Stat 598W: Homework 2

Due Saturday Feb 14 (sveinno@purdue.edu). Include a word/pdf document with your code and relevant output.

Problem 1

(a) Write an integeter Stack class with bounded capacity, similar to the one in Lecture 4. Add a method *peek* that returns the value of the first element without removing it from the stack, a method *reverse* that reverses the order of the elements in the stack (using only Stack objects), and a method *expand* that should be called when the Stack is full and allocate more memory to it. As before, include overloaded versions of the copy and assignment operators, as well as the << operator to print out the content of the stack. Test your implementation.

(b) Implement a simple Queue class using your Stack class (Hint: You need to maintain 2 stack objects, an in-stack and an out-stack). Besides a constructor and a destructor, it should at least contain the member functions enqueue, dequeue, reverse, and peek, as well as the same overloaded operators as the Stack class. Test your implementation.

(c) Use your Stack class to implement a simple calculator the evaluates expressions in Reverse Polish Notation (RPN). For simplicity, it only should accept the following operators: $+, -, *, /, \%$, and only deal with integers. Also assume the user enters the expression terminated with the \$ sign, e.g. $15 + 3 * 4 - 3$ should be entered as 15 3 4 $*$ + 3 $-$ \$ by the user. For more information on RPN, see Wikipedia.

Problem 2

Assume the node structure

```
struct node{
  int value;
  node* next;
};
```

and write code for AT LEAST three of the following:

(a) Check whether a list is a palindrome, i.e. reads the same from the front and the back. For example, 3, $1 \to 5 \to 7 \to 5 \to 1$, and $2 \to 1 \to 1 \to 2$, are examples of palindromes. The function should have signature

```
bool palindrome(node* list)
```

(b) For a given value of x, partition the list around that value, such that all nodes less than x come before all nodes greater than or equal to x. The function should have signature

```
void partition(node* &list)
```

(c) Remove duplicates from a linked lists. The function should have signature

```
void removeDuplicates(node* list)
```

(d) Weave the reverse of the list into the original. For example, if list is $1 \to 4 \to 2$, it becomes $1 \to 2 \to 4 \to 4 \to 2 \to 1$. If list is 3, it becomes $3 \to 3$. If list is $1 \to 3 \to 6 \to 10 \to 15$, it becomes $1 \to 15 \to 3 \to 10 \to 6 \to 6 \to 10 \to 3 \to 15 \to 1$. The function should have the signature

```
void braid(node* list)
```

(e) Given two sorted linked lists of potentially different lengths, merge the two into a single list (You shouldn't allocate any new memory, but instead use the nodes making up the two originals). The function should have signature

```
node* merge(node *list1, node* list2)
```

(f) Given a linked list, return the node at the beginning of a cycle (a cycle occurs when a node's next pointer points to an earlier node which creates a cycle). If no cycle exists, it should return a NULL pointer. The function should have signature

```
node* cycle(node* list)
```

Problem 3

Write a simple class for European options. It should handle both put and call options, contain a default constructor and a constructor that accepts values from the user (S, K, T, σ, r), a destructor, and functions that compute the price of the option, implied volatility, and Delta and Gamma (Greeks). To compute the implied volatility (given the options price) you should implement a simple root-finding algorithm such as the bisection method or Newton's method (you may find it useful to use pointers to functions, explained in simple terms on the following page: http://www.cplusplus.com/doc/tutorial/pointers/). Also overload the << operator to print out the options information.