CS 315: **Implementations of the ADT Stack****tack**

|  |  |  |
| --- | --- | --- |
| **StackArrayBased class**  **Implements StackInterface**  Private data fields   * An array of Objects called items * The index top | **StackReferenceBased**   * Implements StackInterface * top is a reference to the head of a linked list of items | **An Implementation That Uses the ADT List**   * If the item in position 1 of a list represents the top of the stack * push(newItem) operation is implemented as add(1, newItem) |
|  |  | pop() operation is implemented as get(1) remove(1)  peek() operation is implemented as  get(1) |

**Application: Algebraic Expressions**

1) **Evaluating Postfix Expressions**

When an **operand i**s entered, the calculator

* Pushes it onto a stack

When an **operator** is entered, the calculator

* Applies it to the top two operands of the stack
* Pops the operands from the stack
* Pushes the result of the operation on the stack

Example: 2 3 4 + \*



**Exercise:** Fill in the following table showing how to evaluate the postfix expression: 8 4 / 2 + 5 \* 6 -

|  |  |  |
| --- | --- | --- |
| Key Entered | Calculator action | Stack (bottom, to top) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Converting Infix Expressions to Equivalent Postfix Expressions**

An infix expression can be evaluated by first being converted into an equivalent postfix expression

When converting from infix to postfix

* Operands always stay in the same order with respect to one another
* An operator will move only “to the right” with respect to the operands
* All parentheses are removed

A trace of the algorithm that converts the infix expression *a* - (*b* + *c* \* *d*)/*e* to postfix form



Exercise: How would the postfixExp change if the infix expression: a- ( b\* c + d)/e

Provide a trace of the algorithm that converts the infix expression: a + b/c + (d-e) \* f

|  |  |  |
| --- | --- | --- |
| ch | stack (bottom to top) | postfixExp |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| for (each character ch in the infix expression) {  switch (ch) {  case operand: // append operand to end of postfixExp  postfixExp = postfixExp + ch  break  case '(': // save '(' on stack  aStack.push(ch)  break  case ')': // pop stack until matching '('  while (top of stack is not '(') {  postfixExp = postfixExp + aStack.pop()  } // end while  openParen = aStack.pop() // remove the open parenthesis  break  case operator: // process stack operators of  // greater precedence  while ( !aStack.isEmpty() and  top of stack is not '(' and  precedence(ch) <= precedence(top of stack) ) {  postfixExp = postfixExp + aStack.pop()  } // end while  aStack.push(ch) // save new operator  break  } // end switch  } // end for  // append to postfixExp the operators remaining in the stack  while (!aStack.isEmpty()) {  postfixExp = postfixExp + aStack.pop()  } // end while | **Pseudocode that converts an infix expression to a postfix form** |

|  |
| --- |
| **public interface StackInterface {**  **public boolean isEmpty();**  // Determines whether the stack is empty.  // Precondition: None.  // Postcondition: Returns true if the stack is empty;  // otherwise returns false.  **public void popAll();**  // Removes all the items from the stack.  // Precondition: None.  // Postcondition: Stack is empty.  **public void push(Object newItem) throws StackException;**  // Adds an item to the top of a stack.  // Precondition: newItem is the item to be added.  // Postcondition: If insertion is successful, newItem  // is on the top of the stack.  // Exception: Some implementations may throw  // StackException when newItem cannot be placed on  // the stack.  **public Object pop() throws StackException;**  // Removes the top of a stack.  // Precondition: None.  // Postcondition: If the stack is not empty, the item  // that was added most recently is removed from the  // stack and returned.  // Exception: Throws StackException if the stack is  // empty.  **public Object peek() throws StackException;**  // Retrieves the top of a stack.  // Precondition: None.  // Postcondition: If the stack is not empty, the item  // that was added most recently is returned. The  // stack is unchanged.  // Exception: Throws StackException if the stack is  // empty.  } // end StackInterface  **public class StackException**  extends java.lang.RuntimeException {  public StackException(String s) {  super(s);  } // end constructor  } // end StackException |