Graphical user interface

Description automatically generated with low confidence

TTTK1143

Rekabentuk aturcara dan penyelesaian masalah

Tutorial 6

Stack

|  |  |
| --- | --- |
| Name | Mohamad Saiful Nizam Bin Abd Aziz |
| No. Matric | A179830 |

# TK1143 Program Design STACK

**Section A:**

1. Write a java statement to declare and create:
   1. An object variable myStack that represent a data of student’s matric number.

Stack<String> myStack=new Stack<String>();

* 1. An object variable yourStack that represent a data of student’s weight.

Stack<Double>yourStack=new Stack<Double>();

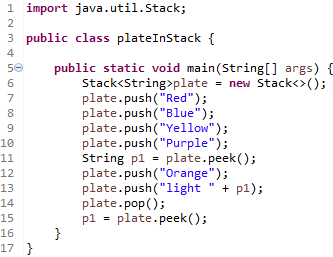
1. Complete the following table with the description of each method in Stack.

|  |  |
| --- | --- |
| **Method** | **Description** |
| push( ) | Push element into the stack |
| pop( ) | Remove the top element of the stack |
| peek( ) | Retrieve top element without removing it and return the element. |
| size( ) | Get the stack’s size |
| empty( ) | This method does not take any parameter. Returns Boolean value. |

1. Complete the following table that shows a series of stack operations and their effects on an initially empty stack S of integers.

|  |  |  |
| --- | --- | --- |
| **Method** | **Return Value** | **Stack Contents** |
| push(9) | - | [9] |
| push(7) | - | [9, 7] |
| push(5) | - | [9, 7, 5] |
| size( ) | 3 | [9, 7, 5] |
| peek( ) | 5 | [9, 7, 5] |
| pop( ) | 5 | [9, 7] |
| isEmpty( ) | false | [9, 7] |
| peek( ) | 7 | [9, 7] |
| pop( ) | 7 | [9] |
| pop( ) | 9 | - |
| isEmpty( ) | true | - |
| push(5) | - | [5] |
| push(2) | - | [5, 2] |
| peek( ) | 2 | [5, 2] |
| push(7) | - | [5, 2, 7] |
| pop( ) | 7 | [5, 2] |
| size( ) | 2 | [5, 2] |
| pop( ) | 2 | [5] |
| Peek( ) | 5 | [5] |

1. Refer to the given program. Draw a diagram to represent the following stack operations, step-by-step as the program executes. Show the output, if any.



Answer:

|  |
| --- |
| purple |

|  |
| --- |
| purple |
| yellow |
| Blue |
| red |

|  |
| --- |
| orange |
| purple |
| yellow |
| blue |
| red |

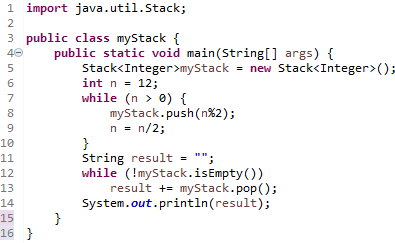
|  |
| --- |
| orange |
| purple |
| yellow |
| blue |
| red |

|  |
| --- |
| Light purple |
| Orange |
| purple |
| Yellow |
| Blue |
| red |

|  |
| --- |
| orange |

# Section B:

1. What is the output of the following program. Draw the content of stack.



Answer :

1

11

110

1100

1. Complete the following program, trace and give the output.

new Stack<String>();

stack1.peek();

stack1.empty()

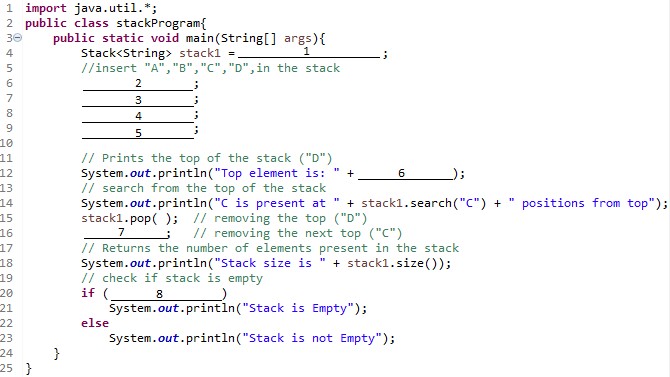
stack1.pop();

stack1.push(“C”);

stack1.push(“D”);

stack1.push(“B”);

stack1.push(“A”);



Answer :

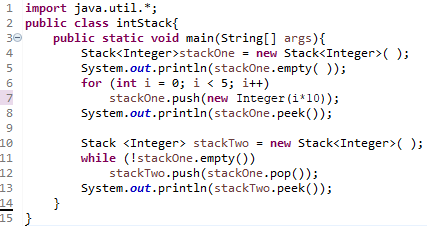
Top element is: D

C is present at 2 position from top

Stack size is 2

Stack is not Empty

1. Trace the output of the following program and explain how the program executes.



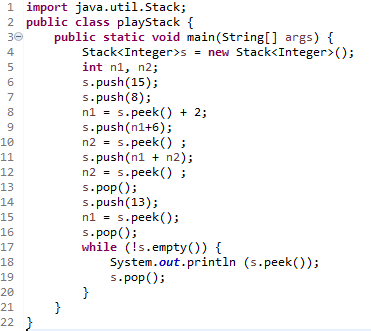
Answer :

true

40

0

1. Trace the given program and draw the content of stack step-by-step as the program executes.



Answer :

15

13

13

15

13

15

16

n1+2

10

15

8

n1+n2

n2

Output:

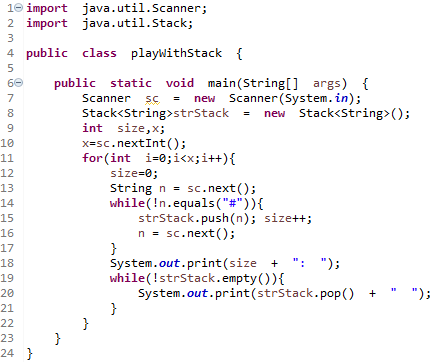
16

8

15

1. Trace the output of the following program using the sample input given. Explain the implementation of stack in this program.

|  |
| --- |
| Sampel Input |
| 2 |
| i like to eat chocolate # |
| java programming # |



Answer :

5: chocolate eat to like i

2: programming java

# Section C:

There are four (4) questions given in this section. Your task is to write the program to implement data structure Stack for each case.

|  |  |
| --- | --- |
| PALINDROME | |
| Input | Standard Input |
| Output | Standard Output |
| Programming Elements | loop |
| Data Structure | stack |

## Problem Description

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

## Input

The first line input contains an integer which determines the number of test cases. Each of following lines represent words.

3

racecar java madam

## Output

For each test case, the output contains respond either ‘It is a Palindrome’ or ‘Not a palindrome’.

It is a Palindrome Not a palindrome It is a Palindrome

Your task is to write a program that will read the word and identify whether it is a palindrome or not.

## Self Study

1. Explain the implementation of Stack in this case study.

|  |  |
| --- | --- |
| 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 Machine. If we give the input 4 2 3 \* +, the program will give the output 10. How the evaluation is done?



**4 2 3 \* +**

**10**

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 machine.

## 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 \* + ;

120 3 4 \* - 5 + ;

4 2 3 \* +

## Output

The output will be an integer. Based on the above example, the output will be: 10

113

ERROR: no END OF STRING in the expression

## Solution

**Postfix Machine 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:

it is pushed 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/solution**: PostfixMachine.

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

## Self Study

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

2 3 4 5 - + \* ?

4 2 \* 3 + ?

1. In your own words, explain Part A1 and Part B1 from the basic structure solution.
2. Among code I, II and III (from the Worked-Example Solution) which code will be used to implement Part A1 and Part B1 (in Basic Structure-Solution)?
3. Try to run the [PostfixMachine program](http://embed.plnkr.co/iNnGlm/) and study the output.
4. Compare your answer from question 1 with the program output from question 4. If you get similar answer with the program, then:
   1. Congratulations. This mean you understand the problem. If the program give different answer from yours, then:
   2. Discuss with your friends and ask your instructor during tutorial session.
5. Study and try run the Postfix Evaluator program from link [Stack | Set 4 (Evaluation of Postfix Expression) - GeeksforGeeks](https://www.geeksforgeeks.org/stack-set-4-evaluation-postfix-expression/).
6. Compare the differences between Postfix Machine program from GeeksforGeeks (Question 6) and your previous program. Which one is more accurate to perform the postfix machine?

|  |  |
| --- | --- |
| INFIX TO 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 conversion.

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?



**4 + 2 \* 3**

**4 2 3 \* +**

Your task is to write a program to perform infix to 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 ;

4 + 2 \* 3

## Output

The output will be a postfix expression as follows: 4 2 3 \* +

ERROR: no END OF STRING in the expression

## 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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | **import** java.util.\*;  **public class** Infix2Postfix {  **public static void** main(String[] args) { Scanner in = **new** Scanner(System.***in***); Stack Stack1 = **new** Stack();  String temp = " ";  **char** symT = ' ';  String infix = in.nextLine(); // input string StringTokenizer st = **new** StringTokenizer(infix); **while** (st.hasMoreTokens()) {  String nextT = st.nextToken();  **if**(!(nextT.equals(";"))) {  **if** (*isOperator*(nextT)){ symT = *atoc*(nextT);  *// >>>>> add your code here <<<<<*  }  **else**  *// >>>>> add your code here if meet operand <<<<<*  }  }  *// >>>>> add your code here <<<<<*  }  **static char** atoc(String str) {  **char** data1 = str.charAt(0);  **return** data1;  } |

|  |  |
| --- | --- |
| 33  34  35  36  37  38  39  40  41  42  43  44 | **static boolean** isHigherThan(**char** op1, **char** op2){  *// >>>>> add your code here <<<<<*  }  **static boolean** isOperator(String tmp) {  *// >>>>> add your code here <<<<<*  }  } |

## Self Study

1. Read [https://codeburst.io/conversion-of-infix-expression-to-postfix-expression-using- stack-data-structure-3faf9c212ab8.](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 4 \* 2 + 3 ?

2 + 3 – 5 \* 1 / 3 ?

1. Run the given [*Infix2Postfix*](http://embed.plnkr.co/iNnGlm/) file.
2. Now, compare your answer from question 2 with the output from [*Infix2Postfix*](http://embed.plnkr.co/iNnGlm/) file . If you get the similar answer with the program, then:
   1. Congratulations. This mean you understand the problem.
   2. If the program file give different answer from yours, then Ask your friends. Then, ask your tutor.

|  |  |
| --- | --- |
| 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.



Challenge yourself to write a program to perform Balance Symbol Checker.

## 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 ) ;

( 4 + ( 2 \* 3 )

## Output

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

Unbalanced

ERROR: no END OF STRING in the expression

## Self Study

1. Can you guess what is the output for the following input?

Input Output

[ ( 5 – 2 ) \* 3 ] ?

( ( A \* ( B + C ) / D ) ?

1. Explain the implementation of Stack in this case study.