite loop.

⇒ Ex: for (; ;) /* Infinite loop */

```
{
  ------
  ------
}
```

⇒
```
for(a=0; a<=10; )
{
   printf("%d", i);
}
```

→ This is also infinite loop, since a is neither increased nor decreased. So the condition is always true.

NESTED for LOOPS:-

⇒ We can also use loop within loops.

⇒ Nested for loops means, one for statement within another for statement.

Ex:

```
for(i=1; i<=10; i++)
{
  ------
  ------
  for(j=1; j<=5; j++)
  {
     -----
     -----
  }
  -----
  -----
}
```

inner loop

outer loop.

⇒ The number of iterations in this type of loops will be equal to the number of iterations in the outer loop multiplied by number of iterations in the inner for loop

⇒ In the above example, the outer loop will be executed 10 times and the inner loop will execute 5 times.

∴ Total number of iterations = 10x5 = 50 times.

⇒ The nesting may continue upto any desired level.

Example program:-

⇒ Write a program to display all the prime numbers between 1 to 100.

```
main()
{
  int i,j, count;
  clrscr();
  for(i=1; i<=100; i++)
  {
     count=0;
     for( j=1; j<=i; j++)
     {
        if(i%j==0)
        count++;
     }
     if(count==2)
     {
       printf("\n %d", i);
     }
  }
  getch();
}
```

⇒ Program to display the following pattern.

```
*
**
***
****
```

```
main()
{
  int i,j,n;
  clrscr();
  printf("Enter n value:");
  scanf("%d", &n);
  for(i=1; i<=n; i++)
  {
     for(j=1; j<=i; j++)
     {
       printf("*");
     }
     printf("\n");
  }
  getch();
```

ARRAYS

ARRAY:-

An array is a collection of homogeneous elements which shares the common name

(OR)

An array is a collection of similar data items.

⇒ All the elements in the array should be same data type.

SIGNIFICANCE:-

⇒ Ordinary variable can store only one value at a time. These variables can be useful to handle small amounts of data.

⇒ In many application, we need to handle large amounts of data.

For example consider we need to store marks of 60 students in a class. Using Ordinary variables we have to declare 60 variables to store marks of 60 students.

int m1, m2, m3, m4, ..., m60; → Because one variable can hold only one value at a time.

This increases program code. To avoid this it is better to use array.

By using arrays we can store marks of 60 students in a class under a single variable name.

Ex: int marks[60];

where marks is an array which stores marks of 60 students.

Examples where arrays can be used:

1. To store list of employees in an organization
2. List of temparatures recorded every hour in a day.
3. List of products
4. To store marks of students in a class.
5. List of customers and their phone numbers

⇒ Arrays can be used to represent a list of numbers or list of names.

ARRAY SUBSCRIPT:-

⇒ Subscript of an array is an integer expression (or) integer constant (or) integer variable that refers to the individual elements in the array.

Ex: int salary[100]; → subscript

→ The above array stores salaries of 100 employees in an organization.

→ We can access the individual salaries by writing index or subscript in brackets after the array name.

Ex: salary[5] → represents the salary of 5th employee

salary[30] represents the salary of 30th employee:

↳ subscript

CLASSIFICATION OF ARRAYS:-

⇒ We can use arrays to represent list of values and table of data in two, three or more dimensions.

⇒ Generally arrays are classified based on the dimensionality.

⇒ Dimensionality is determined by the number of subscripts present in an arr

⇒ Arrays are classified into: 1. One-dimensional arrays
2. Two-dimensional arrays
3. Multi-dimensional arrays.

⇒ If the array contains one subscript, then it is called one-dimensional arrays.
If the array contains two subscripts, it is called two-dimensional array.
Multi-dimensional arrays contains more than two subscripts.

ONE-DIMENSIONAL ARRAYS:-

⇒ One-dimensional arrays have only one subscript.

⇒ A list of items can be given one variable name using only one subscript and su

⇒ A list of items can be represented by a single variable name using only on subscript and such variables is called one-dimensional array (or) single-subs variable.

DECLARATION OF ONE-DIMENSIONAL ARRAYS:-

⇒ Arrays must be declared before they are used in the program.

Syntax: `data-type array-name[size];`

Where, Data-type → specifies type of elements that can be stored inside the arr such as int, float or char.

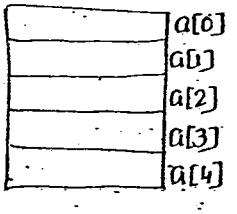
array-name → is a valid identifier

size → size indicates the maximum number of elements that can be stored inside the array.

Example: If we want to represent a set of 5 integer numbers by an array variable 'a', then we can declare the array as follows:

```
int a[5];
```

Now: the computer reserves five storage locations as shown below:



→ The values to the array elements can be assigned as follows:

a[0] = 10
a[1] = 20
a[2] = 30
a[3] = 40
a[4] = 50

⇒ The elements are stored in continuous memory locations.

a[0]	a[1]	a[2]	a[3]	a[4]
10	20	30	40	50
1000	1002	1004	1006	1008

⇒ The size of the array should be unsigned integer constant or a symbolic constant.

Ex: int number[10];
float height[50];
char name[10];

NOTE: Size should not be empty, it should not be negative value and floating-point value.

⇒ When we declare an array, the compiler allocates memory space based on the size and type of the array.

Total memory space allocated to an array = number of datatypes (bytes) occupied by the datatype * size of the array.

Ex: int a[10];

→ Total memory space allocated = 2*10 = 20 bytes.
(2 ↓ sizeof int, 10 ↓ no. of elements)

Ex: char name[10];

→ Total memory space allocated is: 1×10 = 10 bytes

Ex: float height[50];

→ Total memory space allocated = 4*50 = 200 bytes.

INITIALIZATION OF ONE-DIMENSIONAL ARRAYS:

⇒ After an array is declared, its elements must be initialized.

⇒ Assigning values to the array elements is known as initialization of arrays.

⇒ An array can be initialized at either of the following stages:

1. At compile time
2. At run time

1. Compile time initialization:-

⇒ We can initialize array elements at the place of their declaration itself.

Syntax: | Datatype array-name[size] = {list of values seperated by commas};

Example: int x[6] = {1, 2, 17, 14, 3, 8};
char name[5] = {'x', 'y', 'z', 'a', 'b'};

Accessing array elements:-

The individual elements of an array can be accessed by writing index or subscript in brackets after the array name.

Ex: int x[6] = {10, 100, 20, 30, 40, 50}

The subscript or index ranges from 0 to size-1.

i.e. x[0] = 10
x[1] = 100
x[2] = 20
x[3] = 30
x[4] = 40
x[5] = 50

Examples: int number[5] = {1, 2, 3, 4, 5};
Char C[4] = {'A', 'B', 'C', 'D'};
float height[3] = {5.1, 4.9, 6.3};

⇒ Suppose the list of elements specified are less than the size of the array, then that many elements are ~~assig~~ initialized. Remaining elements are assigned ~~garbage values~~ zero

Ex: int P[5] = {1, 2, 3};

P[0] = 1, P[1] = 2, P[2] = 3, P[3] = ~~garbage value~~ 0, P[4] = ~~garbage value~~ 0.

⇒ During compile time initialization, if we don't specify the size of the array, the compiler fixes the size of the array based on number of elements in the

Ex: int x[] = {1, 5, 7, 9};

Since the array x has 4 values in the list, the size of array is 4.

⇒ Any access to the array elements outside its boundaries (i.e. outside its limits) would not ~~access~~ cause any error. It results in unpredictable results.

Ex: int x[5] = {10, 20, 30, 40, 50};

Suppose if we try to access x[6] element, it will not show any error
↳ Here we are accessing beyond the limit

→ C performs no bounds checking. So you should ensure that the array indi are within the declared limits.

⇒ An array can be initialized at run time. This approach is used for initializing large arrays.

⇒ We can use scanf() function to initialize an array at runtime.

Ex: int x[3];

→ scanf("%d %d %d", &x[0], &x[1], &x[2]);

(OR)

```
for(i=0; i<3; i++)
{
   scanf("%d", &x[i]);
}
```

⇒ Reading:

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

Displaying:

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

Example program ①:-

⇒ Write a program to read and display array of 5 integers. Read array elements at compile time.

```
#include<stdio.h>
void main()
{
   int a[5] = {10, 20, 30, 40, 50}; /* Reading */
   clrscr();
   printf("%d %d %d %d %d", a[0], a[1], a[2], a[3], a[4]); /* Displaying */
   getch();
}
```

Example program ②

⇒ Write a program to read and display array of 5 real numbers (i.e. floating point value). Read array elements at run time.

```
#include<stdio.h>
void main()
{
   float a[5]; int i;
   printf("Enter 5 Real numbers:");
   for(i=0; i<5; i++)
   {
      scanf("%f", &a[i]);
   }
   printf("\n Array elements are:");
   for(i=0; i<5; i++)
      printf("\t %f", a[i]);
}
```

Example program ③

Write a program to display array elements in reverse order

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[5],i;
   printf("Enter the array elements:");
   for(i=0;i<5;i++)
   {
     scanf("%d",&a[i]);
   }
   printf("Array elements in reverse order:");
   for(i=4;i>=0;i--) /*Display elements from size-1 to 0 i.e. from 4 to 1)
                                (5-1)
   {
     printf("\t%d",a[i]);
   }
   getch();
}
```

output:

```
Enter the array elements:
10 20 30 40 50
Array elements in reverse orde
50 40 30 20 10
```

Example ④

Write a program to implement a one-dimensional array and prints its index and corresponding value

```
#include<stdio.h>
void main()
{
   int a[5],i;
   printf("Enter array elements:");
   for(i=0;i<5;i++)
   {
     scanf("%d",&a[i]);
   }
   printf("Index\t Value");
   for(i=0;i<5;i++)
   {
     printf("\n%d\t%d",i,a[i]);   a[i] is value
                                  i is index
   }
   getch();
}
```

Input: Enter array elements: 13 12 25 17 9

output:

Index	value
0	13
1	12
2	25
3	17
4	9

⇒Write a program to calculate sum of array elements

```
#include<stdio.h>
Void main()
{
   int a[5], i, sum=0;
   printf("Enter array elements:");
   for(i=0;i<5;i++)
   scanf("%d",&a[i]);
   for(i=0;i<5;i++)
   {
      sum=sum+a[i];
   }
   printf("Sum of array elements is:%d", sum);
}
```

Input: Enter array elements:
1 2 3 4 5

Output:
Sum of array elements is: 15

Example⑥-

⇒Write a program to calculate average marks obtained by 50 students

```
#include<stdio.h>
Void main()
{
   int marks[50], i, sum=0;
   float average;
   printf("Enter 50 students marks");
   for(i=0;i<50;i++)
   {
      scanf("%d", &marks[i]);
   }
   for(i=0;i<50;i++)
   {
      sum=sum+marks[i];
   }
   average=sum/50;
   printf(" Average marks is:%f", average);
}
```

Example⑦-

⇒Write a program to calculate the total prices of 25 items which are stored in prices array.

```
#include <stdio.h>
Void main()
{
   float prices[25], total=0.0;
   int i;
   clrscr();
   printf("Enter prices of 25 items:");
   for(i=0;i<25;i++)
   {
      scanf("%f", &price[i]);
   }
   for(i=0;i<25;i++)
   {
      total=total+prices[i];
   }
   printf(" total price is: %f", total);
   getch();
}
```

Example(5):-

⇒ Write a program to find smallest and largest number in a list of integers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a[5], i, large, small;
   clrscr();
   printf("Enter array elements:");
   for(i=0;i<5;i++)
   {
     scanf("%d",&a[i]);
   }
   small=a[0];
   large=a[0];
   for(i=1;i<5;i++)
   {
      if(a[i]>large)
         large=a[i];
      if(a[i]<small)
         small=a[i];
   }
   printf("\n Smallest element in the array is:%d", small);
   printf("\n Largest element in the array is:%d", large);
   getch();
}
```

Input:-

Enter array elements: 100 5 105 107

output:

Smallest element in the array is: 5

Largest element in the array is: 107

Some more programs:

Try to practice the following programs:

① Write a program to calculate the sum of floating point numbers

② Write a program to display only -ve elements of an array.

③ Write a program to display only even numbers in an array.

④ Write a program to calculate sum of positive numbers and sum of negative nur in an array.

⑤ Write a program to add two integer arrays.

⑥ Write program to read and display a character array.

⇒ Write a program to sort the array elements in ascending order.

```
#include<stdio.h>
Void main()
{
   int a[10], n, i, temp, j;
   clrscr();
   printf("\n Enter number of array elements:");
   scanf("%d", &n);
   printf("\n Enter array elements:");
   for(i=0; i<n; i++)
   {
      scanf("%d", &a[i]);
   }
   for(i=0; i<n; i++)
   {
      for(j=i+1; j<n; j++)
      {
         if(a[i]>a[j])
         {
            temp=a[i];
            a[i] = a[j];
            a[j] =temp;
         }
      }
   }
   printf("\n Array elements in ascending order:\n");
   for(i=0; i<n; i++)
   {
      printf("\t %d", a[i]);
   }
   getch();
}
```

Input: Enter the number of array elements: 5

Enter array elements: 4 1 6 5 3

output: Array elements in ascending order: 1 3 4 5 6

⇒ Write a program to sort the array elements in descending order.

Logic:
```
for(i=0;i<n;i++)
{
  for(j=i+1;j<n;j++)
  {
    if(a[i]<a[j])
    {
      temp=a[i];
      a[i]=a[j];
      a[j]=temp;
    }
  }
}
```

⇒ Write a program to find k^{th} smallest element in an array

Logic:
1. Read array elements
2. Arrange the array elements in ascending order.
3. Print k^{th} element of sorted array.

⇒One-dimensional arrays can store list of values.

⇒There are situations, where we want to store a table of values. Two-dimensional arrays are used to represent or store table of values.

Two-dimensional array:

An array which contains two subscripts is known as two-dimensional array.

Example: Matrix

↳ Matrix contains two dimensions i.e. Rows & Columns.

$$A = \begin{array}{c} \\ \text{Row } 0 \\ \text{Row } 1 \end{array} \begin{array}{c} \begin{array}{cc} \text{Column } 0 & \text{Column } 1 \end{array} \\ \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \end{array}$$

In matrix, a particular element is represented by two subscripts. For example A_{ij} refers to the value in the i^{th} Row and j^{th} column in matrix A.

$A_{00} = 1$, $A_{01} = 2$, $A_{10} = 3$, $A_{11} = 4$

⇒In C language matrix can be represented by two-dimensional arrays.

DECLARATION OF TWO-DIMENSIONAL ARRAYS:-

⇒Two-dimensional arrays are declared as follows:-

Syntax: `datatype array-name[rowsize][columnsize];`

Ex: `int a[3][3];` → Contains 3 rows and 3 columns
`int b[3][2];` → Contains 3 rows and 2 columns
`char c[5][4];` → Contains 5 rows and 4 columns.

⇒Total number of elements in a two-dimensional array is = Rowsize * Columnsize

Space occupied by the two-dimensional array = number of bytes occupied by the datatype * total number of elements.

Example: `int a[5][4];`
total number of elements = 5*4 = 20
Space = 2*20 = 40 bytes.

⇒Each dimension of the array is indexed from 0 to its maximum-size − 1.

⇒The first index selects the row and second index selects column within that row.

INITIALIZATION:-

1. Compile time initialization:-

⇒ Initializing two-dimensional arrays at the time of declaration itself.

Ex: int table[2][3] = {0,0,0,1,1,1};

The above statement initializes first row to 0 and second row to 1. The initi
ion is done row by row. i.e. $\begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$

⇒ We can also initialize two-dimensional array in the form of matrix as shown bel

Ex: int table[2][3] = { {0,0,0}, {1,1,1} };

int a[3][2] = { {4,5}, {1,7}, {8,9} };

⇒ When the array is completely initialized with all values, then no need to specify size of the first dimension.

Ex: int a[][3] = { {4,5,6}, {10,17,1} };

⇒ If the values are missing in initialization, they are automatically set to 0.

Ex: int a[2][3] = { {1,2}, {4} };

a[0][0] = 1 a[1][0] = 4
a[0][1] = 2 a[1][1] = 0
a[0][2] = 0 a[1][2] = 0

Accessing two-dimensional arrays:-

⇒ Individual elements of two-dimensional arrays can be accessed by using two dimensions.

Ex: a[i][j] represents elements in i^{th} row and j^{th} column.

array name ← a[ ][ ] → Row, Column

Ex: int a[2][2] = { {10,20}, {30,40} }

(00 = 10, 01 = 20, 10 = 30, 11 = 40)

	0	1
0	10	20
1	30	40

i.e. a[0][0] = 10
a[0][1] = 20
a[1][0] = 30
a[1][1] = 40

Arrangement of two-dimensional array elements in memory:-

⇒ Either it is a two-dimensional array or one-dimensional array, all the array elements are stored in continuous memory locations.

Ex: int a[2][2] = { {10,20}, {30,40} };

a[0][0]	a[0][1]	a[1][0]	a[1][1]
10	20	30	40

3. Run-time initialization:-

⇒ Two-dimensional arrays can be initialized at run time using scanf function.

Reading:-

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

Displaying:-

```
for(i=0;i<rowsize;i++)
{
   for(j=0;j<columnsize;j++)
   {
      printf("%3d",a[i][j]);
   }
   printf("\n");
}
```

Example program:-

⇒ Write a program to read and display 3x3 matrix.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
   int a[3][3],i,j;
   clrscr();
   printf("Enter a 3x3 matrix:");
   for(i=0;i<3;i++)
   {
      for(j=0;j<3;j++)
      {
         scanf("%d",&a[i][j]);
      }
   }
   printf("The entered matrix is:\n");
   for(i=0;i<3;i++)
   {
      for(j=0;j<3;j++)
      {
         printf("%3d",a[i][j]);
      }
      printf("\n");
   }
   getch();
}
```

Reading

displaying

Input:-

Enter a 3x3 matrix: 1 4 5
7 8 9
10 2 3

Output:

The entered matrix is:

1 4 5
7 8 9
10 2 3

⇒ Example program:-

⇒ Write a program to perform addition of two matrices.

```
#include<stdio.h>
Void main()
{
   int a[10][10], b[10][10], c[10][10], r1, r2, c1, c2, i, j;
   clrscr();
   printf("Enter number of rows & columns in matrix a:");
   scanf("%d %d", &r1, &c1);
   printf("\n Enter matrix a:");
   for(i=0; i<r1; i++)
   {
      for(j=0; j<c1; j++)
      {
         scanf("%d", &a[i][j]);
      }
   }
   printf("\n Enter number of rows and columns in matrix b:");
   scanf("%d %d", &r2, &c2);
   printf("\n Enter matrix b:");
   for(i=0; i<r2; i++)
   {
      for(j=0; j<c2; j++)
      {
         scanf("%d", &b[i][j]);
      }
   }
   if(r1==r2 && c1==c2)
   {
      printf("matrix addition is possible");
      for(i=0; i<r1; i++)
      {
         for(j=0; j<c1; j++)
         {
            c[i][j]=a[i][j]+b[i][j];
         }
      }
      printf("\n Addition of two matrices:\n");
      for(i=0; i<r1; i++)
      {
         for(j=0; j<c1; j++)
         {
            printf("%3d", c[i][j]);
         }
         printf("\n");
      }
   }
   else
   {
      printf("matrix addition is not possible");
   }
   getch();
}
```

```
#include<stdio.h>
void main()
{
 int a[10][10], b[10][10], c[10][10], i, j, k;
 int r1, r2, c1, c2;
 clrscr();
 printf("Enter no.of rows & columns in a:");
 scanf("%d %d", &r1, &c1);
 printf("\n Enter matrix a:");
 for(i=0; i<r1; i++)
 {
   for(j=0; j<c1; j++)
   {
     scanf("%d", &a[i][j]);
   }
 }
printf("\n Enter no.of rows & columns in b:");
 scanf("%d %d", &r2, &c2);
printf ("\n Enter matrix b:");
 for(i=0; i<r2; i++)
 {
   for(j=0; j<c2; j++)
   {
     scanf("%d", &b[i][j]);
   }
 }
if( c1==r2)
{
   printf("matrix multiplication is possible");
   for(i=0; i<r1; i++)
   {
      for(j=0; j<c2; j++)
      {
         c[i][j]=0;
         for(k=0; k<c1; k++)
           c[i][j]=c[i][j]+a[i][k]*b[k][j];
      }
   }
   printf("\n Resultant matrix is:\n");
   for(i=0; i<r1; i++)
   {
      for(j=0; j<c1; j++)
      {
        printf("%4d", c[i][j]);
      }
      printf("\n");
   }
}
else
{
 printf("matrix multiplication is not possible.
}
getch();
}
```

⇒ Write a program to generate transpose of a matrix.

```
#include<stdio.h>
Void main()
{
  int a[3][3], b[3][3], i, j;
  clrscr();
  Printf("Enter a matrix:");
  for(i=0; i<3; i++)
  {
    for(j=0; j<3; j++)
     scanf("%d", &a[i][j]);
  }
  for(i=0; i<3; i++)
  {
     for(j=0; j<3; j++)
     {
        b[i][j] = a[j][i];
     }
  }
  Printf("Transpose of given matrix is:\n");
  for(i=0; i<3; i++)
  {
     for(j=0; j<3; j++)
     {
        printf("%d", b[i][j]);
     }
     printf("\n");
  }
  getch();
}
```

Input:

Enter a matrix: 1 2 3
4 5 6
7 8 9

Output:

Transpose of given matrix is:

1 4 7
2 5 8
3 6 9

HOME WORK:-

① Write a program to perform subtraction of two matrices.

② Write a program to calculate sum of all elements in a matrix.

③ Write a program to display diagnol elements and its sum in a matrix.

↳ if(i==j)
printf("%d", a[i][j]);

a[1][1]
a[2][2]
a[3][3]

④ Write a program to find whether the given matrix is symmetric or not

$A = A^T$ ⇒ symmetric

MULTI-DIMENSIONAL ARRAYS:-

⇒ An array which contains more than two subscripts is known as multi-dimension array.

⇒ Multi-dimensional arrays are used to represent survey data.

Declaration:-

Multi-dimensional arrays are declared as follows:

Syntax: datatype array-name [S1][S2][S3].....[Sm];

→ where Si is the size of i^{th} dimension.

Examples:- int survey[3][5][12];

float table[5][4][3][4];

Here survey is a 3-dimensional array, which contains 180 integer elements and table is a 4-dimensional array.

Example①:-

To represent a survey data of rainfall during the last 3 years in 5 cities from January to december, the array can be declared as follows:

```
int survey[3][5][12];
        ↓  ↓  ↓
      year city month
```

year1

month / City	1	2	3	...	12
1					
2					
3					
4					
5					

year2

month / City	1	2	3	...	12
1					
2					
3					
4					
5					

year3

month / City	1	2	...	12
1				
2				
3				
4				
5				

For example, the element survey[2][3][10] represents the rainfall in the month of october during the second year in city 3.

Example②:-

To represent gold rates during the last 2 years for from January to december every day, the array can be declared as follows:

int gold_rate[2][12][31];

INITIALIZATION:-

1. Compile time initialization:-

Multi-dimensional arrays can be initialized at compile time

Ex: int a[3][3][3] = { {{1,2,3}, {4,5,6}, {7,8,9}},
{{1,1,1}, {1,7,9}, {10,6,5}},
{{4,2,8}, {1,5,11}, {3,6,9}}
};

2. Run-time initialization:-

Multi-dimensional arrays can be initialized at run time using scanf function

Ex:
```
int x[3][2][5];
for(i=0; i<3; i++)
{
   for(j=0; j<2; j++)
   {
      for(k=0; k<5; k++)
      {
         scanf("%d", &x[i][j][k]);
      }
   }
}
```