STACK

Objective: At the end of this topic, students should be able to:

1. Understand the concept of Stack
2. Design and write programs for solving problems by using Stack data structure.

Domain : An Arithmetic Calculator

Undestand how Stack is used/related to solve Postfix Machine, Infix-to Postfix Converter and Balance Symbol Checker problems.

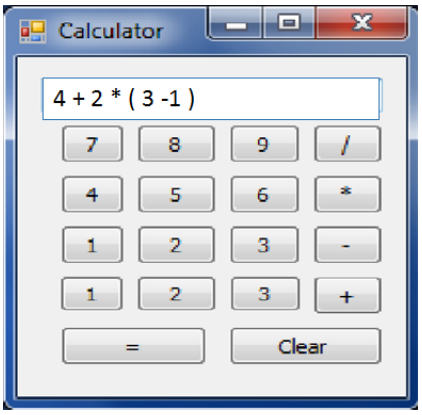
Woked-Example:

Lab 1. Postfix Machine

Lab 2. Infix-Postfix Converter

Lab 3: Balance Symbol Checker

An Arithmetic Calculator



In this topic, we will learn how *stack* is used in perform artihmetic calculation by a calculator. Our calculator is not like an ordinary calculator where you input the operand (value) and the operator one after another, and the calculator will show the result everytime you press the operator or equal symbol. Our calculator will receive the arithmetic expression at once, and then display the final result.

The knowledge in your first programming is not enough for you to create this calculator program. Look at the below situation:

Case	Input Example	Possible program

Calculator that receives exactly two known operads and one known operator.	4 + 2	`int value;`
		`value = 4 + 2;`
		`System.out.println("Total " + value);`

What if, the two operands are unknown, and one known operator.	? + ?	`int a, b, value;`
		`Scanner sc = new Scanner (System.in);`
		`a= sc.nextLine();`
		`b= sc.nextLine();`
		`value = a + b;`
		`System.out.println("Total " + value);`

Calculator that receives exactly three known operands and two known operators.	4 + 2 * 3	`int value;`
		`value = 4 + 2 * 3;`
		`System.out.println("Total " + value);`

What if, three operands are unknown, and two known operands.	? + ? * ?	`int a, b, c, value;`
		`Scanner sc = new Scanner (System.in);`
		`a= sc.nextLine();`
		`b= sc.nextLine();`
		`c= sc.nextLine();`
		`value = a + b * c;`
		`System.out.println("Total " + value);`

What if, three operands are unknown, and two operators are also unknown?	? op ? op ?	`int a, b, c, value;`
		`char op1, op2;`
		`Scanner sc = new Scanner (System.in);`
		`a= sc.nextLine();`
		`op1= sc.nextLine();`
		`b= sc.nextLine();`
		`op2= sc.nextLine();`
		`c= sc.nextLine();`
		`switch (op1):`
		`  case '+': value = a+b; break;`
		`  case '-': value = a-b; break;`
		`  case '*': value = a*b; break;`
		`  case '/': value = a/b; break;`
		`switch (op2):`

```
                                                    case '+': value += c; break;
                                                    case '-': value -=c; break;
                                                    case '*': value *=c; break;
                                                    case '/': value /=c; break;

                                                cout << "Total " << value << endl;
```

So, how the program looks like if the input consists of any number of unknown operators and any number of unknown operands such as follows:

? op ? op ? op ? op ? …… ? op ? op ?

Besides that, how to handle operator precedence problem for example, 4 + 2 * 3 ≠ 4 * 2 + 3.

Hence in this topic we will use stack to handle the operator precedence as well as to handle any length of expression. The main algorithm for this problem is:

1. Get input: Infix expression.
2. Convert: Infix expression to postfix expression
3. Evaluate: Postfix expression
4. Output result.

For example:

1. Get input: 4 + 2 * 3
2. Convert: 4 + (2 * 3) → 4 2 3 * +
3. Evaluate: 4 2 3 * +
   = 4 6 +
   = 10
4. Output: 10

Therefore, in this topic we will divide the problem into 2 subproblems that are:

1. Infix to Postfix Converter, to convert from infix expression to postfix expression.
2. Postfix Evaluator, to evaluate postfix expression.

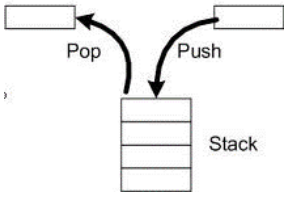
And we add one more subproblem as follows:

3. Balance Symbol Checker, to check the bracket balancing in the expression.

About STACK

The **java.util.Stack** class is the package that contains the collection of stack classes. It represents a last-in-first-out (LIFO) stack of objects.

Concept: First in last out.



How to declare a stack?

```
Stack <Integer> myStack = new Stack();// creating myStack object
Stack <String> yourStack = new Stack();// creating yourStack of string object
```

Common methods used in this problem:

Push: to put data onto stack.

```
int input = Integer.valueOf(4);
yourStack.push("hello");
myStack.push(input);
```

Pop: to remove the object at the top of this stack and returns that object as the value of this function.

```
int a = myStack.pop();
System.out.println("Removed object:" + yourStack.pop());
```

Peek: to look at the object at the top of stack without removing it from the stack.

```
Integer data1, data2;
data1 = myStack.peek();
data2 = (Integer) yourStack.peek();
int d1 = data1.intValue();
int d2 = data2.intValue();
System.out.println("Total:" + (d1+d2));
```

Empty: to check whether the stack is empty.

```
if (myStack.empty())
       System.out.println("No data in myStack");
System.out.println("Is yourStack empty: " + yourStack.empty());
```

Tutorial Activity:

1. Explained the concept of Stack.
2. List out any 3 methods in Stack class and define its purpose.
3. Can you directly perform arithmetic calculation using the object from a stack? Why?

Hands-on Activity:

1. Create java program that read 10 integer numbers. If the number is greater than 0, push them onto a *PositiveStack*. Then pop and display all the elements of Positive*Stack*, and its size. Display your output in this format: Size of stack, followed by a colon symbol, followed by a space followed by the elements of the stack that are separated by a space.

   Sample IO:

Input	Output
34 1 8 -6 7 0 7 10 -2 42	7: 42 10 7 7 8 1 34
1 2 3 4 0 0 -2 -9 5 4	6: 4 5 4 3 2 1

2. ***Palindrome*** is a sequence of characters, a word, phrase, number or sequence of words which reads the same backward as forward. Example of palindrome words are katak, civic and anna. Write a program that will read the word and identify whether it is a palindrome or not.
   Sample I/O:

   Input: racecar
   Ouput: racecar not a palindrome

   Input: p(;(p
   Ouput: p(;(p is a Palindrome

3. Create java program that will convert a *decimal number* into a *binary number*.

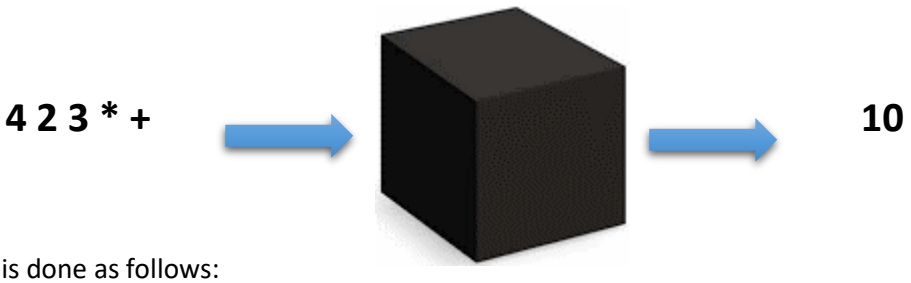
   Sample IO:
   Input in decimal number : 23
   Output in binary number : 10111

Problem 1: Postfix Evaluator

POSTFIX MACHINE	
Input	Standard Input
Output	Standard Output
Java Elements	Loop, Selection
Data Structure	stack

Problem Description

The Postfix Machine is a program that receives a postfix expression and calculate the value of the expression. The postfix form represents a natural way to evaluate expressions because precedence rules are not required. The black box shown below represents the Postfix Evaluator. If we give the input 4 2 3 * +, the program will give the output 10. How the evaluation is done?



The evaluation is done as follows:

4 2 3 * +	[Compute the last two operands with the first operator that is 2 * 3]
4 6 +	[Repeat the computation]
10	[This is the result]

Scope of the program:

- The operator is a binary operator: addition, multiplication, division and subtraction.
- The operand is not more than 3 digit integer.
- The expression ends with symbol semicolon.
- The expression only contains operand, operator and semicolon that are separated by spaces.

Your task is to write a program for postfix evaluator.

Input

The input contains only symbol +, -, *, / and positive number with not more than 3 digits. Each of the symbol and numbers are separated with a single space. The input ends with symbol semicolon. Input example:

4 2 3 * + ;

Output

The output will be an integer. Based on the above example, the output will be:

10

Solution

Postfix Evaluator Algorithm:

Read the string
Split the string into tokens
Repeat until token is semicolon

When a binary operator is seen:
the two operands are popped from the stack,
the operator is evaluated, and the result is pushed back onto the stack.
When an operand is seen:
push it onto a stack.

Read next token

When the complete postfix expression is evaluated:
the result should be a single item on the stack

Basic Structure of the program:

```
public static void main(String[] args) {
       Scanner in = new Scanner(System.in);
       Stack myStack = new Stack();
       String InStr;

       InStr = in.nextLine(); // input string
       StringTokenizer st = new StringTokenizer(InStr);
       while (st.hasMoreTokens()) {
            String nextT = st.nextToken();
            if(!(nextT.equals(";"))) {
                 PART A1
            }
            else
               break;
       }
       PART B1
}
```

Complete structure of the program: PostfixEvaluator.

```
1   import java.util.*;
2
3   public class test {
4
5     public static void main(String[] args) {
6          Scanner in = new Scanner(System.in);
7          Stack myStack = new Stack();
8          String InStr;
9          Integer f1, f2;
10         int result;
11
12         InStr = in.nextLine(); // input string
13         StringTokenizer st = new StringTokenizer(InStr);
14         while (st.hasMoreTokens()) {
15              String nextT = st.nextToken();
16              if(!(nextT.equals(";"))) {
17                  if (isOperator(nextT)){ //
18                       // >>> ADD YOUR CODE I HERE <<<
19
20                   }
21                   else {
22                       // >>> ADD YOUR CODE II HERE <<<
23                   }
24               }
25               else
26                  break;
27          }
28        // >>> ADD YOUR CODE III HERE <<<
29
30    } // end main()
31
32   static boolean isOperator(String tmp) {
33       if ((tmp.equals("+"))||(tmp.equals("-"))||(tmp.equals("*"))||
34            (tmp.equals("/")))
35                return true;
36         return false;
37    }
38
39    static int evaluate(Integer op1, Integer op2, String s1) {
40         int data1 = op2.intValue();
41         int data2 = op1.intValue();
42
43         if (s1.equals("+")) return (data1 + data2);
44                else if (s1.equals("-")) return (data1 - data2);
45                else if (s1.equals("*")) return (data1 * data2);
46                else if (s1.equals("/")) return (data1 / data2);
47                else return -1;
48    }
49
50  } // end class
```

Self Activity

1. Can you guess what is the value for the following postfix expression:

Postfix expression	Value
2 3 4 5 - + *	?
4 2 * 3 +	?

2. Among code I, II and III, which will used to impliment Part A1 and Part B1?
3. Try to run the PostfixEvaluator program and study the output.
4. Now, compare your answer with the program output.
5. If you get similar answer with the program, then:
    a. Congratulations. This mean you understand the problem.
6. If the program give different answer from yours, then:
    a. Discuss with your friends and ask your instructor during tutorial session.

Tutorial Activity

1. List out and explain the aims of the standard function/method and user defined function/method used in the given program.
2. Identify main variable and supportive variables.
3. Which part in the program shows the following:
    a. The operator is a binary operator: addition, multiplication, division and subtraction.
    b. ~~The operand is not more than 3 digit integer~~.
    c. The expression ends with symbol semicolon.
4. Discuss possible errors that might occures and how to control it. For example, if the user key in symbols other than operator, operand and semicolon, we can produce a message “Unknown symbol”, and then exit the program, as in line 24-25.
5. Discuss the missing code in the given program (Code I, II and III).
6. Study and try run the Postfix Evaluator program from link Stack | Set 4 (Evaluation of Postfix Expression) - GeeksforGeeks
7. Try to complete and run the Postfix Machine code in this modul (page Stack8). List out the differences between Postfix Evaluator program from GeeksforGeeks (Question 6) and this modul. Which one is more accurate to perform the postfix evaluator?

Lab 1:

1. Problem-solving: Complete the *PostfixEvaluator* program.
2. Challenge yourself: Implement the error handling.

Problem 2: Infix to Postfix Converter

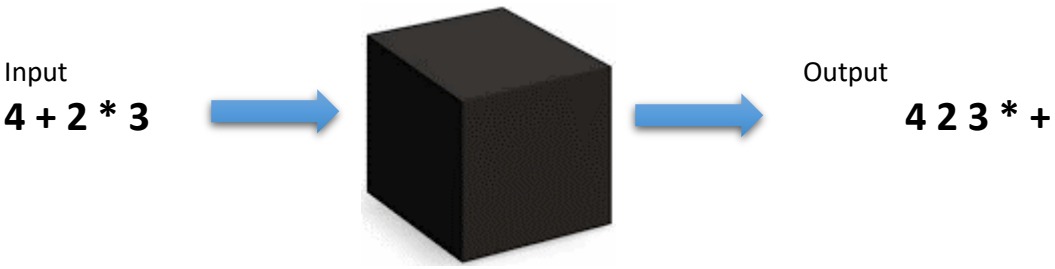
INFIX-POSTFIX CONVERTER	
Input	Standard Input
Output	Standard Output
Java Elements	Loop, Selection
Data Structure	stack

Problem Description

Human can understand better when they read mathematical expression in infix form. In this discussion, infix form means the operator is in the middle of two operands. For example, 4 + 2, I am sure all of you understand this expression. However, for a machine, infix form of an expression is hard to process, it prefers postfix expression. For example, 4 2 + which the results is similar to 4 + 2 that is 6.

The issue is human will provide the input, and machine will process it. Therefore, we need a mechanism to convert the input which is in infix expression (given by the human) to postfix expression. In the real calculator, this process is hidden. However, in this topic, we will display the result of the convertion.

The black box shown below represents the Infix to Postfix Converter. If we give the input 4 + 2 * 3, the converter will give the output 4 2 3 * +. How the conversion is done?



Your task is to write a program to perform infix-postfix converter.

Input

The input contains only symbol +, -, *, / and positive number with not more than 3 digits. Each of the symbol and numbers are separated with a single space. The input ends with symbol semicolon. Input example:

4 + 2 * 3 ;

Output

The output will be a postfix expression as follows:

4 2 3 * + ;

Solution

Infix to Postfix Converter Algorithm:

Read the first string

Repeat until meet semicolon

When meet Operator:
Pop all stack symbols until a symbol of lower precedence or a right-associative symbol of equal precedence appears.
Then push the operator.
When meet Operand:
Immediately output.

Read next string

When meet End of input:
Pop all remaining stack symbols.

Basic Structure of the program:

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

   Scanner in = new Scanner(System.in);
   Stack Stack1 = new Stack();
   String temp = " ";
   char symT = ' ';

   String infix = in.nextLine();
   StringTokenizer st = new StringTokenizer(infix);
   while (st.hasMoreTokens()) {
       String nextT = st.nextToken();
       if(!(nextT.equals(";"))) {
```

PART A2

```
       }
    }
```

PART B1

```
   }
```

Complete structure of the program: Infix2Postfix Converter

```
1  import java.util.*;
2
3  public class infix2postfix {
4
5    public static void main(String[] args)  {
6       Scanner in = new Scanner(System.in);
7       Stack Stack1 = new Stack();
8       String temp = " ";
9       char symT = ' ';
10
11      String infix = in.nextLine(); // input string
12      StringTokenizer st = new StringTokenizer(infix);
13      while (st.hasMoreTokens()) {
14         String nextT = st.nextToken();
15         if(!(nextT.equals(";"))) {
16             if (isOperator(nextT)){
17                symT = atoc(nextT);
18                 // >>>>> add your code here <<<<<
19             }
20             else
21               // >>>>> add your code here if meet operand <<<<<
22          }
23      }
24         // >>>>> add your code here  <<<<<
25      }
26
27    static char atoc(String str) {
28       char data1 = str.charAt(0);
29          return data1;
30    }
31
32    static boolean isHigherThan(char op1, char op2){
33
34              // >>>>> add your code here  <<<<<
35      }
36
37    static boolean isOperator(String tmp) {
38
39             // >>>>> add your code here  <<<<<
40    }
41
42 }
```

Self Activity

1. Read https://codeburst.io/conversion-of-infix-expression-to-postfix-expression-using-stack-data-structure-3faf9c212ab8.
2. Can you guess what is the postfix expression for the following infix expression:

Infix expression	Postfix expression
120 - 4 * 2 + 3	?
((A * (B + C)) / D)	?

3. Try to run Infix2Postfix program.
4. Now, input the infix expressions from question 2, compare your answer with the program output.
5. If you get the similar answer with the program, then:
    a. Congratulations. This mean you understand the problem.
6. If the program give different answer from yours, then:
    a. Ask your friends. Then, ask your tutor....

Tutorial Activity

1. Identify code similarities between *Postfix Machine* and Infix-Postfix Converter.
2. (i) List out and (ii) explain the aims of the *user defined function* used in the given program.
3. Identify *main* variable and *supportive* variables.
4. Refer to Self Activity no. 2, use the given algorithm to obtain the answer.
5. Write a Java code for the following statement:
    Pop all stack symbols until a symbol of lower precedence or a right-associative symbol of equal precedence appears.
6. Discuss possible errors that might occures and how to control it.
7. Discuss the missing code in the given program.

Lab 2

1. Complete the Infix-*Postfix Converter* program.
2. Implement the error handling.

Challenge yourself:

1. The original algorithm of Infix-Postfix Converter should consider Parenthesis symbol that is (and), as follows. Enhanced your Infix2Postfix Converter program by considering the parenthesis symbol.

> When meet Operand:
> immediately output.
> ***When meet Close Parenthesis:***
> ***Pop stack symbols until an open parenthesis appears.***
> When meet Operator:
> Pop all stack symbols until a symbol of lower precedence or a right-associative symbol of equal precedence appears.
> Then push the operator.

BALANCE SYMBOL CHECKER

Input	Standard Input
Output	Standard Output
Programming Elements	loop
Data Structure	stack

Problem Description

Compilers check your programs for syntax errors. Frequently, however, a lack of one symbol, such as a missing * / comment-ender, causes the compiler to produce numerous lines of diagnostics without identifying the real error. A useful tool to help debug compiler error messages is a program that checks whether symbols are balanced. In other words, every { must correspond to a }, every [to a], every (to a) and so on.

However, simply counting the numbers of each symbol is insufficient. For example, the sequence [( )] is legal, but the sequence [( ]) is wrong. A stack is useful here because we know that when a closing symbol such as) is seen, it matches the most recently seen unclosed (. Therefore, by placing an opening symbol on a stack, we can easily determine whether a closing symbol makes sense.

The black box shown below represents the Balance Symbol Checker. If we give the input (4 + (2 * 3), the program will display the error message. How this is done?

(4 + (2 * 3)  **Error: Symbol ‘)’ expected.**

Balance Symbol Checker Algorithm

Read the first symbol
Repeat until meet semicolon

If the symbol is an opening symbol,
Push it onto the stack.
If it is a closing symbol do the following.
If the stack is empty, report an error.
Otherwise, pop the stack. If the symbol popped is not the corresponding opening symbol, report an error.

Read next symbol

When meet End of input:
If the stack is not empty, report an error

Challenge your self to implement the Balance Symbol Checker. Good luck!!!

Input

The input contains any symbols including alphabets or numbers. Each of them are separated with a single space. The input ends with symbol semicolon. Input example:

(4 + (2 * 3) ;

Output

The output will be a word either 'Balanced' or 'Unbalanced'. The above input example will give the following output:

Unbalanced

Other Applications of Stack

Besides the previous problems, stack is also used in many other applications. Some of the examples are as below:

1. Converting a decimal number into a binary number:
   Input in decimal number : 23
   Output in binary number: 10111

2. In a puzzle called maze, if the path we have chosen is wrong, we need to find a way by which we can return to the beginning of the latest junction.