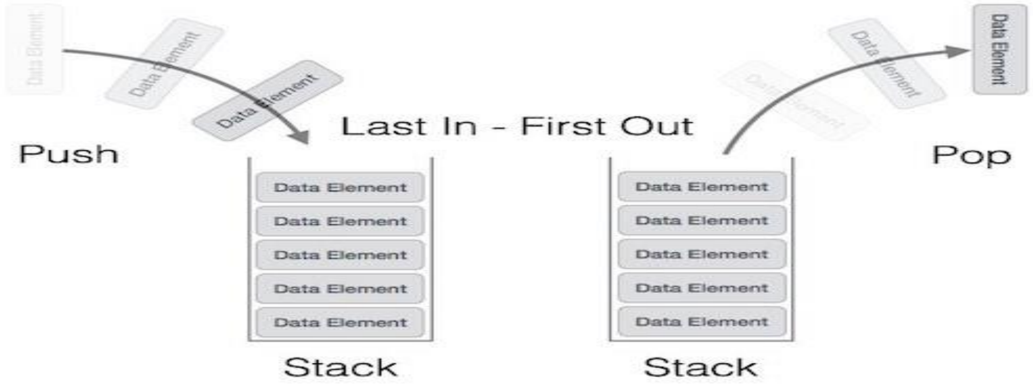
NAME: ADWAIT S PURAO

UID: 2021300101

EXP NO. :1

AIM: Infix to Postfix conversion using Stack

THEORY: A Stack is a linear data structure that follows the **LIFO (Last-In-First-Out)** principle. Stack has one end, whereas the Queue has two ends (**front and rear**). It contains only one pointer **top pointer** pointing to the topmost element of the stack. Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack. In other words, a ***stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.***



Examples of stack from real world:

- Plates stacked over one another in a canteen. Where the plate at the top is removed first.
- A stack of plates in a cupboard
- Undo and redo functions in word, excel etc.

Basic Operations

Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −

- **push()** − Pushing (storing) an element on the stack.
- **pop()** − Removing (accessing) an element from the stack.

When data is PUSHed onto stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks −

- **peek()** − get the top data element of the stack, without removing it.
- **isFull()** − check if stack is full.
- **isEmpty()** − check if stack is empty.

Applications of stack:

1. Polish Notation
2. Reversing action
3. Recursion
4. Backtracking

Polish and Reverse Polish Notation:

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

Conversion of Infix to Polish Notation:

1. Read expression from Left-to-Right and:
2. If an operand is read, copy it to the ouput.
3. If the operator is '(', then push it into the stack.
4. If the operator is ')', then print the top element of the stack and pop it and repeat it until '(' is found. When that occurs, both parantheses are discarded.
5. If an operator is read and has a higher precedence than the topmost element of the stack, the operator is pushed into the stack.
6. If an operator is read and has a lower precedence or equal precedence than the topmost element of the stack, the topmost element of the stack is printed and the popped out and then the read operator is pushed into the stack.
7. After reading the input, Print all the elements of stack by popping it.

ALGORITHM:

Class Stack: 1)Stack constructor
Set user defined size
Set top to -1
Initialize char array

Boolean isEmpty() method

Return true if top equals to -1
Boolean isFull() method
Return true if top equals to size – 1

Void Pop() method

If stack is not empty decrement the top by one

Void Push(char element) method

If stack is not full increment the top by one and insert the given element at top

Char getTop() method

If stack is not empty return the topmost element

Class InfixToPostfix

1) Intialize a Stringbuilder postfix

2) Take a string input from the user

3)Pass the length the length of the string to stackimp object

4)Iterate a for loop from i=0 to i< infix.length

5) if(s.isEmpty() && i!=infix.length()-1 && !Character.isAlphabetic(ch))

then push ch into the stack

6) else if(ch=='(' && !s.isEmpty())

Then push ch in the stack

7) else if(ch=='^' && s.getTop()!='^' && !s.isEmpty())

Then append the topmost element of stack to postfix and then pop it

While the top is equal to ch go on popping and at the end push ch into the stack

8) Do similar operations for *,/ and +,-

9) If ch is a alphabet append it to the string

10) else if(ch==')' && !s.isEmpty())

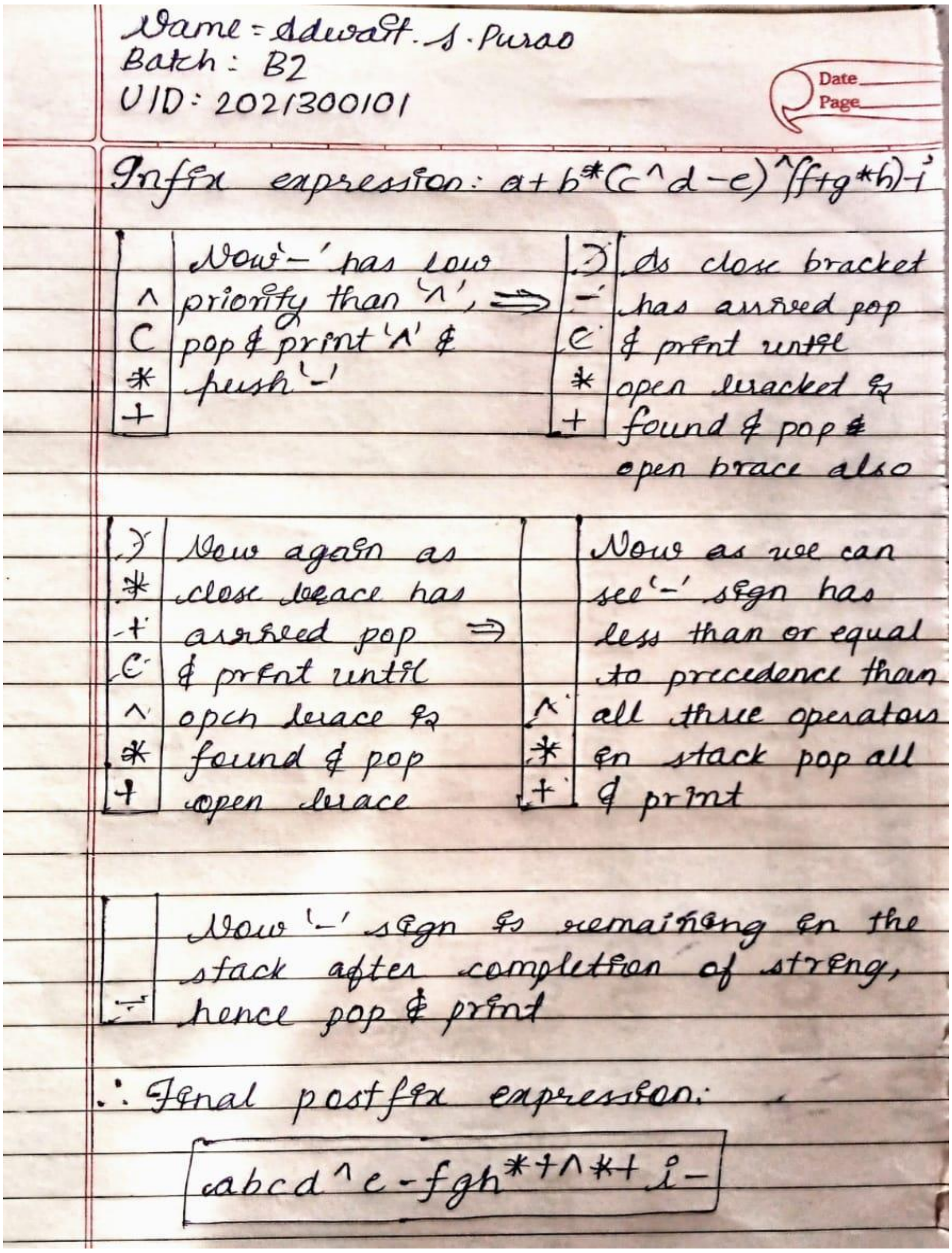
Then pop the stack while ch is not equal to (and pop one more time to remove (

11)After iterating through the entire for loop, check whether the stack is empty or not outside the loop

12)If stack is empty Print the postfix string

13)Else append the remaining characters of the stack to the string and pop the stack and print the string

PROBLEM SOLVING ON CONCEPT:



Date
Page
^
C
*
+
)
-
C
*
+
)
*
+
C
^
*
+
^
*
+
-

Name = Adwait. S. Purao
Batch : B2
UID : 2021300101

Infix expression: a+b*(c^d-e)^(f+g*h)-i

Now '-' has low priority than '^', pop & print '^' & push '-'

As close bracket has arrived pop & print until open bracket is found & pop open brace also

Now again as close brace has arrived pop & print until open brace is found & pop open brace

Now as we can see '-' sign has less than or equal to precedence than all three operators in stack pop all & print

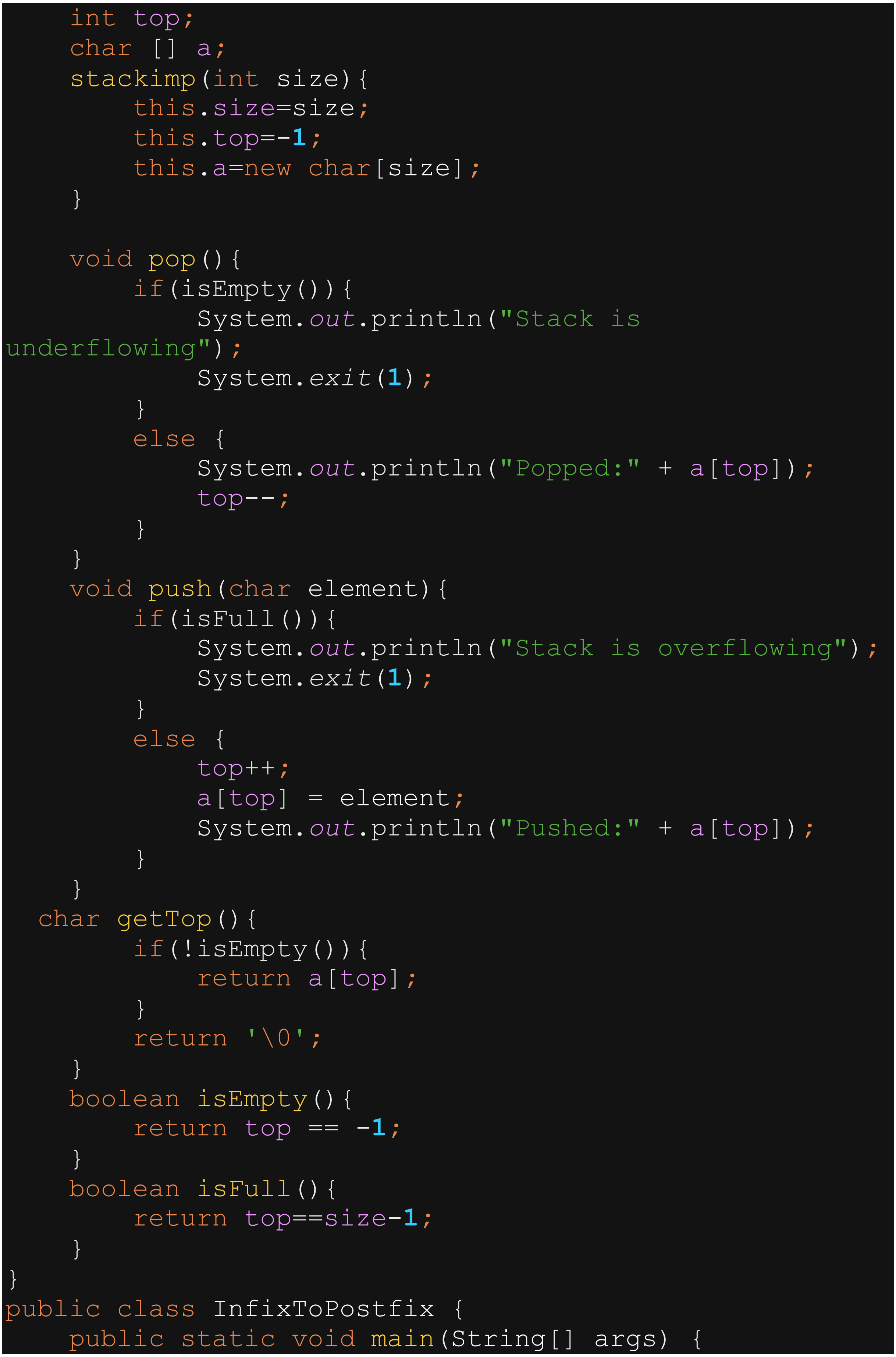
Now '-' sign is remaining in the stack after completion of string, hence pop & print

∴ Final postfix expression:

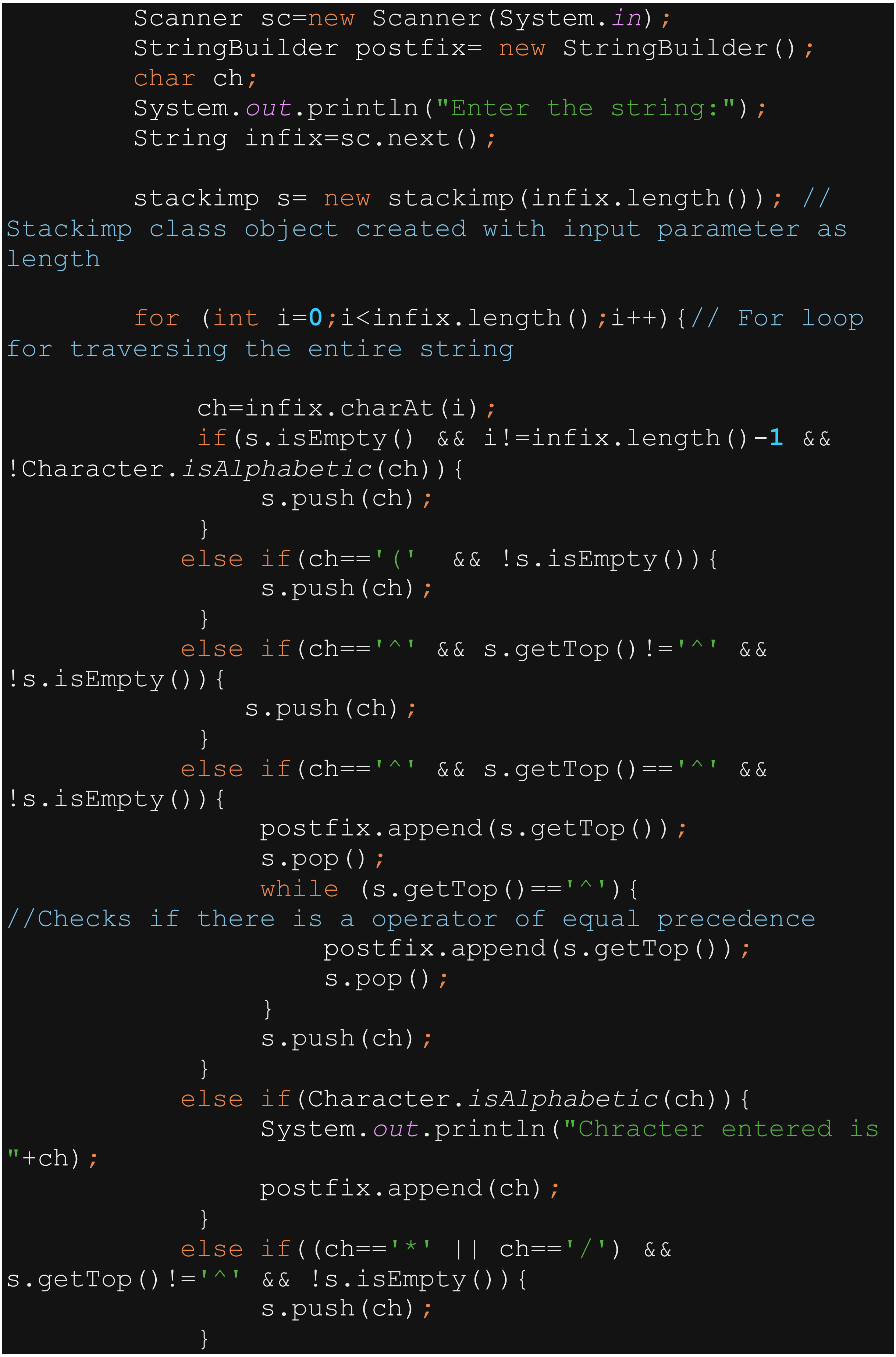
abcd^e-fgh*+^*+i-

CODE:

```
import java.util.Scanner;
class stackimp{
    int size;
```



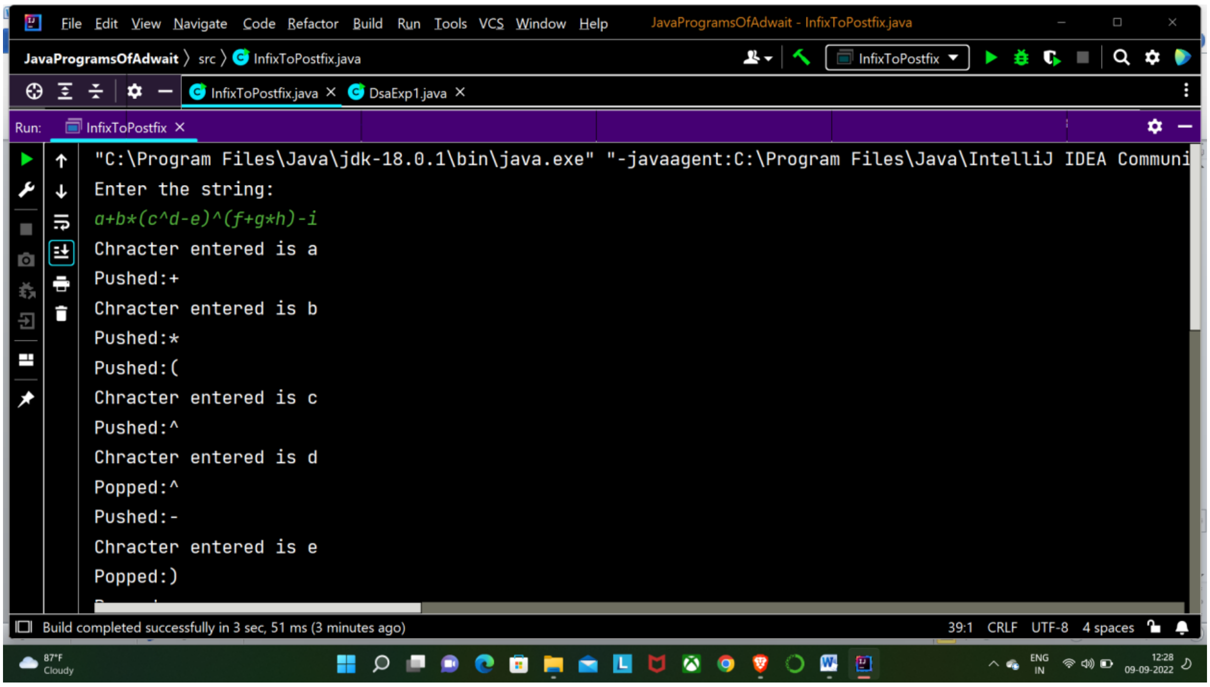
```
    int top;
    char [] a;
    stackimp(int size){
        this.size=size;
        this.top=-1;
        this.a=new char[size];
    }

    void pop(){
        if(isEmpty()){
            System.out.println("Stack is
underflowing");
            System.exit(1);
        }
        else {
            System.out.println("Popped:" + a[top]);
            top--;
        }
    }
    void push(char element){
        if(isFull()){
            System.out.println("Stack is overflowing");
            System.exit(1);
        }
        else {
            top++;
            a[top] = element;
            System.out.println("Pushed:" + a[top]);
        }
    }
  char getTop(){
        if(!isEmpty()){
            return a[top];
        }
        return '\0';
    }
    boolean isEmpty(){
        return top == -1;
    }
    boolean isFull(){
        return top==size-1;
    }
}
public class InfixToPostfix {
    public static void main(String[] args) {
```



```
        Scanner sc=new Scanner(System.in);
        StringBuilder postfix= new StringBuilder();
        char ch;
        System.out.println("Enter the string:");
        String infix=sc.next();

        stackimp s= new stackimp(infix.length()); //
Stackimp class object created with input parameter as
length

        for (int i=0;i<infix.length();i++){// For loop
for traversing the entire string

            ch=infix.charAt(i);
            if(s.isEmpty() && i!=infix.length()-1 &&
!Character.isAlphabetic(ch)){
                s.push(ch);
            }
           else if(ch=='('  && !s.isEmpty()){
                s.push(ch);
            }
           else if(ch=='^' && s.getTop()!='^' &&
!s.isEmpty()){
               s.push(ch);
            }
           else if(ch=='^' && s.getTop()=='^' &&
!s.isEmpty()){
                postfix.append(s.getTop());
                s.pop();
                while (s.getTop()=='^'){
//Checks if there is a operator of equal precedence
                    postfix.append(s.getTop());
                    s.pop();
                }
                s.push(ch);
            }
           else if(Character.isAlphabetic(ch)){
                System.out.println("Chracter entered is
"+ch);
                postfix.append(ch);
            }
           else if((ch=='*' || ch=='/') &&
s.getTop()!='^' && !s.isEmpty()){
                s.push(ch);
            }
           else if((ch=='*' || ch=='/') &&
s.getTop()=='^' && !s.isEmpty()) {
                postfix.append(s.getTop());
                s.pop();
                while((s.getTop()=='*' ||
s.getTop()=='/' || s.getTop()=='^')&& !s.isEmpty()){
//Checks if there is a operator of Higher of equal
precedence
                    postfix.append(s.getTop());
                    s.pop();
                }
                s.push(ch);
            }
            else if((ch=='-'|| ch=='+') &&
(s.getTop()=='^' ||s.getTop()=='*'||
s.getTop()=='/'||s.getTop()=='+' || s.getTop()=='-' )&&
!s.isEmpty()) {
                postfix.append(s.getTop());
                s.pop();
                while((s.getTop()=='+' ||
s.getTop()=='-'||s.getTop()=='^' ||s.getTop()=='*'||
s.getTop()=='/') && !s.isEmpty()){
                    postfix.append(s.getTop());
//Checks if there is a operator of Higher of equal
precedence
                    s.pop();
                }
                s.push(ch);
            }
           else if((ch=='-'|| ch=='+') &&
(s.getTop()!='^' ||s.getTop()!='*'|| s.getTop()!='/'
)&& !s.isEmpty()){
                s.push(ch);
            }
           else if(ch==')' && !s.isEmpty()){
                System.out.println("Popped:)");
                while (s.getTop()!='(') {
                    postfix.append(s.getTop());
                    s.pop();
                }
                s.pop();
            }
        }
        if(!s.isEmpty()){       // If at the end the
stack is not empty it empty's the stack and prints the
```

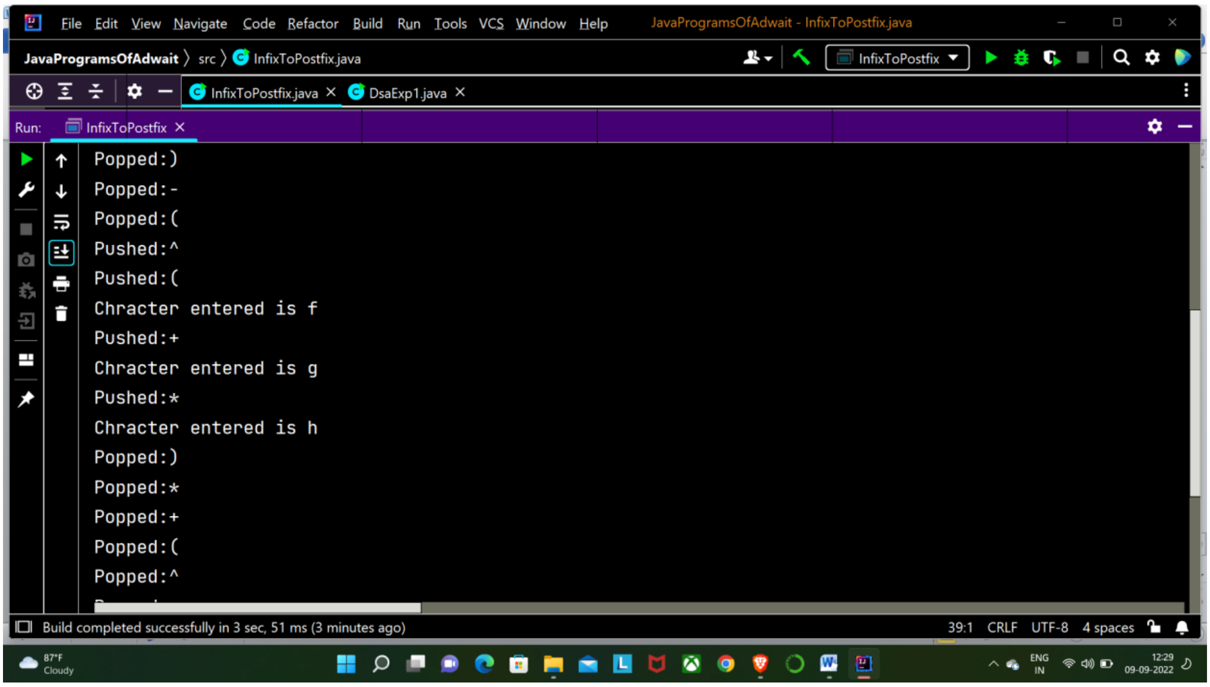
```
operators
                while (!s.isEmpty()) {
                    postfix.append(s.getTop());
                    s.pop();
                }
            }
            System.out.println("Final answer: ");
            System.out.println(postfix);
        }
}
```


OUTPUT SCREENSHOT:



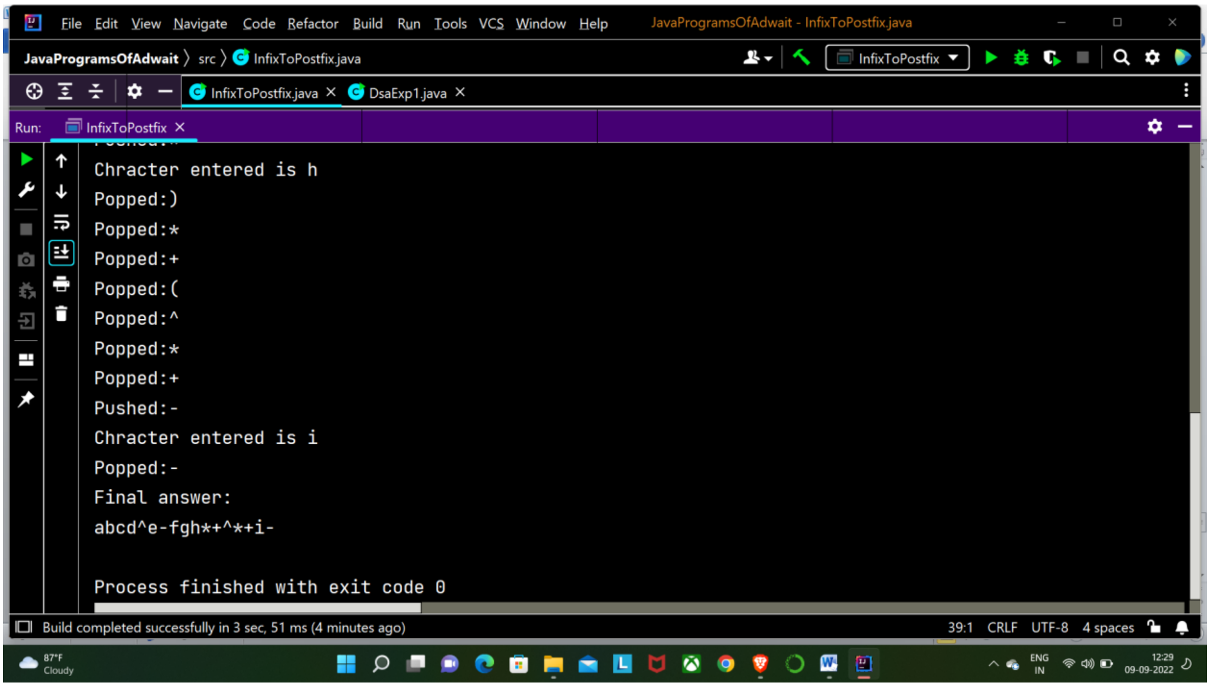
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
JavaProgramsOfAdwait - InfixToPostfix.java
JavaProgramsOfAdwait 〉 src 〉 InfixToPostfix.java
InfixToPostfix
InfixToPostfix.java ×
DsaExp1.java ×
Run: InfixToPostfix ×
Build completed successfully in 3 sec, 51 ms (3 minutes ago)
39:1 CRLF UTF-8 4 spaces
87°F Cloudy
ENG IN
12:28 09-09-2022

```
"C:\Program Files\Java\jdk-18.0.1\bin\java.exe" "-javaagent:C:\Program Files\Java\IntelliJ IDEA Communi
Enter the string:
a+b*(c^d-e)^(f+g*h)-i
Chracter entered is a
Pushed:+
Chracter entered is b
Pushed:*
Pushed:(
Chracter entered is c
Pushed:^
Chracter entered is d
Popped:^
Pushed:-
Chracter entered is e
Popped:)
```


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
JavaProgramsOfAdwait - InfixToPostfix.java
JavaProgramsOfAdwait 〉 src 〉 InfixToPostfix.java
InfixToPostfix
InfixToPostfix.java
DsaExp1.java
Run: InfixToPostfix

```
Popped:)
Popped:-
Popped:(
Pushed:^
Pushed:(
Chracter entered is f
Pushed:+
Chracter entered is g
Pushed:*
Chracter entered is h
Popped:)
Popped:*
Popped:+
Popped:(
Popped:^
```

Build completed successfully in 3 sec, 51 ms (3 minutes ago)
39:1 CRLF UTF-8 4 spaces
87°F
Cloudy
ENG
IN
12:29
09-09-2022


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
JavaProgramsOfAdwait - InfixToPostfix.java
JavaProgramsOfAdwait 〉 src 〉 InfixToPostfix.java
InfixToPostfix
InfixToPostfix.java
DsaExp1.java
Run: InfixToPostfix

```
Chracter entered is h
Popped:)
Popped:*
Popped:+
Popped:(
Popped:^
Popped:*
Popped:+
Pushed:-
Chracter entered is i
Popped:-
Final answer:
abcd^e-fgh*+^*+i-

Process finished with exit code 0
```

Build completed successfully in 3 sec, 51 ms (4 minutes ago)
39:1 CRLF UTF-8 4 spaces
87°F
Cloudy
ENG
IN
12:29
09-09-2022

CONCLUSION:

In the above experiment we learned about the linear data structure stack and various methods associated with it like push , pop , isEmpty and isFull. With the help of these methods we wrote a program that converts an infix expression to an postfix expression(which is understood by compiler). We used the stack to store the operators and to push and pop them according to the required conditions.