Programming Paradigms Lab

Program List

Introduction to C++

C++ is a statically typed, compiled, general purpose, case sensitive, free form programming language that supports procedural, object oriented, and generic programming. C++ is regarded as a middle level language, as it comprises a combination of both high level and low level language features. C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983. C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

Major features of C++ are

1.Data encapsulation
2.Data Hiding
3.Polymorphism

It has two additional features: Inheritance and dynamic binding

Encapsulation

All C++ programs are composed of the following two fundamental elements:

1.Program statements (code): This is the part of a program that performs actions and they are called functions.

2.Program data: The data is the information of the program which gets affected by the program functions.

Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding.

Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

Data Hiding

It is the process of hiding the background details. Data encapsulation led to the concept data hiding.

Polymorphism

The word polymorphism means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance.

C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Concepts of C++

Class

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performedon such an object.

A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations.

Synatax

```
    class class_name
    {
        // Data members
        // Member functions
    }
```

Objects:

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types.

Inheritance:

Inheritance is the ability of one class to inherit the properties of another.

Constructor:

A class constructor is a special function in a class that is called when a new object of the class is created.

Destructor:

A destructor is also a special function which is called when created object isdeleted.

Use of C++

C++ is used by hundreds of thousands of programmers in essentially every application domain. C++ is being highly used to write device drivers and other software that rely on direct manipulation of hardware under real time constraints. C++ is widely used for teaching and research because it is clean enough for successful teaching of basic concepts.Anyone who has used either an Apple Macintosh or a PC running Windows has indirectly used C++ because the primary user interfaces of these systems are written in C++.

Introduction to JAVA programming

The basic and most importent features are

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

Simple

According to Sun, Java language is simple because of the following reasons

1. Syntax is based on C++ (so easier for programmers to learn it after C++).
2. Removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.
3. No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

Object-oriented

Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour. Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.

Basic concepts of OOPs are

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent

A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's a software-based platform that runs on top of other hardware-based platforms.

It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java is platform independent Java code can be run on multiple platforms e.g.Windows,Linux,Sun Solaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode.This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

Secured

Java is secured mainly because of

1. No explicit pointer
2. Programs run inside virtual machine sandbox.

Classloader- adds security by separating the package for the classes of the local file system from those that are imported from network sources.
Bytecode Verifier- checks the code fragments for illegal code that can violate access right to objects.
Security Manager- determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL,JAAS,cryptography etc.

Robust
Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

Architecture-neutral

There is no implementation dependent features e.g. size of primitive types is set.

Portable

We may carry the java bytecode to any platform.

High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)
Distributed
We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it shares the same memory. Threads are important for multi-media, Web applications etc.

Introduction to LISP

LISP stands for LISt Programming. John McCarthy invented LISP in 1958, shortly after the development of FORTRAN. It was first implemented by Steve Russell on an IBM 704 computer. It is particularly suitable for Artificial Intelligence programs, as it processes symbolic information efficiently. Common LISP originated during the decade of 1980 to 1990, in an attempt to unify the work of several implementation groups, as a successor of Maclisp like ZetaLisp and New Implementation of LISP (NIL) etc.It serves as a common language, which can be easily extended for specific implementation.

Programs written in Common LISP do not depend on machine specific characteristics, such as word length etc.

Features of Common LISP

1.It is machine
2.independent
3.It uses iterative design methodology
4.It has easy extensibility
5.It allows to updatethe programs dynamically
6.It provides high level debugging.
7.It provides advanced object
8.oriented programming.
9.It provides convenient macro system.
10.It provides wide ranging data types like, objects, structures, lists, vectors,adjustable arrays, hash tables, and symbols.
11. It is expression based.
12. It provides an object oriented condition system.
13. It provides complete I/O library.
14. It provides extensive control structures.

Applications Developed in LISP

The following applications are developed in LISP:

1.Emacs: It is across platform editor with the features of extensibility, customizability, self document ability and real time display.
2.G2
3.AutoCad
4.Igor Engraver
5.Yahoo Store

Basic Blocks

LISP programs are made up of three basic building blocks:

1. atom
2. list
3. string

An atom is a number or string of contiguous characters. It includes numbers and special characters.
Following are examples of some valid atoms:

```
hello-from-tutorials-point
name
123008907
*hello*
Block#221
abc123
```

A list is a sequence of atoms and/or other lists enclosed in parentheses. Following are examples of some valid lists:

```
( i am a list)
(a ( a b c) d e fgh)
(father tom ( susan bill joe))
(sun mon tue wed thur fri sat)
( )
```

A string is a group of characters enclosed in double quotation marks. Following are examples of some valid strings:

```
" I am a string"
"a ba c d efg #$%^&!"
"Please enter the following details :"
"Hello from 'Tutorials Point'! "
```

Introduction to PROLOG

Prolog (programming in logic) is one of the most widely used programming languages in articial intelligence research. As opposed to imperative languages such as C or Java (the latter of which also happens to be object-oriented) it is a declarativeprogramming language. That means, when implementing the solution to a problem, instead of specifying how to achieve a certain goal in a certain situation, we specify what the situation (rules and facts) and the goal (query) are and let the Prolog interpreter derive the solution for us. Prolog is very useful in some problem areas, such as articial intelligence, natural language processing, databases, . . . , but pretty useless in others, such as graphics or numerical algorithms.

Terms

The central data structure in Prolog is that of a term. There are terms of four kinds:

- atoms
- numbers
- variables
- compound terms

Atoms and numbers are sometimes grouped together and called atomic terms

Atoms.

Atoms are usually strings made up of lower- and uppercase letters, digits, and the underscore, starting with a lowercase letter.

The following are all valid Prolog atoms:

elephant, b, abcXYZ, x_123, another_pint_for_me_please

On top of that also any series of arbitrary characters enclosed in single quotes denotes an atom.

'This is also a Prolog atom.'

Finally, strings made up solely of special characters like + - * = < > : & (check the manual of your Prolog system for the exact set of these characters) are also atoms.

Examples:

+, ::, <------>, ***

Numbers.

All Prolog implementations have an integer type: a sequence of digits, optionally preceded by a - (minus). Some also support oats. Check the manual for details.

Variables.

Variables are strings of letters, digits, and the underscore, starting with a capital letter or an underscore. Examples:

X, Elephant, _4711, X_1_2, MyVariable, _ The last one of the above examples (the single underscore) constitutes a special case. It is called the anonymous variable and is used when the value of a variable is of no particular interest. Multiple occurrences of the anonymous variable in one expression are assumed to be distinct, i.e., their values don't necessarily have to be the same. More on this later.

Compound terms.

Compound terms are made up of a functor (a Prolog atom) and a number of arguments (Prolog terms, i.e., atoms, numbers, variables, or other compound terms) enclosed in parentheses and separated by commas.

The following are some examples for compound terms:

```
is_bigger(horse, X), f(g(X, _), 7), 'My Functor'(dog)
```

It's important not to put any blank characters between the functor and the opening parentheses, or Prolog won't understand what you're trying to say. In other places, however, spaces can be very helpful for making programs more readable.

The sets of compound terms and atoms together form the set of Prolog predicates. A term that doesn't contain any variables is called a ground term.

Clauses, Programs and Queries

In the introductory example we have already seen how Prolog programs are made up of facts and rules. Facts and rules are also called clauses.

Facts.

A fact is a predicate followed by a full stop. Examples:

```
bigger(whale, _).
life_is_beautiful.
```

The intuitive meaning of a fact is that we define a certain instance of a relation as being true.

Rules.

A rule consists of ahead (a predicate) and a body. (a sequence of predicates separated by commas). Head and body are separated by the sign:- and, like everyProlog expression, a rule has to be terminated by a full stop.

Examples:

```
is_smaller(X, Y) :- is_bigger(Y, X).
aunt(Aunt, Child) :-
sister(Aunt, Parent),
parent(Parent, Child).
```

The intuitive meaning of a rule is that the goal expressed by its head is true, if we (or rather the Prolog system) can show that all of the expressions (subgoals) in the rule's body are true.

Programs.

A Prolog program is a sequence of clauses.

Queries.

After compilation a Prolog program is run by submitting queries to the interpreter. A query has the same structure as the body of a rule, i.e., it is a sequence of predicates separated by commas and terminated by a full stop. They can be entered at the Prolog prompt, which in most implementations looks something like this:?-. Whenwriting about queries we often include the?-. Examples:

```
?- is_bigger(elephant, donkey).
?- small(X), green(X), slimy(X).
```

Intuitively, when submitting a query like the last example, we ask Prolog whether all its predicates are provably true, or in other words whether there is an X such that small(X) ,green(X), and slimy(X) are all true

C++

Program 1:

Aim: Write a c++ program to print the hello world.

```
#include<iostream>

using namespace std;

int main()
{
        cout << "Hello World\n";
        return 0;
}
```

Program 2:

Aim: Write a c++ program to print the arithematic series.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

int main()
{
   double firstTerm, difference, numTerms;

   cout << "What is the first term of the series?";
   cin >> firstTerm;
   cout << endl;
   cout << "What is the difference between adjacent terms?";
   cin >> difference;
   cout << endl;
   cout << "How many terms does the series have?";
   cin >> numTerms;
   cout << endl;
   cout << endl;

   if (numTerms <= 0)
   {
     cout << "  Error  ->  The number of terms must be a positive integer.";
     cout << endl;
     cout << endl;
```

```
      return 0;
    }
    else
    {
    double sum, maxValue, i, d, n;

    maxValue = (firstTerm) + (difference * (numTerms - 1));

    cout << "s(" << firstTerm << "," << difference << "," << numTerms << ") = ";

    i = firstTerm;
    d = difference;
    n = numTerms;

    sum = 0;

    for (i = firstTerm; abs (i) <= abs (maxValue); i = i + d)
    {
        sum = sum + i;

        if ((i) >= 0)
        {
            if (i != abs (maxValue))
            {
                cout << i << " + ";
            }
            else if (i = maxValue)
            {
                cout << i << " = " << sum;
            }
        }
        else if ((i) <= 0)
        {
            if (i != maxValue)
            {
                cout << " - " << abs (i);
            }
            else if (i = maxValue)
            {
               cout << " - " << abs (i) << " = " << sum;
            }
        }
        }
    }
    cout << endl;
    cout << endl;

    return 0;

}
```

Program 3:

Aim: Write a c++ program to implement inheritance in C++. Define a base class student and a derived class marks

```
#include<iostream.h>
#include<stdio.h>
#include<dos.h>
class student
{
	int roll;
	char name[25];
	char add [25];
	char *city;
	public: student()
	{
		cout<<"welcome in the student information system"<<endl;
	}
	void getdata()
	{
		cout<<"\n enter the student roll no.";
		cin>>roll;
		cout<<"\n enter the student name";
		cin>>name;
		cout<<\n enter ther student address";
		cin>>add;
		cout<<"\n enter the student city";
		cin>>city;
	}
	void putdata()
	{
		cout<,"\n the student roll no:"<<roll;
		cout<<"\n the student name:"<<name;
		cout<<"\n the student coty:"<<city;
	}
};
class mrks: public student
{
	int sub1;
	int sub2;
	int sub3;
	int per;
	public: void input()
	{
		getdata();
		cout<<"\n enter the marks1:"
		cin>>sub1:
		cout<<"\n enter the marks2:";
		cin>>sub2;
		cout<<\n enter the marks3:";
```

```
                cin>>sub3;
        }
        void output()
        {
                putdata();
                cout<<"\n marks1:"<<sub1;
                cout<<"\n marks2:"<<sub2;
                cout<<"\n marks3:"<<sub3;
        }
        void calculate ()
        {
                per= (sub1+sub2+sub3)/3;
                cout<<"\n tottal percentage"<<per;
        }
};

void main()
{
        marks m1[25];
        int ch;
        int count=0;
        do
        {
                cout<<\n1.input data";
                cout<<\n2.output data";
                cout<<\n3. Calculate percentage";
                cout<<\n4.exit";
                cout<<\n enter the choice";
                cin>>ch;
                switch (ch)
                {
                        case 1:
                        m1.input();
                        count++;
                        break;

                case2:
                        m1.output();
                        break;

                        case3:
                        m1.calculate();
                        break;
                }
        } while (ch!=4);
}
```

Program 4:

Aim: Write a c++ program to implement virtual function.

```
#include <iostream>
using namespace std;
class B
{
   public:
    virtual void display()      /* Virtual function */
        { cout<<"Content of base class.\n"; }
};

class D1 : public B
{
   public:
      void display()
        { cout<<"Content of first derived class.\n"; }
};

class D2 : public B
{
   public:
      void display()
        { cout<<"Content of second derived class.\n"; }
};

int main()
{
   B *b;
   D1 d1;
   D2 d2;

/* b->display();  // You cannot use this code here because the function of base class is virtual. */

   b = &d1;
   b->display();   /* calls display() of class derived D1 */
   b = &d2;
   b->display();   /* calls display() of class derived D2 */
   return 0;
}
```

Program 5:

Aim: Write a c++ program to implement inheritance.

```
#include <iostream>
using namespace std;

class Shape {
  protected:
    int width, height;
  public:
    Shape( int a=0, int b=0)
    {
      width = a;
      height = b;
    }
    int area()
    {
      cout << "Parent class area :" <<endl;
      return 0;
    }
};
class Rectangle: public Shape{
  public:
    Rectangle( int a=0, int b=0):Shape(a, b) { }
    int area ()
    {
      cout << "Rectangle class area :" <<endl;
      return (width * height);
    }
};
class Triangle: public Shape{
  public:
    Triangle( int a=0, int b=0):Shape(a, b) { }
    int area ()
    {
      cout << "Triangle class area :" <<endl;
      return (width * height / 2);
    }
};
int main( )
{
  Shape *shape;
  Rectangle rec(10,7);
  Triangle  tri(10,5);

  // store the address of Rectangle
  shape = &rec;
  // call rectangle area.
  shape->area();

  // store the address of Triangle
```

```
   shape = &tri;
   // call triangle area.
   shape->area();

   return 0;
}
```

}

JAVA

Program 6:

Aim: Write a java program to implement Binary Search Tree.

```
public class BinarySearchTree {
	public static Node root;
	public BinarySearchTree(){
		this.root = null;
	}

	public boolean find(int id){
	Node current = root;
	while(current!=null){
		if(current.data==id){
			return true;
		}else if(current.data>id){
			current = current.left;
		}else{
			current = current.right;
		}
	}
	return false;
}

public boolean delete(int id){
	Node parent = root;
	Node current = root;
	boolean isLeftChild = false;
	while(current.data!=id){
		parent = current;
		if(current.data>id){
			isLeftChild = true;
			current = current.left;
		}else{
			isLeftChild = false;
			current = current.right;
		}
		if(current ==null){
		return false;
	}
}

//if i am here that means we have found the node
//Case 1: if node to be deleted has no children

if(current.left==null && current.right==null){
        if(current==root){
        root = null;
}

if(isLeftChild ==true){
        parent.left = null;
}else{
        parent.right = null;
        }
}
//Case 2 : if node to be deleted has only one child
else if(current.right==null){
        if(current==root){
        root = current.left;
}else if(isLeftChild){
        parent.left = current.left;
}else{
        parent.right = current.left;
        }
}

else if(current.left==null){
        if(current==root){
        root = current.right;
}else if(isLeftChild){
        parent.left = current.right;
}else{
        parent.right = current.right;
        }
}else if(current.left!=null && current.right!=null){
        //now we have found the minimum element in the right sub tree
        Node successor = getSuccessor(current);
        if(current==root){
                root = successor;
                }else if(isLeftChild){
                parent.left = successor;
        }else{
                parent.right = successor;
        }
        successor.left = current.left;
        }
        return true;
}

public Node getSuccessor(Node deleleNode){
        Node successsor =null;
        Node successsorParent =null;
        Node current = deleleNode.right;
        while(current!=null){
                successsorParent = successsor;
                successsor = current;
                current = current.left;
```

```
}
//check if successor has the right child, it cannot have left child for sure
// if it does have the right child, add it to the left of successorParent.
// successsorParent

if(successsor!=deleleNode.right){
            successsorParent.left = successsor.right;
            successsor.right = deleleNode.right;
}
return successsor;
}

public void insert(int id){
            Node newNode = new Node(id);
            if(root==null){
            root = newNode;
            return;
}

Node current = root;
Node parent = null;
while(true){
parent = current;

if(id<current.data){
            current = current.left;
            if(current==null){
            parent.left = newNode;
            return;
}
}else{
            current = current.right;
            if(current==null){
            parent.right = newNode;
            return;
            }
        }
   }
}

public void display(Node root){
            if(root!=null){
            display(root.left);
            System.out.print(" " + root.data);
            display(root.right);
      }
}

public static void main(String arg[]){
            BinarySearchTree b = new BinarySearchTree();
            b.insert(3);b.insert(8);
            b.insert(1);b.insert(4);b.insert(6);b.insert(2);b.insert(10);b.insert(9);
            b.insert(20);b.insert(25);b.insert(15);b.insert(16);
            System.out.println("Original Tree : ");
            b.display(b.root);
            System.out.println("");
            System.out.println("Check whether Node with value 4 exists : " + b.find(4));
            System.out.println("Delete Node with no children (2) : " + b.delete(2));
            b.display(root);
            System.out.println("\n Delete Node with one child (4) : " + b.delete(4));
            b.display(root);
            System.out.println("\n Delete Node with Two children (10) : " + b.delete(10));
            b.display(root);
      }
}
class Node{
            int data;
            Node left;
            Node right;
            public Node(int data){
            this.data = data;
            left = null;
            right = null;
    }
}
```

Program 7:

Aim: Write a java program to inheritance

```
public abstract class Shape {
    public abstract double area();
    public abstract double perimeter();
}
```

Rectangle.java

```
public class Rectangle extends Shape {
    private final double width, length; //sides

    public Rectangle() {
        this(1,1);
    }
    public Rectangle(double width, double length) {
        this.width = width;
        this.length = length;
    }


    public double area() {
        // A = w * l
        return width * length;
    }


    public double perimeter() {
        // P = 2(w + l)
        return 2 * (width + length);
    }

}
```

Circle.java

```
public class Circle extends Shape {
    private final double radius;
    final double pi = Math.PI;

    public Circle() {
        this(1);
    }
    public Circle(double radius) {
        this.radius = radius;
    }
    public double area() {
        // A = π r^2
        return pi * Math.pow(radius, 2);
    }

    public double perimeter() {
        // P = 2πr
        return 2 * pi * radius;
    }
}
```

Triangle.java

```
public class Triangle extends Shape {
    private final double a, b, c; // sides

    public Triangle() {
        this(1,1,1);
    }
    public Triangle(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }


    public double area() {
        // Heron's formula:
        // A = SquareRoot(s * (s - a) * (s - b) * (s - c))
        // where s = (a + b + c) / 2, or 1/2 of the perimeter of the triangle
        double s = (a + b + c) / 2;
        return Math.sqrt(s * (s - a) * (s - b) * (s - c));
    }


    public double perimeter() {
        // P = a + b + c
        return a + b + c;
    }
}
```

TestShape.java

```
public class TestShape {
    public static void main(String[] args) {

        // Rectangle test
        double width = 5, length = 7;
        Shape rectangle = new Rectangle(width, length);
        System.out.println("Rectangle width: " + width + " and length: " + length
                + "\nResulting area: " + rectangle.area()
                + "\nResulting perimeter: " + rectangle.perimeter() + "\n");

        // Circle test
        double radius = 5;
        Shape circle = new Circle(radius);
        System.out.println("Circle radius: " + radius
            + "\nResulting Area: " + circle.area()
            + "\nResulting Perimeter: " + circle.perimeter() + "\n");

        // Triangle test
        double a = 5, b = 3, c = 4;
        Shape triangle = new Triangle(a,b,c);
        System.out.println("Triangle sides lengths: " + a + ", " + b + ", " + c
                + "\nResulting Area: " + triangle.area()
                + "\nResulting Perimeter: " + triangle.perimeter() + "\n");
    }
}
```

Program 8:

Aim: Write a java program to implement Polymorphism and virtual function

Employee.java

```
public class Employee
{
   private String name;
   private String address;
   private int number;
   public Employee(String name, String address, int number)
   {
      System.out.println("Constructing an Employee");
      this.name = name;
      this.address = address;
      this.number = number;
   }
   public void mailCheck()
   {
      System.out.println("Mailing a check to " + this.name
       + " " + this.address);
   }
   public String toString()
   {
      return name + " " + address + " " + number;
   }
   public String getName()
   {
      return name;
   }
   public String getAddress()
   {
      return address;
   }
   public void setAddress(String newAddress)
   {
      address = newAddress;
   }
   public int getNumber()
   {
     return number;
   }
}
```

Salary.java

```
public class Salary extends Employee
{
   private double salary; //Annual salary
   public Salary(String name, String address, int number, double
      salary)
   {
       super(name, address, number);
       setSalary(salary);
   }
   public void mailCheck()
   {
       System.out.println("Within mailCheck of Salary class ");
       System.out.println("Mailing check to " + getName()
       + " with salary " + salary);
   }
   public double getSalary()
   {
       return salary;
   }
   public void setSalary(double newSalary)
   {
       if(newSalary >= 0.0)
       {
          salary = newSalary;
       }
   }
   public double computePay()
   {
      System.out.println("Computing salary pay for " + getName());
      return salary/52;
   }
}
```

VirtualDemo.java

```
public class VirtualDemo
{
   public static void main(String [] args)
   {
      Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);
      Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);
      System.out.println("Call mailCheck using Salary reference --");
      s.mailCheck();
      System.out.println("\n Call mailCheck using Employee reference--");
      e.mailCheck();
    }
}
```

Program 9:

Aim: Write a java program to implement Least Common ansister

```
// Recursive Java program to print lca of two nodes

// A binary tree node
class Node {

    int data;
    Node left, right;

    Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree {

    static Node root;

    /* Function to find LCA of n1 and n2. The function assumes that both
     n1 and n2 are present in BST */
    Node lca(Node node, int n1, int n2) {
        if (node == null) {
            return null;
        }

        // If both n1 and n2 are smaller than root, then LCA lies in left
        if (node.data > n1 && node.data > n2) {
            return lca(node.left, n1, n2);
        }

        // If both n1 and n2 are greater than root, then LCA lies in right
        if (node.data < n1 && node.data < n2) {
            return lca(node.right, n1, n2);
        }

        return node;
    }

    public static void main(String args[]) {

        // Let us construct the BST shown in the above figure
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(20);
        tree.root.left = new Node(8);
        tree.root.right = new Node(22);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(12);
        tree.root.left.right.left = new Node(10);
        tree.root.left.right.right = new Node(14);

        int n1 = 10, n2 = 14;
        Node t = tree.lca(root, n1, n2);
        System.out.println("LCA of " + n1 + " and " + n2 + " is " + t.data);

        n1 = 14;
        n2 = 8;
        t = tree.lca(root, n1, n2);
        System.out.println("LCA of " + n1 + " and " + n2 + " is " + t.data);

        n1 = 10;
        n2 = 22;
        t = tree.lca(root, n1, n2);
        System.out.println("LCA of " + n1 + " and " + n2 + " is " + t.data);

    }
}
```

Program 10:

Aim: Write a java program to implement Reader writer problem

```
class ReadWriteBuffer<T> {

    private static final Object writeLock = new Object();

    private static final Object readWriteLock = new Object();

    private int numberReaders = 0;

    private int numberWriters = 0;

    private int numberWriteRequests = 0;

    public void write(T object) throws InterruptedException {

        numberWriteRequests++;

        synchronized (readWriteLock) {
            while (numberReaders > 0) {
                readWriteLock.wait();
            }
        }

        synchronized (writeLock) {

            numberWriteRequests--;

            numberWriters++;

            System.out.println("Writer #" + Thread.currentThread().getId() + " started writing.");
            Thread.sleep(3000);
            System.out.println("Writer #" + Thread.currentThread().getId() + " finished writing.");

            numberWriters--;

            synchronized (readWriteLock) {
                readWriteLock.notifyAll();
            }
        }
    }



    public void read() throws InterruptedException {

        synchronized (readWriteLock) {
            while (numberWriters > 0 || numberWriteRequests > 0) {
                readWriteLock.wait();
            }
        }

        numberReaders++;

        System.out.println("Reader #" + Thread.currentThread().getId() + " started reading.");
        Thread.sleep(1000);
        System.out.println("Reader #" + Thread.currentThread().getId() + " finished reading.");

        numberReaders--;

        synchronized (readWriteLock) {
            readWriteLock.notifyAll();
        }
    }

}

class Reader implements Runnable {

    private final ReadWriteBuffer<Object> buffer;

    private int priority;

    public Reader(int priority, ReadWriteBuffer<Object> buffer) {
        this.priority = priority;
        this.buffer = buffer;
    }

    @Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(priority);
                buffer.read();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Writter implements Runnable {

    private final ReadWriteBuffer<Object> buffer;

    private int priority;

    public Writter(int priority, ReadWriteBuffer<Object> buffer) {
        this.priority = priority;
        this.buffer = buffer;
    }

    @Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(priority);
                buffer.write("Object");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class ReadersWriters {

    public static void main(String[] args) {

        ReadWriteBuffer<Object> sharedBuffer = new ReadWriteBuffer<Object>();

        new Thread(new Reader(1, sharedBuffer)).start();
        new Thread(new Reader(700, sharedBuffer)).start();
        new Thread(new Reader(100, sharedBuffer)).start();
        new Thread(new Writter(2000, sharedBuffer)).start();
        new Thread(new Writter(4000, sharedBuffer)).start();
    }
}
```

LISP

Program 11:

Aim: Write a LISP program to implement Quick Sort

```
(defun qsort (L)
   (cond
   ((null L) nil)
   (t(append
      (qsort (list< (car L) (cdr L)))
      (cons (car L) nil)
      (qsort (list>= (car L) (cdr L)))))))

(defun list< (a b)
    (cond
    (( or(null a)(null b) nil))
    (( < a (car b)) (list< a (cdr b)))
    (t(cons (car b) (list< a (cdr b))))))

(defun list>= (a b)
    (cond
    (( or( null a)(null b) nil))
    (( >= a (car b)) (list> a (cdr b)))
    (t(cons (car b) (list> a (cdr b))))))
```

Program 12:

Aim: Write a LISP program to implement set operation

```
; creating myset as an empty list
(defparameter *myset* ())
(adjoin 1 *myset*)
(adjoin 2 *myset*)

; adjoin did not change the original set
;so it remains same
(write *myset*)
(terpri)
(setf *myset* (adjoin 1 *myset*))
(setf *myset* (adjoin 2 *myset*))

;now the original set is changed
(write *myset*)
(terpri)

;adding an existing value
(pushnew 2 *myset*)

;no duplicate allowed
(write *myset*)
(terpri)

;pushing a new value
(pushnew 3 *myset*)
(write *myset*)
(terpri)
```

Program 13:

Aim: Write a LISP program to implement Binary Search Tree

```
(defstruct node
  (data nil) (left nil) (right nil))

(defun insert-data (root-node d &key (lessp #'<)

                              (equalp #'=)
                              (key #'identity))

  "Insert datum d in binary search tree."
  (loop  initially
    (when (null-node-p root-node)
      (setf (node-data root-node) d)
      (return root-node))

    (when (find-data root-node d :lessp lessp :equalp equalp :key key)
      (return root-node))
    with cnode = root-node
    with pnode = nil
    with nnode = (make-node :data d)
    until (null-node-p cnode)

    do (let ((data (funcall key (node-data cnode))))
         (cond
          ((funcall lessp d data)
           (setf pnode cnode)
           (setf cnode (node-left cnode)))
          (t
           (setf pnode cnode)
           (setf cnode (node-right cnode)))))

    finally
    (if (funcall lessp d (funcall key (node-data pnode)))
        (setf (node-left pnode) nnode)
      (setf (node-right pnode) nnode))
    (return root-node)))

(defun find-data (root-node d &key (lessp #'<) (equalp #'=) (key #'identity))
  "Find datum d in binary search tree."

  (loop with cnode = root-node
    until (null-node-p cnode)

    do (let ((data (funcall key (node-data cnode))))
         (cond
          ((funcall equalp d data) (return cnode))
          ((funcall lessp d data)
           (setf cnode (node-left  cnode)))
          (t
         (setf cnode (node-right cnode)))))
    finally (return cnode)))

(defun find-node-and-parent (root d &key (lessp #'<) (equalp #'=) (key #'identity))
  "Find node whose data is d and its parent."

  (loop
    with cnode = root
    with pnode = nil
    until (null-node-p cnode)

    do (let ((data
              (funcall key (node-data cnode))))
         (cond
          ((funcall equalp d data)
           (return (values cnode pnode)))
          ((funcall lessp d data)
           (setf pnode cnode)
           (setf cnode (node-left cnode)))
          (t
           (setf pnode cnode)
           (setf cnode (node-right cnode)))))
    finally (return (values cnode pnode))))

(defun delete-data (root d &key (lessp #'<) (equalp #'=) (key #'identity))
  "Delete node whose data is d from binary search tree."
  (multiple-value-bind (dnode pnode)
    (find-node-and-parent root d :lessp lessp :equalp equalp :key key)
    (let ((new-root root))
      (if (null-node-p dnode)
              new-root
            (let ((rnode nil))
              (cond
               ((null-node-p (node-right dnode))
                (setf rnode (node-left dnode)))
               ((null-node-p (node-left dnode))
                (setf rnode (node-right dnode)))
               (t
                (let ((prnode dnode))
                  (setf rnode (node-left dnode))
                  (loop until (null-node-p (node-right rnode))
                            do (setf prnode rnode)
                               (setf rnode (node-right rnode)))
                  (if (eq prnode dnode)
                          (setf (node-right rnode)
                                  (node-right dnode))
                        (progn
                          (setf (node-right prnode)
                                  (node-left rnode))
                          (setf (node-left rnode)
                                  (node-left dnode))
                          (setf (node-right rnode)
                                (node-right dnode)))))))
              (if (null-node-p pnode)
                  (setf new-root rnode)
                (if (funcall lessp (funcall key (node-data dnode))
                                   (funcall key (node-data pnode)))
                        (setf (node-left pnode) rnode)
                  (setf (node-right pnode) rnode)))
              new-root)))))

(defun depth (root-node)
  "Compute depth of binary search tree."
  (if (null-node-p root-node)
      -1
    (1+ (max (depth (node-left root-node))
             (depth (node-right root-node))))))

(defun bst-to-list (root)
  "Convert binary search tree to list."
  (if (null-node-p root)
      nil
    `(,@(bst-to-list (node-left root))
        ,(node-data root)
        ,@(bst-to-list (node-right root)))))

(defun bst-to-list2 (root)
  "Convert binary search tree to list."
  (if (null-node-p root)
      nil
    (let ((left-list (bst-to-list2 (node-left root)))
          (right-list (bst-to-list2 (node-right root))))
      (append left-list
              (list (node-data root))
              right-list))))

(defun bst-to-sorted-array (root &optional (order :nondecreasing))
  "Convert binary search tree to sorted array."
  (let ((ary (make-array 10 :adjustable t :fill-pointer 0)))
    (ecase order
      (:nondecreasing
       (scan-bst-inorder root #'(lambda (n)
                                  (vector-push-extend (node-data n)
                                                      ary)))
       ary)
      (:nonincreasing
       (scan-bst-inorder root #'(lambda (n)
                                  (vector-push-extend (node-data n) ary)))
       (reverse ary)))))

(defun bst-to-sorted-array2 (root &optional (order :nondecreasing))
  (let ((elts (bst-to-list root)))
    (ecase order
      (:nondecreasing
      (make-array (length elts)
                  :initial-contents elts))
     (:nonincreasing
      (make-array (length elts)
                  :initial-contents (reverse elts))))))

(defun random-bst (num-nodes from upto)
  "Build binary search tree of random integers."
  (let ((r (make-node))
        (range (1+ (- upto from))))
    (dotimes (i num-nodes r)
      (insert-data r (+ from (random range))))))

(defun scan-bst-inorder (root scan-function)
  "Scan binary search tree inorder."
  (unless (null-node-p root)
    (scan-bst-inorder (node-left root) scan-function)
    (funcall scan-function root)
    (scan-bst-inorder (node-right root) scan-function)))

(defun bst-p (root &key (lessp #'<) (key #'identity))
  "Is this a binary search tree?"
  (let ((x nil))
    (catch :no-bst-p
           (scan-bst-inorder root
                             #'(lambda (n)
                                 (let ((d (funcall key (node-data n))))
                                   (if (or (null x)
                                           (not (funcall lessp d x)))
                                         (setf x d)
                                     (throw :no-bst-p nil)))))
           t)))
(defun null-node-p (n)
  "Check if n is empty node."
  (or (null n) (null (node-data n))))

(defun test-delete-data (root data-list num-tests &key (lessp #'<)
                              (equalp #'=)
                              (key #'identity))
  "Test deletion of data from binary search tree rooted at root."
  (loop initially (assert (bst-p root :lessp lessp :key key))
           (format t "yes, it is a binary search tree~%")
           with upper    = (length data-list)
           with new-root = (copy-bst root)
           with deleted  = '()
           for i from 1 upto num-tests
           as d = (nth (random upper) data-list)
           do (format t "Deleting ~S~%" d)
              (push d deleted)
              (setf new-root
                    (delete-data new-root d :lessp lessp :equalp equalp :key key))
              (if (bst-p new-root :lessp lessp :key key)
                (format t "bst so far~%")
                  (progn
                    (format t "not a bst~%")
                    (return (values new-root deleted))))))

(defun copy-bst (root)
  "Create a copy of binary search tree."
  (if (null-node-p root)
      nil
    (make-node :data (node-data root)
               :left (copy-bst (node-left root))
               :right (copy-bst (node-right root)))))

;;; end-of-file
```

PROLOG

Program 14:

Aim: Write a PROLOG program to implement GCD

```
gcd(X,Y):-X=Y,write('GCD of two numbers is '),write(X);
X=0,write('GCD of two numbers is '),write(Y);
Y=0,write('GCD of two numbers is '),write(X);
Y>X,Y1 is Y-X,gcd(X,Y1);
X>Y,Y1 is X-Y,gcd(Y1,Y).
```

Program 15:

Aim: Write a PROLOG program to implement NFA.

```
nfa([], _, S, F) :- member(S, F).
nfa([A|X], D, S, F) :- member((S, A, T), D), nfa(X, D, T, F).
```