|  |  |
| --- | --- |
| **Name** | **Hatim Yusuf Sawai** |
| **UID no.** | **2021300108** |
| **Experiment No.** | **1** |

|  |  |
| --- | --- |
| **AIM:** | **Evaluation of Post-fix expression using Stack** |
| **Program** | |
| **THEORY:** | **STACK:**  A stack is an ordered list in which insertion and deletion are done at one end, called the top. The last element inserted is the first one to be deleted. Hence, it is called the Last in First out **(LIFO)**. When an element is inserted in a stack, the concept is called **push**, and when an element is removed from the stack, the concept is called **pop**. Trying to pop out an empty stack is called **underflow** and trying to push an element in a full stack is called **overflow.**  **Stack Operations:**  1. **push(data):** Inserts data onto stack.  2. **pop():** Removes and returns the last inserted element from the stack.  3. **Top():** Returns the last inserted element without removing it.  4. **Size():** Returns the number of elements stored in the stack.  5. **IsEmptyStack():** Indicates whether any elements are stored in the stack or not.  6. **IsFullStack():** Indicates whether the stack is full or not.  **Applications of Stack:**  1. Balancing of symbols  2. Infix-to-postfix conversion  3. Evaluation of postfix expression  4. Implementing function calls (including recursion)  5. Finding of spans (finding spans in stock markets, refer to Problems section)  6. Page-visited history in a Web browser [Back Buttons]  7. Undo sequence in a text editor  8. Matching Tags in HTML and XML  **Postfix:**  A postfix expression (also called Reverse Polish Notation) is a single letter or an operator, preceded by two postfix strings. Every postfix string longer than a single variable contains first and second operands followed by an operator.  **Postfix Evaluation Using Stack:**  1. Scan the Postfix string from left to right.  2. Initialize an empty stack.  3. Repeat steps 4 and 5 till all the characters are scanned.  4. If the scanned character is an operand, push it onto the stack.  5. If the scanned character is an operator, and if the operator is a unary operator, then pop an element from the stack. If the operator is a binary operator, then pop two elements from the stack. After popping the elements, apply the operator to those popped elements. Let the result of this operation be returned onto the stack.  6 After all characters are scanned, we will have only one element in the stack.  7 Return top of the stack as result.  **Example:**  Assume that the postfix string is **123\*+5-**  1. Initially the stack is empty. Now, the first three characters scanned are 1, 2 and 3, which are operands. They will be pushed into the stack in that order:  2. The next character scanned is “\*”, which is an operator. Thus, we pop the top two elements from the stack and perform the “\*” operation with the two operands. The second operand will be the first element that is popped:  3. The next character scanned is “+”, which is an operator. Thus, we pop the top two elements from the stack and perform the “+” operation with the two operands.  4. The value of the expression (1+6) that has been evaluated (7) is pushed into the stack. The next character scanned is “5”, which is added to the stack.  5. The next character scanned is “-”, which is an operator. Thus, we pop the top two elements from the stack and perform the “-” operation with the two operands. The second operand will be the first element that is popped. The value of the expression(7-5) that has been evaluated(23) is pushed into the stack  6. Now, since all the characters are scanned, the remaining element in the stack (there will be only one element in the stack) will be returned. End result:  1. **Postfix String:** 123\*+5-  2. **Result:** 2 |
| **ALGORITHM:** | **Class PostfixEval**  **Main Method**:   1. Input equation as string from user 2. Store length of string in size 3. Convert string to CharArray 4. Initialise **IntStack** Object 5. Int a,b vars for saving popped operands 6. Start for loop: i=0 to i<size: 7. If char is digit then push into stack 8. Else If char is ‘~’ then pop top and push with negation 9. Else: 10. A = pop() 11. B = pop() 12. Switch case to match operators: +,-,\*,/,^ 13. Perform operation and push result to stack 14. Display final result stack[top]   **Class IntStack**  **Members**:   1. Int[] stack 2. Int capacity 3. Int front,rear   **Constructor:**   1. Initialize stack array 2. Top = -1   **Push Method:**   1. Check if stack is full or not, then add element to array   **Pop Method:**   1. Check if stack is empty or not, then decrease top by 1 and return stack[top--]   **IsEmptyMethod:**   1. Return true if top = -1   **IsFull Method:**   1. Return true if top = capacity-1 |
| **PROBLEM SOLVING:** |  |
| **PROGRAM:** | **PostfixEval.java:**  import *java*.*util*.*Scanner*;  import *java*.*lang*.*Math*;  import *stackds*.*IntStack*;  *public* *class* PostfixEval {  *public* *static* void main(String[] args) {          Scanner sc = new Scanner(System.*in*);          System.*out*.print("Enter postfix equation: ");          String exp = sc.nextLine();          int size = exp.length();          char[] eq = exp.toCharArray();          int a,b;          IntStack s = new IntStack(size);          try {              for(int i=0;i<size;i++) {                  if(Character.isDigit(eq[i])) {                      s.push(eq[i]-'0');                  } else if(eq[i] == '~') {                      a = s.pop();                      a = -a;                      s.push(a);                  } else {                      b = s.pop();                      a = s.pop();                      switch(eq[i]) {                          case '+':                              s.push(a+b);                              System.*out*.printf("%d+%d = %d\n",a,b,a+b);                              System.*out*.println(s.printStack());                              break;                          case '-':                              s.push(a-b);                              System.*out*.printf("%d-%d = %d\n",a,b,a-b);                              System.*out*.println(s.printStack());                              break;                          case '\*':                              s.push(a\*b);                              System.*out*.printf("%d\*%d = %d\n",a,b,a\*b);                              System.*out*.println(s.printStack());                              break;                          case '/':                              s.push(a/b);                              System.*out*.printf("%d/%d = %d\n", a, b, a / b);                              System.*out*.println(s.printStack());                              break;                          case '^':                              s.push((int)Math.pow(a,b));                              System.*out*.printf("%d^%d = %d\n", a, b, s.Top());                              System.*out*.println(s.printStack());                              break;                      }                  }              }              System.*out*.println("Result: "+s.Top());          }          catch(Exception ex) {              System.*out*.println(ex.getMessage());          }          sc.close();      }  }  **IntStack.java:**  package *stackds*;  *public* *class* IntStack {      int[] stack;      int top;      int capacity;  *public* IntStack(int size) {          stack = new int[size];          capacity = size;          top = -1;      }  *public* void push(int e) *throws* Exception {          if(isFullStack()) {              throw new Exception("Stack is full!");          }          stack[++top] = e;      }  *public* int pop() *throws* Exception {          if(isEmptyStack()) {              throw new Exception("Stack is empty!");          }          return stack[top--];      }  *public* int Top() *throws* Exception {          if(isEmptyStack()) {              throw new Exception("Stack is empty!");          }          return stack[top];      }  *public* String printStack() {          String s = "[";          if(size()>0) {              s += stack[0];          }          if(size()>1) {              for(int i=1;i<size();i++) {                  s += "," + stack[i];              }          }          return s += "]";      }  *public* boolean isEmptyStack() {          return top == -1;      }  *public* boolean isFullStack() {          return top == capacity - 1;      }  *public* int size() {          return top+1;      }  } |
| **OUTPUT:** | |
| **CONCLUSION:** | In this experiment, we learned how to implement a stack using an array and use the stack to solve given postfix equations effectively. |