Thadomal Shahani Engineering College

Bandra (W.), Mumbai - 400 050.

CERTIFICATE

Certify that Mr./Miss PRATHAM ASNANI

of IT Department, Semester III with

Roll No. 72 has completed a course of the necessary

experiments in the subject CPP2 - LAB under my

supervision in the **Thadomal Shahani Engineering College**

Laboratory in the year 2024 - 2025

Teacher In- Charge

Head of the Department

Date 18/10/2024

Principal

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

CONTENTS

SR. NO.	EXPERIMENTS	PAGE NO.	DATE	TEACHERS SIGN.
1	Scope of variables and call by value reference, stack implementation	1-8	15/7/24	
2	C++ Encapsulation, inheritance and operation overloading	9-15	22/07/24	
3	Java program for polymorphism exception handling, static and dynamic initialization	16-23	29/7/24	A
4	Arithmetic processing using prolog	24-26	5/8/24	Done
5	Prolog queries, basic and advanced prolog	27-30	30/8/24	
6	Arithmetic and logical function in haskel	31-33	2/9/24	
7	Recursive and higher order function in haskel	34-37	16/9/24	
8	Java and C++ program to display number and character concureency in C++	38-40	23/9/24	
9	Password validation using Java script	41-43	11/10/24	
10	Writen assignment	44-46	11/10/24	

1.1:-To study scope of global and local variables

CODE:-

```
#include<stdio.h>
 #include"firstName.h"
extern char pratham; char pratham="a";
int asnani=10;
void f1 ()
{
float pratham= 8.5;

printf("\nValue of pratham (local to a block) is %f", pratham);
printf("\nValue of asnani (global variable) is %d ", asnani);
char pratham='a';
printf("\n");
}
char pratham

void f2()

{

char pratham = 'b';
int asnani=33;
printf("\nValue of pratham (local to a block) is %c", pratham);
printf("\nValue of asnani (local to a function) is %d", asnani);
printf("\n"); }
}
void main()
{
char asnani= 'x';
printf("\nValue of asnani(local to a function) is %c", asnani);
printf("\nValue of pratham (extern variable) is %c", pratham);
printf("\n");
}
f1();
f2();
}

char pratham='a';
```

OUTPUT :-

```
value of asnani(local to function)is x
value of pratham(extern variable)is a

value of pratham(local to block)is 8.500000
value of asnani(global variable)is 10

value of pratham(local to block)is b
value of asnani(local to function)is 33
```

SCOPE OF VARIABLES WITH SAME NAME AND SAME TYPE

CODE :-

```
#include<stdio.h>
#include"lect3lib.c"
extern char c; char a='p';
void f2()
{ char a ='j'; int i;
printf("\nThe value of c at entry in f2 is %c\n",c);
printf("The value of a at entry in f2 is %c\n",a);
a = 'k';
for(i = 0 ; i < 4 ; i++)
{ char c = a++;
printf("The value of c after modification in loop witin f2 is %c\n",c); }
printf("The value of a after modification in f2 is %c\n",a);
printf("The value of c after modification in f2 is %c\n\n",c); }
void main()
{ printf("\nThe value of c at entry in main function is %c\n",c);
printf("The value of a at entry in main function is %c\n",a);
c = 'y'; a = 'q'; f2();
printf("The value of c after modification in main function is %c\n",c);
printf("The value of a after modification in main function is %c\n",a);
}
```

OUTPUT :-

```
The value of c at entry in main function is x
The value of a at entry in main function is p

The value of c at entry in f2 is y
The value of a at entry in f2 is j
The value of c after modification in loop witin f2 is k
The value of c after modification in loop witin f2 is l
The value of c after modification in loop witin f2 is m
The value of c after modification in loop witin f2 is n
The value of a after modification in f2 is o
The value of c after modification in f2 is y

The value of c after modification in main function is y
The value of a after modification in main function is q
```

SCOPE OF VARIABLES WITH SAME NAME AND DIFFERENT TYPES

CODE :-

```
#include<stdio.h>
char a='p';
void f1()
{ int a = 100;
S# printf("\nThe value of integer a in f2 is %d\n\n",a); }
void main()
{ printf("\nThe value of character a at entry in main is %c\n",a);
a = 'q'; f1();
printf("The value of character a after modification in main is %c\n",a);
}
```

OUTPUT :-

```
The value of character a at entry in main is p

The value of integer a in f2 is 100

The value of character a after modification in main is q

Process returned 57 (0x39)   execution time : 0.054 s
Press any key to continue.
```

Name:Pratham Asnani
Roll no:72
Batch:s13

LO mapping:-LO1

1.2:-Call by value vs call by reference, show stack working for prog

AIM:- WAP in C to show difference between call by value and call by reference parameter passing methods. Show working of stack for program.

Call by value call by reference:-

Call by value:-In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.

Call by reference:- "Call by reference" is a method of passing arguments to a function where instead of passing the actual value of the argument, a reference or address to the variable is passed. This means that the function can modify the variable directly, and the changes will be reflected outside the function.

Single Function Call

CODE:-

```
int a = 100;
void f(int x)
{
   int y = 7;
     printf("\nLocal variabe x = %d and has address %p in callee f in Stack Segment \n",x,&x);
     printf("Local variabe y = %d and has address %p in callee f in Stack Segment \n",y,&y); }
void main()
{
  int z = 9;
    printf("\nGlobal variable a = %d and has address %p in Data Segment\n\n",a,&a);
    printf("Local variabe z = %d and has address %p in caller main in Stack Segment\n",z,&z);
 f(z);
```

```
    printf("\nAddress of function main is %p\n",main);
    printf("\nAddress of function f is %p\n",f);
}
```

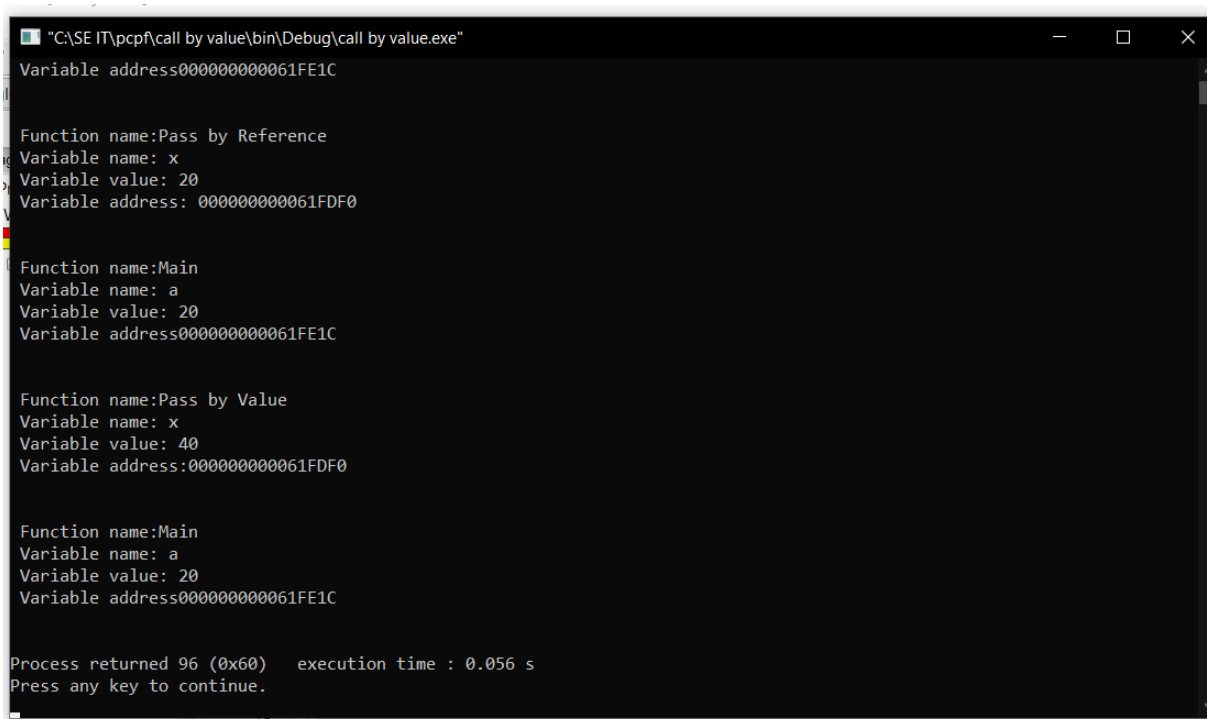
OUTPUT :-

Multiple Function Calls

CODE:-

```
void f2(int b)
 {
printf("\nLocal variable b of callee f2 has value = %d and address %p\n\n",b,&b); }
void f1(int a)
 {
printf("\nLocal variable a of callee f1 has value = %d and address %p\n\n",a,&a); }
void main()
 {int z = 9;
printf("\nLocal variable z of caller function main has value = %d and address %p\n",z,&z);
f1(z);
f2(z);
}
```

OUTPUT :-

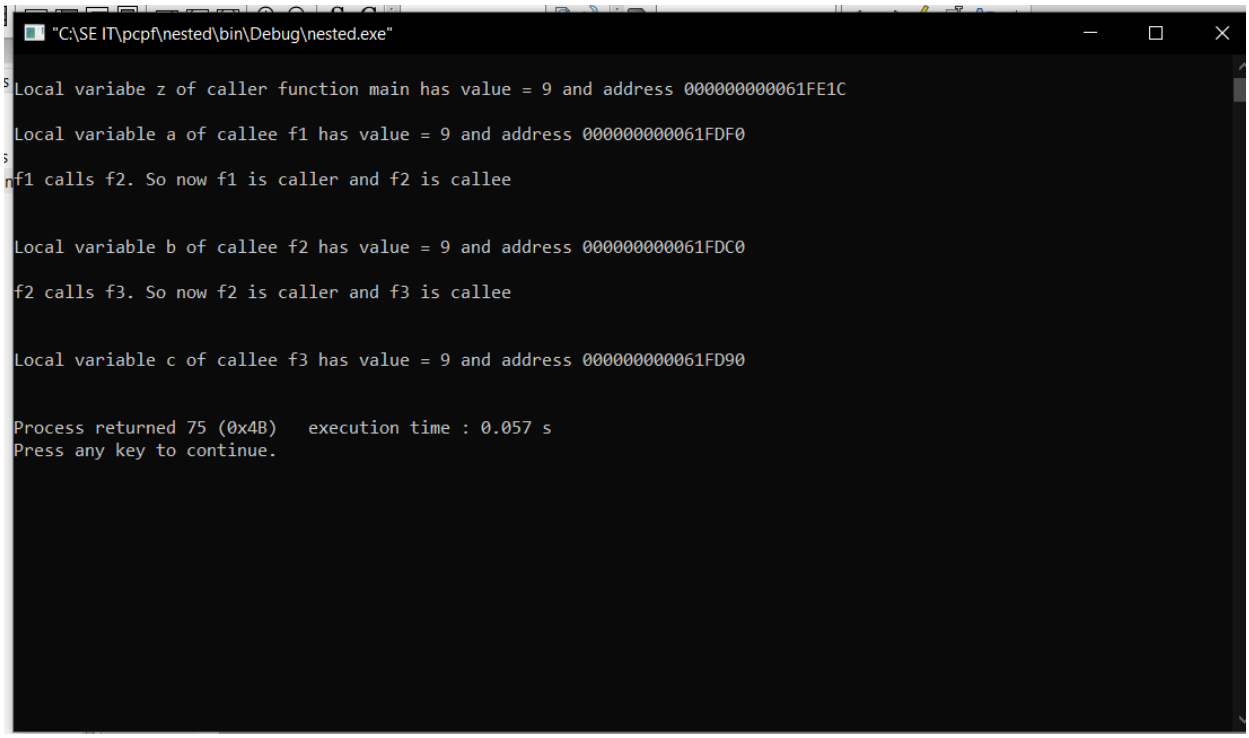


Nested Function Calls

CODE:-

```
void f3(int c)
{
printf("\nLocal variable c of callee f3 has value = %d and address %p\n\n",c,&c);
}
void f2(int b)
{
printf("\nLocal variable b of callee f2 has value = %d and address %p\n\n",b,&b);
printf("f2 calls f3. So now f2 is caller and f3 is callee\n\n"); f3(b);
}
void f1(int a)
{
printf("\nLocal variable a of callee f1 has value = %d and address %p\n\n",a,&a);
printf("f1 calls f2. So now f1 is caller and f2 is callee\n\n"); f2(a);
 }
void main()
{
   int z = 9;
printf("\nLocal variabe z of caller function main has value = %d and address %p\n",z,&z);
f1(z);
 }
```

OUTPUT :-



CONCLUSION:-

Call by value involves passing a copy of the parameter's value to a function, maintaining the original data outside the function. Call by reference passes the actual address of the parameter, enabling direct modification of the original data within and outside the function.

Name:-Pratham Asnani
BAtch:-S13
Roll no:-72

2.WAP in c++ for Encapsulation, Operator overloading, Single and multiple inheritance

Aim:-Write a program in C++ for encapsulation, operator overloading, single multiple inheritance

1.Encapsulation

Code:-

```
#include <iostream>

class rectangle {

private:
   int length;
   int width;
   int rectArea;

public:

   rectangle(int len, int wid) {
      length = len;
      width = wid;
      rectArea = 0;
   }


   void show() {
      std::cout << "The rectangle has length " << length << " units and width " << width << " units" << std::endl;
   }


   void calculateArea() {
      rectArea = length * width;
      std::cout << "Area of the rectangle: " << rectArea << " square units" << std::endl;
   }
};

int main() {

   rectangle r(5, 6);


   r.show();


   r.calculateArea();

   return 0;
}
```

Output :-

The rectangle has length 5 units and width 6 units
Area of the rectangle: 30 square units

Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.

2,Operator overloading

Code:-

```
#include <iostream>

using namespace std;

class n {

private:
   int value;

public:

   n(int v = 0)
   {
      value = v;
   }


   n operator-(n const& obj)
   {
      n result;
      result.value = value - obj.value;
      return result;
   }


   void print()
   {
      cout << "Value: " << value << '\n';
   }
};

int main()
{
```

```
    n num1(10), num2(5);


    n num3 = num1 - num2;

    num3.print();

    return 0;
}
```

Output:-

```
Value: 5

Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.
```

3.Single Inheritance

Code:-

```
#include <iostream>

class regpoly {
private:
    int nsides;

protected:
    int lside;

public:

    regpoly() : nsides(0), lside(0) {}


    regpoly(int n, int l) : nsides(n), lside(l) {}


    void show() {
        std::cout << "The regular polygon has " << nsides << " sides of length " << lside << " units." << std::endl;
    }


    virtual void showarea() {
        std::cout << "The regular polygon has " << nsides << " sides of length " << lside << " units." << std::endl;
    }
```

```
};

class square : public regpoly {
private:
   int area, peri;

public:

   square(int l) : regpoly(4, l), area(0), peri(0) {}


   void calarea() {
      area = lside * lside;
   }


   void calperi() {
      peri = 4 * lside;
   }


   void show()  {

std::cout << "The square has sides of length " << lside <<" units" << std::endl;

}

void showareaperi()

{

std::cout << "The area and perimeter of square are " << area <<" sq units and " <<peri <<" units respectively"
<< std::endl;

}

};

int main()

{

std::cout<< "Regular polygon r" <<std::endl;
```

```
regpoly r(4,6); r.show();

std::cout<< "Square" <<std::endl;

square s(3);

s.show();

s.calarea();

s.calperi();

s.showareaperi();

}
```

Output:-

```
Regular polygon r
The regular polygon has 4 sides of length 6 units.
Square
The square has sides of length 3 units
The area and perimeter of square are 9 sq units and 12 units respectively

Process returned 0 (0x0)   execution time : 0.029 s
Press any key to continue.
```

4.Multiple Inheritance

Code:-

```
#include <iostream>

class rectangle {
public:
   double length;
   double breadth;


   rectangle(double len, double bre) : length(len), breadth(bre) {}


   double area() {
      return length * breadth;
```

```
    }
};

class painting {
public:
    double ratePerSquareUnit;


    painting(double rate) : ratePerSquareUnit(rate) {}
};

class wall : public rectangle, public painting {
public:

    wall(double len, double bre, double rate)
        : rectangle(len, bre), painting(rate) {}


    double costofpainting() {
        return area() * ratePerSquareUnit;
    }
};

int main() {
    double length, breadth, rate;

    std::cout << "Enter the length and breadth of the wall: ";
    std::cin >> length >> breadth;

    std::cout << "Enter the rate per square unit of painting the wall: ";
    std::cin >> rate;


    wall w(length, breadth, rate);


    double cost = w.costofpainting();


    std::cout << "Cost of painting the wall will be: " << cost << std::endl;

    return 0;
}
```

Output:-
Enter the length and breadth of the wall: 4 6
Enter the rate per square unit of painting the wall: 22
Cost of painting the wall will be: 528

Process returned 0 (0x0)   execution time : 6.883 s
Press any key to continue.

Conclusion:-We learnt how to write and use C++ for encapsulation, operator overloading, single multiple inheritance

Name:Pratham Asnani
Roll No: 72
Batch: S13

3.1 Java code for Polymorphism, Exception handling

Aim:- Write a program in java code for Polymorphism, Exception handling

Single Exception handling
Code:-

```
public class Exceptional {
   public static void main(String[] args) {
      int a = 5, b = 0;

      try {
         System.out.println(a / b);
      } catch (Exception e) {
         System.out.println("Division by zero");
      }
   }
}
```

Output:-

```
C:\SE IT\pcpf>javac Exceptional.java

C:\SE IT\pcpf>java Exceptional
Division by zero
```

Multiple Exception Handling

Code:-

```
public class Exceptional {
   public static void main(String[] args) {
      int a = 5, b = 0;

      try {
         System.out.println(a / b);
      } catch (Exception e) {
         System.out.println("Division by zero");
      }
   }
}
```

Output:-

```
C:\SE IT\pcpf>javac Exceptional2.java

C:\SE IT\pcpf>java Exceptional2
Cannot divide by zero.
```

Polymorphism method overloading across classes

Code:-

```
class Pig
{
        public void animalSound()
        {
                System.out.println("The pig says: wee wee");
                }
                }
class Dog
{
        public void animalSound()
        {
                System.out.println("The dog says: bow wow");
                }
                }
class polymorphism1
{
        public static void main(String[] args)
        {
                Pig myPig = new Pig();
                Dog myDog = new Dog();
                myPig.animalSound();
```

```
        myDog.animalSound();
        }
        }
```

Output:-

```
C:\SE IT\pcpf>javac Pig.java

C:\SE IT\pcpf>java Pig
The pig says: wee wee
The dog says: bow wow
```

Polymorphism method overloading within class

Code:-

```
class poly
{
        public int add(int x, int y)
        { return( x + y);
        }
public double add(double x, double y)
{
        return( x + y);
        }
public static void main(String[] args)
{
        poly a = new poly() ;
        System.out.println("Integer Addition "+ a.add(4,3) );
        System.out.println("Floating point Addition " +a.add(4.0,3.0) );
        }
        }
```

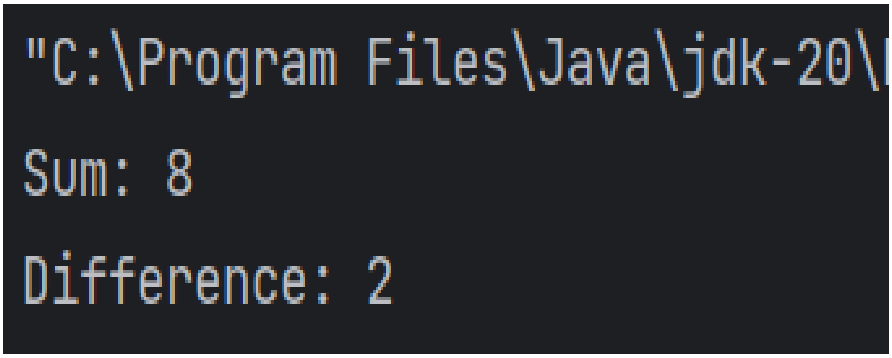
Output:-

```
C:\SE IT\pcpf>java poly
Integer Addition 7
Floating point Addition 7.0
```

1.Static binding

```
class call {
    int add(int a, int b) {
        return a + b;
    }
    int sub(int a, int b) {
        return a - b;
    }
}
public class Static {
    public static void main(String[] args) {
        call calculator = new call();
        int sum = calculator.add(5, 3);
        int difference = calculator.sub(5, 3);
        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
    }
}
```

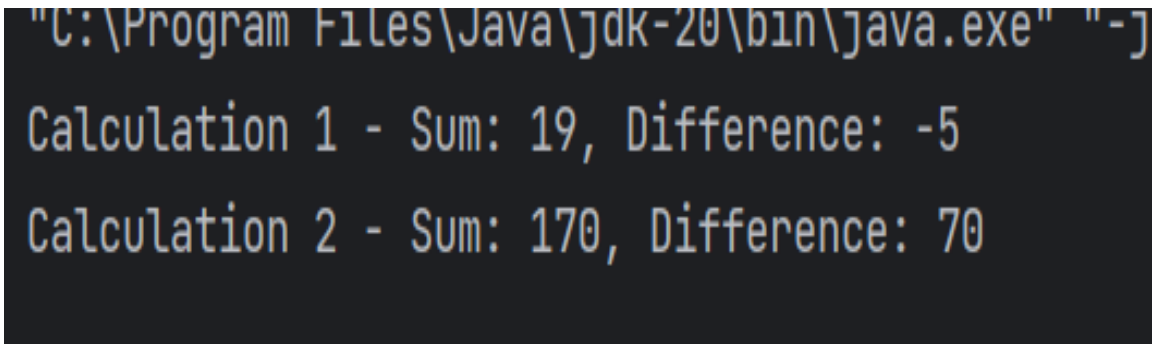
output:-



2.Initialization:-

```
class Calculator1 {
    int number1;
    int number2;

    Calculator1(int a, int b) {
        number1 = a;
        number2 = b;
    }

    int add() {
        return number1 + number2;
    }

    int subtract() {
        return number1 - number2;
    }
}

public class Initialization {
    public static void main(String[] args) {
        Calculator1 calc1 = new Calculator1(7, 12);
        Calculator1 calc2 = new Calculator1(120, 50);

        int sum1 = calc1.add();
        int difference1 = calc1.subtract();

        int sum2 = calc2.add();
        int difference2 = calc2.subtract();

        System.out.println("Calculation 1 - Sum: " + sum1 + ",
Difference: " + difference1);
        System.out.println("Calculation 2 - Sum: " + sum2 + ",
Difference: " + difference2);
    }
}
```

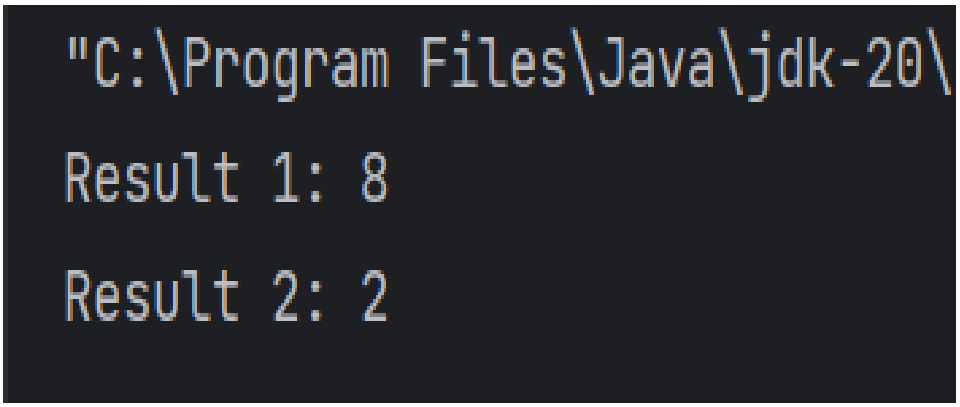
output:-



3.Finalization:-

```
class Calculator {
    int calculate(int a, int b) {
        return 0;
    }
}

class Addition extends Calculator {
    int calculate(int a, int b) {
        return a + b;
    }
}

class Subtraction extends Calculator {
    int calculate(int a, int b) {
        return a - b;
    }
}

public class Dynamic {
    public static void main(String[] args) {
        Calculator operation;

        operation = new Addition();
        int result1 = operation.calculate(5, 3);

        operation = new Subtraction();
        int result2 = operation.calculate(5, 3);

        System.out.println("Result 1: " + result1);
        System.out.println("Result 2: " + result2);
    }
}
```

output:-



4.Finalization:-

```
class CalculationWithFinalization {
    private int result;
    CalculationWithFinalization(int a, int b) {
        result = a + b;
    }
    void displayResult() {
        System.out.println("Result: " + result);
    }
    @Override
    protected void finalize() throws Throwable {
        System.out.println("Finalizing the CalculationWithFinalization object.");
        super.finalize();
    }
}
public class bac {
    public static void main(String[] args) {
        CalculationWithFinalization calc = new CalculationWithFinalization(5, 3);
        calc.displayResult();
        calc = null;
        System.gc();
    }
}
```

Ouput:-

```
G:\java>java bac
Result: 8
Finalizing the CalculationWithFinalization object.
```

Name-Pratham Asnani

Roll no.-72

Batch-s13

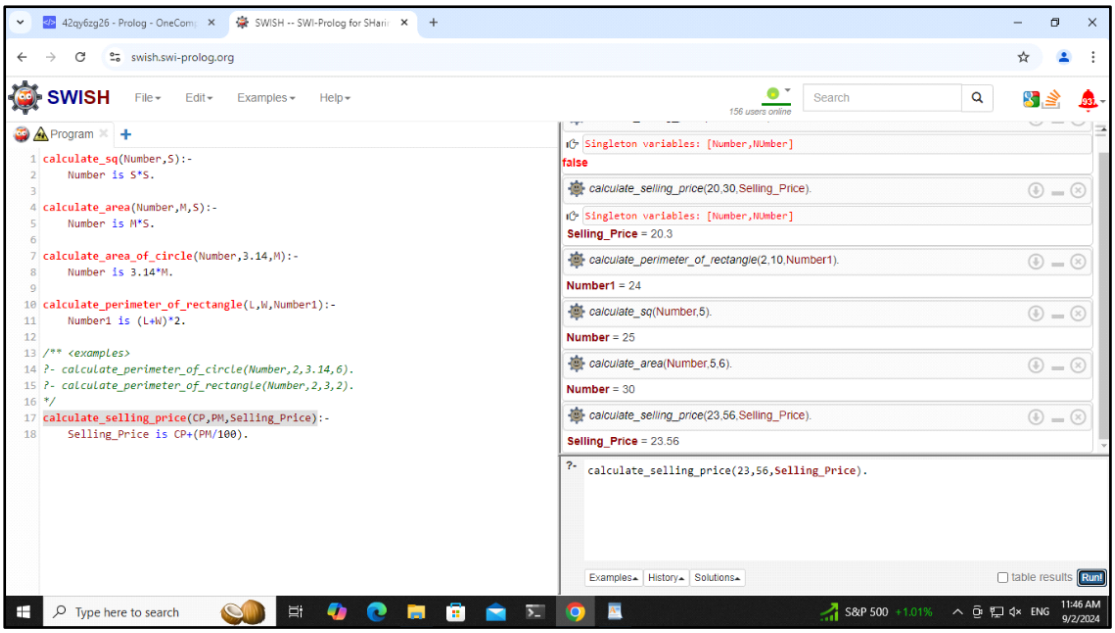
4.Prolog

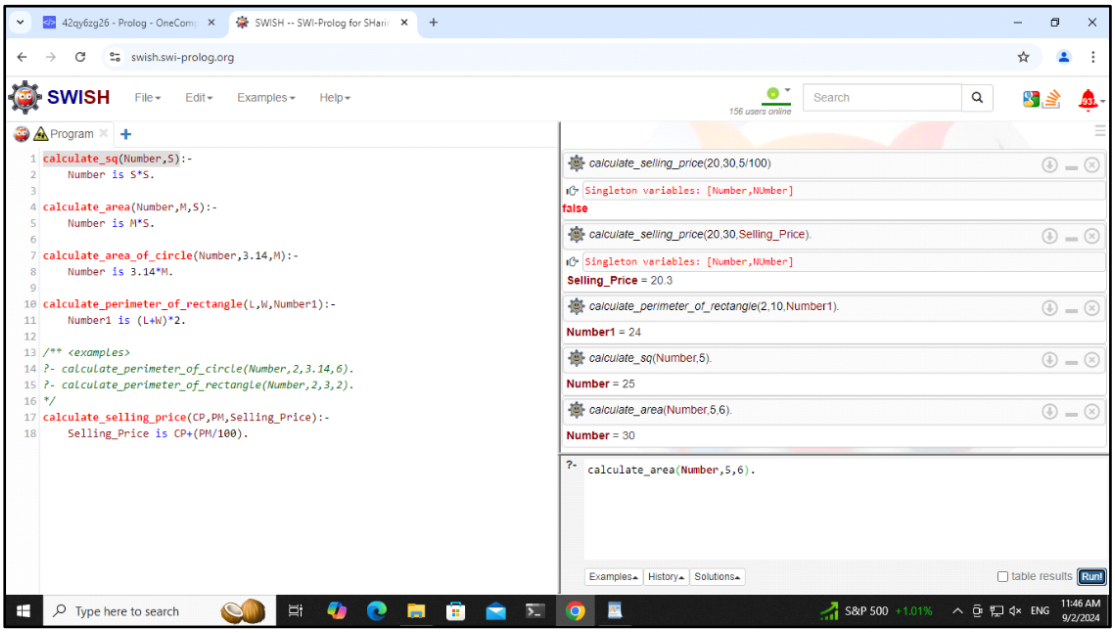
Aim:-Perform arthmetic processing using prolog

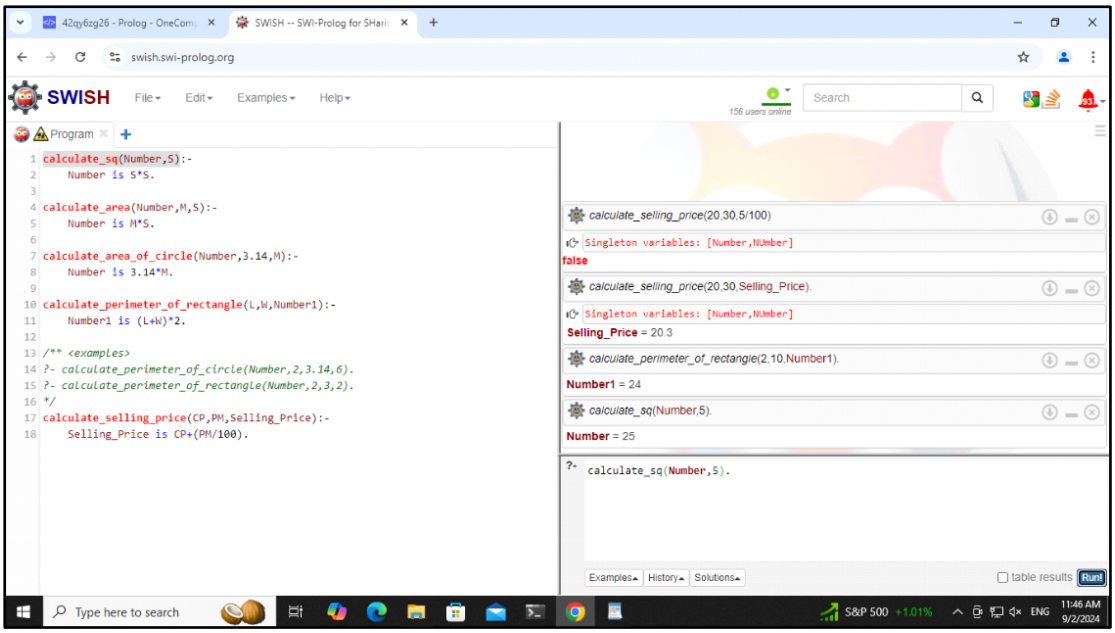
Theory:-Prolog is a logic programming language used primarily for solving problems related to artificial intelligence and computational linguistics. It operates based on formal logic, allowing users to define rules and facts that the system uses to infer new information and solve queries. Programs in Prolog consist of a series of logical statements and can be used to model complex relationships and reasoning processes.

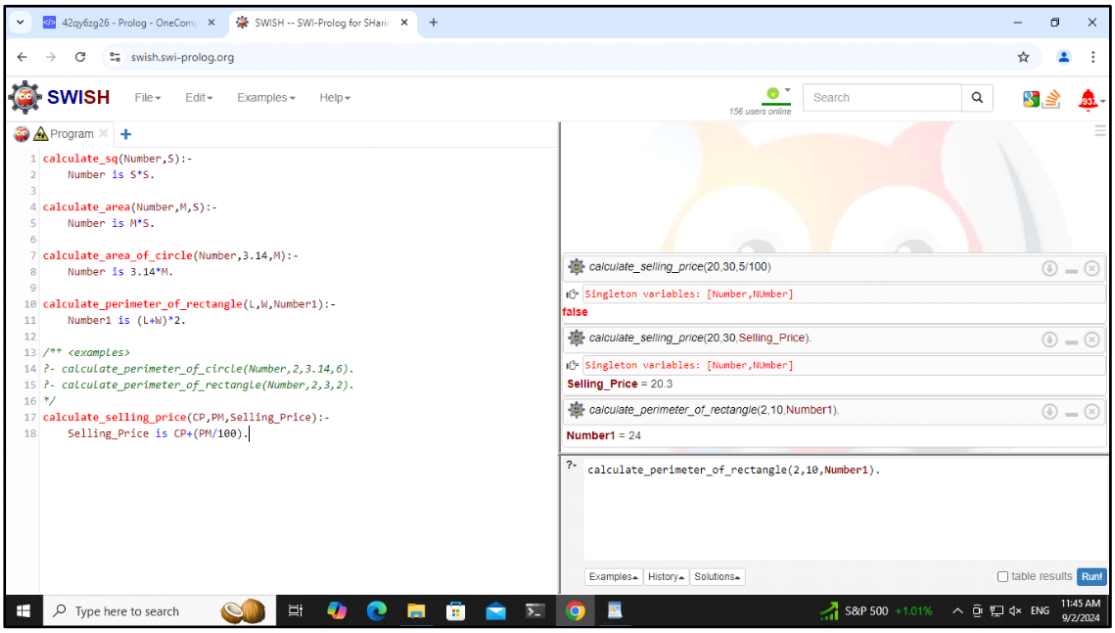
Code:-

```
calculate_sq(Number,S):-
    Number is S*S.


calculate_area(Number,M,S):-
    Number is M*S.


calculate_area_of_circle(Number,3.14,M):-
    Number is 3.14*M.


calculate_perimeter_of_rectangle(L,W,Number1):-
    Number1 is (L+W)*2.


calculate_selling_price(CP,PM,Selling_Price):-
    Selling_Price is CP+(PM/100).
```









Conclusion:-

In Prolog, arithmetic operators help perform and integrate mathematical calculations within logical rules.

LO mapping:- LO 2

Name: Pratham Asnani
Roll no: 72
Batch:-s13

5.1. Basic Prolog

3A]

Aim: To use the sentences given below answer the queries: What grows? What's edible? Wheat is a monocot. Monocots are plants. Plants grow. Monocots are edible.

Theory:
Prolog is a high-level programming language that is primarily associated with artificial intelligence (AI) and computational linguistics. Unlike procedural programming languages like C or Java, Prolog is a declarative language, which means that you specify what should be done, rather than detailing how it should be done. Prolog is based on first-order logic (predicate logic), where the programmer defines relationships, facts, and rules, and the Prolog engine derives logical conclusions from them. The logical and declarative nature of Prolog can be difficult to grasp for those used to procedural Programming.

Program:

```
% Facts based on the statements
monocot(wheat). % Wheat is a monocot
plant(X) :- monocot(X). % Monocots are plants
grow(X) :- plant(X). % Plants grow
edible(X) :- monocot(X). % Monocots are edible
% Queries:
% To check what grows: ?- grow(X).
% To check what's edible: ?- edible(X).
```

Output:

```
?- grow(X).
X = wheat.
?- edible(X).
X = wheat.
```

Conclusion:
Thus we implement the basic prolog program to display output based on the sentences provided.

3B]

Aim: To write a program in prolog to input length and calculate area and perimeter of a square.

Theory:
In Prolog, a program is essentially a knowledge base that consists of facts and rules, and queries are used to reason about the knowledge base. The logic programming paradigm focuses on describing relations and deriving conclusions based on them.
The rule square_properties(Side, Area, Perimeter) takes the side of the square as input and calculates:
Area as side*side
Perimeter as 4*side
The is operator is used to evaluate the arithmetic expressions in Prolog.

Program:
```
% Rule to calculate area and perimeter of a square
square_properties(Side, Area, Perimeter) :-
 Area is Side * Side,
 Perimeter is 4 * Side.
% To run the program, use the following query:
% ?- square_properties(5, Area, Perimeter).
```

Output:
Area = 25,
Perimeter = 20.

Conclusion:
Thus we write basic program in prolog to display the output calculating the area and perimeter of area a square taking side as the input.

Name: Pratham Asnani
Roll No: 72
Batch:-s13

5.2. Advance Prolog

Aim: To write a program in prolog to calculate length of a list.

Theory:
The base case is list_length([], 0). The recursive case keeps breaking

the list down by removing the first element (denoted by _) and calling the function on the rest of the list (Tail). Once the list is reduced to an empty list ([]), the base case returns a length of 0. As the recursion unwinds, it adds 1 for each element in the original list, eventually calculating the total length.

Program:

```
% Base case: The length of an empty list is 0.
list_length([], 0).
% Recursive case: The length of a non-empty list is 1 plus the length
of the tail.
list_length([_ | Tail], Length) :-
 list_length(Tail, TailLength),
 Length is TailLength + 1.
```

Output:
For the input,

```
?- list_length([a, b, c, d], Length).
Length = 4.
```

Conclusion:
Thus we implement a code in prolog to calculate the length of the List.

3B]

Aim: In given 2 lists to subtract corresponding elements of second list from first list.

Theory:
[H1 | T1] and [H2 | T2] represent the two input lists. H1 and H2 are the heads (first elements) of the two lists, and T1 and T2 are the tails (rest of the lists). The result of subtracting H2 from H1 is stored in Result, which is the head of the resulting list. The program then recursively calls list_subtract(T1, T2, ResultTail) to process the remaining elements in the lists. When both input lists are empty, the result is also an empty list. This serves as the base case for recursion. Both input lists should be of the same length for the program to work correctly. If they are of different lengths, the program will fail. You can add a check for equal list lengths if needed.

Program:

```
% Base case: If both lists are empty, the result is an empty list.
list_subtract([], [], []).
```

```
% Recursive case: Subtract the heads of the two lists and recurse on
the tails.
list_subtract([H1 | T1], [H2 | T2], [Result | ResultTail]) :-
Result is H1 - H2,
list_subtract(T1, T2, ResultTail).
```

Output:
Result = [9, 18, 27].

Conclusion:
Thus we demonstrate the code to subtract the corresponding elements of second list from the first list.

Name:-Pratham Asnani
Rollno:-72
Batch:-s13

6.1 Perform arithmetic and logical operations using haskell

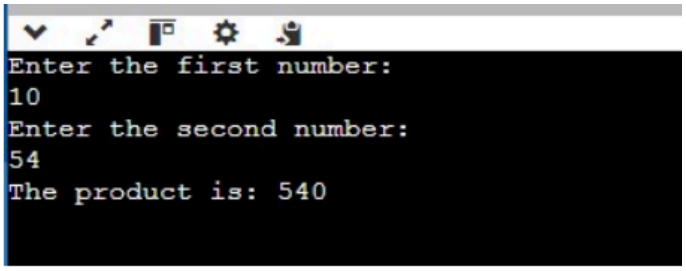
AIM - Perform Arithmetic & Logical operations using haskell.

Theory - Haskell is a purely functional programming language that emphasizes immutability, first-class functions, and type safety. It was designed to be a clean, mathematically sound language for research, education, and practical software development.

CODE -
1. Multiplication of Two Number:
CODE -

```
multiply :: Int -> Int -> Int
multiply x y = x * y
main = do
 putStrLn "Enter the first number:"
 input1 <- getLine
 putStrLn "Enter the second number:"
 input2 <- getLine
 let num1 = read input1 :: Int
 let num2 = read input2 :: Int
 let result = multiply num1 num2
 putStrLn ("The product is: " ++ show result)
```

OUTPUT:

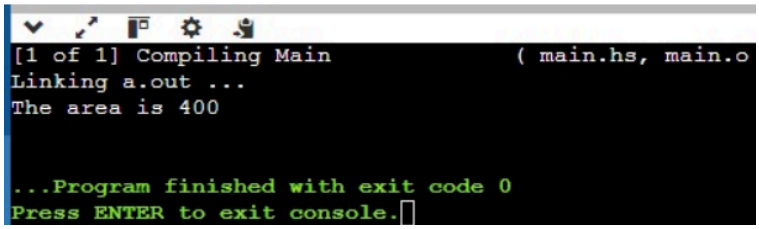


2.Area Of Rectangle:

CODE -

```
main = do
let x= 20
putStr"The area is "
print(x*x)
```

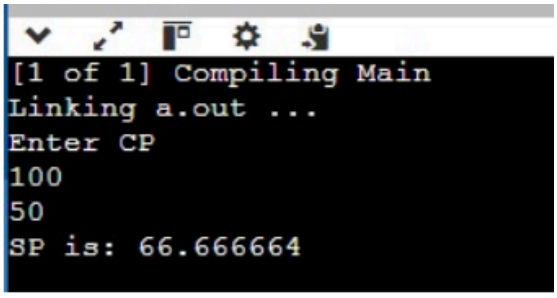
OUTPUT:



3. Selling Price and cost price:

```
main = do
putStrLn("Enter CP")
ain <- getLine
let x= (read ain ::Float)
bin<-getLine
let y =(read bin :: Float)
let c = x/(1+(y/100))
putStr("SP is: ")
print(c)
```

OUTPUT:



Name:-Pratham Asnani
Rollno:-72
Batch:-s13

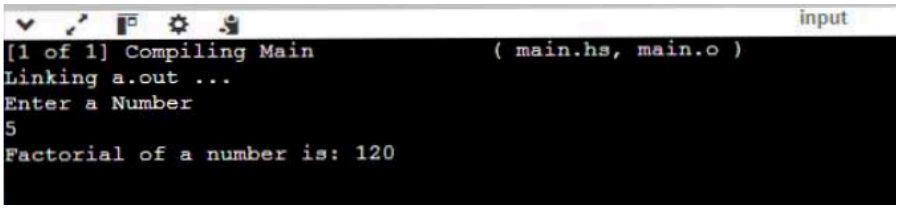
6.1 Perform recursive function in haskell

AIM:-Perform recursive function in haskell

CODE -
1.
 Factorial OF Number
CODE:

```
fact::Int->Int
fact n|n==0=1
fact n|n/=0=n*fact(n-1)
main = do
putStrLn"Enter a Number"
number<-getLine
let n = (read number::Int)
putStr"Factorial of a number is: "
print(fact n)
```

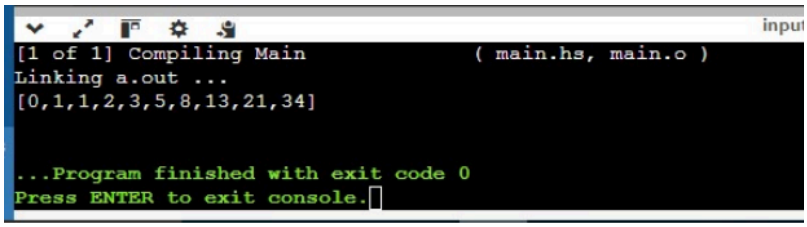
OUTPUT:



2. Fibonacci Series
CODEfibonacci :: [Integer]
fibonacci = 0 : 1 : zipWith (+) fibonacci (tail fibonacci)

```
fiboN :: Int -> Integer
fiboN n = fibonacci !! n
main :: IO ()
main = print $ take 10 fibonacci
```

OUTPUT

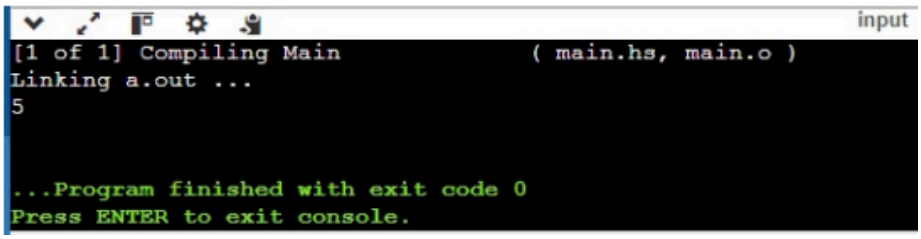


3. Length of the Striing

CODE:

```
listLength :: [a] -> Int
listLength [] = 0
listLength (_:xs) = 1 + listLength xs
main = do
 print (listLength [1, 2, 3, 4, 5])
```

OUTPUT:



Name:Pratham Asnani
Roll no: 72
Batch:-S13

7.2. Advance Haskell

Aim: To WAP in Haskell to implement (i) triple of square of a number and (ii) double of tripple using higher order functions.

Theory:
In the main function we first compute the triple of the square using triple square. Then, we compute the double of the triple using double (triple square). square is a simple function that squares a

number, triple is a higher-order function that triples the result of another function (f) applied to a number (x), double is another higher-order function that doubles the result of a given function (f) applied to a number (x).

Program:

```
-- Function to square a number
square :: Num a => a -> a
square x = x * x
-- Higher-order function to triple a result
triple :: Num a => (a -> a) -> a -> a
triple f x = 3 * f x
-- Higher-order function to double the result of another function
double :: Num a => (a -> a) -> a -> a
double f x = 2 * f x
-- Main function
main :: IO ()
main = do
 putStrLn "Enter a number:"
 input <- getLine
 let num = read input :: Int
 -- (i) Triple of the square of a number
 let tripleSquare = triple square num
 putStrLn ("Triple of square: " ++ show tripleSquare)

 -- (ii) Double of the triple of square of a number
 let doubleTripleSquare = double (triple square) num
 putStrLn ("Double of triple of square: " ++ show
doubleTripleSquare)
```

Output:

```
Enter a number:
4
Triple of square: 48
Double of triple of square: 96
```

B]

Aim: To subtract corresponding elements second list from the first List.

Theory:

In the main function, two example lists are provided (list1 and list2), and their corresponding elements are subtracted using subtractLists. subtractLists uses zipWith (-) to subtract corresponding elements

from two lists. The zipWith function applies the subtraction (-) to pairs of elements from the two lists.

Program:

```
-- Function to subtract corresponding elements of two lists
subtractLists :: [Int] -> [Int] -> [Int]
subtractLists = zipWith (-)
-- Main function
main :: IO ()
main = do
 let list1 = [10, 20, 30]
 let list2 = [1, 2, 3]
 let result = subtractLists list1 list2
 print result
```

Output:
[9,18,27]
Conclusion:
Thus we implement the Haskell programs for triple of square, double of triple using the higher order functions and also the demonstration to subtract corresponding elements second list from the first list.

Name-Pratham
Roll no.:-72
Batch:-s13

8.Write a Java/c++ program to display number and alphabets concurrently

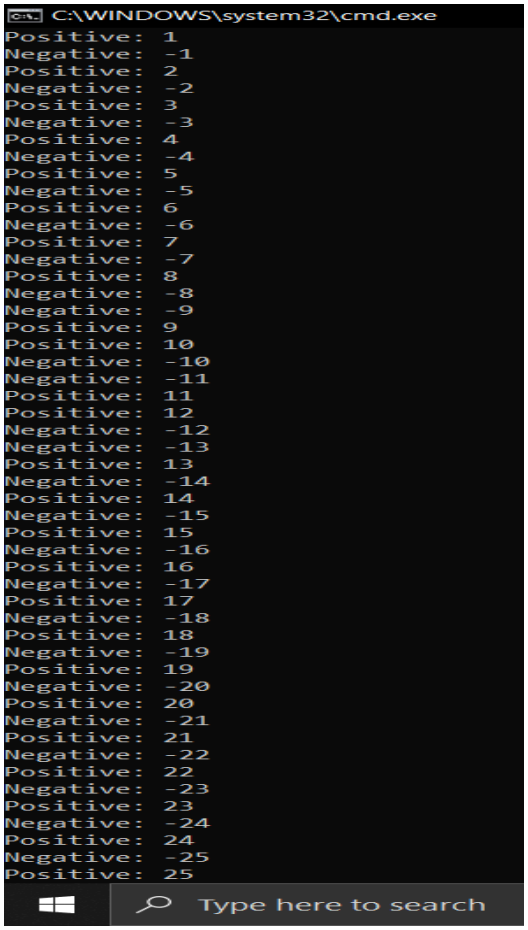
Aim:-Write a Java/c++ program to display number and alphabets concurrently

Display Numbers
Code:-

```
public class Multithreading12 {

    public static void main(String[] args) {
        Thread positiveThread = new Thread(new PositiveNumbersRunnable());
        positiveThread.start();
```

```
        Thread negativeThread = new Thread(new NegativeNumbersRunnable());
        negativeThread.start();
    }
}

class PositiveNumbersRunnable implements Runnable {

    public void run() {
        for (int i = 1; i <= 25; i++) {
            System.out.println("Positive: " + i);
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class NegativeNumbersRunnable implements Runnable {

    public void run() {
        for (int i = -1; i >= -25; i--) {
            System.out.println("Negative: " + i);
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Output:-



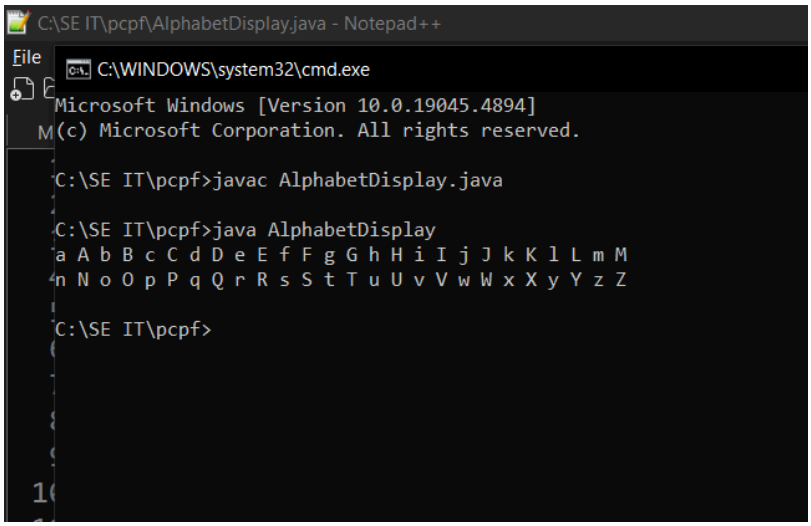
Display Alphabet

Code:-

```
public class AlphabetDisplay {
    public static void main(String[] args) {
        char lowerCase = 'a';
        char upperCase = 'A';
        for (int i = 0; i < 26; i++) {
            System.out.print(lowerCase + " ");
            System.out.print(upperCase + " ");
            lowerCase++;
            upperCase++;
            if (i % 13 == 12) {
                System.out.println();
            }
```

```
    }
  }
}
```

Output:-



Conclusion:-

In the Java program that displays numbers and alphabets concurrently, multithreading is used to run two tasks in parallel, ensuring efficient execution of both sequences. This approach highlights the power of concurrency in handling multiple operations simultaneously.

Name:Pratham Asnani
Roll No: 72
Batch:-s13

ASSIGNMENT 9

AIM: To create a java script to accept a password of length between 6 to 8 that has 1 capital alphabet and 1 special character.

THEORY:
Password validation is an essential part of web security. By enforcing specific rules such as
length, inclusion of uppercase letters, and special characters, we ensure the password is
strong enough to resist simple guessing or brute-force attacks. In JavaScript, regular
expressions (regex) are commonly used to match patterns in strings, making them suitable for validation password constraints.

CODE:

```
/<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initialscale=1.0">
 <title>Password Validator</title>
</head>
<body>
 <h2>Password Validator</h2>
 <form id="passwordForm">
 <label for="password">Enter your password:</label>
 <input type="password" id="password" name="password"
required>
 <button type="submit">Submit</button>
 </form>
 <p id="message"></p>
 <script>

document.getElementById("passwordForm").addEventListener("sub
mit", function(event) {
 event.preventDefault(); // Prevent form submission

 let password = document.getElementById("password").value;
 let message = document.getElementById("message");
 // Regular expression to check password rules:
 // - Length between 6 to 8 characters
 // - At least 1 uppercase letter
 // - At least 1 special character
 let passwordPattern = /^(?=.*[A-Z])(?=.*[!@#$%^&*])[A-Zaz\d!@#$%^&*]{6,8}$/;
 if (passwordPattern.test(password)) {
 message.textContent = "Password is valid!";
 message.style.color = "green";
 } else {
 message.textContent = "Password must be 6-8 characters
long, with at least one uppercase letter and one special character.";
 message.style.color = "red";
 }
 });
 </script>
</body>
</html>
```

OUTPUT:

CONCLUSION:
Thus we create a password validator including the constrains such as password of length 6 to 8, having 1 capital alphabet and 1 special character.

Pratham Asnani
SI3 72

(A) Sunday

THADOMAL SHAHANI TSEC ENGINEERING COLLEGE

1 Write program in C, Java, prolog and Haskell for implementing merge sort over 2 list

i Java

```
Class merge sort {
  public static void main (string [] args)
  { int [] list 1 = {-4, 5, 7, 10, 25};
    int [] list 2 = {-1, 2, 3, 4, 9, 7, 99, 103};
    merge (list 1, list 2); }
    public static void ~~main~~ merge (int [] list 1, list 2)
  { int len 1 = list 1 length;
    int len 2 = list 2 length;
    int len 3 = list 3 length;
    int index 1 = 0;
    int index 2 = 0;
    For (int i = 0; i < len 3; i++);
    if (index 1 < len 1 && index 2 < len 2)
  { if (list 1 [index 1] < list 2 [index 2])
      list 3 [i] = list 1 [index 1];
      index 1++; }
  else { (index 1 < len 1)
      { list 3 [i] = list 1 [index 1++];
      }
      else { list 3 [i] = list 2 [index 2++];
      }
    }
  System out println (Arrays to string (list 3));
  }
}
```

ii Haskell

```
merge [Int] -> int -> int
merge [] [] z = reverse z
merge [] (Y:Y) z = merge [] ys [y:z]
merge (x:xs) [] z = merge xs [] (x:z)
merge (x:xs) (y:ys) = z (x<=y = merge xs (y) (y) (x:z)
merge (x:x1 (y:ys) z|x>y = merge (x:xs)
main =do
let l1 = [-1, 23, 44, 57]
let l2 = [-4, 3, 88, 22]
print ( merge l1, l2 [])
```

iii Prolog

```
merge ([], L)
merge ( L, [], L)
merge [[H1|T1], [H2|T2], [H1, R]);
H1=< H2
merge (T1, [H2|T2], R)
merge ( [H1|T1], [H2|T2], [H2|R]);
merge ([H|T1], T2, R)
```

Output

i Java

{-4, -1, 2, 3, 4, 5, 7, 9, 10, 25, 77, 99, 103}

ii Prolog

```
?- merge ([-1, 3, 5] [-2, 6, 9], L)
L = [-2, -1, 3, 5, 6, 19]
```

iii Haskell

```
[1 of 1] Compiling main (main L, main o)
Linking a- out
[-4, -1, 3, 23, 44, 57, 88, 222]
```

TSEC
ENGINEERING COLLEGE

Criteria	Procedural Programming	Object Oriented Programming	Function Programming
Ease of writting Code	Easier for simple tasks, complex for large programs	Modular and Scalable but more complex due to class setup	Requires a different mindset complex for beginners
Lines of Code	More lines of repetitive code	Fewer lines due to reuses of methods and inheritance	Fewer line due to concise higher order function
Data Type and Data Structure support	Support basic data types and structure	Strong support for customs data types	Strong support for immutable data type and recursion
Ease of debugging	Easier for small programs harder for large are	Debugging can be complex due to class interactions	Debugging can be challenging due to recursion and immutability
Use of Compiler	Typically uses both (C compiler, python use interpreter)	Generally uses compiler (Java C++)	Often interpreted (Java Script, Python) but also compiler in same language.
Time required for Execution	Generally faster for small tasks	Slightly slower due to object management overhead	Can be slower due to recursion and immutability overhead