Data structure:-

* A data structure is a specialised format which is used for organising and storing the data in a proper manner.

* Usually data structures are classified into 2 types they are 1) linear data structures
2) non-linear data structures

Linear data structures:- These data structures will organise and store the data in a linear manner (or) sequencial order

Ex:- Arrays, stacks, queues, linked lists

Non-linear data structures:- These data structures will organise and store the data in a non-linear manner (or) random order

ex:- Trees, Graphs.

* Data structures are mainly used in computer memory to perform various tasks by the computer the computer will perform different opeations by using different data structures inside its memory

Data:- whatever we are entering into computer is known as data.

Process:- the data in the execution is known as process

Information:- The processed data is known as Information

Program:- A program is a set of instructions.

Algorithm:- An algorithm is a step by step process of a program (or) problem

Software:- A software is a set of programs.

Ex:- MS-office, OS, Unix, Linux etc...

Hard ware:- Hard ware is an inter connection of various hard ware devices such as mouse, keyboard, hard disk etc..

Language:- A language is a medium between user and computer the language has been divided into two types in the computer they are 1) low level language 2) high level language

Low level language:- It is the combination of two languages such as machine level language and assembly level language

Machine level language:- It is computer understandable language and it can also be called as binary language why because it will be in the form of 0's and 1's.

Ex:- 1010 → 10
1000 → 8

Assembly level language:- It can be understable by human and it can also be called as symbolic language why because it would be in the form symbolic notations.

Ex:- load, XCH, MOV, Add, sub etc...

High level language :- It would be in the form of english statements by combining numericals, alphabets and special characters.

* In the high level language has be divided into 2 types

1) procedure oriented programing language
2) object oriented programing language

procedure oriented programing language :- In the procedure oriented programing language a problem (or) program will be divided into group of functions

Ex :- C, COBAL, PASCAL, FORTRAN, BASIC etc..

object oriented programing language :- In this problem (or) program is divided into group of objects

Ex :- C++, JAVA, .Net, small talks, etc..

Translator :- A translator is a software program which converts source language instructions into destination language instructions

Compiler :- compiler is a translator which converts high level language instructions into machine level language instructions and it also compile (or) check the errors in the program.

Assembler :- It is also one of the translator which converts assembly language instructions into machine language instructions.

UNIT-I

Arrays

Basic concepts of OOP's:- The basic concepts of Oop's have been classified into six major types

1) class
2) object
3) Inheritance
4) polymorphism
5) Encapsulation
6) Abstraction

class:- A class is a collection of data members (variables (or) attributes (or) fields (or) properties) and member functions (functions (or) operations)

* In order to create a class use a keyword known as class
* usually the class is used to generate user defined data type. through which objects will be created

Syntax:-

```
class classname
{
    private:
        Data members;
        member functions;
    protected:
        Data members;
        member functions;
    public:
        Data members;
        member functions;
};
```

private, protected, public } access specifiers

Private:- If we declare the data members and member functions under private access specifier then they can be accessed with in the class only.

protected:- If we declare the data members and member functions under protected access specifiers then they can be accessed with in the class and inherited class

public:- If we declare the data members and member functions under public access specifiers then they can be accessed with in the class and inherited class and outside the class.

Student:-

Ex:-
```
class student
{
private:
    int sno;
    char sname;
    void scount();
protected:
    ----
    ----
public:
    ---
    ---
};
```

Object:- Each class variable can be called as an object

* An object is mainly used to invoke (call) the member functions for their execution.

Syntax:- classname obj name;

Ex:- Student s1;

Simple Program for class & object:-

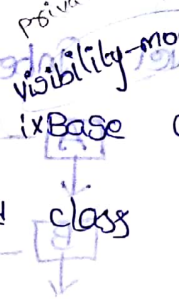
```
#include<iostream.h>
#include<conio.h>
class person
{
 char name[30];
 int age;
public:
  void getdata();
  void display();
};
void person::getdata()
{
 cout<<"Enter the name:";
 cin>>name;
 cout<<"Enter the age:";
 cin>>age;
}
void person::display()
{
 cout<<"\n Name is:"<<name;
 cout<<"\n Age is:"<<age;
}
int main()
{
 person p;
 p.getdata();
 p.display();
 return 0;
}
```

Inheritance:- The process of creating a new class from already existed class is known as an Inheritance.

where the existed class can be called as Base class (or) parent class (or) Super class and the new class can be called as derived class (or) child class (or) sub class.

Syntax:-

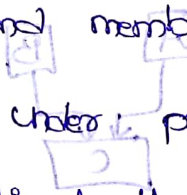
```
class derived class-name :visibility-mode Base class-name
{
  ....  //members of derived class
}
```



Inheritance is mainly used for achieving code reusability

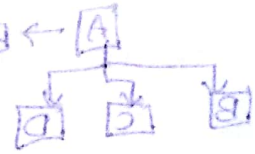
Note:- If we not declare data members and member functions under another any access specifiers in the class then those data members and member functions will be considered by default under private access specifier

* The visibility mode is optional that may be either private (or) public

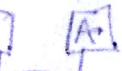


Types of Inheritance:- An Inheritance has been classified into 5 types

1) Single level Inheritance
2) Multi-level Inheritance
3) Multiple inheritance
4) Hierarchical Inheritance
5) Hybrid Inheritance

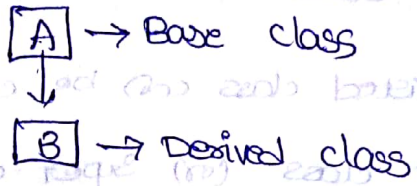


Single level Inheritance:-

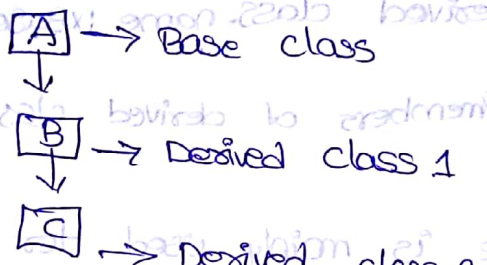


If a class is derived from only single base

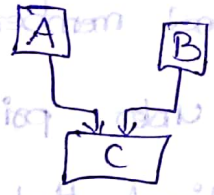
class then such inheritance is known as Single level Inheritance.



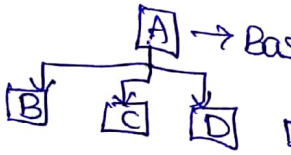
Multi-level Inheritance:- If a class is derived from already derived class then such Inheritance is known as multi-level Inheritance



Multiple Inheritance:- If a class is derived from more than one base class then such type of Inheritance is known as multiple Inheritance.



Hierarchical Inheritance:- If more no. of derived class are derived from single base class such as Inheritance is known as hierarchical Inheritance.



Hybrid Inheritance:- combination of more than one Inheritances is called hybrid inheritance

Example program for Single level Inheritance:-

```
#include<iostream.h>
#include<conio.h>
class A
{ public:
  int a,b;
public:
   void getdata();
   };
class B: public A
  {
    int c,d;
  public:
    void add();
    void sub();
    void display();
  };
void A::getdata()
  {
  cout<<"Enter a value:"<<endl;
  cin>>a;
  cout<<"enter b value:"<<endl;
  cin>>b;
  }
void B::add()
  {
  c=a+b;
  }
void B::sub()
  {
  d=a-b;
  }
void B::display()
  {
  cout<<"Addition is:"<<c<<endl;
  cout<<"substraction is:"<<d;
  }
int main()
{
B obj;
clrscr();
obj.getdata();
obj.add();
obj.sub();
obj.display();
getch();
return 0;
}
```

\n occupies space in the memory

endl does not occupies

Polymorphism:-

* The polymorphism is a greats term and which is used to represent single form into more than one form
* The polymorphism has been divided into two types they are 1) compile time polymorphism
2) Run time polymorphism

Compile time polymorphism:-

If the polymorphism is done during the compilation time then such polymorphism is said to be compile time polymorphism.

Eg:- function overloading, operator overloading.

Run time Polymorphism:-

If the polymorphism is done during the run time then such polymorphism is said to be run time polymorphism

Eg:- Virtual functions.

Encapsulation:- The process of wrapping (or) grouping the data members and members functions into a single unit (class) is known as encapsulation

* The encapsulation concept can we acheived by creating class
* Using the encapsulation we can provide security for the data members and member functions. by making them as private in the class.

Abstraction:- The process of hiding unnecessary information inside and revealing (or) displaying required information outside is known as abstraction.

Abstract data types:-

* An abstract data type is a collection of data elements and associated operations
* An abstract data type is also a user defined data type like as class
* Usually Arrays, linked lists, stacks, queues, sets, Graphs are said to be abstract data types.

→ for stack ADT (Abstract data type) basic operations may be push & pop

→ for the queue ADT basic operations may be enqueue and Dequeue

→ for the set ADT the basic operations may be union, intersection, complement and size etc..

→ for the graph ADT the basic operations may be no.of vertices, no.of edges etc.

- the main purpose of abstract data type is write the operations of ADT once in the program and perform their from any part of the program by calling their appropriate functions

Polynomial representation using arrays:-

A polynomial is an expression which contains more than one term.

means combination of operators & operands

Each term may be made with one (or) more exponents and co-efficients

Ex:-

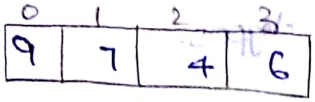
$$P(x) = 6x^3 + 4x^2 + 7x + 9$$

In order to represent single variable polynomial we

use single dimension array

similarly to represent multivariable polynomial we use multi-dimensional array.

The indexes of the array can be considered as exponents of the polynomial and the coefficients will be stored at particular indexes of the array



Array

// Arrays as an abstract data type:-

An array is the basic abstract data type because which contains collection of homogeneous elements and associated operations such as traversing, Insertion, Deletion, searching, sorting, Merging

The array elements can be stored in consecutive locations in the memory and they can be accessed by using integer index set and the array index starts with 'zero'.

The no. of elements stored in the array will be called as the length of the array and which can be determined by the following formula

$$\text{Length} = UB - LB + 1$$

where UB = upper boundary index

LB = lower boundary index

The elements of the array can be denoted using different notations

subscript notation $(A_0, A_1, A_2 \ldots)$

parantheses (A(0), A(1), A(2), ...)

Bracket notation (A[0], A[1], A[2], ...)

let A[K] be the array and 'K' is an index (or) subscript

Representation in the of arrays in the memory

Consider A[K] be the array and loc(A[K]) is the x location addr

Usually the elements of the array will be stored in consecutive memory location in the computer

A[ ] = {10, 20, 30, 40, 50};

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

Base — 1000 1001 1002 1003 1004

* The computer does not need to keep the addresses of all the elements in the array but needs to keep only the address of first element in the array. which is known as base address of the array and which can be denoted by Base(A)

* the computer can easily calculate the location address of any element using the base address of the array by the following formula.

$$Loc(A[K]) = Base(A) + W(K - LB)$$

W = memory space allocated for each element

K = index

LB = lower boundary index

Construct the elements A[6] be the array

A[6] = {10, 20, 30, 40, 50}; and the memory space is allocated for each element is 1 byte

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]
10	20	30	40	50	
100	101	102	103	104	105

find out location address of (A[4])

$Loc(A[K]) = Base(A) + w(K - LB)$

$Loc(A[4]) = 100 + 1(4 - 0)$

$= 100 + 4$

$LOC(A[4]) = 104$

// Polynomial addition:-

Algorithm:-

Assume these are two polynomials P & Q

The addition of two polynomials is stored in third polynomial Sum

Step (1) Start

Step (2) while P & Q are not null, repeat Step 3

Step (3) If the powers of two terms of the polynomi are equal then

Insert the addition of two terms into Sum polynomial

Else if the x power of first polynomial > x power of secon polynomial then, Insert the term of first polynomial int Sum polynomial

Else

Insert the term of second polynomial into Sum polynom

Step (4)

Copy the remaining terms from the two polynomials into Sum polynomial

Step (5)

Print Sum polynomial

Step (6)

Stop.

Eg:- $P(x) = 7x^5 + 5x^2 + x$

$Q(x) = 5x^4 + 4x^3 + 2x^2 + 3x - 14$

$Sum(x) = 7x^5 + 5x^4 + 4x^3 + 7x^2 + 4x - 14$

Program-

```
# include <iostream.h>
# include <conio.h>
void main( )
{
 int  a[6], b[6], c[6] , i, flag=0;
 clrscr( );
 for(i=0; i<6; i++)
 {
  a[i]=0 ;
  b[i] =0;
 }
 a[1] =1; a[2]= 5; a[5] = 7;
 b[0] = -14; b[1]= 3; b[2]= 2;
 b[3] =4; b[4] =5;
   for (i=5 ; i>0; i--)
   {
    if(a[i] ! =0 && i>0)
    {
     cout <<a[i] <<"x" <<i;
      flag = 1;
    }
  if(i>0&&a[i-1]!=0&&flag==1)
     cout << "+";
     if (a[i]! =0& & i==0)
     cout <<a[i];
    }
    flag=0;
```

flag = variable

a[6]

a[5]	a[4]	a[3]	a[2]	a[1]	a[0]
7	0	0	5	1	0

$7x^5 + 5x^2 + 1x^1$

b[6]

b[5]	b[4]	b[3]	b[2]	b[1]	b[0]
0	5	4	2	3	-14

$5x^4 + 4x^3 + 2x^2 + 3x^1 - 14$

c[6]

c[5]	c[4]	c[3]	c[2]	c[1]	c[0]
7	5	4	7	4	-14

$7x^5 + 5x^4 + 4x^3 + 7x^2 + 4x^1 - 14$

```
cout<<"second polynomial is:";
for (i=5; i>=0; i--)
{
 if (b[i]!=0 && i>0)
   {
    cout << b[i] <<"x"<<i;
        flag=1;
       }
    if (i>0 && b[i-1]>0 && flag ==1)
    cout<<"+";
    if (b[i]!=0 && i==0)
    cout << b[i];
   }
  cout<<" polynomial addition is:";
      for (i=0; i>0; i++)
          {
           c[i] = a[i]+b[i];
          }
cout<<"The resultant polynomial is:";
 for (i=5; i>=0; i--)
 {
  if (c[i]!=0 && i>0)
  {
   cout <<c[i] <<"x"<<i;
    flag=1;
  }
 if (i>0 && c[i-1]>0 && flag==1)
   cout<<" + ";
  if(c[i]!=0 && i==0)
   cout<<c[i];
 }getch();
```

Eg:- a[6]

a[5]	a[4]	a[3]	a[2]	a[1]	a[0]
7	0	0	5	1	0

$\Rightarrow 7x5 + 5x2 + 1x1$

b[6]

b[5]	b[4]	b[3]	b[2]	b[1]	b[0]
0	5	4	2	3	-14

$\Rightarrow 5x4 + 4x3 + 2x2 + 3x1 - 14$

c[6]

c[5]	c[4]	c[3]	c[2]	c[1]	c[0]
7	5	4	7	4	-14

$\Rightarrow 7x5 + 5x4 + 4x3 + 7x2 + 4x1 - 14$

Matrix Multiplication:-

```
#include<iostream.h>
#include <conio.h>
  int main()
  {
int a[10][10], b[10][10], c[10][10];
int m,n,p,q, i,j,k;
clrscr();
cout<<"Enter the no.of rows & columns of first matrix:"<<
 endl;
cin >>m>>n;
cout<<"Enter the no.of rows & columns of second matrix;"<<endl;
cin>>p>>q;
if (n==p)
{
cout<<"Enter the elements of first matrix is:"<<endl;
for (i=0; i<m; i++)
 {
 for (j=0; j<n; j++)
  {
   cin >>a[i][j];
 }}
```

```
cout<<" Enter the element of second matrix is:"<<endl;
for(i=0; i<p; i++)
{
 for(j=0; j<q; j++)
 {
  cin>>b[i][j];
 }
}
cout<<"the matrix A is:"<<endl;
 for(i=0; i<m; i++)
  {
  for(j=0; j<n; j++)
   {
   cout<<" "<<" "<<" "<<a[i][j]<<endl;
   }
  }
  cout<<"the matrix B is:"<<endl;
   for(i=0; i<p; i++)
    {
     for(j=0; j<q; j++)
      {
       cout<<" "<<" "<<" "<< b[i][j]<<endl;
      }
    }
    cout<<"The resultant matrix is:"<<endl;
     for(i=0; i<m; i++)
     {
      for(j=0; j<q; j++)
      {
       c[i][j]=0;
       for(k=0; k<p; k++)
       {
       c[i][j] = c[i][j] + (a[i][k] * b[k][j]);
       }
```

o/p:-

```
enter the no.of rows and columns of first matrix:
3 3
enter the no.of rows and columns of second matrix:
3 3
enter the elements of first matrix is:
1 4 5
2 3 6
7 8 9
enter the elements of second matrix is:
4 5 6
1 0 3
2 6 9
the matrix A is:
1
4
5
2
3
6
7
8
9
the matrix B is:
4
5
6
1
0
3
2
6
9
The resultant matrix is:
18
35
63
23
46
75
54
89
147
```

```
        }
    }
    cout<<" "<<" "<<" "<<c[i][j]<<endl;
    }
else
cout<<"matrix multiplication is not possible"<<endl;
  getch();
  return 0;
}
```

Sparse Matrix:-

* Usually the two dimensional array can be used to represent Matrices.
* If a matrix contains more number of zero values than non zero values. then such type of matrix can be called as "sparse matrix."
* In contrast, If a matrix contains more number of non zero values than zero values then such type of matrix can be called as "dense matrix."

Eg:- consider a matrix size of 5x6 & contains 6 non-zero elements

	0	1	2	3	4	5
0	0	0	0	0	9	0
1	0	8	0	0	0	0
2	4	0	0	2	0	0
3	0	0	0	0	0	5
4	0	20	02	0	0	0

* when the sparse matrix is represented in 2 dimensional array then while implementing in computers memory, A lot of memory space will be wasted and C.P.U access time is also increased while accessing non-zero elements

by scaning all the elements (0 and non-zero elements) in the sparse matrix

* In order to avoid memory space wastage and to decrease cpu access time we represent the sparse matrix into two ways they are 1) Triplet representation
2) linked representation

Triplet Representation:-

* In the Triplet representation we consider only non-zero values along with their rows and columns
* In this representation, the 0th row stores the total no.of rows, total no.of columns and total no. non-zero values existed in the sparse matrix
From the above example the Sparse matrix can be represented in triplet representation as

	0	1	2	3	4	5
0	0	0	0	0	9	0
1	0	8	0	0	0	0
2	4	0	0	2	0	0
3	0	0	0	0	0	5
4	0	2	0	0	0	0

Row	column	value
5	6	6
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2

Linked representation:-

* In the linked representation the sparse matrix can be represented using linked lists.
In this, the linked lists use two different nodes they are
1. Header node
2. element node

Header Node:- The header node can be used to stored (or) specify index values which can be either row wise (or) column wise

* The header node contains 3 fields such as Index value, down, right fields

Header node

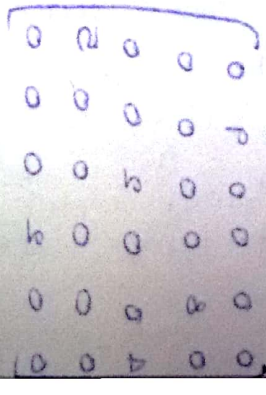
Index value	
down	right

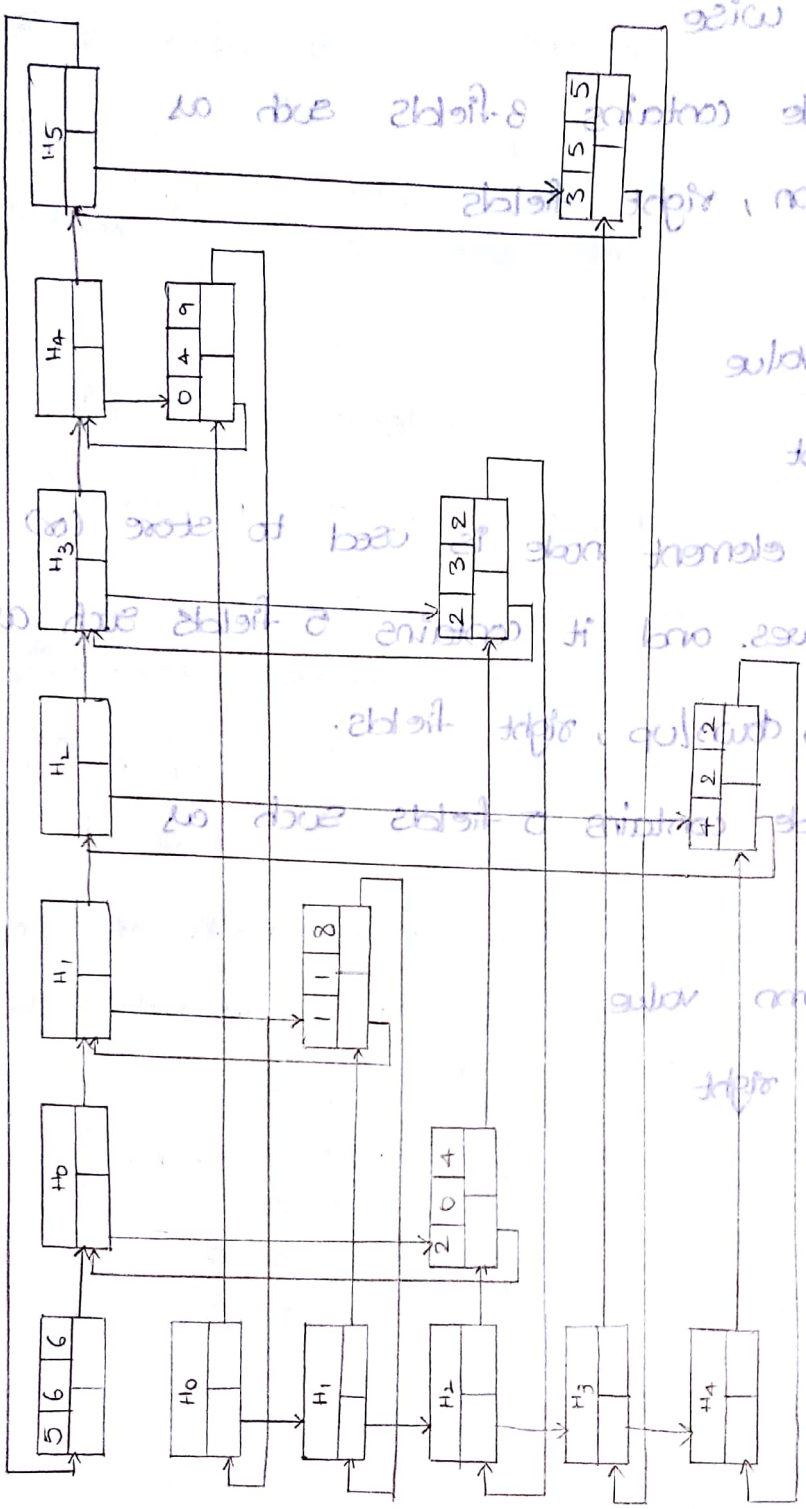
Element node:- An element node is used to store (or) specify non-zero values. and it contains 5 fields such as row, column, value, down/up, right fields.

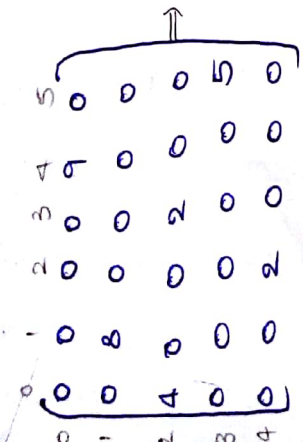
* The element node contains 5 fields such as

Element node:-

row	column	value
down/up	right	




H5
H4
H3
H2
H1
H0
H0
H1
H2
H3
H4



Transposing from Triplet Sparse Matrix:-

```
#include<iostream.h>
#include<conio.h>
Void main()
{
 int S[7][3], t[7][3];
 int r,c,nzvs,i;
 clrscr();
 cout<<" Enter no.of rows, columns and non zero values of
      Sparse triplet from matrix:"<<endl;
 cin>>r>>c>>nzvs;
  S[0][0]=r;
  S[0][1]=c;
  S[0][2]=nzvs;
 cout<<" Reading the elements of sparse triplet from matrix:"
                                                  <<endl;
for(i=0;i<=nzvs;i++)
{   cout<<"Enter the next triplet (row, column, value):"<<endl;
    cin>>S[i][0]>>S[i][1]>>S[i][2];
cout<<"The Sparse triplet from matrix is:"<<endl;
for(i=0;i<=nzvs;i++)
{
cout<<"rows"<<"\t"<<"columns"<<"\t"<<"values"<<endl;
cout<<S[i][0]<<"\t"<<S[i][1]<<"\t"<<S[i][2]<<endl;
}
cout<<"transposing Sparse triplet from matrix is:"<<endl;
for(i=0;i<=nzvs;i++)
 {
  t[i][0]=S[i][1];
  t[i][1]=S[i][0];
  t[i][2]=S[i][2];
 }
cout<<" the transposed sparse triplet from matrix is:"<<
                                                  endl;
```

```
for(i=0; i<=nzvs;i++)
{
 cout<<"row"<<"\t"<<"column"<<"\t"<<"value"<<endl;
 cout<<t[i][0]<<"\t"<<t[i][1]<<"\t"<<t[i][2]<<endl;
}
getch();
}
```

Olp:-

Enter no of rows, columns and non zero values of sparse triplet fo

matrix: 5 6 6

reading the elements into sparse triplet form matrix:

enter the next triplet (row, column value):
0 4 9

enter the next triplet (row, column value):
1 1 8

enter the next triplet (row, column value):
2 0 4

enter the next triplet (row, column value):
2 3 2

enter the next triplet (row, column value):
3 5 5

enter the next triplet (row, column value):
4 2 2

enter the next triplet (row, column value):
6 6 3

rows	columns	values
0	4	9
1	1	8
2	0	4
2	3	2
3	5	5
4	2	2
6	6	3

transposing sparce triplet from matrix is:
the transposed sparce triplet from matrix is:

	0	1	2	3	4	5	6
0	0	0	4	0	0	0	0
1	0	8	0	0	0	0	0
2	0	0	0	2	0	0	0
3	0	0	2	0	0	0	0
4	9	0	0	0	0	0	0
5	0	0	0	5	0	0	0
6	0	0	0	0	0	0	3

Representation of Arrays:-

* An array is a collection of homogeneous elements

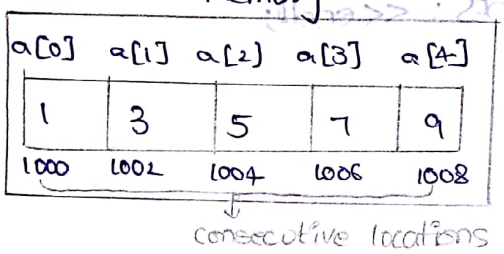
Usually arrays are classified into two major types

1. Single dimensional array
2. Multi dimensional array

Representation of single dimensional array in the memory:-

* The single dimensional array can be represented in the memory as in the form of consecutive locations which mean that the elements of single dimensional array will be stored in the memory in consecutive locations

Eg:- int a[] = {1,3,5,7,9};



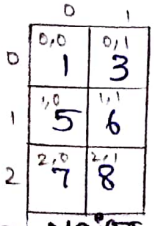
Representation of multi dimensional array in the memory:-

* The multi dimensional array can be represented in the memory in the form of matrix by using 2 types of forms they are i) Row-major order
ii) column-major order

Row-Major order:-

In this multi-dimensional array elements will be stored (or) represented row by row

Eg:- int a[3][2];



Column Major order:-

In this the elements of multi-dimensional array can be represented column by column in row wise

Eg:- int a[3][2]

o/p →

	0	1	2
0	1	5	7
1	3	6	8

Input :

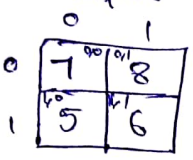
1 3
5 6
7 8

Program to illustrate row-major order & column major-order

```
#include <iostream.h>
#include <conio.h>
 void main( )
{
int a[10][10], m,n,i,j;
clrscr( );
cout<<" Enter the no.of rows & columns :"<<endl;
cin >> m>>n;
cout<<"Enter the elements :"<<endl;
for (i=0; i<m; i++)
{
 for (j=0; j<n ; j++)
 {
 cin>>a[i][j];
}
}
cout<<" the row-major order is:"<<endl;
for (i=0;  i<m; i++)
{
 for(j=0 ;j<n ; j++)
 {
 cout <<a[i][j] <<"\t";
 }
 cout <<endl;
}
cout<<" the column -major order is:"<<endl;
for(i=0;i<m; i++)
{
```

```
for (j=0;j<m; j++)
 {
 cout<< a [j][i]<<"\t";
 }
 cout<<endl;
 }
 getch();
}
```

O/p:- Input



O/p:- Enter the no.of rows and columns:
2 2
Enter the elements:
7 8
5 6

Row-major order is:

7 8
5 6

column-major order is:

7 5
8 6