Expression Tree

**1.Testing function: string covertInfixToPostfix(string infixExpression)**

**Specification:** Function will transform an infix math expression into a postfix and return the postfix expression, by using Stack data structure.

**Error:** Function will not convert expression correctly.

**A)Test plan for: string covertInfixToPostfix(string infixExpression)**

**a)**Testing function with multiple expressions **Expected output for following expressions:**

**x+y\*a/b:** x y a \* b / + **a\*b/c+e:** a b \* c / e + **x+z-y\*b:** x z + y b \* - **2+3-8+6:** 2 3 + 8 - 6 + **x\*y+3/b-1**: x y \* 3 b / + 1 -

**2.Testing class TreeClass{};**

**A)Testing function TreeNode\*newNode(ItemType item):**

**Expected Output:** Function should allocate memory from heap and create a new TreeNode that will be returned by that function.

**Error:** No more available memory to dynamically allocate from heap

a)Testing function with bad allocation:

First function will check if memory is available to dynamically allocate a new TreeNode by calling helper function is full. If exception accusers bad\_alloc exception will be thrown and error message will be printed. Function will terminate returning nullptr.

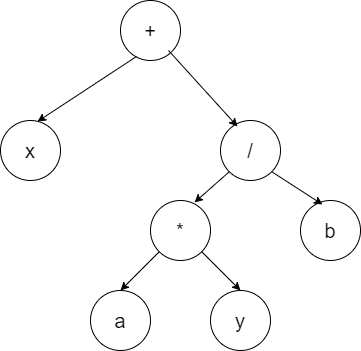
b)Testing function with available memory:

Function will allocate a new node and return it.

**B)Testing function: void createExpressionTree(string postFixExpression)**

**Expected Output:** Function will receive a postfix expression that will and will create a binary expression tree.

**Example output: for postfix expression xya \* b / + (infix: x+y\*a/b ) Expected Tree:**



**Error: Expression Tree not built correctly.**

a) Function will be tested with multiple expression, and tree will be traversed in in order, post order ad preorder to make sure tree is built correctly.

**C)Testing Function: void PrintOrders()const**

**Expected Output:** Function should print the Inorder, postorder, and preorder traversal of the expression tree by calling helper functions printPreorder(root); printPostorder(root); printInorder(root); Which recursively travers the tree and print each element in the specific order.

Output example for expression from above (expression: **x+y\*a/b):**

Preorder: + x / \* y a b

Postorder: x y a \* b / +

Inorder: ( (x) + ( ( (y) \* (a) ) / (b) ) )

**Error:** Outputting the wrong traversals, or root = nullptr (Empty tree)

a)Testing function with empty tree:

Nothing will be displayed because tree is empty, function will terminate, and program will continue running.

b)Testing function with expression tree not empty:

The traversals of the tree will be displayed like in the example above. **D)Testing function makeEmpty():**

Function description: Function will delete al the elements in expression tree by calling helper function void Destroy(TreeNode\*& tree) which recursively deletes each element of the binary tree.

a)Function will be tested with and existing expression tree, after call tree must be empty and all dynamically allocated memory must be deallocated.



**3. Testing template<class StackType> class Stack{}**

**A)Testing function bool isFull() and bool isEmpty()**

**Description bool isFull():** Function will return true top ==STACK\_SIZE else

function will return false.

**Description bool isEmpty():**Function will return true if top == -1 else function

will return false

**B)Testing** void push(StackType data)

**Description:** when functionis called and a StackType element is passed as a parameter, function will add the new element at the top of the stack;

**Error:** Stack overflow, if the stack is full an error message will be printed to the screen, and new element would not be added to the stack;

a)Testing push function with enough space on stack:

Expected output: When function void push(StackType data) will be called with a StackType element, the new element will be added at the top of the Stack.

b)Testing push function with full stack;

Expected output: Error message will be printed to the screen and new element is not going to be added to the stack.

1. **Testing** void pop();

**Description:** Element at the top of the Stack will be removed from the stack

**Error:** Stack underflow, stack is empty nothing to remove;

a)Testing pop function with elements available in stack.

Expected output: After every call of pop() the top element will be removed from the stack.

b)Testing function with empty stack

Expected output: Error message explaining that the Stack is empty will be printed to the screen.

1. **Testing** StackType peek():

**Description:** Function will return a copy of the top element from the stack.

**Error:** Stack Under Flow, stack is Empty.

a)Testing function with elements available:

Expected output: Function will return copy of StackType object, from top of the stack.

b)Testing function with empty stack:

Expected Output: Before calling function, a precondition will be added to check if stack is empty, if the stack is empty, function will not execute.

**4.Testing function:** int precedence(char operator);

**Description:** If character is an operator, function will check which type of operator it is and return the appropriate integer values corresponding to the precedence of the operator.

**Error**: input character is not an operator.

A)Testing function with a multitude of operators:

Expected output: Function will return the appropriate int values corresponding to the precedence of the operator.

B) Testing function with character input not a operator:

Expected output: Function will return -1 for all values that are not one of the operators:

**+ - \* / ^**

**5.Testing function void dipslayTraversals():**

**Description:** Function will open “ExpressionFile.txt” and read each expression line by line,

After and expression is read, the expression is converted to postfix by calling function string convertInfixToPostfix(string infixExpression), after converting the function void creatExpressionTree(string postFixExpression) will be called a will create the appropriate expression tree. After that, all three traversals (in order, post order, preorder)will be printed to the screen. After expression is displayed the tree gets deleted and a new tree is created. This process happens until we reach the end of file.

**Error:** Cannot open file or file is empty. A)Testing function with a good file:

Expected output: Function will display each individual expression with the appropriate

traversals.

B)Testing function with empty file or file not found.

Expected output: Appropriate error message will be printed program will terminate.