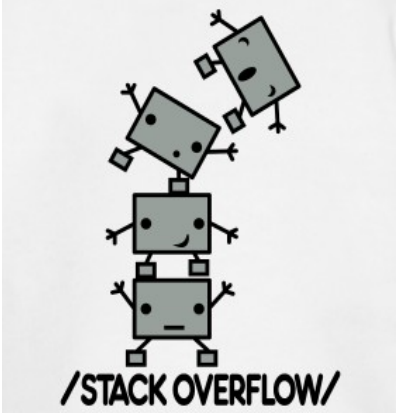
CS 202
Assignment #13

Purpose: Learn to about templates and stacks using linked lists
Due: Tuesday (4/23)
Points: 150

Assignment:



Create a C++ class to implement a stack a stack (using a linked list), ***linkedStack*** using templates. A main will be provided that can be used to test the ***linkedStack*** class.

When the ***linkedStack*** class is fully functional, create a client program (i.e., *eval.cpp*), to evaluate a postfix[1] expression.

The count of postfix expressions to prompt for and evaluate should be read from the command line. If the no arguments are provided on the command line, the program should display a usage message detailed what should be entered (i.e., **`Usage: ./eval -c <countValue>`**). If the count specifier, **`-c`**, is not provided or incorrect, an error message should be displayed (i.e., **`Error, must specify '-c' count specifier.`**). If the count value is not provided or incorrect, an error message should be displayed (i.e, **`Error, invalid count value.`**). The final postfix express should be formatted as shown in the example output.

The postfix evaluation program should include at least two functions as follows:

- Function *bool inValidNumber(const string)* that will check each character in the string and return true if each is a digit and false otherwise.
- Function *int stringToInt(const string)* that will convert a string to an integer using stringstream. This function should only be called with valid numeric strings.

A test script will be provided to test the evaluation program.

Algorithm to Evaluate Postfix Expressions

The algorithm to evaluate a postfix expression is outlined as follows:

- Create an empty stack.
- Repeat the following until the end of the expression is encountered:
  - Get the next token (constant, variable, arithmetic operator) in the postfix expression.
  - If the token is an operand (i.e., number), push it onto the stack.
  - If it is an operator then do the following:
    - Pop the top two values from the stack. If the stack does not contain two items, an error due to a malformed postfix expression has occurred and evaluation is terminated.
    - Apply the operator to these two values.
    - Push the resulting value back onto the stack.
- When the end of the expression is encountered, its value is on the top of the stack. In fact, it must be the only value in the stack and if it is not, an error due to a malformed postfix expression has occurred.

The algorithm outline must be converted into C++ code and the error handling must be addressed.

---

1 For more information, refer to: http://en.wikipedia.org/wiki/Reverse_Polish_notation

Linked Stack Class

- Linked List Stack Class
  The linked stack class will implement a stack with a linked list including the specified functions. We will use the following node structure definition.

```
template <class myType>
struct nodeType {
      myType info;
      nodeType<myType> *link;
};
```

linkedStack<myType>
-nodeType<myType> *stackTop
-count: int
+operator == (const linkedStack<myType>&): bool
+operator != (const linkedStack<myType>&): bool
+operator = (const linkedStack<myType>&): linkedStack<myType>
+linkedStack()
+linkedStack(const linkedStack<myType>&)
+~linkedStack()
+deleteStack(): void
+isStackEmpty() const: bool
+push(const myType&): void
+peek() const: myType
+pop(): myType
+stackCount() const: int
+stackSum() const: myType
-rSum(nodeType<myType> *) const: myType
-copyStack(const linkedStack<myType>&): void

Function Descriptions

- The *linkedStack()* constructor should initialize the stack to an empty state (*stackTop* = NULL and *count*=0). The *linkedStack(const linkedStack<myType>&)* copy constructor should create a new, deep copy of the passed stack. The *~linkedStack()* destructor should delete the stack (including releasing all the allocated memory).

- The *isStackEmpty()* function should determine whether the stack is empty, returning *true* if the stack is empty and *false* if not.

- The *push(const myType&)* function will add the passed item to the top of the stack.

- The *pop()* function return and remove the top item from the stack. The *peek()* function will return the current top of the stack (without changing the stack). If the stack is empty, an error message should be displayed.

- The *stackCount()* function should return the count of items currently on the stack.

- The *stackSum()* function should return the sum of all items currently on the stack. The *stackSum()* function should use the recursive *rSum()* function.

- The overloaded *operator = (const linkedStack<myType>&)* should create and return a deep copy of the stack.

- The overloaded *operator == (const linkedStack<myType>&)* should return *true* if the current and passed stacks are the same and *false* otherwise. This will require comparing the data values for each element in each stack. The overloaded *operator != (const linkedStack<myType>&)* should return *false* if the current and passed stacks are the same and *true* otherwise.

Refer to the example executions for output formatting. Make sure your program includes the appropriate documentation. See Program Evaluation Criteria for CS 202 for additional information. ***Note, points will be deducted for especially poor style or inefficient coding.***

Make File:

You will need to develop a make file. You should be able to type:

```
make stackTest
make eval
```

Which should create each of the respective executables. Typing **`make`** with no arguments should make both executables. The makefile should be able to build either of the executables; the one for the testing (i.e., testStack.cpp) and the other for the expression evaluation program (eval.cpp). The makefile will be similar to the previous assignment makefiles.

Submission:

- Submit a compressed zip file of the program source files, header files, and makefile via the on-line submission by 23:50.

All necessary files must be included in the ZIP file. The grader will download, uncompress, and type **make** (so you must have a valid, working *makefile*).

Example Execution:

Below is an example program execution for the main.

```
ed@-vm% ./testStack
**********************************************************************
CS 202 - Assignment #13
Linked List Stacks
Testing for Stack Operations

----------------------------------------------------------------------
Stack Testing -> Integers

Original List:  2 4 6 8 10 12 14 16 18 20

Sum is: 110

Reverse Original List:  20 18 16 14 12 10 8 6 4 2
Reverse, copy A:  20 18 16 14 12 10 8 6 4 2
Reverse, copy B (modified):  21 18 16 14 12 10 8 6 4 2

Stack empty errors...
Error (pop), cannot pop from an empty stack.
```

```
Error (peek), stack is empty.
Error (pop), cannot pop from an empty stack.
Error (peek), stack is empty.


---------------------------------------------------------------------
Stack Testing -> Doubles

Original List:  1.1 3.3 5.5 7.7 9.9 11.11 13.13 15.15 12.1

Sum is: 78.99

Reverse Original List:  12.1 15.15 13.13 11.11 9.9 7.7 5.5 3.3 1.1
Reverse, copy A:  12.1 15.15 13.13 11.11 9.9 7.7 5.5 3.3 1.1
Reverse, copy B (modified):  13.1 15.15 13.13 11.11 9.9 7.7 5.5 3.3 1.1

Stack empty errors...
Error (pop), cannot pop from an empty stack.
Error (peek), stack is empty.
Error (pop), cannot pop from an empty stack.
Error (peek), stack is empty.


---------------------------------------------------------------------
Stack Testing -> Strings

Original List:  enters green in a jumps big familiar dog hills
Reverse Original List:  hills dog familiar big jumps a in green enters
Reverse, copy A:  hills dog familiar big jumps a in green enters
Reverse, copy B (modified):  hello dog familiar big jumps a in green enters

Stack empty errors...
Error (pop), cannot pop from an empty stack.
Error (peek), stack is empty.
Error (pop), cannot pop from an empty stack.
Error (peek), stack is empty.


---------------------------------------------------------------------
Game Over, thank you for playing.
ed-vm%
ed-vm%
ed-vm% ./eval
Usage: ./eval -v <countValue>
ed-vm%
ed-vm% ./eval -c 1
-------------------------------------------------------------
Enter Postfix Expression:
12 6 4 + 2 17 - * 4 - 15 54 - / +

ed-vm% ./eval -x 1
Error, must specify '-c' count specifier.
ed-vm%
ed-vm%  ./eval -c #
Error, must provide expression prompt count.
ed-vm%
ed-vm% ./eval -c 2
-------------------------------------------------------------
Enter Postfix Expression:
223  342  443  +  *  211  17  /  -

Postfix expression: 223  342  443  +  *  211  17  /  -
 evaluated is: 175043

-------------------------------------------------------------
Enter Postfix Expression:
345 4a3 +
```

```
Error, invalid expression.
ed-vm%
ed-vm% ------------------------------------------------------------
Enter Postfix Expression:
12  6 4 +  2 17 -   * 4 - 15 54 - / +

Postfix expression: 12  6 4 +  2 17 -   * 4 - 15 54 - / +
 evaluated is: 15

------------------------------------------------------------
Enter Postfix Expression:
12  6 4 +  2 17 -   * 4 - 15 54 - / %
Error, invalid expression.

ed-vm%
ed-vm%
ed-vm%
```