Department of Computer Science and Engineering

CS6311 PROGRAMMING AND DATA STRUCTURES II LABORATORY

III SEMESTER/ II YEAR

CS6311 PROGRAMMING AND DATA STRUCTURE LABORATORY II
SYLLABUS

OBJECTIVES:

The student should be made to:

- Be familiarized with good programming design methods, particularly Top- Down design.
- Getting exposure in implementing the different data structures using C++
- Appreciate recursive algorithms.

LIST OF EXPERIMENTS:

IMPLEMENTATION IN THE FOLLOWING TOPICS:

1. Constructors & Destructors, Copy Constructor.
2. Friend Function & Friend Class.
3. Inheritance.
4. Polymorphism & Function Overloading.
5. Virtual Functions.
6. Overload Unary & Binary Operators Both as Member Function & Non Member Function.
7. Class Templates & Function Templates.
8. Exception Handling Mechanism.
9. Standard Template Library concept.
10. File Stream classes.
11. Applications of Stack and Queue

12. Binary Search Tree
13. Tree traversal Techniques
14. Minimum Spanning Trees
15. Shortest Path Algorithms

TOTAL: 45 PERIODS

OUTCOMES:

At the end of the course, the student should be able to:

- Design and implement C++ programs for manipulating stacks, queues, linked lists, trees, and graphs.
- Apply good programming design methods for program development.
- Apply the different data structures for implementing solutions to practical problems.
- Develop recursive programs using trees and graphs.

EX NO: 1 SIMPLE C++ PROGRAM

AIM:

To implement a simple C++ program to display the name and number

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the class as person with data members name and number
Step 3: Create an object for the class person
Step 4: Get the name and number as input
Step 5: Access the data member using object and display the output.

PROGRAM:

```
#include <iostream.h>
#include<conio.h>
using namespace std;
class person
{
public:
  string name;
  int number;
};
int main()
{
  person obj;
  cout<<"Enter the Name :";
  cin>>obj.name;
  cout<<"Enter the Number :";
  cin>>obj.number;
  cout << obj.name << ": " << obj.number << endl;
  getch();
  return 0;
}
```

SAMPLE OUTPUT:

Enter the Name: Byron
Enter the Number: 100
Byron: 100

RESULT:

Thus the simple C++ program to display the name and number is implemented successfully.

EX NO: 2

CONSTRUCTOR

AIM:

To implement a C++ program to calculate prime number using Constructor.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the class as Prime with data members and Member functions.
Step 3: Consider the argument constructor Prime () with integer Argument.
Step 4: To cal the function calculate () and do the following steps.
Step 5: For i=2 to a/2 do
Step 6: Check if a%i==0 then set k=0 and break.
Step 7: Else set k value as 1.
Step 8: Increment the value i as 1.
Step 9: Check whether the k value is 1 or 0.
Step 10: If it is 1 then display the value is a prime number.
Step 11: Else display the value is not prime.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
using namespace std;
class prime
{
int a,k,i;
 public:
 prime(int x)
 {
 a=x;
 }
void calculate()
 {
 k=1;
 {
 for(i=2;i<=a/2;i++)
 if(a%i==0)
 {
 k=0;
 break;
 }
 else
 {
 k=1;
 }
 }
 }
 void show()
```

```
 {
 if(k==1)
 cout<<"\n"<<a<<" is Prime Number.";
 else
 cout<<"\n"<<a<<" is Not Prime Numbers.";
 }
};
int main()
{
 int a;
 cout<<"Enter the Number:";
 cin>>a;
 prime obj(a);
 obj.calculate();
 obj.show();
 getch();
 return 0;
}
```

SAMPLE OUTPUT:

Enter the Number: 10
10 is Not Prime Numbers.

Enter the Number:7
7 is Prime Number.

RESULT:

Thus a C++ program to check the prime number using constructor is implemented successfully.

EX NO: 3

DESTRUCTOR

AIM:

To implement a C++ program to print student details using constructor and destructor

ALGORITHM:

Step 1: Include the header files
Step 2: Invoke The Classes
Step 3: Call The Read() Function
Step 4: Get The Inputs Name ,Roll Number And Address
Step 5: Call The Display() Function
Step 6: Display The Name, Roll Number, And Address Of The Student

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
class stu
{
                private: char name[20],add[20];
                         int roll,zip;
                public: stu ( );//Constructor
                         ~stu( );//Destructor
                         void read( );
                         void disp( );
};
stu :: stu( )
{
        cout<<"This is Student Details"<<endl;
}
void stu :: read( )
{
        cout<<"Enter the student Name";
        cin>>name;
        cout<<"Enter the student roll no ";
        cin>>roll;
        cout<<"Enter the student address";
        cin>>add;
        cout<<"Enter the Zipcode";
        cin>>zip;
}
void stu :: disp( )
{
        cout<<"Student Name :"<<name<<endl;
        cout<<"Roll no   is      :"<<roll<<endl;
        cout<<"Address is       :"<<add<<endl;
        cout<<"Zipcode is       :"<<zip;
}
```

```
stu : : ~stu( )
{
        cout<<"Student Detail is Closed";
}
void main( )
{
        stu s;
        clrscr( );
s.read ( );
s.disp ( );
getch( );
}
```

SAMPLE OUTPUT:

```
Enter the student Name
James
Enter the student roll no
01
Enter the student address
Newyork
Enter the Zip code
919108

Student Name: James
Roll no is : 01
Address is : Newyork
Zip code is :919108
```

RESULT:

Thus a C++ program to print student details using constructor and destructor is implemented successfully.

EX NO: 4

COPY CONSTRUCTOR

AIM:

To implement a C++ program to calculate factorial of a given number using copy constructor.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the class name as copy with data members and member functions.
Step 3: The constructor copy () with argument to assign the value.
Step 4: To cal the function calculate () do the following steps.
Step 5: for i=1 to var do
Step 6: Calculate fact*i to assign to fact.
Step 7: Increment the value as 1.
Step 8: Return the value fact.
Step 9: Print the result.
.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
class copy
{
        int var,fact;
        public:

          copy(int temp)
          {
           var = temp;
          }

          double calculate()
          {
                fact=1;
                for(int i=1;i<=var;i++)
                {
                fact = fact * i;
                }
                return fact;
          }
};
void main()
{
   clrscr();
   int n;
   cout<<"\n\tEnter the Number : ";
   cin>>n;
   copy obj(n);
```

```
    copy cpy=obj;
    cout<<"\n\t"<<n<<" Factorial is:"<<obj.calculate();
    cout<<"\n\t"<<n<<" Factorial is:"<<cpy.calculate();
    getch();
}
```

SAMPLE OUTPUT:

```
Enter the Number: 5
Factorial is: 120
Factorial is: 120
```

RESULT:

Thus a C++ program to calculate factorial of a given number using copy constructor is implemented successfully.

EX NO: 5

FRIEND FUNCTION

AIM:

To implement a C++ program to find the mean value of a given number using friend function

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the class name as base with data members and member functions.
Step 3: The function get () is used to read the 2 inputs from the user.
Step 4: Declare the friend function mean (base ob) inside the class.
Step 5: Outside the class to define the friend function and do the following.
Step 6: Return the mean value (ob.val1+ob.val2)/2 as a float.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
class  base
{
   int val1,val2;
  public:
   void get()
   {
     cout<<"Enter two values:";
     cin>>val1>>val2;
   }
   friend float mean(base ob);
};
float mean(base ob)
{
  return float(ob.val1+ob.val2)/2;
}
void main()
{
   clrscr();
   base obj;
   obj.get();
   cout<<"\n Mean value is : "<<mean(obj);
   getch();
}
```

SAMPLE OUTPUT:

Enter two values: 10, 20
Mean Value is: 15

RESULT:

Thus a C++ program to find the mean value of a given number using friend function is implemented successfully.

EX NO: 6

SINGLE INHERITANCE

AIM:

To implement a C++ program to find out the payroll system using single inheritance

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the base class emp.
Step 3: Define and declare the function get () to get the employee details.
Step 4: Declare the derived class salary.
Step 5: Declare and define the function get1 () to get the salary details.
Step 6: Define the function calculate () to find the net pay.
Step 7: Define the function display ().
Step 8: Create the derived class object.
Step 9: Read the number of employees.
Step 10: Call the function get (), get1 () and calculate () to each employee.
Step 11: Call the display ().

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
class emp
{
  public:
    int eno;
    char name[20],des[20];
    void get()
    {
            cout<<"Enter the employee number:";
            cin>>eno;
            cout<<"Enter the employee name:";
            cin>>name;
            cout<<"Enter the designation:";
            cin>>des;
    }
};
class salary:public emp
{
    float bp,hra,da,pf,np;
  public:
    void get1()
    {
            cout<<"Enter the basic pay:";
            cin>>bp;
            cout<<"Enter the Humen Resource Allowance:";
            cin>>hra;
            cout<<"Enter the Dearness Allowance :";
```

```
            cin>>da;
            cout<<"Enter the Profitablity Fund:";
            cin>>pf;
    }
    void calculate()
    {
            np=bp+hra+da-pf;
    }
    void display()
    {
            cout<<eno<<"\t"<<name<<"\t"<<des<<"\t"<<bp<<"\t"<<hra<<"\t"<<da<<"\t"<<pf<<"\t"<<np<
<"\n";
    }
};
 void main()
{
   int i,n;
   char ch;
   salary s[10];
   clrscr();
   cout<<"Enter the number of employee:";
   cin>>n;
   for(i=0;i<n;i++)
   {
            s[i].get();
            s[i].get1();
            s[i].calculate();
   }
   cout<<"\ne_no \t e_name\t des \t bp \t hra \t da \t pf \t np \n";
   for(i=0;i<n;i++)
   {
            s[i].display();
   }
   getch();
}
```

SAMPLE OUTPUT:

```
Enter the Number of employee: 1
Enter the employee No: 150
Enter the employee Name: ram
Enter the designation: Manager
Enter the basic pay: 5000
Enter the HR allowance: 1000
Enter the Dearness allowance: 500
Enter the profitability Fund: 300
 E.No  E.name  des     BP   HRA   DA   PF    NP
150    ram     Manager  5000  1000  500  300   6200
```

RESULT:

Thus a C++ program to find out the payroll system using single inheritance is implemented successfully.

EX NO: 7

MULTIPLE INHERITANCE

AIM:

To implement a C++ program to find out the student details using multiple inheritance.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the base class student.
Step 3: Declare and define the function get () to get the student details.
Step 4: Declare the other class sports.
Step 5: Declare and define the function getsm() to read the sports mark.
Step 6: Create the class statement derived from student and sports.
Step 7: Declare and define the function display () to find out the total and average.
Step 8: Declare the derived class object, call the functions get(),getsm() and display().

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
 class student
{
   protected:
     int rno,m1,m2;
   public:
             void get()
            {
                        cout<<"Enter the Roll no :";
                        cin>>rno;
                        cout<<"Enter the two marks   :";
                        cin>>m1>>m2;
            }
};
class sports
{
   protected:
     int sm;                  // sm = Sports mark
   public:
             void getsm()
            {
              cout<<"\nEnter the sports mark :";
              cin>>sm;

            }
};
class statement:public student,public sports
{
   int tot,avg;
   public:
```

```
    void display()
            {
              tot=(m1+m2+sm);
              avg=tot/3;
              cout<<"\n\n\tRoll No    : "<<rno<<"\n\tTotal      : "<<tot;
             cout<<"\n\tAverage    : "<<avg;
            }
};
void main()
{
  clrscr();
  statement obj;
  obj.get();
  obj.getsm();
  obj.display();
  getch();
}
```

SAMPLE OUTPUT:

```
Enter the Roll no: 100

Enter two marks

90
80

Enter the Sports Mark: 90

Roll No: 100
Total    : 260
Average: 86.66
```

RESULT:

Thus a C++ program to find out the student details using multiple inheritance is implemented successfully.

EX NO: 8

HIERARCHICAL INHERITANCE

AIM:

To implement a C++ program to find out the area of the rectangle and triangle using hierarchical inheritance

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the base class polygon.
Step 3: Declare and define the function input () to get the width and height.
Step 4: Declare the two derived classes rectangle and triangle
Step 5: Calculate the area using width and height
Step 6: Create objects for the derived classes.
Step 7: Declare the derived class object, call the functions input () and calculate the area

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
 class polygon
{
 protected:
  int width, height;
 public:
  void input(int x, int y)
  {
   width = x;
   height = y;
  }
};
 class rectangle : public polygon
{
 public:
  int areaR ()
  {
   return (width * height);
  }
};

class triangle : public polygon
{
 public:
  int areaT ()
  {
   return (width * height / 2);
  }
};

void main ()
```

```
{
 clrscr();
 rectangle rect;
 triangle tri;
 rect.input(6,8);
 tri.input(6,10);
 cout <<"Area of Rectangle: "<<rect.areaR()<< endl;
 cout <<"Area of Triangle: "<<tri.areaT()<< endl;
 getch();
}
```

SAMPLE OUTPUT:

Area of Rectangle: 48

Area of triangle: 30

RESULT:

Thus a C++ program to find out the area of the rectangle and triangle using hierarchical inheritance **is** implemented successfully.

EX NO: 9

MULTILEVEL INHERITANCE

AIM:

To implement a C++ program to find out the student details using multilevel inheritance.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the base class student.
Step 3: Declare and define the function get_ number ( ) and put_number( ).
Step 4: Declare the derived class test for student
Step 5: Declare and define the function get_ marks ( ) and put_marks( ).
Step 6: Declare the derived class result for test
Step 7: Declare and define the function display ( )
Step 8: Create objects for the derived class.
Step 9: Declare the derived class object, call the functions get_ number ( ), get_ marks ( ), display ( ).

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
 class student
{
 protected:
  int roll;
 public:
  void get_number(int a)
  {
   roll = a;
  }
   void put_number()
  {
   cout<<"Roll Number: "<<roll<<"\n";
  }
};
class test : public student
{
 protected:
  float sub1;
  float sub2;
 public:
  void get_marks(float x,float y)
  {
   sub1 = x;
   sub2 = y;
  }
  void put_marks()
  {
   cout<<"Marks in Subject 1 = "<<sub1<<"\n";
   cout<<"Marks in Subject 2 = "<<sub2<<"\n";
```

```
  }

};
class result : public test
{
 private:
 float total;
 public:
 void display()
  {
   total = sub1 + sub2;
   put_number();
   put_marks();
   cout<<"Total = "<<total<<"\n";
  }
};
void main()
{
 clrscr();
 result student;
 student.get_number(83);
 student.get_marks(99.0,98.5);
 student.display();
 getch();
}
```

SAMPLE OUTPUT:

Roll No: 83
Marks in sub 1:99
Marks in sub 2:98.5
Total: 197.5

RESULT:

Thus a C++ program to find out the student details using multilevel inheritance is implemented successfully.

EX NO: 10

HYBRID INHERITANCE

AIM:

To implement a C++ program to find out the student details and sport score using hybrid inheritance.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the base class stu.
Step 3: Declare and define the function get_ no () and put_no().
Step 4: Declare the derived class test for stu
Step 5: Declare and define the function get_ mark () and put_marks().
Step 6: Declare another base class sports
Step 7: Declare and define the function \getscore () and putscore ()
Step 8: Declare the derived class result from test and sports
Step 9: Declare and define the function display ()
Step 8: Create objects for the derived class.
Step 9: Declare the derived class object, call the functions get_ no ( ),get_ mark ( ),getsscore (),display ()

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
class stu
{
protected:
            int rno;
public:
            void get_no(int a)
{
rno=a;
 }
void put_no(void)
 {
 cout<<"Roll no"<<rno<<"\n";
 }
};
class test:public stu
{
 protected:
 float part1,part2;
public:
  void get_mark(float x,float y)
  {
 part1=x;
 part2=y;
  }
void put_marks()
{
 cout<<"Marks obtained:"<<"part1="<<part1<<"\n"<<"part2="<<part2<<"\n";
```

```
}
};
class sports
{
 protected:
 float score;
 public:
 void getscore(float s)
 {
 score=s;
 }
void putscore(void)
{
 cout<<"sports:"<<score<<"\n";
 }
};
 class result: public test, public sports
{
float total;
public:
void display(void);
};
void result::display(void)
{
total=part1+part2+score;
put_no();
put_marks();
putscore();
cout<<"Total Score="<<total<<"\n";
}
int main()
{
clrscr();
result stu;
stu.get_no(123);
stu.get_mark(27.5,33.0);
stu.getscore(6.0);
stu.display();
return 0;
}
```

SAMPLE OUTPUT:

```
Roll no 123
Marks obtained: part1=27.5
Part2=33
Sports=6
Total score = 66.5
```

RESULT:

Thus a C++ program to find out the student details and sport score using hybrid inheritance. is implemented successfully.

EX NO: 11

POLYMORPHISM

AIM:

To implement a C++ program to add and subtract two numbers using polymorphism

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the base class base
Step 3: Declare and define the function setVar() and getresult()
Step 4: Create a pure virtual function op ()
Step 5: Create two derived classes add and sub from base class
Step 6: Declare the virtual function op ()
Step 7: Create an object pointer for base class
Step 8: Create an object for derived classes and access the member functions to find the result.

PROGRAM:

```
#include <iostream.h>
 using namespace std;
 // abstract base class
class base
{
   protected:  // attribute section
      int num1;
      int num2;
      int result;
   public:  // behavior section
   void setVar(int n1,int n2)
      {
         num1 = n1;
         num2 = n2;
      }
      virtual void op() = 0;  // pure virtual function
      int getresult() {return result;}
};
class add: public base  // add class inherits from base class
{
  public:
     void op() {result = num1 + num2;}
};
 //sub class inherit base class
class sub: public base
{
  public:
     void op() {result = num1 - num2;}
};
 int main()
{
   int x,y;
   base *m; //pointer variable declaration of type base class
```

```
    add ad;  //create object1 for addition process
    sub su;  //create object2 for subtraction process
    cout << "\nEnter two numbers separated by space, or press Ctrl+z to Exit: ";
     while(cin >> x >> y)
    {
        m = &ad;
        m->setVar( x , y );
        m->op(); //addition process, even though call is on pointer to base!
        cout << "\nResult of summation = " << m->getresult();
        m = &su;
        m->setVar( x , y );
        m->op(); //subtraction process, even though call is on pointer to base!
        cout << "\nResult of subtraction = " << m->getresult() << endl << endl;
        cout << "\nEnter two numbers seperated by space or press Ctrl+z to Exit: ";
    }
    return 0;
}
```

SAMPLE OUTPUT:

```
Enter two numbers separated by space, or press Ctrl+z to Exit: 88  9

Result of summation = 97
Result of subtraction = 79


Enter two numbers separated by space or press Ctrl+z to Exit: ^Z

Process returned 0 (0x0) execution times: 102.711 s
Press any key to continue.
*/
```

RESULT:

Thus a C++ program to add and subtract two numbers using polymorphism is implemented successfully.

EX NO: 12

FUNCTION OVERLOADING

AIM:

To implement a C++ program to calculate the area of circle, rectangle and triangle using function overloading.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the class name as fn with data members and member functions.
Step 3: Read the choice from the user.
Step 4: Choice=1 then go to the step 5.
Step 5: The function area() to find area of circle with one integer argument.
Step 6: Choice=2 then go to the step 7.
Step 7: The function area() to find area of rectangle with two integer argument.
Step 8: Choice=3 then go to the step 9.
Step 9: The function area() to find area of triangle with three arguments, two as Integer and one as float.
Step 10: Choice=4 then exit

PROGRAM:

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#define pi 3.14
class fn
{
     public:
       void area(int);  //circle
       void area(int,int);  //rectangle
       void area(float ,int,int);  //triangle
};
void fn::area(int a)
{
cout<<"Area of Circle:"<<pi*a*a;
}
void fn::area(int a,int b)
{
     cout<<"Area of rectangle:"<<a*b;
}
void fn::area(float t,int a,int b)
{
     cout<<"Area of triangle:"<<t*a*b;
}
 void main()
{
    int ch;
    int a,b,r;
    clrscr();
```

```
    fn obj;
    cout<<"\n\t\tFunction Overloading";
    cout<<"\n1.Area of Circle\n2.Area of Rectangle\n3.Area of Triangle\n4.Exit\n:”;
    cout<<”Enter your Choice:";
    cin>>ch;
    switch(ch)
    {
          case 1:
            cout<<"Enter Radious of the Circle:";
            cin>>r;
            obj.area(r);
            break;
          case 2:
            cout<<"Enter Sides of the Rectangle:";
            cin>>a>>b;
            obj.area(a,b);
            break;
          case 3:
            cout<<"Enter Sides of the Triangle:";
            cin>>a>>b;
            obj.area(0.5,a,b);
            break;
          case 4:
            exit(0);
    }
getch();
}
```

SAMPLE OUTPUT:

```
Function Overloading
1. Area of Circle
2. Area of Rectangle
3. Area of Triangle
4. Exit
Enter Your Choice: 2

Enter the Sides of the Rectangle: 5 5

Area of Rectangle is: 25

1. Area of Circle
2. Area of Rectangle
3. Area of Triangle
4. Exit
Enter Your Choice: 4
```

RESULT:

Thus a C++ program to calculate the area of circle, rectangle and triangle using function overloading is implemented successfully.

EX NO: 13

VIRTUAL FUNCTION

AIM:

To implement a C++ program to illustrate virtual function

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the base class base.
Step 3: Declare and define the virtual function show().
Step 4: Declare and define the function display().
Step 5: Create the derived class from the base class.
Step 6: Declare and define the functions display() and show().
Step 7: Create the base class object and pointer variable.
Step 8: Call the functions display() and show() using the base class object and pointer.
Step 9: Create the derived class object and call the functions display() and show() using the derived class object and pointer.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
class base
{
   public:
     virtual void show()
     {
            cout<<"\n  Base class show:";
     }
     void display()
     {
           cout<<"\n  Base class display:" ;
     }
};
 class drive:public base
{
  public:
     void display()
     {
           cout<<"\n  Drive class display:";
     }
     void show()
     {
           cout<<"\n  Drive class show:";
     }
};
 void main()
{
  clrscr();
```

```
  base obj1;
  base *p;
  cout<<"\n\t P points to base:\n"  ;
   p=&obj1;
  p->display();
  p->show();
 cout<<"\n\n\t P points to drive:\n";
  drive obj2;
  p=&obj2;
  p->display();
  p->show();
  getch();
}
```

SAMPLE OUTPUT:

P points to Base

Base class display
Base class show

P points to Drive

Base class Display
Drive class Show

RESULT:

Thus a C++ program to illustrate virtual function is implemented successfully.

EX NO: 14

UNARY OPERATOR OVERLOADING

AIM:

To implement a C++ program to find the complex numbers using unary operator overloading.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the class.
Step 3: Declare the variables and its member function.
Step 4: Using the function getvalue() to get the two numbers.
Step 5: Define the function operator ++ to increment the values
Step 6: Define the function operator - -to decrement the values.
Step 7: Define the display function.
Step 8: Declare the class object.
Step 9: Call the function getvalue
Step 10: Call the function operator ++() by incrementing the class object and call the function display.
Step 11: Call the function operator - -() by decrementing the class object and call the function display.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
 class complex
{
    int a,b,c;
   public:
      complex(){}
      void getvalue()
     {
            cout<<"Enter the Two Numbers:";
            cin>>a>>b;
     }
    void operator++()
     {
            a=++a;
            b=++b;
     }
       void operator--()
     {
            a=--a;
            b=--b;
      }
        void display()
     {
            cout<<a<<"+\t"<<b<<"i"<<endl;
      }
};
```

```
void main()
{
    clrscr();
    complex obj;
    obj.getvalue();
    obj++;
    cout<<"Increment Complex Number\n";
    obj.display();
    obj--;
    cout<<"Decrement Complex Number\n";
    obj.display();
    getch();
}
```

SAMPLE OUTPUT:

```
Enter the two numbers: 3 6
Increment Complex Number
4 +             7i
Decrement Complex Number
3 +             6i
```

RESULT:

Thus a C++ program to find the complex numbers using unary operator overloading is implemented successfully.

EX NO: 15

BINARY OPERATOR OVERLOADING

AIM:

To implement a C++ program to add two complex numbers using binary operator overloading.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the class.
Step 3: Declare the variables and its member function.
Step 4: Using the function getvalue() to get the two numbers.
Step 5: Define the function operator +() to add two complex numbers.
Step 6: Define the function operator –()to subtract two complex numbers.
Step 7: Define the display function.
Step 8: Declare the class objects obj1,obj2 and result.
Step 9: Call the function getvalue using obj1 and obj2
Step 10: Calculate the value for the object result by calling the function operator + and operator -.
Step 11: Call the display function using obj1 and obj2 and result.
Step 12: Return the values.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
 class complex
{
        int a,b;
   public:
        void getvalue()
        {
          cout<<"Enter the value of Complex Numbers a,b:";
          cin>>a>>b;
        }
        complex operator+(complex ob)
        {
                complex t;
                t.a=a+ob.a;
                t.b=b+ob.b;
                return(t);
        }
        complex operator-(complex ob)
        {
                complex t;
                t.a=a-ob.a;
                t.b=b-ob.b;
                return(t);
        }
        void display()
        {
                cout<<a<<"+"<<b<<"i"<<"\n";
```

```
        }
};
void main()
{
  clrscr();
  complex obj1,obj2,result,result1;
   obj1.getvalue();
  obj2.getvalue();
   result = obj1+obj2;
  result1=obj1-obj2;
   cout<<"Input Values:\n";
  obj1.display();
  obj2.display();
    cout<<"Result:";
  result.display();
 result1.display();
    getch();
}
```

SAMPLE OUTPUT:

```
Enter the value of Complex Numbers a, b
4               5
Enter the value of Complex Numbers a, b
2               2
Input Values
4 + 5i
2 + 2i
Result
6 +  7i
2 +  3i
```

RESULT:

Thus a C++ program to add two complex numbers using binary operator overloading is implemented successfully.

EX NO: 16

FUNCTION TEMPLATE

AIM:

To implement a C++ program to swap the numbers using the concept of function template.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the template class.
Step 3: Declare and define the functions to swap the values.
Step 4: Declare and define the functions to get the values.
Step 5: Read the values and call the corresponding functions.
Step 6: Display the results.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
template<class t>
 void swap(t &x,t &y)
{
  t temp=x;
  x=y;
  y=temp;
}
void fun(int a,int b,float c,float d)
{
  cout<<"\na and b before swaping :"<<a<<"\t"<<b;
  swap(a,b);
  cout<<"\na and b after swaping  :"<<a<<"\t"<<b;
  cout<<"\n\nc and d before swaping :"<<c<<"\t"<<d;
  swap(c,d);
  cout<<"\nc and d after swaping  :"<<c<<"\t"<<d;
}
void main()
{
   int a,b;
   float c,d;
   clrscr();
   cout<<"Enter A,B values(integer):";
   cin>>a>>b;
   cout<<"Enter C,D values(float):";
   cin>>c>>d;
   fun(a,b,c,d);
   getch();
}
```

SAMPLE OUTPUT:

Enter A, B values (integer): 10 20
Enter C, D values (float): 2.50 10.80
A and B before swapping: 10 20
A and B after swapping: 20 10
C and D before swapping: 2.50 10.80
C and D after swapping: 10.80 2.50

RESULT:

Thus a C++ program to swap the numbers using the concept of function template is implemented successfully.

EX NO: 17

EXCEPTION HANDLING

(a) AIM:

To implement a C++ program to perform exception handling for Divide by zero Exception

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the variables a,b,c.
Step 3: Read the values a,b,c,.
Step 4: Inside the try block check the condition.
a. if(a-b!=0) then calculate the value of d and display.
b. otherwise throw the exception.
Step 5: Catch the exception and display the appropriate message.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
void main()
{
  int a,b,c;
  float  d;
  clrscr();
  cout<<"Enter the value of a:";
  cin>>a;
  cout<<"Enter the value of b:";
  cin>>b;
  cout<<"Enter the value of c:";
  cin>>c;
    try
  {
          if((a-b)!=0)
          {
            d=c/(a-b);
            cout<<"Result is:"<<d;
          }
          else
          {
            throw(a-b);
          }
  }
   catch(int i)
  {
          cout<<"Answer is infinite because a-b is:"<<i;
  }
    getch();
}
```

SAMPLE OUTPUT:

```
Enter the value for a: 20
Enter the value for b: 20
Enter the value for c: 40

Answer is infinite because a-b is: 0
```

RESULT:

Thus a C++ program to perform exception handling for Divide by zero Exception is implemented successfully.

(b) AIM:

To implement a C++ program to perform exception handling with multiple catch

ALGORITHM:

Step 1: Include the header files
Step 2: Declare and define the function test().
Step 3: Within the try block check whether the value is greater than zero or not.
a. if the value greater than zero throw the value and catch the corresponding exception.
b. Otherwise throw the character and catch the corresponding exception.
Step 4: Read the integer and character values for the function test().

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
void test(int x)
{
  try
  {
          if(x>0)
            throw x;
      else
            throw 'x';
  }
   catch(int x)
  {
          cout<<"Catch a integer and that integer is:"<<x;
  }
   catch(char x)
  {
          cout<<"Catch a character and that character is:"<<x;
  }
}
```

```
 void main()
{
   clrscr();
   cout<<"Testing multiple catches\n:";
   test(10);
   test(0);
   getch();
}
```

SAMPLE OUTPUT:

```
Testing multiple catches
Catch a integer and that integer is: 10
Catch a character and that character is: x
```

RESULT:

Thus a C++ program to perform exception handling with multiple catch is implemented successfully.

EX NO: 18

STANDARD TEMPLATE LIBRARY

AIM:

To implement a C++ program to illustrate the concept of standard template library.

ALGORITHM:

Step 1: Include the header files
Step 2: Create a vector to store int
Step 3: Display the original size to store int
Step 4: Push 5 values into the vector using for loop
Step 5: Display extended size of vec
Step 6: Access 5 values from the vector
Step 7: Use iterator to access the values

PROGRAM:

```
#include <iostream.h>
#include <vector>
using namespace std;
 int main()
{
   vector<int> vec;
   int i;
   cout << "vector size = " << vec.size() << endl;
   for(i = 0; i < 5; i++)
{
      vec.push_back(i);
   }
   cout << "extended vector size = " << vec.size() << endl;
      for(i = 0; i < 5; i++)
{
      cout << "value of vec [" << i << "] = " << vec[i] << endl;
   }
   vector<int>::iterator v = vec.begin();
   while( v != vec.end()) {
      cout << "value of v = " << *v << endl;
      v++;
   }
   return 0;
}
```

SAMPLE OUTPUT:

```
vector size = 0
extended vector size = 5
value of vec [0] = 0
value of vec [1] = 1
value of vec [2] = 2
value of vec [3] = 3
value of vec [4] = 4
value of v = 0
value of v = 1
value of v = 2
value of v = 3
value of v = 4
```

RESULT:

Thus a C++ program to illustrate the concept of standard template library is implemented successfully.

EX NO: 19

FILE STREAM CLASSES

(a) AIM:

To implement a C++ program to read the content of a file.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the variables.
Step 3: Get the file name to read.
Step 4: Using ifstreamin(filename) check whether the file exist.
Step 5: If the file exist then check for the end of file condition.
Step 6: Read the contents of the file.
Step 7: Print the contents of the file.
Step 8: Stop the program.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
void main()
{
        char c,fname[10];
        clrscr();
        cout<<"Enter file name:";
        cin>>fname;
        ifstream in(fname);
        if(!in)
        {
                cout<<"File Does not Exist";
                getch();
                return;
        }
        cout<<"\n\n";
        while(in.eof()==0)
        {
                in.get(c);
                cout<<c;
        }
        getch();
}
```

SAMPLE OUTPUT:

```
Enter File name: one.txt
Master of Computer Applications
```

RESULT:

Thus a C++ program to read the content of a file is implemented successfully.

(b) AIM:

To implement a C++ program to perform the write operation within a file.

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the variables.
Step 3: Read the file name.
Step 4: open the file to write the contents.
Step 5: writing the file contents up to reach a particular condition.

PROGRAM:

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<fstream.h>
void main()
{
        char c,fname[10];
        ofstream out;
        cout<<"Enter File name:";
        cin>>fname;
        out.open(fname);
        cout<<"Enter contents to store in file (Enter # at end):\n";
        while((c=getchar())!='#')
        {
                out<<c;
        }
        out.close();
        getch();
}
```

SAMPLE OUTPUT:

```
Enter File name: one.txt
Enter contents to store in file (enter # at end)

Master of Computer Applications#
```

RESULT:

Thus a C++ program to perform the write operation within a file is implemented successfully

(c) AIM:

To implement a C++ program to convert lowercase to uppercase using files

ALGORITHM:

Step 1: Include the header files
Step 2: Declare the variables.
Step 3: Read the file name.
Step 4: open the file to write the contents.
Step 5: writing the file contents up to reach a particular condition.
Step 6: write the file contents as uppercase.
Step 7: open the file to read the contents.

PROGRAM:

```
#include<fstream.h>
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<iostream.h>
#include<conio.h>
void main()
{
        char c,u;
        char fname[10];
        clrscr();
        ofstream out;
        cout<<"Enter File Name:";
        cin>>fname;
        out.open(fname);
        cout<<"Enter the text(Enter # at end)\n";   //write contents to file
        while((c=getchar())!='#')
        {
                u=c-32;
                out<<u;
        }
        out.close();
        ifstream in(fname);        //read the contents of file
        cout<<"\n\n\t\tThe File contains\n\n";
        while(in.eof()==0)
        {
                in.get(c);
                cout<<c;
        }
        getch();
}
```

SAMPLE OUTPUT:

```
Enter File Name: two.txt
Enter contents to store in file (enter # at end)
oops programming
The File Contains
OOPS PROGRAMMING
```

RESULT:

Thus a C++ program to convert lowercase to uppercase using files is implemented successfully

EX NO: 20

STACK APPLICATION –TOWERS OF HANOI

AIM:

To implement a C++ program to illustrate the concept of towers of hanoi problem

ALGORITHM:

Step 1: Include the header files

Step 2: If n=1, move the disk from A to C

Step 3: If n=2, move the 1st disk from A to B then move the 2nd disk from A to C and finally move the 1st disk from B to C

Step 4: If n=3, repeat step 3 to move the first 2 disks from A to B using C as intermediate. Then the 3rd disk is moved from A to C. Then repeat step 3 to move 2 disks from B to C using A as intermediate.

PROGRAM:

```
#include <iostream.h>
#include <conio.h>
void tower(int a,char from,char aux,char to)
{
   if(a==1)
{
     cout<<"\t\tMove disc 1 from "<<from<<" to "<<to<<"\n";
     return;
   }
  else
{
     tower(a-1,from,to,aux);
     cout<<"\t\tMove disc "<<a<<" from "<<from<<" to "<<to<<"\n";
     tower(a-1,aux,from,to);
   }
}
void main()
{
    clrscr();
    int n;
    cout<<"\n\t\t*****Tower of Hanoi*****\n";
    cout<<"\t\tEnter number of discs : ";
    cin>>n;
    cout<<"\n\n";
    tower(n,'A','B','C');
    getch();
}
```

SAMPLE OUTPUT:

```
****Towers of Hanoi****
Enter number of disks: 2
Move disc 1 from A to B
Move disc 2 from A to C
Move disc 1 from B to C
```

RESULT: Thus a C++ program to illustrate the concept of towers of hanoi problem is implemented successfully

EX NO: 21

BINARY SEARCH TREE AND TREE TRAVERSAL

AIM:

To implement a C++ program to illustrate binary search tree and tree traversal techniques

ALGORITHM:

Step 1: Include the header files

Step 2: Declare a structure with two pointer fields; for left child information and right child information, one data field to store the balance factor of each node and another data field to store the node data.

Step 3: Get the elements one by one and insert it using a insert () function.

Insert () function:

(i) Place the first element as root.

(ii) Perform the following steps for the next subsequent insertions.

a) If the element is less than one root and its left sub tree is NULL then place it as the left child of the root.

b) Else loop the insertion function taking the first left sub tree element as the root.

c) If the element is greater than the root and its right sub tree is NULL then place element as the right child of the root.

d) Else loop the insertion function taking the first right sub tree element as the root.

Step 4: For searching any element get the element to search and perform the following steps:

(i) Compare the element with the root node data if it is same then display element is found and its parent is present.

(ii) If the element is less than the root loop the search function taking the first left sub tree element as root.

(iii) If the element is greater than the root loop the search function taking the first right sub tree element as root.

(iv) If the element is not present in the binary search tree then display element is not found.

Step 5: For deleting any element get the element to be deleted and perform the following steps.

(i) If the element is leaf node then delete the node.

(ii) Else if the element has one child deletes it and links its parent node directly with its child node.

(iii) Else if the element has two children then replace with the smallest element of the right sub tree.

Step 6: Display the element of the binary search tree.

Step 7: To traverse a non-empty binary search tree in pre-order, perform the following operations recursively at each node, starting with the root node:

(i) Visit the root.

(ii) Traverse the left sub-tree.

(iii) Traverse the right sub-tree.

Step 8: To traverse a non-empty binary search tree in in-order, perform the following operations recursively at each node, starting with the root node:

(i) Traverse the left sub-tree.

(ii) Visit the root.

(iii) Traverse the right sub-tree.

Step 9: To traverse a non-empty binary search tree in post-order, perform the following operations recursively at each node, starting with the root node:

(i) Traverse the left sub-tree.
(ii) Traverse the right sub-tree.
(iii) Visit the root.

PROGRAM:

```
# include <iostream.h>
# include <cstdlib>
using namespace std;
struct node
{
   int info;
   struct node *left;
   struct node *right;
}*root;
class BST
{
   public:
      void find(int, node **, node **);
      void insert(int);
      void del(int);
      void case_a(node *,node *);
      void case_b(node *,node *);
      void case_c(node *,node *);
      void preorder(node *);
      void inorder(node *);
      void postorder(node *);
      void display(node *, int);
      BST()
      {
         root = NULL;
      }
};
int main()
{
   int choice, num;
   BST bst;
   node *temp;
   while (1)
   {
      cout<<"-----------------"<<endl;
      cout<<"Operations on BST"<<endl;
      cout<<"-----------------"<<endl;
      cout<<"1.Insert Element "<<endl;
      cout<<"2.Delete Element "<<endl;
      cout<<"3.Inorder Traversal"<<endl;
      cout<<"4.Preorder Traversal"<<endl;
      cout<<"5.Postorder Traversal"<<endl;
      cout<<"6.Display"<<endl;
      cout<<"7.Quit"<<endl;
```

```
        cout<<"Enter your choice : ";
        cin>>choice;
        switch(choice)
        {
        case 1:
           temp = new node;
           cout<<"Enter the number to be inserted : ";
                cin>>temp->info;
           bst.insert(root, temp);
        case 2:
           if (root == NULL)
           {
              cout<<"Tree is empty, nothing to delete"<<endl;
              continue;
           }
           cout<<"Enter the number to be deleted : ";
           cin>>num;
           bst.del(num);
           break;
        case 3:
           cout<<"Inorder Traversal of BST:"<<endl;
           bst.inorder(root);
           cout<<endl;
           break;
            case 4:
           cout<<"Preorder Traversal of BST:"<<endl;
           bst.preorder(root);
           cout<<endl;
           break;
        case 5:
           cout<<"Postorder Traversal of BST:"<<endl;
           bst.postorder(root);
           cout<<endl;
           break;
        case 6:
           cout<<"Display BST:"<<endl;
           bst.display(root,1);
           cout<<endl;
           break;
        case 7:
           exit(1);
        default:
           cout<<"Wrong choice"<<endl;
        }
     }
}
 void BST::find(int item, node **par, node **loc)
{
    node *ptr, *ptrsave;
    if (root == NULL)
    {
```

```
        *loc = NULL;
        *par = NULL;
        return;
    }
    if (item == root->info)
    {
        *loc = root;
        *par = NULL;
        return;
    }
    if (item < root->info)
        ptr = root->left;
    else
        ptr = root->right;
    ptrsave = root;
    while (ptr != NULL)
    {
        if (item == ptr->info)
        {
            *loc = ptr;
            *par = ptrsave;
            return;
        }
        ptrsave = ptr;
        if (item < ptr->info)
            ptr = ptr->left;
             else
                ptr = ptr->right;
    }
    *loc = NULL;
    *par = ptrsave;
}
 void BST::insert(node *tree, node *newnode)
{
    if (root == NULL)
    {
        root = new node;
        root->info = newnode->info;
        root->left = NULL;
        root->right = NULL;
        cout<<"Root Node is Added"<<endl;
        return;
    }
    if (tree->info == newnode->info)
    {
        cout<<"Element already in the tree"<<endl;
        return;
    }
    if (tree->info > newnode->info)
    {
        if (tree->left != NULL)
```

```
        {
            insert(tree->left, newnode);
             }
             else
             {
            tree->left = newnode;
            (tree->left)->left = NULL;
            (tree->left)->right = NULL;
            cout<<"Node Added To Left"<<endl;
            return;
        }
    }
    else
    {
        if (tree->right != NULL)
        {
            insert(tree->right, newnode);
        }
        else
        {
            tree->right = newnode;
            (tree->right)->left = NULL;
            (tree->right)->right = NULL;
            cout<<"Node Added To Right"<<endl;
            return;
        }
    }
}
void BST::del(int item)
{
    node *parent, *location;
    if (root == NULL)
    {
        cout<<"Tree empty"<<endl;
        return;
    }
    find(item, &parent, &location);
    if (location == NULL)
    {
        cout<<"Item not present in tree"<<endl;
        return;
    }
    if (location->left == NULL && location->right == NULL)
        case_a(parent, location);
    if (location->left != NULL && location->right == NULL)
        case_b(parent, location);
    if (location->left == NULL && location->right != NULL)
        case_b(parent, location);
    if (location->left != NULL && location->right != NULL)
        case_c(parent, location);
    free(location);
```

```
}
void BST::case_a(node *par, node *loc )
{
   if (par == NULL)
   {
      root = NULL;
   }
   else
   {
      if (loc == par->left)
         par->left = NULL;
      else
         par->right = NULL;
   }
}
 void BST::case_b(node *par, node *loc)
{
   node *child;
   if (loc->left != NULL)
      child = loc->left;
   else
      child = loc->right;
   if (par == NULL)
   {
      root = child;
   }
   else
   {
      if (loc == par->left)
         par->left = child;
      else
         par->right = child;
   }
}
void BST::case_c(node *par, node *loc)
{
   node *ptr, *ptrsave, *suc, *parsuc;
   ptrsave = loc;
   ptr = loc->right;
   while (ptr->left != NULL)
   {
      ptrsave = ptr;
      ptr = ptr->left;
   }
   suc = ptr;
   parsuc = ptrsave;
   if (suc->left == NULL && suc->right == NULL)
      case_a(parsuc, suc);
   else
      case_b(parsuc, suc);
   if (par == NULL)
```

```
    {
        root = suc;
    }
    else
    {
        if (loc == par->left)
            par->left = suc;
        else
            par->right = suc;
    }
    suc->left = loc->left;
    suc->right = loc->right;
}
void BST::preorder(node *ptr)
{
    if (root == NULL)
    {
        cout<<"Tree is empty"<<endl;
        return;
    }
    if (ptr != NULL)
    {
        cout<<ptr->info<<"  ";
        preorder(ptr->left);
        preorder(ptr->right);
    }
}
void BST::inorder(node *ptr)
{
    if (root == NULL)
    {
        cout<<"Tree is empty"<<endl;
        return;
    }
    if (ptr != NULL)
    {
        inorder(ptr->left);
        cout<<ptr->info<<"  ";
        inorder(ptr->right);
    }
}
void BST::postorder(node *ptr)
{
    if (root == NULL)
    {
        cout<<"Tree is empty"<<endl;
        return;
    }
    if (ptr != NULL)
    {
        postorder(ptr->left);
```

```
        postorder(ptr->right);
        cout<<ptr->info<<"  ";
    }
}
void BST::display(node *ptr, int level)
{
    int i;
    if (ptr != NULL)
    {
        display(ptr->right, level+1);
        cout<<endl;
        if (ptr == root)
            cout<<"Root->:  ";
        else
        {
            for (i = 0;i < level;i++)
                cout<<"       ";
             }
        cout<<ptr->info;
        display(ptr->left, level+1);
    }
}
```

SAMPLE OUTPUT:

```
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 8
Root Node is Added
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:
```

```
Root->:  8
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 9
Node Added To Right
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:

            9
Root->:  8
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 5
Node Added To Left
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
```

```
7.Quit
Enter your choice : 6
Display BST:

          9
Root->:   8
          5
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 11
Node Added To Right
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:

               11
          9
Root->:   8
          5
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 3
Node Added To Left
-----------------
```

```
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 7
Node Added To Right
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:

                11
          9
Root->:  8
                7
          5
                3
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 10
Node Added To Left
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
```

```
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:

                11
                    10
        9
Root->:  8
                7
        5
                3
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 2
Enter the number to be deleted : 10
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:

                11
        9
Root->:  8
                7
        5
                3
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
```

```
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 3
Inorder Traversal of BST:
3  5  7  8  9  11
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 4
Preorder Traversal of BST:
8  5  3  7  9  11
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 5
Postorder Traversal of BST:
3  7  5  11  9  8
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 2
Enter the number to be deleted : 8
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
```

```
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:

            11
Root->:  9
                7
            5
                3
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 10
Node Added To Left
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:

            11
                10
Root->:  9
                7
            5
                3
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
```

```
6.Display
7.Quit
Enter your choice : 1
Enter the number to be inserted : 15
Node Added To Right
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:

                15
          11
                10
Root->:  9
                7
          5
                3
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 4
Preorder Traversal of BST:
9  5  3  7  11  10  15
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 5
Postorder Traversal of BST:
3  7  5  10  15  11  9
```

```
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 6
Display BST:

                15
        11
                10
Root->:  9
                7
        5
                3
-----------------
Operations on BST
-----------------
1.Insert Element
2.Delete Element
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Display
7.Quit
Enter your choice : 7
```

RESULT:

Thus a C++ program to illustrate binary search tree and tree traversal techniques is implemented successfully.

EX NO: 22

PRIM'S ALGORITHM

AIM:

To implement a C++ program for Prim's Algorithm to find MST of an undirected graph.

ALGORITHM:

Step 1: Include the header files
Step 2: Get the number of vertices n, vertices and edges weight.
Step 3: Consider any vertex in graph process the vertex and add the vertex to the tree.
Step 4: Find the smallest edge from the graph connecting the edge of the vertex in the tree such that it does not from a cycle.
Step 5: Add that vertex to the tree.
Step 6: Repeat from step 4, until the tree contains all the n vertices.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
using namespace std;
struct node
{
   int fr,to,cost;
}p[6];
int c = 0,temp1 = 0,temp = 0;
void prims(int *a,int b[][7],int i,int j)
{
   a[i] = 1;
   while (c < 6)
   {
      int min = 999;
      for (int i = 0; i < 7; i++)
      {
         if (a[i] == 1)
         {
            for (int j = 0; j < 7; )
            {
               if (b[i][j] >= min || b[i][j] == 0)
               {
                  j++;
               }
               else if (b[i][j] < min)
               {
                  min = b[i][j];
                  temp = i;
                  temp1 = j;
               }
            }
         }
      }
```

```
        a[temp1] = 1;
        p[c].fr = temp;
        p[c].to = temp1;
        p[c].cost = min;
        c++;
        b[temp][temp1] = b[temp1][temp]=1000;
    }
    for (int k = 0; k < 6; k++)
    {
        cout<<"source node:"<<p[k].fr<<endl;
        cout<<"destination node:"<<p[k].to<<endl;
        cout<<"weight of node"<<p[k].cost<<endl;
    }
}
int main()
{
    int a[7];
    for (int i = 0; i < 7; i++)
    {
        a[i] = 0;
    }
    int b[7][7];
    for (int i = 0; i < 7; i++)
    {
        cout<<"enter values for "<<(i+1)<<" row"<<endl;
        for (int j = 0; j < 7; j++)
        {
            cin>>b[i][j];
        }
    }
    prims(a,b,0,0);
    getch();
}
```

SAMPLE OUTPUT:

```
enter values of adjacency matrix for a 7 node graph:

enter values for 1 row
0
3
6
0
0
0
0
enter values for 2 row
3
0
2
4
```

0
0
0
enter values for 3 row
6
2
0
1
4
2
0
enter values for 4 row
0
4
1
0
2
0
4
enter values for 5 row
0
0
4
2
0
2
1
enter values for 6 row
00
0
2
0
2
0
1
enter values for 7 row
0
0
0
4
1
1
0
MINIMUM SPANNING TREE AND ORDER OF TRAVERSAL:

source node:0
destination node:1
weight of node3
source node:1
destination node:2
weight of node2

source node:2
destination node:3
weight of node1
source node:2
destination node:5
weight of node2
source node:5
destination node:6
weight of node1
source node:6
destination node:4
weight of node1

RESULT:

Thus a C++ program for Prim's Algorithm to find MST of an undirected graph is implemented successfully.

EX NO: 23

KRUSKAL'S ALGORITHM

AIM:

To implement a C++ program for Kruskal's Algorithm to find MST of an undirected graph.

ALGORITHM:

Step 1: Include the header files
Step 2: Get the number of vertices n, vertices and edges weight.
Step 3: Get the edge weights and place it in the priority queue in ascending order.
Step 4: Create a disjoint sets where each vertex will be separate disjoint set.
Step 5: Find the minimum value from the priority queue and its connecting vertices.
Step 6: Make a union of those vertices and mark the edges as accepted if it does not form a cycle in the MST and delete the minimum from the priority queue.
Step 7: Repeat from step 3 till all the vertices are connected.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
using namespace std;
int cost[10][10],i,j,k,n,m,c,visit,visited[10],l,v,count,count1,vst,p;
main()
{
int dup1,dup2;
cout<<"enter no of vertices";
cin >> n;
cout <<"enter no of edges";
cin >>m;
cout <<"EDGE Cost";
for(k=1;k<=m;k++)
{
cin >>i >>j >>c;
cost[i][j]=c;
cost[j][i]=c;
}
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(cost[i][j]==0)
cost[i][j]=31999;
visit=1;
while(visit
{
v=31999;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
if(cost[i][j]!=31999 && cost[i][j]
{
int count =0;
for(p=1;p<=n;p++)
{
```

```
if(visited[p]==i || visited[p]==j)
count++;
}
if(count >= 2)
{
for(p=1;p<=n;p++)
if(cost[i][p]!=31999 && p!=j)
dup1=p;
for(p=1;p<=n;p++)
if(cost[j][p]!=31999 && p!=i)
dup2=p;
if(cost[dup1][dup2]==-1)
continue;
}
l=i;
k=j;
v=cost[i][j];
}
cout <<"edge from " <<<"-->"<
cost[l][k]=-1;
cost[k][l]=-1;
visit++;
int count=0;
count1=0;
for(i=1;i<=n;i++)
{
if(visited[i]==l)
count++;
if(visited[i]==k)
count1++;
}
if(count==0)
visited[++vst]=l;
if(count1==0)
visited[++vst]=k;
}
}
```

SAMPLE OUTPUT:

```
enter no of vertices4
enter no of edges4
EDGE Cost
1 2 1
2 3 2
3 4 3
1 3 3
edge from 1-->2edge from 2-->3edge from 1-->3
```

RESULT:

Thus a C++ program for Kruskal's Algorithm to find MST of an undirected graph is implemented successfully.

EX NO: 24

DIJKSTRA'S ALGORITHM

AIM:

To implement a C++ program for Dijkstra's Algorithm to find the shortest path algorithm.

ALGORITHM:

Step 1: Include the header files
Step 2: Save the adjacency matrix in a file.
Step 3: Get as input the source node and destination node.
Step 4: Search the shortest path from source to destination.
Step 5: Initialize all the distances in the shortest path graph to infinity and all the nodes status to infinity.
Step 6: Start with the source node.
Step 7: Find the adjacent node and its shortest distance from the source node.
Step 8: Take the minimum distance value node and repeat step(7) till all the nodes become visited.

PROGRAM:

```
#include<iostream>
#define INFINITY 999
 using namespace std;
 class Dijkstra
{
   private:
      int adjMatrix[15][15];
      int predecessor[15],distance[15];
      bool mark[15];    //keep track of visited node
      int source;
      int numOfVertices;
   public:
      void read();
      void initialize();
       int getClosestUnmarkedNode();
      void calculateDistance();
      void output();
      void printPath(int);
};
 void Dijkstra::read()
{
   cout<<"Enter the number of vertices of the graph(should be > 0)\n";
   cin>>numOfVertices;
   while(numOfVertices <= 0)
 {
      cout<<"Enter the number of vertices of the graph(should be > 0)\n";
      cin>>numOfVertices;
   }
   cout<<"Enter the adjacency matrix for the graph\n";
   cout<<"To enter infinity enter "<<INFINITY<<endl;
   for(int i=0;i<numOfVertices;i++)
```

```
 {
      cout<<"Enter the (+ve)weights for the row "<<i<<endl;
      for(int j=0;j<numOfVertices;j++)
 {
         cin>>adjMatrix[i][j];
         while(adjMatrix[i][j]<0)
{
            cout<<"Weights should be +ve. Enter the weight again\n";
            cin>>adjMatrix[i][j];
         }
      }
   }
   cout<<"Enter the source vertex\n";
   cin>>source;
   while((source<0) && (source>numOfVertices-1))
{
      cout<<"Source vertex should be between 0 and"<<numOfVertices-1<<endl;
      cout<<"Enter the source vertex again\n";
      cin>>source;
   }
}
 void Dijkstra::initialize()
{
   for(int i=0;i<numOfVertices;i++)
 {
      mark[i] = false;
      predecessor[i] = -1;
      distance[i] = INFINITY;
   }
   distance[source]= 0;
}
int Dijkstra::getClosestUnmarkedNode()
{
   int minDistance = INFINITY;
   int closestUnmarkedNode;
   for(int i=0;i<numOfVertices;i++)
{
      if((!mark[i]) && ( minDistance >= distance[i]))
{
         minDistance = distance[i];
         closestUnmarkedNode = i;
      }
   }
   return closestUnmarkedNode;
}
 void Dijkstra::calculateDistance()
{
   initialize();
   int minDistance = INFINITY;
   int closestUnmarkedNode;
   int count = 0;
```

```
   while(count < numOfVertices)
 {
      closestUnmarkedNode = getClosestUnmarkedNode();
      mark[closestUnmarkedNode] = true;
      for(int i=0;i<numOfVertices;i++)
 {
         if((!mark[i]) && (adjMatrix[closestUnmarkedNode][i]>0) )
{
            if(distance[i] > distance[closestUnmarkedNode]+adjMatrix[closestUnmarkedNode][i])
{
               distance[i] = distance[closestUnmarkedNode]+adjMatrix[closestUnmarkedNode][i];
               predecessor[i] = closestUnmarkedNode;
            }
         }
      }
      count++;
   }
}
void Dijkstra::printPath(int node)
{
   if(node == source)
      cout<<(char)(node + 97)<<"..";
   else if(predecessor[node] == -1)
      cout<<"No path from “<<source<<”to "<<(char)(node + 97)<<endl;
   else
{
      printPath(predecessor[node]);
      cout<<(char) (node + 97)<<"..";
   }
}
void Dijkstra::output()
{
   for(int i=0;i<numOfVertices;i++)
 {
      if(i == source)
         cout<<(char)(source + 97)<<".."<<source;
      else
         printPath(i);
      cout<<"->"<<distance[i]<<endl;
   }
}
 int main()
{
   Dijkstra G;
   G.read();
   G.calculateDistance();
   G.output();
   return 0;
}
```

SAMPLE OUTPUT:

```
Enter the number of vertices of the graph(should be > 0)
6
Enter the adjacency matrix for the graph
To enter infinity enter 999
Enter the (+ve)weights for the row 0
0   4    2   999   999   999
Enter the (+ve)weights for the row 1
4   0    1   5   999   999
Enter the (+ve)weights for the row 2
2   1    0   8   10   999
Enter the (+ve)weights for the row 3
999   5   8    0   2    6
Enter the (+ve)weights for the row 4
999    999     10   2   0   3
Enter the (+ve)weights for the row 5
999   999    999   6   3   0
Enter the source vertex
0
a..0->0
a..c..b..->3
a..c..->2
a..c..b..d..->8
a..c..b..d..e..->10
a..c..b..b..e..f->13
```

RESULT:

Thus a C++ program for Dijkstra's Algorithm to find the shortest path algorithm is implemented successfully.