ALAGAPPA UNIVERSITY

(Accredited with 'A+' Grade by NAAC (with CGPA: 3.64) in the Third Cycle and Graded as category - I University by MHRD-UGC)
(A State University Established by the Government of Tamilnadu)

KARAIKUDI – 630 003

DIRECTORATE OF DISTANCE EDUCATION

B.Sc. (COMPUTER SCIENCE)

Second Year – Third Semester

13034 – DATA STRUCTURE AND ALGORITHIM

Author:
Dr. A. Sumathi,
Assistant Processor,
Department of Computer Science,
SRC, Sastra University,
Kumbakonam -612 001.

CONTENTS

C++ Introduction

C++ is a **multi-paradigm programming language** that **supports object-oriented programming (OOP)**, created by **Bjarne Stroustrup** in **1983** at **Bell Labs**, C++ is an extension(superset) of C programming and the programs are written in C language can run in C++ compilers.

Uses of C++

C++ is used by programmers to create computer software. It is used to create general *systems software*, *drivers* for various computer devices, software for servers and software for specific applications and also widely used in the creation of video games.
C++ is used by many programmers of different types and coming from different fields. C++ is mostly used to write *device* driver programs, system software, and applications that depend on direct hardware manipulation under real-time constraints. It is also used to teach the basics of object-oriented features because it is simple and is also used in the fields of research. Also, many primary user interfaces *and* system files of Windows and Macintosh are written using C++. So, C++ is a popular, strong and frequently used programming language of this modern programming era.

Object-oriented programming and C++

C++ supports Object-Oriented Programming (OOP), with four significant principles of object-oriented development:

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

Abstraction

Data abstraction allows a program to ignore the details of how a data type is represented. Abstraction refersto the act of representing essential features without including the background details or explanations.

Encapsulation

Unable to deal with the complexity of an object, human chooses to ignore its non-essential details and concentrate on the details which are essential to our understanding. You can place a simple context of the object model, that abstraction is "looking for what you want" within an object or class. But there is another major concept connected to abstraction which is called encapsulation.

So basically, encapsulation can be defined as the process of hiding all of the details of an object that do not throw in or dealt with its essential characteristics. Encapsulation can also be defined as preventing access to non-essential details of classes or its objects. Abstraction and encapsulation have close bonding among each other. Encapsulation assists abstraction by providing a means of suppressing the non-essential details.

Inheritance

The technique of deriving a new class from an old one is called inheritance. The old class is referred to as base class and the new class is referred to as derived class or subclass. Inheritance concept allows programmers to define a class in terms of another class, which makes creating and maintaining application easier. When writing a new class, instead of writing new data member and member functions all over again, programmers can make a bonding of the new class with the old one that the new class should inherit the members of the existing class. A class can get derived from one or more classes, which means that it can inherit data and functions from multiple base classes.

Here is the syntax how inheritance is performed in C++:

```
class derived-class: visibility-mode base-class
```

Polymorphism

Polymorphism is another concept of object-oriented programming (OOPs). The attitude which lies beneath this concept is "single interface having multiple implementations." This provides a single interface for controlling access to a general class of actions. Polymorphism can be gained in both ways:

- Compile time
- Run time

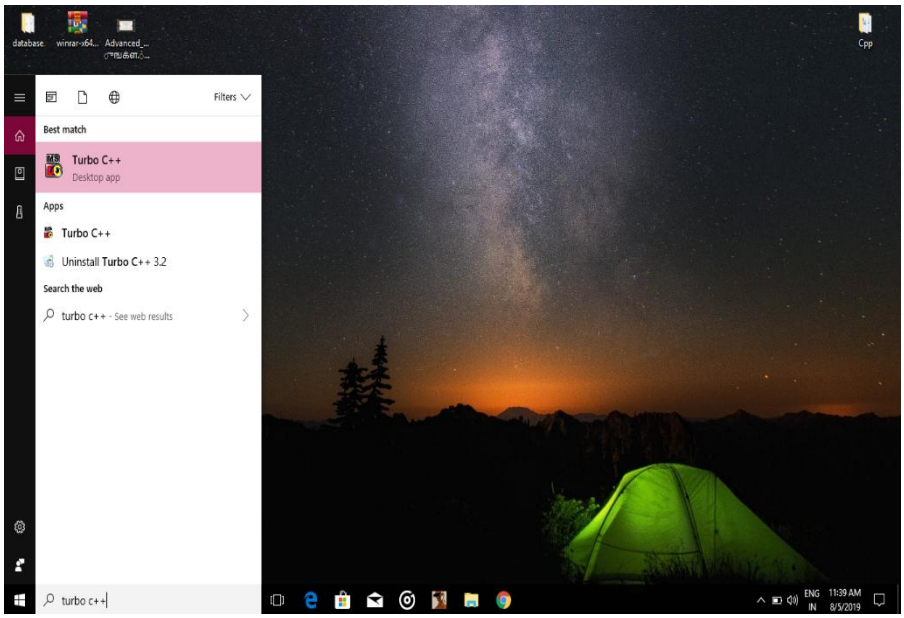
A common and simple example of polymorphism is when you used >> and << as operator overloading in C++, for *cin* and *cout* statements respectively. This bitwise shift operator at that time acts as a inclusion operator and its overloaded meaning is defined in *iostream* header file.

How to execute C++ program:

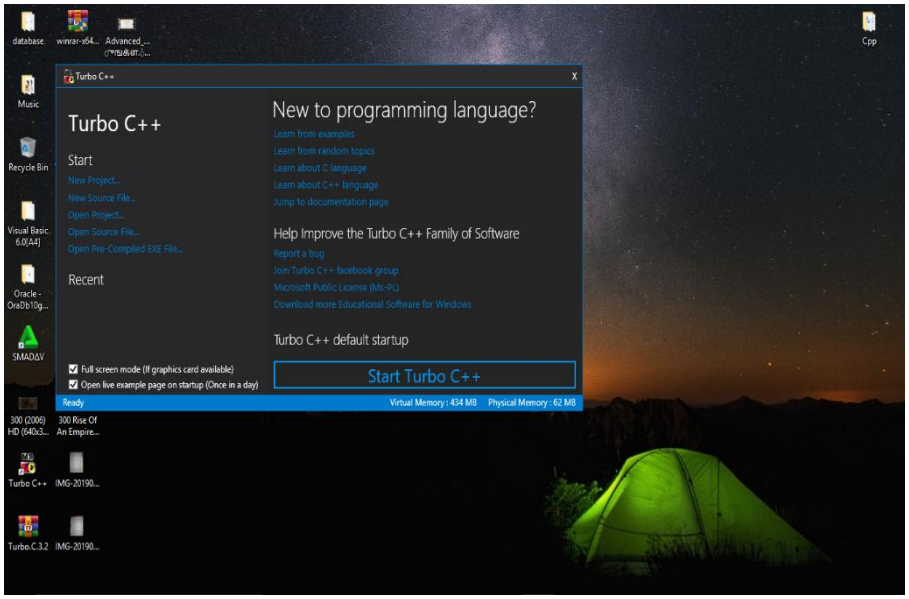
eSoftner

Through this app we can execute C++ programs.

How to open:



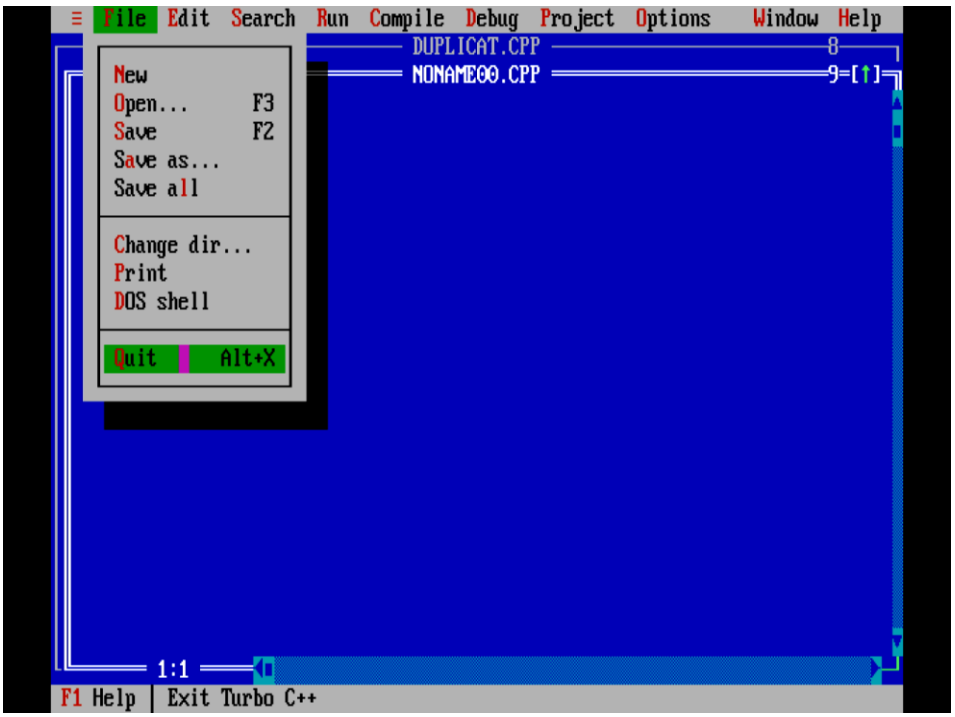
Go to Start and search for Turbo C++:



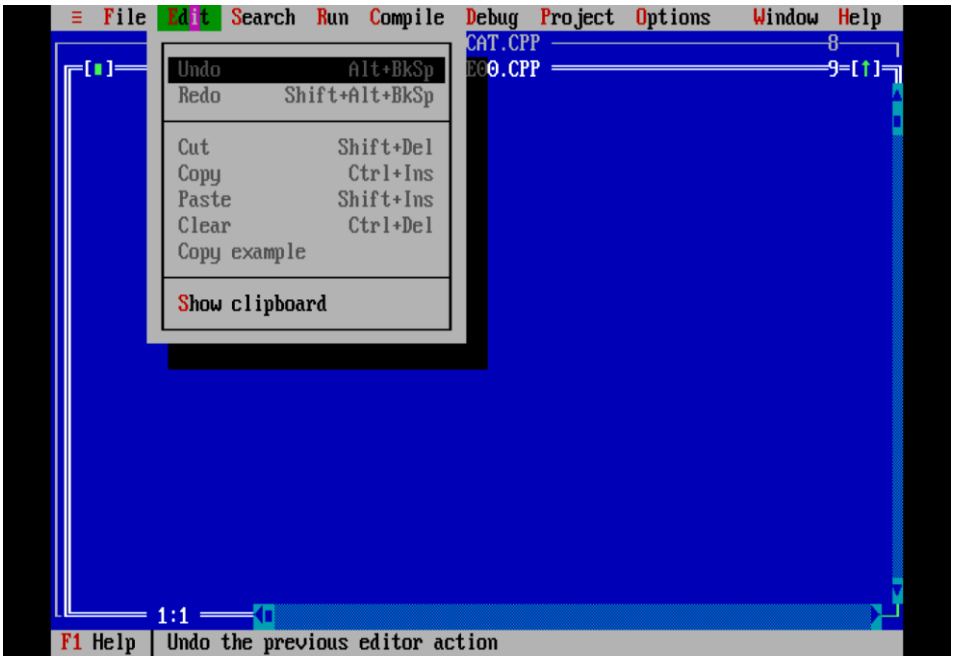
Notes

Notes

In this app we have File menu:

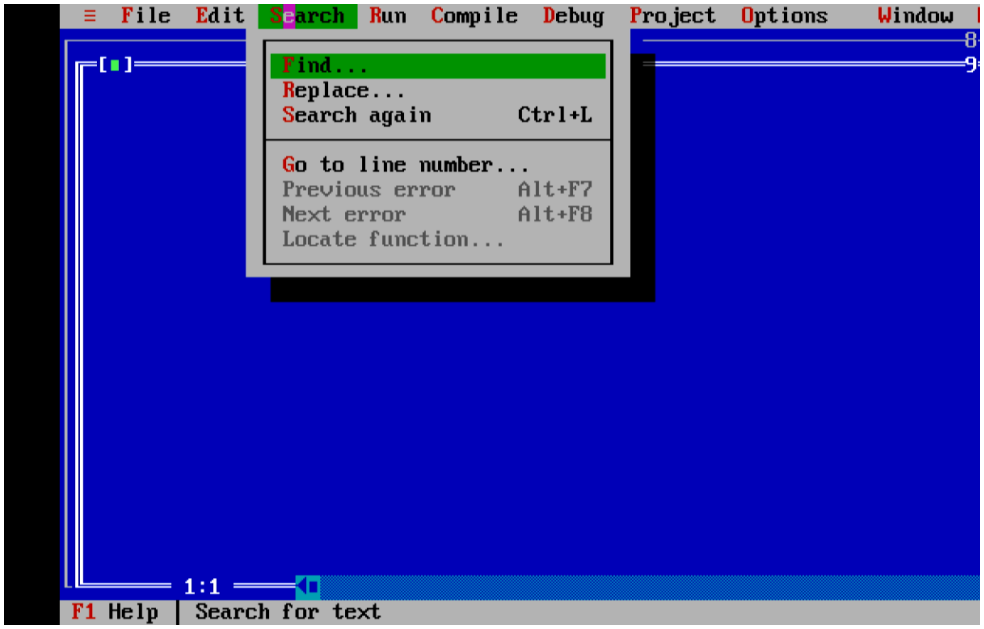


Edit menu:

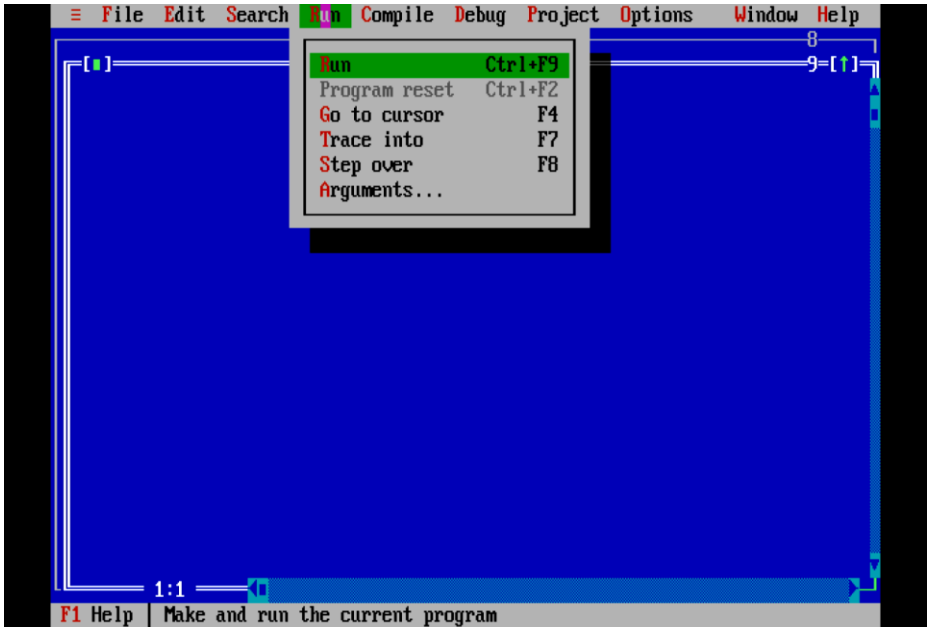


Notes

Search Menu:

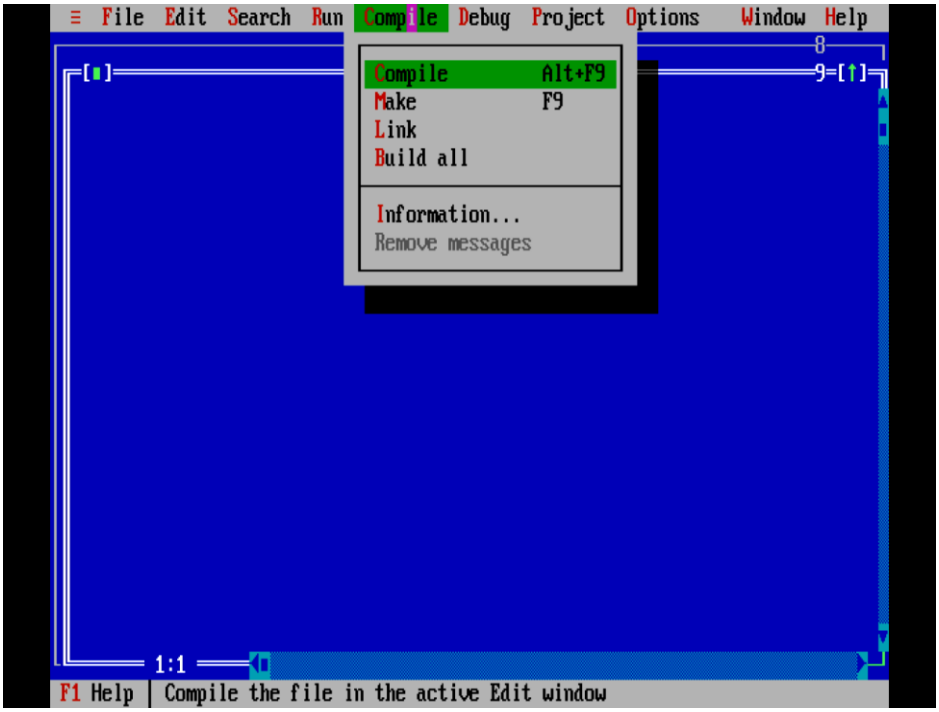


Run Menu:

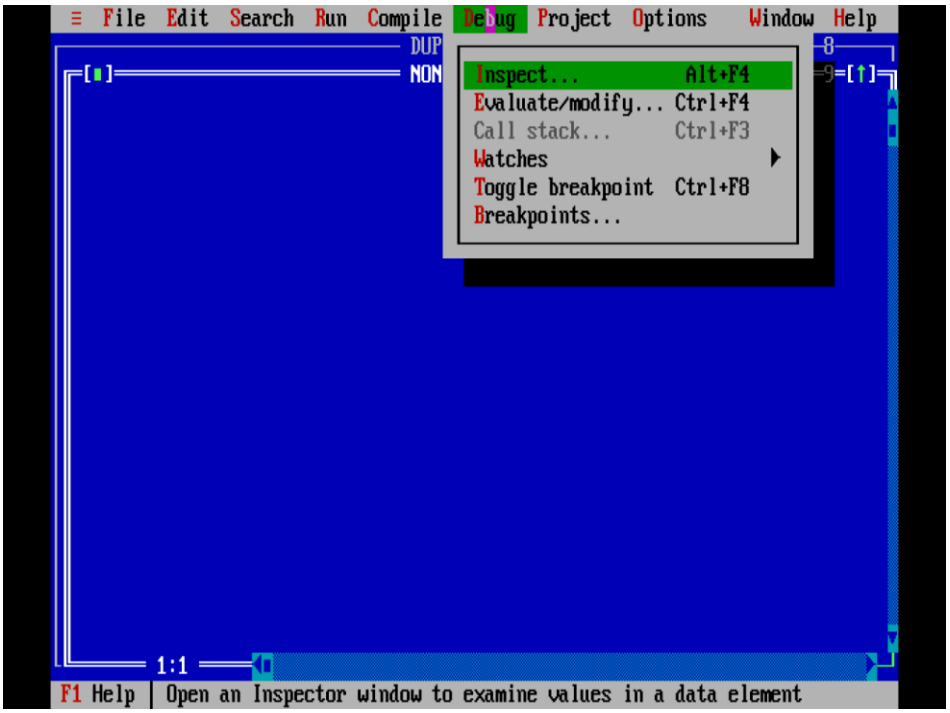


Notes

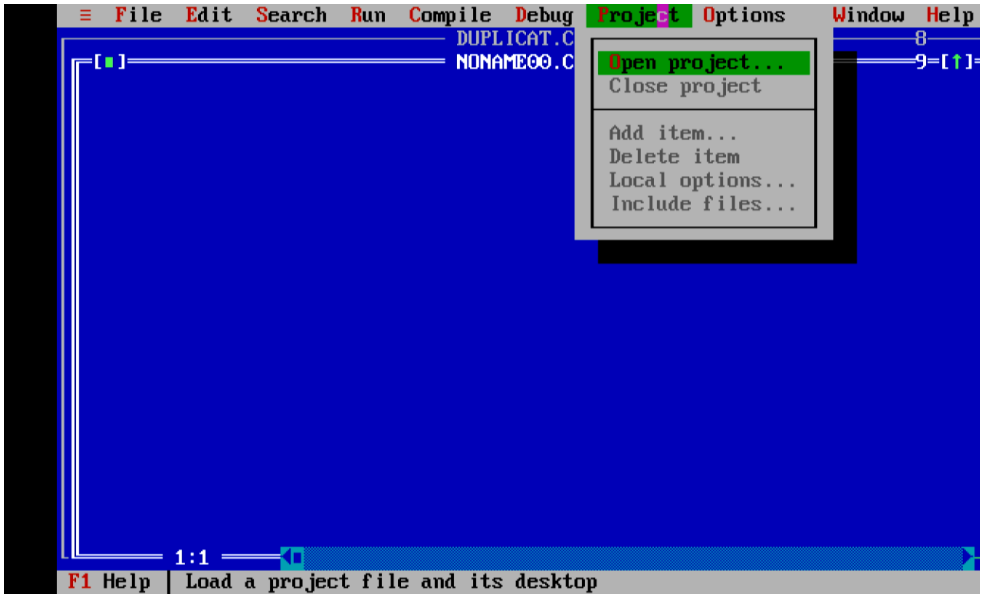
Compile Menu:



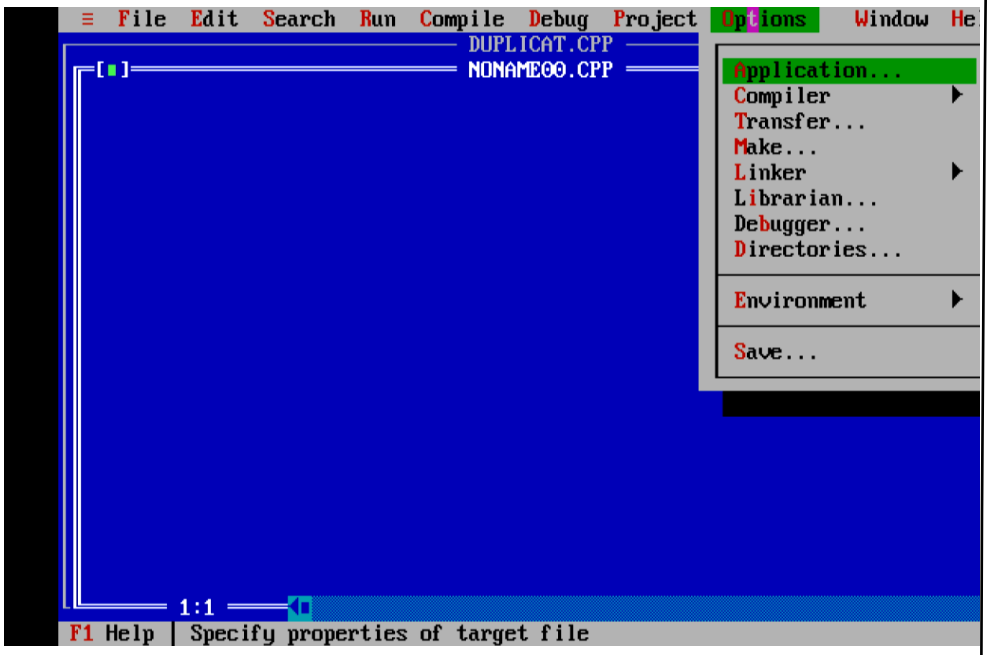
Debug Menu:



Project Menu:



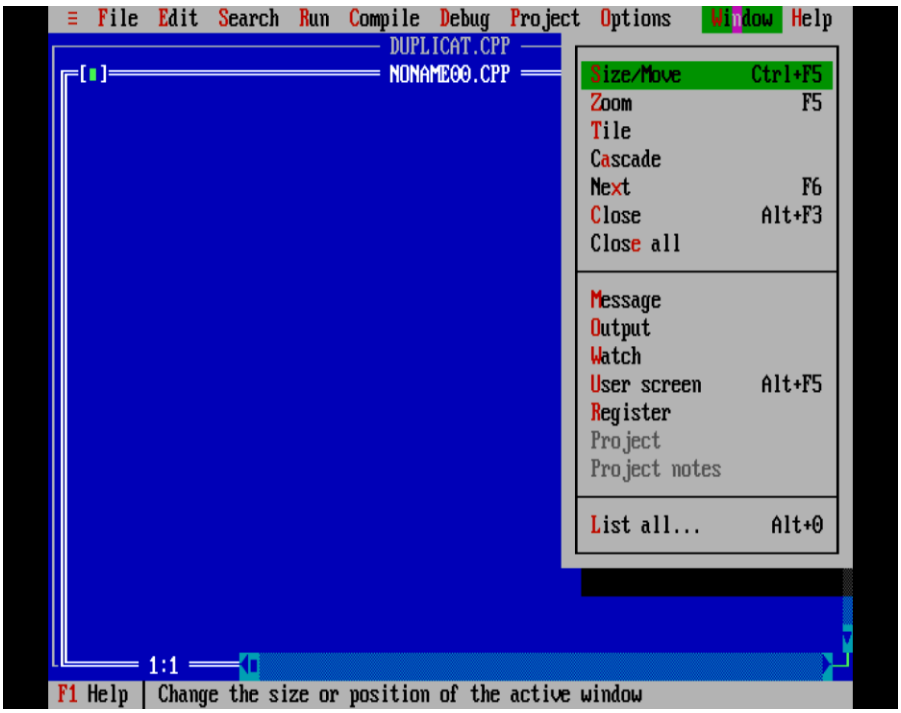
Option Menu:



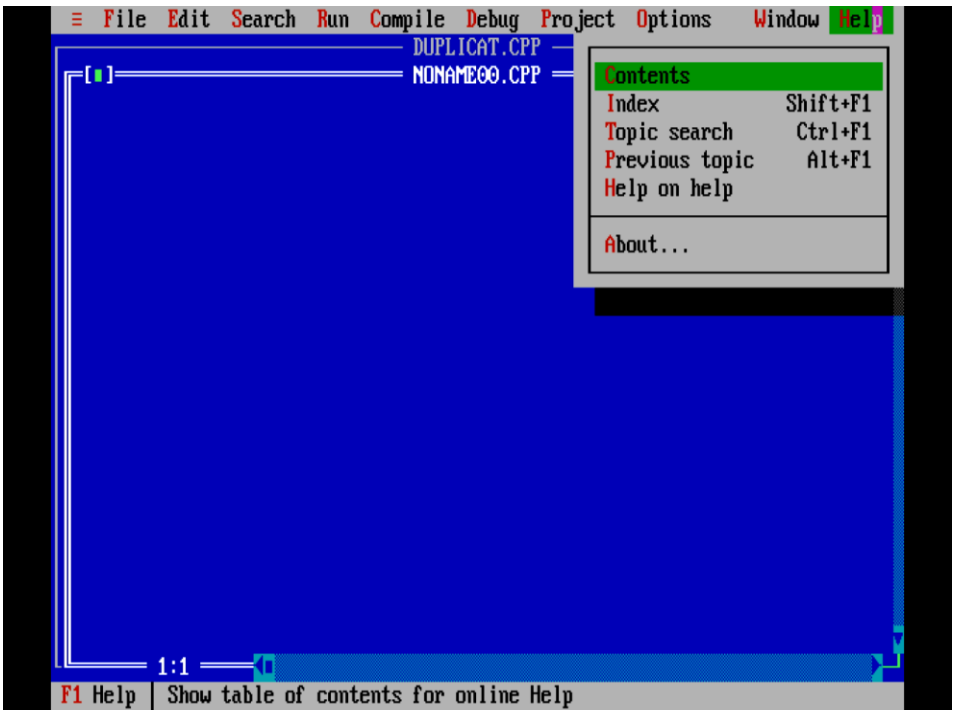
Notes

Notes

Window Menu:



Help Menu:



BLOCK – I SIMPLE C++ PROGRAMS

UNIT 1
Introduction: Simple C++ Programs

1.1 Write a program in C++ to print a message on output screen:

```
#include<iostream.h>
int main()
{
        cout<<"Hello world";
        return 0;
}
```

Output:
Hello world

1.2Write a program in C++ to print an integer value:

```
#include<iostream>
int main()
{
        int a;
        cout<<"Enter integer\n";
        cin>>a;
        cout<<"Integer is "<<a;
        return 0;
}
```

Output:
Enter integer
10
Integer is 10

1.3Write a program in C++ to add two numbers:

```
#include<iostream.h>
int main()
{
        int a,b,sum;
```

```
        cout<<"Enter the 1st number\n";
        cin>>a;
        cout<<"Enter the 2nd number\n";
        cin>>b;
        sum=a+b;
        cout<<"Sum of two number "<<a;
        return 0;
}
```

Output:

```
Enter the 1st number
10
Enter the 2nd number
20
Sum of two number 30
```

1.4Write a program in C++ to swap two numbers:

```
#include<iostream>
int main()
{
        int a,b,c;
        cout<<"Enter the 1st number\t";
        cin>>a;
        cout<<"\nEnter the 2nd number\t";
        cin>>b;
        c=a;
        a=b;
        b=c;
        cout<<"First number is\t"<<a;
        cout<<"\nSecond number  is\t<<b;
        return 0;
}
```

Output:

```
Enter the first number          12
Enter the second number         13
First number is                 13
Second number is        12
```

1.5Write a program in C++ to swap two numbers without the help of third variable:

```
#include<iostream>
int main()
{
        int a,b;
```

```
        cout<<"Enter the 1st number\t";
        cin>>a;
        cout<<"\nEnter the 2nd number\t";
        cin>>b;
        a=a+b;
        b=a-b;
        a=a-b;
        cout<<"First number is\t"<<a;
        cout<<"\Second number is\t"<<b;
        return 0;
}
```

Output:

```
Enter the first number      12
Enter the second number     13
First number is             13
Second number is    12
```

UNIT – 2
Control Structures using if and switch constructs program

2.1 The simple If statement:
Write a program to find the negative number:

```
#include<iostream>
int main()
{
        cout<<"Please enter a number:\n ";
        int x;
        cin>>x;
        // this program only checks if the number is negative or not
        if(x<0)
        {
                cout<<"\nNegative";
        }
        return 0;
}
```

Output:

```
Please enter a number:
-10
Negative
```

Notes

2.2 If – Else control structure
Write a program to find whether the given number is negative or not:

```
#include<iostream>
int main()
{
        cout<<"Please enter a number:\n ";
        int x;
        cin>>x;
        // this program only checks if the number is negative or not
        if(x<0)
        {
                cout<<"\nNegative\n";
        }
        else
        {
                cout<<"\nThe number is not negative";
        }
        return 0;
}
```

Output:

```
Please enter a number:
10
The number is not negative
```

2.3If – Else – Else If controls structure:
Write a program to find whether the given number is positive or negative or zero:

```
#include<iostream>
int main()
{
        cout<<"Please enter a number:\n ";
        int x;
        cin>>x;
        // this program only checks if the number is negative or not
        if(x<0)
        {
                cout<<"\nNegative\n";
        }
```

```
        else if(x>0)
        {
                cout<<"\nPositive\n";
        }
        else
        {
                cout<< "\nThe number is 0";
        }
        return 0;
}
```

Output:

Please enter a number:0
The number is 0

2.4Switch case:

Write a program to find weekdays based on the given number:

```
#include<iostream>
int main()
{
        cout<<"Please enter a number between 1 and 7:\n ";
        int x;
        cin>>x;
        switch(num)
        {
                case 1:
                        cout<<"Sunday";
                        break;
                case 2:
                        cout<<"Monday";
                        break;
                case 3:
                        cout<<"Tuesday";
                        break;
                case 4:
                        cout<<"Wednesday";
                        break;
                case 5:
                        cout<<"Thursday";
                        break;

                case 6:
                        cout<<"Friday";
                        break;
                case 7:
                        cout<<"Saturday";
```

```
                    break;
            // optional
            default:
                    cout<<"Invalid Input";
        }
        return 0;
}
```

Output:

Please enter a numer between 1 and 7:
1
Sunday

UNIT – 3

Looping, Arrays, Structure statements: for, while, do-while, Strings and Matrices Programs

3.1 Write a Program to find factorial of a number using For Loop

```
#include<iostream>
int main()
{
            int i ,n,factorial=1;
            cout<<"Enter a number:";
            cin>>n;
            for(i=1;i<=n;i++)
            {
                    factorial=factorial*i;
            }
            cout<<"Factorial of "<<n<<" ="<<factorial;
            return 0'
}
```

Output :

Enter a number: 5
Factorial of 5 = 120

3.2 Write a Program for Adam's Square Number Using While Loop

```
#include<iostream.h>
#include<conio.h>
```

```
void main()
{
	int n,ss1,r,s=0,ss,ns,sr,sum=0;
	clrscr();
	cout<<"Enter a number:";
	cin>>n;
	cout<<"\nGiven number is:"<<n;
	ns=n*n;
	cout<<"\nSquare of given number is:"<<ns;
	while(n!=0)
	{
		r=n%10;
		n=n/10;
		s=s*10+r;
	}
	cout<<"\nReverse of given number is:"<<s;
	ss=s*s;
	cout<<"\nSquare of reverse number is:"<<ss;
	while(ss!=0)
	{
		sr=ss%10;
		ss=ss/10;
		sum=sum*10+sr;
	}
	cout<<"\nReverse of square is:"<<sum;
	if(ns==sum)
	{
			cout<<"\nGiven number is adam's number";
	}
	else
	{
		cout<<"\nNot an adam's number";
	}
	getch();

}
```

Output :

```
Enter a number:12
Given number is:12
Square of given number is:144
Reverse of given number is:21
Square of reverse number is:441
Reverse of square is:144
Given number is adam's number

Enter a number:23
Given number is:23
Square of given number is:529
Reverse of given number is:32
```

Notes

Square of reverse number is:1024
Reverse of square is:4201

Not an adam's number

3.3Write a program to print the sum of n natural numbers using Do While Loop

```
#include<iostream>
#include<conio.h>
int main()
{
        int n,i=1,s=0;
        cout<<"Enter n:";
        cin>>n;
        do
        {
                s=s+i;
                i++;
        }
        while(i<=n);
        cout<<"Sum="<<s;
        getch();
        return 0;
}
```

Output :

```
Enter n:8
Sum=36
```

3.4Write a program to Find Sum of n Array Elements

```
#include<iostream>
#include<conio.h>
void main()
{
        int arr[30],i,n,sum=0;
        cout<<"Enter n:";
        cin>>n;
        cout<<"Enter "<<n<<" numbers :";
        for(i=0;i<n;i++)
        {
                cin>>arr[i];
        }
        for(i=0;i<n;i++)
        {
```

```
            sum=sum+arr[i];
        }
        for(i=0;i<n;i++)
        {
            cout<<"Sum of Numbers =" <<sum;
        }
        getch();
 }
```

Output:

```
Enter  n :  5
Enter  5  numbers :
7 5 7 8 9
Sum of Number = 36
```

3.5Write a program to Count Word in Sentence Using String

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
#include<Stdio.h>
void main()
{
            char strs[100],count=0,strw[20];
            int i,len;
            cout<<"Enter a Sentence : ";
            gets(strs);
            len=strlen(strs);
            for(i=0;i<len;i++)
            {
                if(strs[i]==' ')
                {
                    count++;

                }
            }
            cout<<"Total Num Of Words : "<<count+1;
            getch();
}
```

Output:

```
Enter a Sentence : Hello cpp Program how are you
Total Num Of Words : 6
```

3.6Write a Program for Transpose of Matrix

```
#include<iostream>
int main()
  {
```

Notes

```
        int a[10][10],m,n,i,j;
        cout<<"Enter Rows and Columns : ";
        cin>>m>>n;
        cout<<"Enter Matrix Elements : ";
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        cin>>a[i][j];
                }
        }
        cout<<"Your Matrix : \n ";
        for(i=0;i<m;i++)
        {
                for(j=0;j<n;j++)
                {
                        cout<<a[i][j]<<" ";
                        cout<<"\n";
                }
        }
        cout<<"Transpose Matrix : \n ";
        for(i=0;i<n;i++)
        {
                for(j=0;j<m;j++)
                {
                        cout<<a[i][j]<<" ";
                        cout<<"\n";
                }
        }
        return 0;
}
```

Output :

```
Enter Rows and Columns : 2 2
Enter Matrix Elements : 1  2  3  4
Your Matrix :
  1 2
3  4
Transpose Matrix :
  1  3
  2  4
```

BLOCK -2 OOPS CONCEPTS

UNIT – 4
Functions: static function, friend function, constructor, destructor and operator overloading and Recursive programs.

Public and Private Keywords:

Public:

Public members can be accessed by any other code(members) in the same class or other classes that references it.

Private:

Private members can be accessed only by code in the same class or struct.

4.1 Use of Class: Program for Transaction in Bank

```
#include<iostream.h>
#include<conio.h>
class bank
{

         char cust_name[20];
         int acc_no;
         int amount;
         int bal;
         char cust_address[20];
        public:
                void get();
                void balance();
                void withdraw();
                void disp();

};
void bank :: get(void)
{
        cout<<"\n Cust_Name";
        cin>>cust_name;
        cout<<"\n A/c_No";
        cin>>acc_no;
```

Notes

```
        cout<<"\n Amount";
        cin>>amount;
        cout<<"\n Cust_Address";
        cin>>cust_address;
        bal=0;
        bal=amount;
}
void bank :: balance(void)
{
        cout<<"Balance"<<bal;
}
void bank :: withdraw()
{
        cout<<"\n Enter amount";
        cin>>amount;
        bal=bal-amount;
}
void bank :: disp()
{
        if(bal<100)
        {
                cout<<"No withdrawl";
        }
        else
        {
                cout<<"\n Cust_Name:"<<cust_name<<"\n";
                cout<<"\n A/c_No:"<<acc_no<<"\n";
                cout<<"\n Cust_Address:"<<cust_address<<"\n";
                cout<<"\n Balance:"<<bal<<"\n";
        }
}
void main()
{
        clrscr();
        bank b;
        b.get();
        b.balance();
        b.withdraw();
        b.disp();
        getch();
}
```

OUTPUT:

```
Cust_Name:    Elango
A/c_No:       58585
Amount:       5000
Cust_Address: 4, gandhinagar, Trichy

Balance              5000
Enter amount: 2000
```

Cust_Name: Elango
A/c_No: 58585
Cust_Address: 4,gandhinagar, Trichy
Balance: 3000

4.2 Write a Program Using Static Variable and Static Function

```
#include<iostream>
class demo
{
        private:
                static int x;
                static int y;
        public:
                static void print()
                {
                        cout<<"Value Of x : "<<x;
                        cout<<"Value Of y : "<<y;
                }
};
int demo ::x=10;
int demo ::y=20;
int main()
{
        demo ob;
        cout<<"Printing Through Object Name : ";
        ob.print();
        cout<<"Printing Through Class Name : ";
        demo::print();
        return 0;
}
```

Output :

Printing Through Object Name :
Value of x :10
Value of y :20
Printing Through Class Name :
Value of x :10
Value of y :20

Characteristics of Static Function:

- It is initialized to zero when the first object of it's class is created no other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all the object of the class.

Notes

- It is visible only within the class but its life time is the entire program.

4.3Write a Program to find swapping of two values Using Friend function

```
#include<iostream.h>
#include<conio.h>
class B;
class A
{
        private:
                int x;
        public:
                void setx()
                {
                        cout<<"\n enter x";
                        cin>>x;
                }
                friend void swap (A,B);
};

class B
{
        private:
                int y;
        public:
                void sety()
                {
                        cout<<"\n enter y";
                        cin>>y;
                }
                friend void swap (A,B);
};

void swap (A o1,B o2)
{
        int temp;
        cout<<"\nBefore swapping:"<<x<<"\t"<<y;
        temp=o1.x;
        o1.x=o2.y;
        o2.y=temp;
        cout<<"\nAfter swapping x="<<o1.x<<"y="<<o1.y;
}

void main()
{
        A p;
        B q;
```

```
        clrscr();
        p.setx();
        q.sety();
        swap(p,q);
        getch();
}
```

OUTPUT

****

Enter x: 2

Enter y: 3

Before swapping

x=2 y=3

After swapping

x=3 y=2

4.4Write a Program for Matrix Using Constructor and Destructor

```
#include<iostream.h>
#include<conio.h>
class matrix
{
        int **p;
        int d1,d2;
        public:
                matrix(int x,int y);
                void getelement(int i,intj,int value)
                {
                        p[i][j]=value;
                }
                int &putelement(int i,int j)
                {
                        return p[i][j];
                }
                ~matrix();
};
matrix::matrix(int x,int y)
{
        d1=x;
        d2=y;
        p=new int *[d1];
        for(int i=0;i<d1;i++)
                p[i]=new int [d2];
}
matrix::~matrix()
{
```

Notes

```
        for(int i=0;i<d1;i++)
                delete p[i]; //row by row deletion
        delete p;  // Reference for that matrix is delelted.
cout<<"\nDestroyed";

}
void main()
{
        int m,n;
        clrscr();
        cout<<"\n Enter the size of matrix";
        cin>>m>>n;
        matrix a(m,n);
        cout<<"\n Enter matrix elements row by row";
        int i,j,value;
        for(i=0;i<m;i++)
                for(j=0;j<n;j++)
                 {
                        cin>>value;
                        a.getelement(i,j,value);
                 }

                cout<<"\n"<<a.putelement(1,1)<<"is the element in the 1st
row 1st column";
        }
        getch();
}
```

Output:

```
Enter the size of matrix:    3    3
Enter matrix elements row by row
7         8         9
2         3         4
5         6         7
3   is the element in the 1st row 1st column
Destroyed
```

4.5Complex number Addition Using Operator Overloading

```
#include<iostream.h>
class complex
{
        int a,b;
        public:
        complex()
        {
        }
        complex(int x,int y)
```

```
        {
                a=x;
                b=y;
        }
        friend complex operator +(complex,complex);
        void display()
        {
                cout<<"The Complex Number Is...\n";
                cout<<a<<"+"<<b<<"j";
        }
};
complex operator +(complex ob,complex ob1)
{
        complex ob2;
        ob2.a=ob.a+ob1.a;
        ob2.b=ob.b+ob1.b;
        return ob2;
}
int main()
{
        float a1,b1,a2,b2;
        cout<<"Enter The Real Part :";
        cin>>a1>>a2;
        complex m(a1,a2);
        cout<<"Enter The ImaginoryPart :";
        cin>>b1>>b2;
        complex(b1,b2);
        complex s;
        s=m+n;
        s.disp();
        return 0;
}
```

Output :

```
Enter The Real Part :
3  2
Enter The Imaginary Part :
3  2
The Complex Number Is...
6 + 4j
```

4.6 Write a Program for Factorial Using Recursion

```
#include<iostream>
int f(int n)
{
        if(n<=1)
        {
                return 1;
```

```
        }
        else
        {
                return n*f(n-1);
        }
}
int main()
{
        int num;
        cout<<"Enter Number :";
        cin>>num;
        cout<<"Factorial Is ="<<f(num);
        return  0;
}
```

Output :

```
Enter Number : 4
 Factorial Is =24
```

UNIT 5

Inheritance and Polymorphism: Inheritance types and polymorphism, Virtual function

5.1 Write a Program for Product Details using Multilevel Inheritance

```
#include<iostream.h>
#include<conio.h>
class product
{
        protected:
                int pno;
                float up,qty;
                char pname[20];
        public:
                void get();
};
void product::get()
{
        cout<<"\nEnterPno,Pname,Qty and Unit Price(in float):";
        cin>>pno>>pname>>qty>>up;
}
class calculation:public product
{
```

```
        protected:
                float tot;
        public:
                void calc();
};
void calculation::calc()
{
        tot=up*qty;
}
class result:public calculation
{
        public:
                void put();
};
void result::put()
{
        cout<<"\nPNo :"<<pno;
        cout<<"\nPName  :"<<pname;
        cout<<"\nQty  :"<<qty;
        cout<<"\nUPrice:"<<up;
        cout<<"\nTotal:"<<tot;
}
void main()
{
        result o;
        clrscr();
        o.get();
        o.calc();
        o.put();
        getch();
}
```

Output:

```
*********
Enter Pno,Pname,Qty and Unit Price(in float):
101     LUX             2       12.50
PNo     :       101
PName:  LUX
Qty     :       2
UPrice :        12.50
Total   :       25.00
```

Function Overloading:

Using a single function name to perform different type of task is known as Function Overloading.

Notes

5.2 Write a program for volume of different shapes using function overloading

```
#include<iostream.h>
#include<conio.h>
class function
{
        public:
        float volume(int a)
        {
                  int cube;
                cube=a*a*a;
                return(cube);
        }
        float volume(float a,float b)
         {
                float cylinder;
                cylinder=3.14*a*a*b;
                 return(cylinder);
         }
        float volume(int a,intb,int c)
         {
                int rectangle;
                rectangle=a*b*c;
                return(rectangle);
        }
        double volume(double a)
         {
                double sphere;
                sphere=(4/3)*3.14*a*a;
                return(sphere);
         }
};
void main()
{
        function f;
        int ch;
        do
        {
                cout<<"\n 1.Cube";
                cout<<"\n 2.Cylinder";
                cout<<"\n 3.Rectangle";
                cout<<"\n 4.Sphere";
                cout<<"\n 5.Exit";
                cout<<"\n Enter your choice";
                cin>>ch;
                switch(ch)
                {
                        case 1:
                       cout<<"\n cube="<<f.volume(3);
```

```
                    break;
                    case 2:
                    cout<<"\n cylinder="<<f.volume(3.0,2.0);
                    break;
                    case 3:
                    cout<<"\n rectangle="<<f.volume(2,4,3);
                    break;
                    case 4:
                    cout<<"\n sphere="<<f.volume(4);
                    break;
            }
        }while(ch!=5);
getch();
}
```

Output:

1. Cube
2. Cylinder
3. Rectangle
4. Sphere
5. Exit

Enter your choice: 1
Cube:27

Virtual Function:

- A virtual function a member function which is declared within a base class and is redefined(Overridden) by derived class.
- They are mainly used to achieve Runtime polymorphism.
- Functions are declared with a virtual keyword in base class.

5.3Write a Program for Book Details Using Virtual Function

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class media
{
        protected:
                char title[50];
                float price;
        public:
                media(char *s,float a)
                {
                        strcpy(title,s);
```

Notes

```
                price=a;
        }
        virtual void disp() {}
};
class book :public media
{
    int pages;
    public:
        book(char *s,floata,int p) : media(s,a)
        {
            pages=p;
        }
        void disp();
};
class tape : public media
{
    float time;
    public:
        tape(char *s,floata,float t) : media(s,a)
        {
            time=t;
        }
        void disp();
};
void book :: disp()
{
    cout<<"\n title :\t"<<title;
    cout<<"\n pages :\t"<<pages;
    cout<<"\n price :\t"<<price;
}
void tape :: disp()
{
    cout<<"\n title :\t"<<title;
    cout<<"\n play time :\t"<<time<<"mins";
    cout<<"\n price :\t"<<price;
}
void main()
{
    char *title=new char[30];
    float price,time;
    int pages;
    clrscr();
    cout<<"\n **********************";
    cout<<"\n VIRTUAL FUNCTION:";
    cout<<"\n **********************";
    cout<<"\n \n enter book details";
    cout<<"\n title";
    cin>>title;
    cout<<"\n price";
    cin>>price;
    cout<<"\n pages";
```

Notes

```
        cin>>pages;
        book book1(title , price , pages);
        cout<<"\n enter tape details ";
        cout<<"\n  title";
        cin>>title;
        cout<<"\n price";
        cin>>price;
        cout<<"\n play time(mins)";
        cin>>time;
        tape tape1(title , price ,time);
        media * list[2];
        list[0]=&book1;
        list[1]=&tape1;
        cout<<"\n media details";
        cout<<"\n .....Book.....";
        list[0]->disp();
        cout<<"\n .....Tape.....";
        list[1]->disp();
        getch();
}
```

Output:

```
**********************************
          VIRTUAL FUNCTION
**********************************
        Enter Book details
        Title:   roja
        Price:    155
        Pages:   256

        Enter tape details
        Title:   vijay
        Price:                  890
        Playtime(mins):     45

        Media details
        .........................Book.....................
        Title:   roja
        Pages:  256
        Price:   155
        .........................Tape.....................
          Title: vijay
        Playtime:       45mins
        Price:          890
```

Notes

UNIT 6 FILE

Open a File :

The first operation generally performed on an object of one of these classes is to associate it to a real file. This procedure is known as to open a file.

Syntax :

open(filename,mode);

Close a File:

When we are finished with our input and output operations on a file we shall close it is close a file.

Syntax:

myfile.close();

6.1 Write a Program for Student Details Using File Concepts

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
class student
{
        int rno,f,a;
        char name[25],res[25];
        float m[6],tot,avg;
        ofstream of;
        ifstream inf;
        public:
        student(int x)
        {
                 a=x;
                of.open("data.txt",ios::out);
        }
        void get()
        {
        for(int i=0;i<a;i++)
        {
        cout<<"\nEnter the roll no:";
        cin>>rno;
        of<<rno<<endl;
        cout<<"\nEnter the name:";
        cin>>name;
        of<<name<<endl;
        for(int i=1;i<=5;i++)
        {
                cout<<"\nEnter the mark:"<<i<<":";
                cin>>m[i];
```

Notes

```
                of<<m[i]<<endl;
        }
        calc();
 }
of.close();
}
 void calc()
 {
         tot=0;
        for(int l=1;l<=5;l++)
        tot=tot+m[l];
        of<<tot<<endl;
         avg=tot/5;
        of<<avg<<endl;
        f=0;
        for(int a=1;a<=5;a++)
        {
                if(m[a]>40)
                f=f+1;
        }
        if(f==5)
        of<<"Pass"<<endl;
        else
        of<<"Fail"<<endl;
 }
 void disp()
 {
        inf.open("data.txt");
        cout<<”RNO\tName\tMark1\t Mark2\tMark3
        << \tMark4\tMark5\tTot\tAvg\tResult\n”;
        for(int i=0;i<a;i++)
        {
                inf>>rno;
               inf>>name;
                cout<<rno<<"\t"<<name<<endl;
                for(int j=1;j<=5;j++)
                {
                        inf>>m[i];
                        cout<<m[i]<<endl;
                }
                inf>>tot;
                cout<<tot<<"\t";
                inf>>avg;
                cout<<avg<<"\t";
                inf>>res;
                cout<<res<<"\t";
        }
        inf.close();
 }
 };
```

```
void main()
{
        int a;
        clrscr();
        cout<<"\nEnter the no. of students:";
        cin>>a;
        student s(a);
        s.get();
        s.disp();
        getch();
}
```

Output:

```
Enter the no. of students:      2
Enter the name  :       Shahid
Enter the roll no:      112
Enter the mark1 :       50
Enter the mark2 :       50
Enter the mark3 :       50
Enter the mark4 :       50
Enter the mark5 :       50

Enter the name  :       Priyanka
Enter the roll no:      113
Enter the mark1 :       70
Enter the mark2 :       70
Enter the mark3 :       70
Enter the mark4 :       70
Enter the mark5 :       70

RNO   Name        Mark1   Mark2 Mark3 Mark4  Mark5   Tot    Avg
Result
-------------------------------------------------------------------------------
--------
112   Shahid      50      50      50      50      50      250     50
      Pass
113   Priyanka    70      70      70      70      70      350     70
      Pass
```

Input and Output Header Files In Files

- **Ofstream:** Stream class to write on files
- **Ifstream:** Stream class to read from files
- **fstream:** Stream class to both read and write from/to files.

Notes

UNIT 7 POINTERS

7.1Write Program for Swapping Of Two Numbers Using Pointers

```
#include<iostream.h>
#include<conio.h>
void main()
{
        int *a,*b,*temp;
        cout<<"Enter a and b : ";
        cin>>*a>>*b;
        temp=a;
        a=b;
        b=temp;
        cout<<"After Swapping \n ";
        cout<<"a="<<*a<<"\n"<<"b="<<*b;
        getch();
}
```

Output:

```
Enter a and b :5
4
After Swapping
a=4
b=5
```

7.2 Write a Program for Array of Pointers

```
#include<iostream.h>
#include<string.h>
int main()
{
        int i=0;
        char *p[10]={"cpp","java","c#","vb"};
        char s[20];
        cin>>s;
        for(i=0;i<4;i++)
        {
                if(strcmp(s,*p[i]))
                {
                        cout<<"Book Name Exists";
                        break;
                }
```

Notes

```
        }
        if(i==4)
        {
                cout<<"Not Found";
        }
        return 0;
}
```

Output :

```
Enter Book Name :
java
Book Name Exists
C Programming
Not Found
```

BLOCK 3
LINEAR DATA STRUCTURE

UNIT- 8 Stacks: Stack Implementation, expression evaluation, Polish notation

Stack Implementation in C++:

A stack is a linear data structure than serves as a container of objects that are inserted and removed according to the LIFO(last-in-first-out) rule.

Below stack implementation in C++ covers below opertation:

1. **push:** Inserts a new element at the top of the stack, above its current top element.
2. **pop:** Removes the top element on the stack, thereby decrementing its size by one.
3. **isEmpty:** Returns true if stack is empty i.e. its size is zero else it returns false.
4. **isFull:** Returns true if stack is full i.e. its size has reached maximum allocated capacity else it returns false.
5. **size:** Returns the count of elements present in the stack.

8.1 Stack using array

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
int s[20],n,top;
void stack();
void push();
void pop();
void disp();
void main()
{
        clrscr();
        stack();
        int ch;
        do
        {
                cout<<"\nEnter option->1.push 2.pop 3.display
4.exit:";
                cin>>ch;

                switch(ch)
                {
```

Notes

```
                        case 1:push();break;
                        case 2:pop(); break;
                        case 3:disp();break;
                        default:exit(0);
                        break;
                }
                }while(ch<=4);
        }
}
void stack()
{
        top=-1;
        cout<<"Enter the size:";
        cin>>n;
}
void push()
{
        int x;
        if(top==n-1)
        cout<<"\nStack is full";
        else
        {
                cout<<"\nEnter the element:";
                cin>>x;
                top++;
                s[top]=x;
        }
}

void pop()
{
        int x;
        if(top==-1)
                cout<<"\nStack is empty";
        else
        {
                x=s[top];
                top--;
                cout<<"\n"<<x<<"is deleted from the stack\n";
        }
}

void disp()
{
        int i;
        if(top==-1)
                cout<<"\nStack is empty";
        else
        {
                cout<<"\nElements of the stack are:\n";
                for(i=top;i>=0;i--)
```

```
                {
                        cout<<s[i];
                        cout<<endl;
                }
        }
}
```

Output :

```
Enter the size:3
Enter option->1.push 2.pop 3.display 4.exit:1
Enter the element:10

Enter option->1.push 2.pop 3.display 4.exit:1
Enter the element:20

Enter option->1.push 2.pop 3.display 4.exit:1
Enter the element:30

Enter option->1.push 2.pop 3.display 4.exit:1
Stack is full
Enter option->1.push 2.pop 3.display 4.exit:3
Elements of the stack are:
10
20
30
Enter option->1.push 2.pop 3.display 4.exit:2
30is deleted from the stack
Enter option->1.push 2.pop 3.display 4.exit:3
Elements of the stack are:
10
20

Enter option->1.push 2.pop 3.display 4.exit:4
```

Expression evaluation:

This C++ program, using a stack data structure, computes value of postfix expression which pushes operands and pops these values on encountering an operator.

Polish notation:

Polish notation (PN), also known as normal Polish notation (NPN), Polish prefix notation or simply prefix notation, is a mathematical notation in which operators precede their operands, in contrast to the more common infix notation, in which operators are placed between operands, as well as reverse Polish notation (RPN), in which operators follow their operands. It does not need any parentheses as long as each operator has a fixed number of operands.

Notes

UNIT 9
Queues: Queue Implementation, Applications of Queue

Queue Implementation

A queue is an abstract data structure that contains a collection of elements. Queue implements the FIFO mechanism i.e. the element that is inserted first is also deleted first. In other words, the least recently added element is removed firs in a queue.

Applications of Queue Data Structure

Breadth First Search property of Queue makes it also useful in following kind of scenarios.

1. When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
2. When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

9.1 Queue Using Array

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>
class queue
{
int q[20],n,f,r,x,i;
public:
        queue()
        {
        clrscr();
        f=-1;
        r=-1;
        cout<<"\nEnter the n value: \t";
        cin>>n;
        }
        void enqueue();
        void dequeue();
        void disp();
};
void queue::enqueue()
{
        if(r==n-1)
        {
                cout<<"\nQueue is full"<<endl;
        }
        else
        {
```

```
                cout<<"Enter a element to queue: \t";
                cin>>x;
                if(f==-1)
                f=0;
                r=r+1;
                q[r]=x;
        }
}
void queue::dequeue()
{
        if (r==f==-1||r<f)
        {
                cout<<"\nQueue is empty"<<endl;
        }
        else
        {
                cout<< "Deleted from the queue;";
                x=q[f];
                cout<<x<<endl;
                f=f+1;
        }
}
void queue::disp()
{
        if(r==f==-1||r<f)
        {
                cout<<"\nQueue is empty"<<endl;
        }
        else
        {
                cout<<"Elements of the queue are: ";
                for(i=f;i<=r;i++)
                {
                cout<<endl;
                cout<<q[i];
                }
        }
};
void main()
{
        int ch;
        queue q;
        clrscr();
        cout<<"\t\t\tQUEUE\n";
        do
        {
                cout<<"\n\n1.Enqueue 2.Dequeue 3.Display 4.Exit\n";
                cout<<"\nEnter your choice: \t";
                cin>>ch;
                switch(ch)
                {
```

Notes

```
                    case 1:
                            q.enqueue();
                            break;
                    case 2:
                            q.dequeue();
                            break;
                    case 3:
                            q.disp();
                            break;
                    case 4:
                            exit(0);
                            default:
                                    cout<<"\nEnter 1to4 values:  ";
            }
        }while(1)
```

Output:

```
                Queue
Enter the n value:
3
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your choice:
1
Enter a element to queue
10
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your choice:
1
Enter a element to queue
20
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your choice:
1
Enter a element to queue
30
1.Enqueue 2.Dequeue 3.Display 4.Exit
Enter your choice:
1
Queue is full
```

UNIT 10
Linked List programs

List

A list or sequence is an abstract data type that represents a countable number of ordered values, where the same value may occur more than once. An instance of a list is a computer representation of the mathematical concept of a finite sequence; the (potentially) infinite analog of a list is a stream. Lists are a basic example of containers, as they

contain other values. If the same value occurs multiple times, each occurrence is considered a distinct item.

Merging lists

You're given the pointer to the head nodes of two sorted linked lists. The data in both lists will be sorted in ascending order. Change the next pointers to obtain a single, merged linked list which also has data in ascending order. Either head pointer given may be null meaning that the corresponding list is empty.

10.1 Write a C++ program to merge two sorted linked lists

```
#include <bits/stdc++.h>
class Node
{
   public:
   int data;
   Node* next;
};
/* pull off the front node of  the source and put it in dest */
void MoveNode(Node** destRef, Node** sourceRef);
                /* Takes two lists sorted in increasing order, and splices
                their nodes together to make one big sorted list which  is
                returned. */
Node* SortedMerge(Node* a, Node* b)
{
   Node dummy;
   Node* tail = &dummy;
dummy.next = NULL;
   while (1)
   {
      if (a == NULL)
      {
         tail->next = b;
         break;
      }
      else if (b == NULL)
      {
         tail->next = a;
         break;
      }
      if (a->data <= b->data)
MoveNode(&(tail->next), &a);
      else
MoveNode(&(tail->next), &b);
      tail = tail->next;
   }
   return(dummy.next);
}
void MoveNode(Node** destRef, Node** sourceRef)
```

Notes

```
{

   Node* newNode = *sourceRef;
assert(newNode != NULL);
   *sourceRef = newNode->next;
newNode->next = *destRef;
     *destRef = newNode;
}
void push(Node** head_ref, int new_data)
{
   Node* new_node = new Node();
new_node->data = new_data;
new_node->next = (*head_ref);
   (*head_ref) = new_node;
}
                    /* Function to print nodes in a given linked list */
void printList(Node *node)
{
   while (node!=NULL)
   {
cout<<node->data<<" ";
      node = node->next;
   }
}
/* Driver code*/
int main()
{
   /* Start with the empty list */
   Node* res = NULL;
   Node* a = NULL;
  Node* b = NULL;
                    /* Let us create two sorted linked liststo test the functions
                    Created lists, a: 5->10->15, b: 2->3->20 */
push(&a, 15);
push(&a, 10);
push(&a, 5);
push(&b, 20);
push(&b, 3);
push(&b, 2);
   res = SortedMerge(a, b);
cout<< "Merged Linked List is: \n";
printList(res);
   return 0;
}
```

Output:

```
Merged Linked List is:
2 3 5 10 15 20
```

Linked List

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers .

In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list

Single linked list

Singly linked list is a type of data structure that is made up of nodes that are created using self-referential structures. Each of these nodes contain two parts, namely the data and the reference to the next list node. Only the reference to the first list node is required to access the whole linked list. This is known as the head. The last node in the list points to nothing so it stores NULL in that part.

10.2 Program For Implement Singly Linked List

```
#include <iostream>
using namespace std;
struct Node
{
  int data;
  struct Node *next;
};
struct Node* head = NULL;
void insert(int new_data) {
  struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
new_node->data = new_data;
new_node->next = head;
  head = new_node;
}
void display() {
  struct Node* ptr;
ptr = head;
  while (ptr != NULL) {
cout<<ptr->data <<" ";
ptr = ptr->next;
  }
}
int main()
{
insert(3);
insert(1);
insert(7);
insert(2);
insert(9);
cout<<"The linked list is: ";
display();
  return 0;
```

}

Output :
The linked list is: 9 2 7 1 3

Notes

Header linked list

A Header linked list is one more variant of linked list. In Header linked list, we have a special node present at the beginning of the linked list. This special node is used to store number of nodes present in the linked list.

Insertion and Deletion of linked list

In this program, we will learn how to implement Linked List using C++ program

In this example, we will implement a singly linked list with insert, delete and display operations. Here, we will declare Linked List Node, Insert Node at the beginning, Delete Node from beginning and display all linked list Nodes.

10.3Program ForLinked List Implementation

```
#include <iostream>
struct Node{
   int num;
   Node *next;
};
struct Node *head=NULL;
void insertNode(int n){
   struct Node *newNode=new Node;
newNode->num=n;
newNode->next=head;
   head=newNode;
}
void display(){
   if(head==NULL){
cout<<"List is empty!"<<endl;
      return;
   }
   struct Node *temp=head;
   while(temp!=NULL){
cout<<temp->num<<" ";
      temp=temp->next;
   }
cout<<endl;
}
void deleteItem(){
   if(head==NULL){
cout<<"List is empty!"<<endl;
      return;
   }
```

```
cout<<head->num<<" is removed."<<endl;
   head=head->next;
}
int main()
{
display();
insertNode(10);
insertNode(20);
insertNode(30);
insertNode(40);
insertNode(50);
display();
deleteItem(); deleteItem(); deleteItem(); deleteItem(); deleteItem();
deleteItem();
display();
   return 0;
}
```

Output

```
List is empty!
50 40 30 20 10
50 is removed.
40 is removed.
30 is removed.
20 is removed.
10 is removed.
 List is empty!
List is empty!
```

Traversal in a List

In this program the start pointer points to the beginning of the list and rear points to the last node

The function create_new_node() takes one parameter, allocates memory to create a new node and returns the pointer to the new node. (return type: node *)

The function insert_node() takes node * type pointer as argument and inserts this node in the end of the list.

And the function traversal() takes node * type pointer as argument and displays the list from this pointer till the end of the list.

10.4 Program of Linked Lists Traversal in a list

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
struct node
{
        int info;
```

Notes

```
        node *next;
} *start, *newptr, *save, *ptr, *rear;
node *create_new_node(int);
void insert_node(node *);
void travers(node *);
void main()
{
        clrscr();
        start = rear = NULL;
        int inf;
        char ch='y';
        while(ch=='y' || ch=='Y')
        {
                cout<<"Enter Information for the new node: ";
                cin>>inf;
                newptr = create_new_node(inf);
                if(newptr == NULL)
                {
                        cout<<"\nSorry..!!..cannot create new
node..!!..Aborting..!!";
                        cout<<"\nPress any key to exit..";
                        getch();
                        exit(1);
                }
                insert_node(newptr);
                cout<<"Want to enter more nodes ? (y/n)..";
                cin>>ch;
                cout<<"\n";
        }
        cout<<"The list now is:\n";
        travers(start);
        getch();
}
node *create_new_node(int n)
{
        ptr = new node;
        ptr->info = n;
        ptr->next = NULL;
        return ptr;
}
void insert_node(node *np)
{
        if(start==NULL)
        {
                start = rear = np;
        }
        else
        {
                rear -> next = np;
                rear = np;
        }
}
```

```
void travers(node *np)
{
        while(np != NULL)
        {
                cout<<np->info<<" -> ";
                np = np->next;
        }
        cout<<" !!\n";
}
```

Here is the sample run of this C++ program

```
Enter Information for the new node: 10
Want to enter more nodes ? (y/n)..y

Enter Information for the new node: 20
Want to enter more nodes ? (y/n)..y

Enter Information for the new node: 30
Want to enter more nodes ? (y/n)..y

Enter Information for the new node: 40
Want to enter more nodes ? (y/n)..y

The list now is:
10->20-> 30-> 40-> !!
```

Notes

BLOCK 4
NON-LINEAR DATASTRUCTURE

UNIT 11
Tree Programs

Tree:

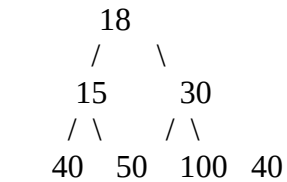
A tree is a collection of nodes connected by directed (or undirected) edges. A tree is a nonlinear data structure, compared to arrays, linked lists, stacks and queues which are linear data structures.

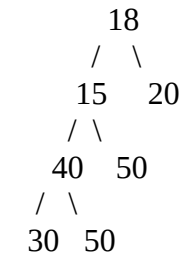
Binary trees:

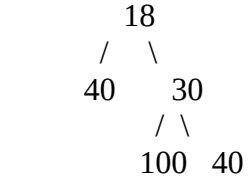
A binary tree is made of nodes, where each node contains a "left" reference, a "right" reference, and a data element. The topmost node in the tree is called the root. ... On the other hand, each node can be connected to arbitrary number of nodes, called children. Nodes with no children are called leaves, or external nodes.

Following are common types of Binary Trees.

Full Binary Tree A Binary Tree is full if every node has 0 or 2 children. Following are examples of a full binary tree. We can also say a full binary tree is a binary tree in which all nodes except leaves have two children.







In a Full Binary, number of leaf nodes is number of internal nodes plus 1

$$L = I + 1$$

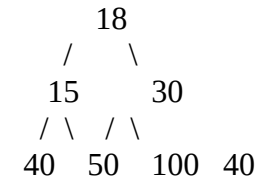
Where L = Number of leaf nodes, I = Number of internal nodes

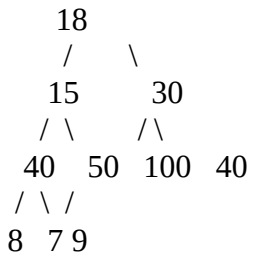
See Handshaking Lemma and Tree for proof.

Complete Binary Tree: A Binary Tree is complete Binary Tree if all levels are completely filled except possibly the last level and the last level has all keys as left as possible

Notes

Following are examples of Complete Binary Trees

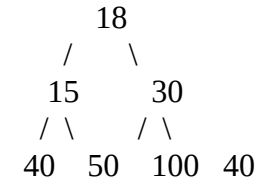


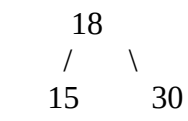


Practical example of Complete Binary Tree is Binary Heap.

Perfect Binary Tree a Binary tree is Perfect Binary Tree in which all internal nodes have two children and all leaves are at the same level.

Following are examples of Perfect Binary Trees.





A Perfect Binary Tree of height h (where height is the number of nodes on the path from the root to leaf) has 2h – 1 node.

Example of a Perfect binary tree is ancestors in the family. Keep a person at root, parents as children, Parents of parents as their children.

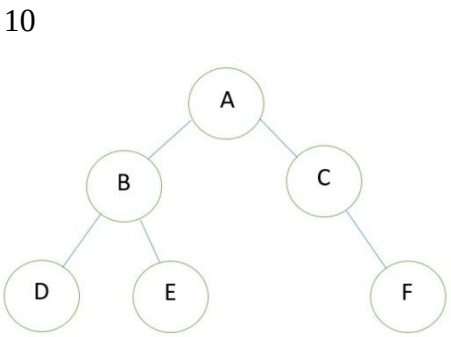
Balanced Binary Tree

A binary tree is balanced if the height of the tree is O(Log n) where n is the number of nodes. For Example, AVL tree maintains O(Log n) height by making sure that the difference between heights of left and right sub trees is 1. Red-Black trees maintain O(Log n) height by

Notes

making sure that the number of Black nodes on every root to leaf paths are same and there are no adjacent red nodes. Balanced Binary Search trees are performance wise good as they provide O(log n) time for search, insert and delete.

A degenerate (or pathological) tree A Tree where every internal node has one child. Such trees are performance-wise same as linked list.



10
/
20
\
30
\
40

Representing Binary Tree in memory

Let T be a Binary Tree. There are two ways of representing T in the memory as follow

LEFT [k]	INFO [k]	RIGHT [k]

- Sequential Representation of Binary Tree.
- Link Representation of Binary Tree.

1) Linked Representation of Binary Tree

Consider a Binary Tree T. T will be maintained in memory by means of a linked list representation which uses three parallel arrays; INFO, LEFT, and RIGHT pointer variable ROOT as follows. In Binary Tree each node N of T will correspond to a location k such that

LEFT [k] contains the location of the left child of node N.
INFO [k] contains the data at the node N.
RIGHT [k] contains the location of right child of node N.

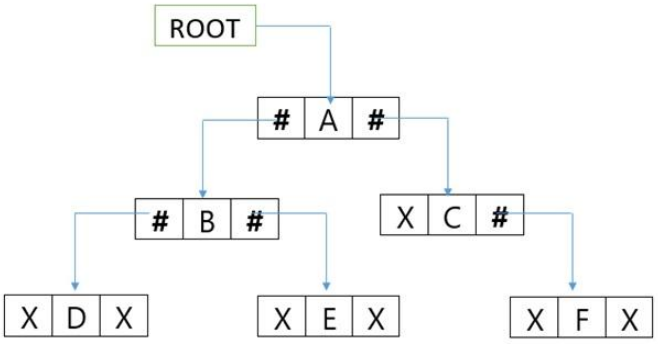
Representation of a node:

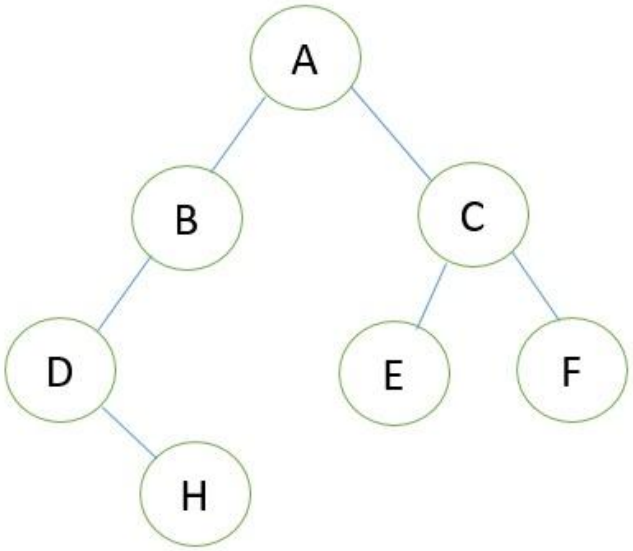
Node Representation

In this representation of binary tree root will contain the location of the root R of T. If any one of the sub tree is empty, then the corresponding pointer will contain the null value if the tree T itself is empty, the ROOT will contain the null value.

Example

Consider the binary tree T in the figure. A schematic diagram of the linked list representation of T appears in the following figure. Observe that each node is pictured with its three fields, and that the empty sub tree is pictured by using x for null entries.





Binary Tree

Binary tree node representations

Linked Representation of Binary Tree

2) Sequential representation of Binary Tree

Let us consider that we have a tree T. let our tree T is a binary tree that us complete binary tree. Then there is an efficient way of representing T in the memory called the sequential representation or array representation of T. This representation uses only a linear array TREE as follows:
The root N of T is stored in TREE [1].
If a node occupies TREE [k] then its left child is stored in TREE [2 * k] and its right child is stored into TREE [2 * k + 1].

For Example:
Consider the following Tree:

Sequential Representation of Binary Tree
Its sequential representation is as follow:

Notes

Sequential Representation of Binary Tree 1

Traversing Binary Trees

Traversing a tree means visiting every node in the tree. You might for instance want to add all the values in the tree or find the largest one. For all these operations, you will need to visit each node of the tree.

Linear data structures like arrays, stacks, queues and linked list have only one way to read the data. But a hierarchical data structure like a tree can be traversed in different ways.
Depending on the order in which we do this, there can be three types of traversal.

Inorder traversal

1. First, visit all the nodes in the left subtree
2. Then the root node
3. Visit all the nodes in the right subtree

inorder(root->left)
display(root->data)
inorder(root->right)

A	B	C	D	-	E	F	-	H		

Preorder traversal

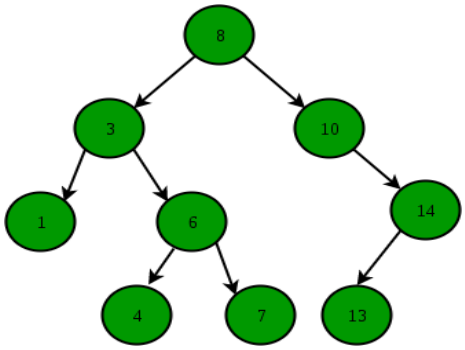
1. Visit root node
2. Visit all the nodes in the left subtree
3. Visit all the nodes in the right subtree

display(root->data)
preorder(root->left)
preorder(root->right)

Postorder traversal

1. visit all the nodes in the left subtree
2. visit the root node
3. visit all the nodes in the right subtree

postorder(root->left)
postorder(root->right)
display(root->data)



Binary search tree

Binary Search Tree is a node-based binary tree data structure which has the following properties:

The left subtree of a node contains only nodes with keys lesser than the node's key.

The right subtree of a node contains only nodes with keys greater than the node's key.

The left and right subtree each must also be a binary search tree.

Insertion in Binary Search Tree:

Check whether root node is present or not(tree available or not). If root is NULL, create root node.

If the element to be inserted is less than the element present in the root node, traverse the left sub-tree recursively until we reach T->left/T->right is NULL and place the new node at T->left(key in new node < key in T)/T->right (key in new node > key in T).

If the element to be inserted is greater than the element present in root node, traverse the right sub-tree recursively until we reach T->left/T->right is NULL and place the new node at T->left/T->right.

Deletion in Binary Search Tree:

How to delete a node from binary search tree?

There are three different cases that needs to be considered for deleting a node from binary search tree.

11.1 Program for Insertion, Deletion and Traversal in Binary Search Tree

```
#include <stdio.h>
#include <stdlib.h>
struct treeNode {
    int data;
    struct treeNode *left, *right;
};
struct treeNode *root = NULL;
struct treeNode* createNode(int data) {
    struct treeNode *newNode;
newNode = (struct treeNode *) malloc(sizeof (struct treeNode));
```

```
newNode->data = data;
newNode->left = NULL;
newNode->right = NULL;
      return(newNode);
  }
  void insertion(struct treeNode **node, int data) {
      if (*node == NULL) {
            *node = createNode(data);
      } else if (data < (*node)->data) {
            insertion(&(*node)->left, data);
      } else if (data > (*node)->data) {
            insertion(&(*node)->right, data);
      }
  }
   void deletion(struct treeNode **node, struct treeNode **parent, int data)
{
      struct treeNode *tmpNode, *tmpParent;
      if (*node == NULL)
            return;
      if ((*node)->data == data) {
            if (!(*node)->left && !(*node)->right) {
                  if (parent)
                        if ((*parent)->left == *node)
                              (*parent)->left = NULL;
                        else
                              (*parent)->right = NULL;
                        free(*node);
                  } else {
                        free(*node);
                  }
            } else if (!(*node)->right && (*node)->left)
tmpNode = *node;
                  (*parent)->right = (*node)->left;
                  free(tmpNode);
                  *node = (*parent)->right;
            } else if ((*node)->right && !(*node)->left) {
tmpNode = *node;
                  (*parent)->left = (*node)->right;
                  free(tmpNode);
                  (*node) = (*parent)->left;
            } else if (!(*node)->right->left)
                  tmpNode = *node;
                  (*node)->right->left = (*node)->left;
                  (*parent)->left = (*node)->right;
                  free(tmpNode);
                  *node = (*parent)->left;
            } else {
                  tmpNode = (*node)->right;
                  while (tmpNode->left) {
tmpParent = tmpNode;
tmpNode = tmpNode->left;
                  }
```

Notes

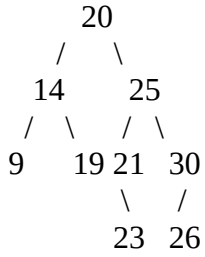
```
tmpParent->left = tmpNode->right;
tmpNode->left = (*node)->left;
tmpNode->right =(*node)->right;
                free(*node);
                *node = tmpNode;
            }
      } else if (data < (*node)->data) {
            deletion(&(*node)->left, node, data);
      } else if (data > (*node)->data) {
           deletion(&(*node)->right, node, data);
      }
  }
  void findElement(struct treeNode *node, int data) {
      if (!node)
            return;
      else if (data < node->data) {
findElement(node->left, data);
      } else if (data > node->data) {
findElement(node->right, data);
      } else
printf("data found: %d\n", node->data);
      return;

  }
  void traverse(struct treeNode *node) {
      if (node != NULL) {
            traverse(node->left);
printf("%3d", node->data);
            traverse(node->right);
      }
      return;
  }
  int main() {
      int data, ch;
      while (1) {
printf("1. Insertion in Binary Search Tree\n");
printf("2. Deletion in Binary Search Tree\n");
printf("3. Search Element in Binary Search Tree\n");
printf("4. Inorder traversal\n5. Exit\n");
printf("Enter your choice:");
scanf("%d", &ch);
            switch (ch) {
                  case 1:
                        while (1) {
                           printf("Enter your data:");
                           scanf("%d", &data);
                           insertion(&root, data);
                           printf("Continue Insertion(0/1):");
                           scanf("%d", &ch);
                        if (!ch)
                              break;
```

```
                    }
                    break;
                case 2:
                    printf("Enter your data:");
                    scanf("%d", &data);
                    deletion(&root, NULL, data);
                    break;
                case 3:
                    printf("Enter value for data:");
                    scanf("%d", &data);
                    findElement(root, data);
                    break;
                case 4:
                    printf("Inorder Traversal:\n");
                    traverse(root);
                    printf("\n");
                    break;
                case 5:
                    exit(0);
                default:
                    printf("u've entered wrong option\n");
                    break;
            }
        }
        return 0;
}
```

Output:

```
1. Insertion in Binary Search Tree
 2. Deletion in Binary Search Tree
 3. Search Element in Binary Search Tree
 4. Inorder traversal
 5. Exit
 Enter your choice:1
 Enter your data:20
 Continue Insertion(0/1):1
 Enter your data:14
 Continue Insertion(0/1):1
 Enter your data:9
 Continue Insertion(0/1):1
 Enter your data:19
 Continue Insertion(0/1):1
 Enter your data:25
 Continue Insertion(0/1):1
 Enter your data:21
 Continue Insertion(0/1):1
 Enter your data:23
 Continue Insertion(0/1):1
 Enter your data:30
 Continue Insertion(0/1):1
 Enter your data:26
 Continue Insertion(0/1):0
```

Resultant Binary Search Tree after insertion operation:



1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit

Enter your choice:4

Inorder Traversal:

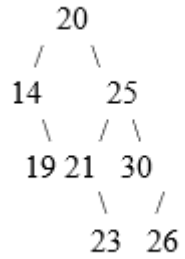
9 14 19 20 21 23 25 26 30

1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit

Enter your choice:2

Enter your data:9

Delete node 9



1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit

Enter your choice:4

Inorder Traversal:

14 19 20 21 23 25 26 30

1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit

Enter your choice:2

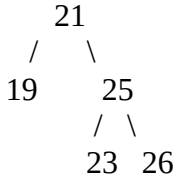
Notes

```
Enter your data:14

Delete node 14
          20
        /    \
      19      25
             /  \
            21   30
             \   /
             23  26

1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:4
Inorder Traversal:
19 20 21 23 25 26 30
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:2
Enter your data:30

Delete node 30
          20
        /    \
      19      25
             /  \
            21   26
             \
             23

1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:4
Inorder Traversal:
19 20 21 23 25 26
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
```

Notes

Enter your choice:2
Enter your data:20

Delete node 20

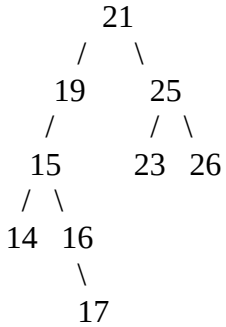


1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit

Enter your choice:4
Inorder Traversal:
19 21 23 25 26

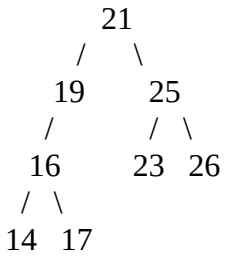
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit

Enter your choice:1
Enter your data:15
Continue Insertion(0/1):1
Enter your data:14
Continue Insertion(0/1):1
Enter your data:16
Continue Insertion(0/1):1
Enter your data:17
Continue Insertion(0/1):0
Binary Search Tree After Insertion Operation:



1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit

```
Enter your choice:4
Inorder Traversal:
14 15 16 17 19 21 23 25 26
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:2
Enter your data:15
Delete Node 15
```




```
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:4
Inorder Traversal:
14 16 17 19 21 23 25 26
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:3
Enter value for data:21
data found: 21
1. Insertion in Binary Search Tree
2. Deletion in Binary Search Tree
3. Search Element in Binary Search Tree
4. Inorder traversal
5. Exit
Enter your choice:5
```

UNIT 12 Graphs

A Graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.

Dijkstra's Algorithm

Dijkstra's algorithm has many variants but the most common one is to find the shortest paths from the source vertex to all other vertices in the graph.

Algorithm Steps:

1. Set all vertices distances = infinity except for the source vertex, set the source distance =0 .
2. Push the source vertex in a min-priority queue in the form (distance , vertex), as the comparison in the min-priority queue will be according to vertices distances.
3. Pop the vertex with the minimum distance from the priority queue (at first the popped vertex = source).
4. Update the distances of the connected vertices to the popped vertex in case of "current vertex distance + edge weight < next vertex distance", then push the vertex
5. With the new distance to the priority queue.
6. If the popped vertex is visited before, just continue without using it.
7. Apply the same algorithm again until the priority queue is empty.

12.1Dijkstra program

```
#include<iostream>
#include<climits>
using namespace std;
#define vertex 7
int minimumDist(int dist[], bool Dset[])
{
        int min=INT_MAX,index;
        for(int v=0;v<vertex;v++)
        {
                if(Dset[v]==false &&dist[v]<=min)
                {
                        min=dist[v];
                        index=v;
                }
        }
        return index;
```

Notes

```
}
void dijkstra(int graph[vertex][vertex],int src)
{
        int dist[vertex];
        bool Dset[vertex];
        for(int i=0;i<vertex;i++)
          {
                dist[i]=INT_MAX;
                Dset[i]=false;
        }
        dist[src]=0;
        for(int c=0;c<vertex;c++)
        {
                int u=minimumDist(dist,Dset);
                Dset[u]=true;
                for(int v=0;v<vertex;v++)
                {
                        if(!Dset[v] && graph[u][v] &&dist[u]!=INT_MAX
                        &&
                        dist[u]+graph[u][v]<dist[v])
                        dist[v]=dist[u]+graph[u][v];
                }
        }
        cout<<"Vertex\t\tDistance from source"<<endl;
        for(int i=0;i<vertex;i++)
        {
                char c=65+i;
                cout<<c<<"\t\t"<<dist[i]<<endl;
        }
}
int main()
{
        int
graph[vertex][vertex]={{0,5,3,0,0,0,0},{0,0,2,0,3,0,1},{0,0,0,7,7,0,0},{2,0
,0,0,0,6,0},{0,0,0,2,0,1,0},{0,0,0,0,0,0,0}, {0,0,0,0,1,0,0}};
        dijkstra(graph,0);
        return 0;
}
```

Output :

```
        Vertex      Distance from source
        A            0
        B            5
        C            3
        D            9
        E            7
        F            8
        G            6
```

Notes

Graphs with Negative Edge costs

C++ Programming examples on "Shortest Path" Bellman–Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. Floyd–Warshall algorithm is an algorithm for finding shortest paths in a weighted graph with positive or negative edge weights.

12.2 Floyd Warshall Algorithm

```
#include <iostream>
#include <vector>
#include <iomanip>
#include <climits>
using namespace std;
		// Data structure to store graph edges
struct Edge
{
	int source, dest, weight;
};
		// Recurive Function to print path of given vertex v from
	source vertex
void printPath(vector<int> const &parent, int v)
{
	if (v < 0)
		return;
	printPath(parent, parent[v]);
	cout<< v << " ";
}
		// Function to run Bellman-Ford algorithm from given
	source
void BellmanFord(vector<Edge> const &edges, int source, int N)
{
		// count number of edges present in the graph
	int E = edges.size();
	vector<int> distance (N, INT_MAX);
	distance[source] = 0;
	vector<int> parent (N, -1);
	int u, v, w, k = N;

		// Relaxation step (run V-1 times)
	while (--k)
	{
		for (int j = 0; j < E; j++)
		{
		// edge from u to v having weight w
			u = edges[j].source, v = edges[j].dest;
			w = edges[j].weight;
```

Notes

```
                        if (distance[u] != INT_MAX && distance[u] + w <
distance[v])
                        {
                        // update distance to the new lower value
                                distance[v] = distance[u] + w;

                        // set v's parent as u
                                parent[v] = u;
                        }
                }
        }
        for (int i = 0; i< E; i++)
        {
                        // edge from u to v having weight w
                u = edges[i].source, v = edges[i].dest;
                w = edges[i].weight;
                if (distance[u] != INT_MAX && distance[u] + w <
distance[v])
                {
                        cout<< "Negative Weight Cycle Found!!";
                        return;
                }
        }
        for (int i = 0; i< N; i++)
        {
                cout<< "Distance of vertex " <<i<< " from the source is "
                        <<setw(2) << distance[i] << ". It's path is [ ";
                printPath(parent, i); cout<< "]" << '\n';
        }
}
                // main function
int main()
{
                // vector of graph edges as per above diagram
        vector<Edge> edges =
        {
                // (x, y, w) -> edge from x to y having weight w
                { 0, 1, -1 }, { 0, 2, 4 }, { 1, 2, 3 }, { 1, 3, 2 },
                { 1, 4, 2 }, { 3, 2, 5 }, { 3, 1, 1 }, { 4, 3, -3 }
        };
                // Set maximum number of nodes in the graph
        int N = 5;
                // let source be vertex 0
        int source = 0;
                // run Bellman-Ford algorithm from given source
        BellmanFord(edges, source, N);
        return 0;
}
```

Output :

Notes

Distance of vertex 0 from the source is 0. It's path is [0]
Distance of vertex 1 from the source is -1. It's path is [01]
Distance of vertex 1 from the source is 2. It's path is [012]
Distance of vertex 1 from the source is -2. It's path is [0143]
Distance of vertex 1 from the source is 1. It's path is [014]

Acyclic graph

Acyclic graph is a directed graph which contains a path from at least one node back to itself. ... An acyclic graph is a directed graph which contains absolutely no cycle, that is no node can be traversed back to itself.

12.3 Program For generate a random DAG (Directed Acyclic Graph) for a given number of edges.

```
#include<iostream>
#include<stdlib.h>
// The maximum number of the vertex for the sample random graph.
#define NOV 20
using namespace std;
// A function to check for the cycle, on addition of a new edge in the
random graph.
bool CheckAcyclic(int edge[][2], int ed, bool check[], int v)
{
        int i;
        bool value;
        // If the current vertex is visited already, then the graph contains
cycle.
        if(check[v] == true)
        {
                return false;
        }
        else
        {
                check[v] = true;
                // For each vertex, go for all the vertex connected to it.
                for(i = ed; i>= 0; i--)
                {
                        if(edge[i][0] == v)
                        {
                        return CheckAcyclic(edge, ed, check, edge[i][1]);
                        }
                }
        }
        // In case, if the path ends then reassign the vertexes visited in that
path to false again.
        check[v] = false;
```

Notes

```
        if(i == 0)
                return true;
}
// A function to generate random graph.
void GenerateRandGraphs(int e)
{
        int i, j, edge[e][2], count;
        bool check[21];
        // Build a connection between two random vertex.
        i = 0;
        while(i< e)
        {
                edge[i][0] = rand()%NOV+1;
                edge[i][1] = rand()%NOV+1;
                for(j = 1; j <= 20; j++)check[j] = false;
                if(CheckAcyclic(edge, i, check, edge[i][0]) == true)
                        i++;
        }
                        // Print the random graph.
        cout<<"\nThe generated random random graph is: ";
        for(i = 0; i< NOV; i++)
        {
                count = 0;
                cout<<"\n\t"<<i+1<<"->{ ";
                for(j = 0; j < e; j++)
                {
                        if(edge[j][0] == i+1)
                        {
                                cout<<edge[j][1]<<"  ";
                                count++;
                        }
                        else if(edge[j][1] == i+1)
                        {
                                count++;
                        }
                        else if(j == e-1 && count == 0)
                                cout<<"Isolated Vertex!";
                }
                cout<<" }";
        }
}
int main()
{
        int e;
        cout<<"Enter the number of edges for the random graphs: ";
        cin>>e;
                        // A function to generate a random undirected graph
with e edges.
        GenerateRandGraphs(e);
}
```

Notes

Runtime Test Cases

Case 1:
Enter the number of edges for the random graphs: 10

The generated random random graph is:

```
1->{  }
2->{ 8  8  12   }
3->{ 5  14   }
4->{ Isolated Vertex! }
5->{  }
6->{ Isolated Vertex! }
7->{ Isolated Vertex! }
8->{ 17   }
9->{ Isolated Vertex! }
10->{ 5   }
11->{ Isolated Vertex! }
12->{ 5   }
13->{ Isolated Vertex! }
14->{  }
15->{ 1   }
16->{ 3   }
17->{  }
18->{ Isolated Vertex! }
19->{ Isolated Vertex! }
20->{ Isolated Vertex! }
```

Case 2:
Enter the number of edges for the random graphs: 50

The generated random random graph is:

```
1->{ 3   }
2->{ 8  8  12  17  12  6   }
3->{ 5  14  14  18   }
4->{ 12  2   }
5->{ 9  15  20  11  13   }
6->{  }
7->{ 6  2  17   }
8->{ 17  7  20  5   }
9->{ 7   }
10->{ 5  13  19  4  2   }
11->{ 3  10   }
12->{ 5  19  9   }
13->{ 3   }
14->{ 5  9  9  16  7   }
15->{ 1   }
16->{ 3  15   }
17->{ 16  1   }
```

Notes

```
18->{ 20   }
19->{ 16  3   }
20->{  }
```

All pair shortest paths algorithm

The all-pairs shortest path problem is the determination of the shortest graph distances between every pair of vertices in a given graph. ... The matrix of all distances between pairs of vertices is called the graph distance matrix, or sometimes the all-pairs shortest path matrix.

12.4 Program to implement dynamic programming algorithm to solve the all pairs shortest path problem

```
#include<iostream>
#include<conio.h>
using namespace std;
int min(int a,int b);
int cost[10][10],a[10][10],i,j,k,c;
int main()
{
        int n,m;
        cout<<"Enter no of vertices";
        cin>> n;
        cout<<"Enter no od edges";
        cin>> m;
        cout<<"Ebter the\nEDGE Cost\n";
        for(k=1;k<=m;k++)
        {
                cin>>i>>j>>c;
                a[i][j]=cost[i][j]=c;
        }
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
                if(a[i][j]== 0 &&i !=j)
                a[i][j]=31999;
        }
        for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        a[i][j]=min(a[i][j],a[i][k]+a[k][j]);
        cout<<"Resultant adj matrix\n";
        for(i=1;i<=n;i++)
        {
        for( j=1;j<=n;j++)
        {
        if(a[i][j] !=31999)
        cout<< a[i][j] <<" ";
```

Notes

```
  }
cout<<"\n";
}
getch();
}
int min(int a,int b)
{
if(a<b)
return a;
else
return b;
}
```

Output :

```
Enter no of vertices3
Enter no od edges5
Enter the
EDGE Cost
1 2 4
2 1 6
1 3 11
3 1 3
2 3 2
Resultant adj matrix
0 4 6
5 0 2
3 7 0
```

Minimum-cost spanning tree.

A Minimum Spanning Tree (MST) works on graphs with directed and weighted (non-negative costs) edges. Consider a graph G with n vertices. The spanning tree is a subgraph of graph G with all its n vertices connected to each other using n-1 edges.

Kruskal's algorithm

Kruskal's algorithm is a minimum-spanning-tree algorithm which finds an edge of the least possible weight that connects any two trees in the forest. It is a greedy algorithm in graph theory as it finds a minimum spanning tree for a connected weighted graph adding increasing cost arcs at each step.

Application of Minimum Spanning Tree

1. Consider n stations are to be linked using a communication network & laying of communication links between any two stations involves a cost.

Notes

2. The ideal solution would be to extract a subgraph termed as minimum cost spanning tree.
3. Suppose you want to construct highways or railroads spanning several cities then we can use the concept of minimum spanning trees.
4. Designing Local Area Networks.
5. Laying pipelines connecting offshore drilling sites, refineries and consumer markets.
6. Suppose you want to apply a set of houses with
   - Electric Power
   - Water
   - Telephone lines
   - Sewage lines

To reduce cost, you can connect houses with minimum cost spanning trees.

12.5 Program for Kruskal's algorithm to find Minimum Spanning Tree of a given connected ,undirected and weighted graph

```
#include <bits/stdc++.h>
using namespace std;
                // a structure to represent a weighted edge in graph
class Edge
{
   public:
   int src, dest, weight;
};
                // a structure to represent a connected, undirected
                // and weighted graph
class Graph
{
   public:
                // V-> Number of vertices, E-> Number of edges
   int V, E;
                // graph is represented as an array of edges.
                // Since the graph is undirected, the edge
                // from src to dest is also edge from dest
                // to src. Both are counted as 1 edge here.
   Edge* edge;
};
                // Creates a graph with V vertices and E edges
Graph* createGraph(int V, int E)
{
   Graph* graph = new Graph;
   graph->V = V;
   graph->E = E;
   graph->edge = new Edge[E];
   return graph;
}
                // A structure to represent a subset for union-find
```

Notes

```
class subset
{
   public:
   int parent;
   int rank;
};
                    // A utility function to find set of an element i
                    // (uses path compression technique)
int find(subset subsets[], int i)
{
                    // find root and make root as parent of i
                    // (path compression)
   if (subsets[i].parent != i)
      subsets[i].parent = find(subsets, subsets[i].parent);
   return subsets[i].parent;
}
                    // A function that does union of two sets of x and y
                    // (uses union by rank)
void Union(subset subsets[], int x, int y)
{
   int xroot = find(subsets, x);
   int yroot = find(subsets, y);
                    // Attach smaller rank tree under root of high
                    // rank tree (Union by Rank)
   if (subsets[xroot].rank< subsets[yroot].rank)
      subsets[xroot].parent = yroot;
   else if (subsets[xroot].rank> subsets[yroot].rank)
      subsets[yroot].parent = xroot;
                    // If ranks are same, then make one as root and
                    // increment its rank by one
   else
   {
      subsets[yroot].parent = xroot;
      subsets[xroot].rank++;
   }
}
                    // Compare two edges according to their weights.
                    // Used in qsort() for sorting an array of edges
int myComp(const void* a, const void* b)
{
   Edge* a1 = (Edge*)a;
   Edge* b1 = (Edge*)b;
   return a1->weight > b1->weight;
}
                    // The main function to construct MST using Kruskal's
          algorithm
void KruskalMST(Graph* graph)
{
   int V = graph->V;
```

Notes

```
    Edge result[V]; // Tnis will store the resultant MST
    int e = 0;
    int i = 0;
qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);
                    // Allocate memory for creating V ssubsets
    subset *subsets = new subset[( V * sizeof(subset) )];
                    // Create V subsets with single elements
    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
                    // Number of edges to be taken is equal to V-1
    while (e < V - 1 &&i< graph->E)
    {
        Edge next_edge = graph->edge[i++];
        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);
        if (x != y)
        {
            result[e++] = next_edge;
Union(subsets, x, y);
        }
                    // Else discard the next_edge
    }
                    // print the contents of result[] to display the
             // built MST
cout<<"Following are the edges in the constructed MST\n";
    for (i = 0; i< e; ++i)
cout<<result[i].src<<" -- "<<result[i].dest<<" ==
"<<result[i].weight<<endl;
    return;
}
// Driver code
int main()
{
    int V = 4;    // Number of vertices in graph
    int E = 5;    // Number of edges in graph
    Graph* graph = createGraph(V, E);
                    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 10;
                    // add edge 0-2
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 6;
           // add edge 0-3
    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;
    graph->edge[2].weight = 5;
```

Notes

```
                // add edge 1-3
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 15;
                // add edge 2-3
    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
    graph->edge[4].weight = 4;
KruskalMST(graph);
    return 0;
}
```

Output:

```
Following are the edges in the constructed MST
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
```

Prims algorithm

Prim's algorithm is a greedy algorithm that finds a minimum spanning tree for a connected weighted undirected graph. It finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized.

The program below implements Prim's algorithm in C++. Although adjacency matrix representation of graph is used, this algorithm can also be implemented using Adjacency List to improve its efficiency.

12.6Program for Prim's Algorithm Implementation

```
#include <iostream>
#include <cstring>
using namespace std;
#define INF 9999999
#define V 5
int G[V][V] = {
  {0, 9, 75, 0, 0},
  {9, 0, 95, 19, 42},
  {75, 95, 0, 51, 66},
  {0, 19, 51, 0, 31},
  {0, 42, 66, 31, 0}
};
int main()
{
  int no_edge;
  int selected[V];
memset (selected, false, sizeof (selected));
```

Notes

```
no_edge = 0;
selected[0] = true;
  int x;           //  row number
  int y;           //  col number
          // print for edge and weight
cout<< "Edge" <<" : " << "Weight";
cout<<endl;
  while (no_edge< V - 1)
{
     int min = INF;
     x = 0;
     y = 0;
     for (int i = 0; i< V; i++) {
       if (selected[i]) {
          for (int j = 0; j < V; j++) {
            if (!selected[j] && G[i][j]) {
               if (min > G[i][j]) {
                  min = G[i][j];
                  x = i;
                  y = j;
               }

            }
         }
       }
     }
cout<< x <<  " - " << y << " :  " << G[x][y];
cout<<endl;
     selected[y] = true;
no_edge++;
   }
  return 0;
}
```

Output:

```
Edge : Weight
0 - 1 :  9
1 - 3 :  19
3 - 4 :  31
3 - 2 :  51
```

Breadth First Traversal

Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

Notes

12.7Programs for the implementation of BFS for a given graph

```
#include<iostream>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int cost[10][10],i,j,k,n,qu[10],front,rare,v,visit[10],visited[10];
main()
{
        int m;
        cout<<"Enterno of vertices";
        cin>> n;
        cout<<"Enter no of edges";
        cin>> m;
        cout<<"\nEDGES \n";
        for(k=1;k<=m;k++)
        {
                cin>>i>>j;
                cost[i][j]=1;
        }
        cout<<"Enter initial vertex";
        cin>>v;
        cout<<"Visitied vertices\n";
        cout<< v;
        visited[v]=1;
        k=1;
        while(k<n)
        {
        for(j=1;j<=n;j++)
        if(cost[v][j]!=0 && visited[j]!=1 && visit[j]!=1)
        {
                visit[j]=1;
                qu[rare++]=j;
        }
        v=qu[front++];
        cout<<v << " ";
        k++;
        visit[v]=0; visited[v]=1;
}
}
```

Output :

```
Enterno of vertices9
Enter  no of edges9
EDGES
```

Notes

```
1 2
2 3
1 5
1 4
4 7
7 8
8 9
2 6
5 7
Enter initial vertex1
Visited vertices
12 4 5 3 6 7 8 9
```

BLOCK 5 SEARCHING AND SORTING ALGORITHMS

UNIT 13 Searching Techniques: Linear and Binary search Programs

Linear Search

A Linear Search is the most basic type of searching algorithm. A Linear Search sequentially moves through your collection (or data structure) looking for a matching value. In other words, it looks down a list, one item at a time, without jumping. Think of it as a way of finding your way in a phonebook.

13.1 Simple Linear Search Example Program (Sequential search)

```
#include<iostream.h>
#include<conio.h>
class linear
{
        int data[50],x;
        int i,n;
        public:
                void get();
                void search();
};
void linear::get()
{
        cout<<"Enter n value:\t";
        cin>>n;
        cout<<"Enter the "<<n<<" values....\n";
        for(i=0;i<n;i++)
        {
                cin>>data[i];
        }
}

void linear::search()
{
        int y,f=0;
        cout<<"\nEnter the value to search:";
        cin>>y;
```

```
        for(i=0;i<n;i++)
        {
        if(y==data[i])
         {
                f=1;
                break;
         }
        }
        if(f==1)
        {
        cout<<"\nThe given search element "<<y<<" found at index "
<<i<<endl;
        }
        else
        {
        cout<<"\n Element Not found";
        }
}

void main()
{
        clrscr();
        linear ob;
        ob.get();
        ob.search();
        getch();
}
```

Output

```
======
Enter n value:  5
Enter the 5 values.... 1 2 3 4 5

Enter the search element:  4
The given search element 4 is found at the index 3
```

Binary Search

Binary search is a fast search algorithm with run-time complexity of O(log n). ... For this algorithm to work properly, the data collection should be in the sorted form. Binary search looks for a particular item by comparing the middle most item of the collection. If a match occurs, then the index of item is returned.

13.2Program for Binary Search

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class binary
{
```

```
        int i,n,a[10],low,high,t,j,flag,x,mid;
        public:
                void getdata();
                void sorting();
                void searching();
};
void binary::getdata()
{
        cout<<"\nEnter the n values:\t";
        cin>>n;
        cout<<"\nEnter the elements one by one:\n";
        for(i=0;i<n;i++)
        {
                cin>>a[i];
        }
}
void binary::sorting()
{
        for(i=0;i<n;i++)
        {
                for(j=n-1;j>i;j--)
                {
                        if(a[j]<a[j-1])
                        {
                                t=a[j];
                                a[j]=a[j-1];
                                a[j-1]=t;
                        }
                }
        }
                cout<<"\nAfter sorting:\t";
                for (i=0;i<n;i++)
                {
                        cout<<a[i]<<"\t";
                }
}
void binary::searching()
{
        int x;
        flag=1;
        low=0;
        high=n-1;
        cout<<"\n\nEnter the searching element:\t";
        cin>>x;
        while(low<=high)
        {
                mid=(low+high)/2;
                if(x<a[mid])
                {
```

```
                high=mid-1;
            }
            else
            {
                low=mid+1;
            }
            if(a[mid]==x)
            {
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
        }
        if(flag==0)
        {
            cout<<"\nThe required Element "<<x<<" is found\n";
        }
        else
        {
            cout<<"Element  not found";
        }
}
void main()
{
        clrscr();
        cout<<"\t\t\tBINARY SEARCH\n";
        binary b;
        b.getdata();
        b.sorting();
        b.searching();
        getch();
}
```

Output:

```
        BINARY SEARCH
Enter the n values:
5
Enter the elements one by one:
50
30
10
20
40
After Sorting
10  20 30 40 50
Enter the search element
40
The required Element 40 is found
```

UNIT 14
Sorting techniques:Bubble sort,Quick sort,Insertion sort,Merge sort

Bubble sort:

Bubble sort, sometimes referred to as sinking sort, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent pairs and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

14.1 Program for BUBBLE SORT

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
class bubble
{
        int i,j,k,n,data[20],t;
        public:
                void getdata();
                void calculation();
                void display();
};
void bubble::getdata()
{
        cout<<"Enter the value of n:\t";
        cin>>n;
        cout<<"Enter the elements:\n";
        for(i=0;i<n;i++)
        {
                cin>>data[i];
        }
}
void bubble::calculation()
{
        for(i=0;i<n;i++)
        {
                for(j=n-1;j>i;j--)
                {
                        if(data[j]<data[j-1])
                        {
                                t=data[j];
                                data[j]=data[j-1];
                                data[j-1]=t;
```

```
                }
            }
            cout<<"\nAfter"<<i<<"iteration\n";
            for(k=0;k<n;k++)
            {
                cout<<data[k]<<"\t";
            }
        }
}
void bubble::display()
{
        cout<<"\nAfter sorting\n";
        for (i=0;i<n;i++)
        {
            cout<<data[i]<<"\t";
        }
}
void main()
{
        clrscr();
        cout<<"\t\t\tBUBBLE SORT\n";
        bubble  i;
        i.getdata();
        i.calculation();
        i.display();
        getch();
}
```

Output:

```
                BUBBLE SORT
Enter the value of n:
5
Enter the elements:
2
8
5
6
1
After  0 iteration
2 8 5 1 6
After  0 iteration
2 8 1 5 6
After  0 iteration
2 1 8 5 6
After  0 iteration
1 2 8 5 6
After  1 iteration
1 2 8 5 6
After  1 iteration
1 2 5 8 6
After  1 iteration
```

```
1 2 5 8 6
After  2 iteration
1 2 5 6 8
After  3 iteration
1 2 5 6 8
After Sorting
1 2 5 6 8
```

Quick sort:

Quick sort is one of the most famous sorting algorithms based on divide and conquers strategy which results in an O(n log n) complexity. So, the algorithm starts by picking a single item which is called pivot and moving all smaller items before it, while all greater elements in the later portion of the list.

14.2Program for Quick Sort

```
#include<iostream.h>
#include<conio.h>
void main()
{
        void quicksort(int [],int,int);
        int data[50],i,lb,mb;
        clrscr();
        cout<<"\t\t\tQUICK SORT\n";
        cout<<"\n\nEnter n value:\t";
        cin>>n;
        cout<<"\n\nEnter the "<<n<<" elements...\n";
        for(i=0;i<n;i++)
        {
                cin>>data[i];
        }
        quicksort(data,0,n-1);
        cout<<"\n\n\nSorted array....\n";
        for(i=0;i<n;i++)
        {
                cout<<data[i]<<endl;
        }
        getch();
}
void quicksort(int data[],int lb,int mb)
{
        int flag,low,high,pivot,t;
        if(lb<mb)
        {
                flag=1;
```

```
                low=lb+1;
                high=mb;
                pivot=data[lb];
                while(flag)
                {
                        while(data[low]<pivot && low<mb)
                        {
                                low++;
                        }
                        while(data[high]>pivot)
                        {
                                high--;
                        }
                        if(low<high)
                        {
                                t=data[low];
                                data[low]=data[high];
                                data[high]=t;
                        }
                        else
                                flag=0;
                }
                t=data[lb];
                data[lb]=data[high];
                data[high]=t;
                cout<<"\nAfter "<<l<<" iteration....\n";
                l++;
                for(int i=0;i<n;i++)
                {
                        cout<<data[i]<<"\t";
                }
                cout<<endl;
                quicksort(data,lb,high-1);
                quicksort(data,high+1,mb);
        }
}

/*OUTPUT
                QUICK SORT
Enter n value: 5

Enter the 5 elements
12 65 3 6 18

After 1 iteration.....
3 6 12 65 18

After 2 iteration....
3 6 12 65 18

After 3 iteration
```

3 6 12 18 65

Sorted array..
3
6
12
18
65

Insertion Sort

Insertion sort is based on the idea that one element from the input elements is consumed in each iteration to find its correct position i.e., the position to which it belongs in a sorted array. Since is the first element has no other element to be compared with, it remains at its position.

Following C++ program ask to the user to enter array size and array element to sort the array using insertion sort technique, then display the sorted array on the screen:

14.3 Programming Code for Insertion Sort

```
#include<iostream.h>
#include<conio.h>
class sort
{
        int data[50],n,i,j,t;
        public:
                void get();
                void sort();
                void display();
};
void sort::get()
{
        cout<<"\nEnter n value:\t";
        cin>>n;
        cout<<"\n\nEnter the "<<n<<" elements...\n";
        for(i=0;i<n;i++)
        {
                cin>>data[i];
        }
}
void sort::sort()
{
        for(i=1;i<=n;i++)
```

```
        {
                t=data[i];
                for(j=i;j>0 && t<data[j-1];j--)
                {
                        data[j]=data[j-1];
                }
                cout<<"\n\nAfter "<<i<<" iteration\n\n";
                for(int k=0;k<n;k++)
                {
                        cout<<data[k]<<"\t";
                }
                data[j]=t;
        }
}
void sort::display()
{
        cout<<"\n\nAfter sorting....\n\n";
        for(i=0;i<n;i++)
        {
                cout<<data[i]<<endl;
        }
}
void main()
{
        clrscr();
cout<<”Insertion Sort \n”;
        sort ob;
        ob.get();
        ob.sort();
        ob.display();
        getch();
}
```

```
Insertion sort
Enter n value: 5
Enter the 5 Elements
15  20  5  8  12
After 1 iteration
15  20  5  8  12
After 2 iteration
15  15  20  8  12
After 3 iteration
5  15  15  20  12
After 4 iteration
5  8  15  15  20
After 5 iteration
5  8  12  15  20

After sorting
5  8  12  15  20
```

Merge sort

Merge sort is a divide-and-conquer algorithm based on the idea of breaking down a list into several sub-lists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list. Idea: Divide the unsorted list into sublists, each containing element.

14.4Program For Merge Sort

```
#include<iostream.h>
#include<conio.h>
int n,data[50];
void merge(int data[],int low,intmid,int high)
{
        int i,j,k,b[50],h;
        i=low;
        h=low;
        j=mid+1;
        while(h<=mid && j<=high)
        {
                if(data[h]<=data[j])
                {
                        b[i]=data[h];
                        h++;
                }
                else
                {
                        b[i]=data[j];
                        j++;
                }
                i++;
        }
        if(h>mid)
        {
                for(k=j;k<=high;k++)
                {
                        b[i]=data[k];
                        i++;
                }
        }
        else
        {
                for(k=h;k<=mid;k++)
                {
                        b[i]=data[k];
                        i++;
```

```
                }
        }
        for(k=low;k<=high;k++)
        {
                data[k]=b[k];
        }
        cout<<endl;

for(i=low;i<=high;i++)
        {
                cout<<data[i]<<" ";
        }
        cout<<endl;
}
void disp()
{
        for(int i=0;i<n;i++)
        {
                cout<<data[i]<<"\t";
        }
        cout<<endl;
}
void mergesort(int data[],int low,int high)
{
        static int i=1;
        int mid;
        if(low<high)
        {
                mid=(low+high)/2;
                mergesort(data,low,mid);
                mergesort(data,mid+1,high);
                merge(data,low,mid,high);
                cout<<"\nAfter "<<i<<" iteration:\t";
                i++;
                disp();
        }
}
void main()
{
        int i;
        clrscr();
        cout<<"Merge Sort"<<endl;
        cout<<"\nEnter n value:\t";
        cin>>n;
        cout<<"\nEnter the "<<n<<" elements.....\n";
        for(i=0;i<n;i++)
        {
                cin>>data[i];
        }
        mergesort(data,0,n-1);
        cout<<"\n\nSorted array.....\n\n";
```

```
        disp();
        getch();
}
```

Output

```
Merge Sort
Enter n value: 5
Enter the 5 elements
6       5       7       1       4

5       6

After 1 iteration: 5    6       7       1       4

5       6       7

After 2 iteration: 5    6       7       1       4

1       4

After 3 iteration: 5    6       7       1       4

1       4       5       6       7

After 4 iteration: 1    4       5       6       7

Sorted array

1       4       5       6       7
```