CS 132 - Intro to Computer Science II - Spring 2005
Week #9 Lab Exercise and Homework #7

Week #9 Lab Exercise due: Wednesday, March 23rd, END of lab
HW #7 due: Wednesday, March 30th, beginning of lab

WEEK #9 LAB EXERCISE

1. INDIVIDUAL-answer, TEAM-verification exercise:

On a sheet of paper with your name on it, **individually** answer all of the questions below. Once you have written out your initial answers **by yourself**, **then** compare them with at least one other classmate's answers. If they differ, discuss why until you both agree on the answer (and change the answer on your paper accordingly if appropriate). Write down the name(s) of all those you conferred with on your paper, then put your name on the "Next:" list on the board to have your answers checked.

Consider the following sequence of stack and queue operations; I want you to **hand-execute** them, to see if you are comfortable with how stacks and queues work. (Assume that, as this begins, aStack is an empty stack of integers, and aQueue is an empty queue of integers.) Their implementation is unimportant, however DO note that they do correspond to the **stack** and **queue** pseudo-UML's discussed in lecture (and available from the course web page).

(Follow the directions carefully; the point here is to make sure you are comfortable with how a stack and queue "behave".)

```
stack<int>      aStack;
queue<int>      aQueue;
int             looky1, looky2, looky3;

aStack.push(2);
aStack.push(4);
aStack.push(6);
aStack.push(8);

/* POINT A: write POINT A, and draw aStack's CONTENTS at this point, clearly
   labeling the stack's TOP */

aStack.pop( );
aStack.push(10);
aStack.pop( );

/* POINT B: write POINT B, and draw aStack's CONTENTS at this point, clearly
   labeling the stack's TOP */

looky1 = aStack.get_top( );
aStack.pop( );

looky2 = aStack.get_top( );
aStack.pop( );

looky3 = aStack.get_top( );
aStack.pop( );

/* POINT C: write out the output of the following statements at this point: */

cout << "POINT C" << endl;
cout << "looky1: " << looky1 << endl;
cout << "looky2: " << looky2 << endl;
cout << "looky3: " << looky3 << endl;
```

```
aQueue.enqueue(2);
aQueue.enqueue(4);
aQueue.enqueue(6);
aQueue.enqueue(8);

/* POINT D: write POINT D, and draw aQueue's CONTENTS at this point, clearly
   labeling the queue's FRONT and REAR */

aQueue.dequeue( );
aQueue.enqueue(10);
aQueue.dequeue( );

/* POINT E: write POINT E, and draw aQueue's CONTENTS at this point, clearly
   labeling the queue's FRONT and REAR */

looky1 = aQueue.get_front( );
aQueue.dequeue( );

looky2 = aQueue.get_front( );
aQueue.dequeue( );

looky3 = aQueue.get_front( );
aQueue.dequeue( );


/* POINT F: write out the output of the following statements at this point: */
cout << "POINT F" << endl;
cout << "looky1: " << looky1 << endl;
cout << "looky2: " << looky2 << endl;
cout << "looky3: " << looky3 << endl;
```

You must turn in your **sheet of paper** with your name, your answers, and the name(s) of those you confirmed/discussed your answers with by the time that lab is over.

The above must be completed and checked before the end of lab.

HOMEWORK #7:

1. On the course web page, with this HW #7 handout, you'll find an implementation of the **node** class that does NOT include the linked-list toolkit functions.

In a new directory **132hw07**, transform this into a **template class** node, in files **node.h** and **node.template**. Write a testing main function **test_node.cpp** for your resulting class that involves tests of at least two nodes with different types of data, and redirect its ouput into **132hw07_1_out**; you'll be submitting **node.h**, **node.template**, **test_node.cpp**, and **132hw07_1_out**

2. We'll continue, now, with some light implementation of stacks.

Consider the stack pseudo-UML provided in lecture, and **stack.h** and **stack.template**, which you copied over for the lab exercise. These are a **dynamic array** implementation of that stack UML. Also consider your **node** template class from problem #1.

Using these as a basis, in directory **132hw07**, create a **stack.h** and a **stack.template** which are an implementation of this same **stack** UML using **linked lists** instead --- BUT it MUST use the node class from problem #1, and it cannot use the linked-list toolkit functions described in the text.

(So, the text's linked list implementation can help you in understanding what you need to do, but you cannot use the linked-list toolkit functions that they use in their version.)

Just to make sure we're clear on this: your code should be indented and documented using the same style currently seen in the provided **stack.h** and **stack.template**. You **may not change** the stack pseudo-UML provided.

Now, adapt the lab exercise's **ck_them.cpp** into **ck_stack.cpp**, that #include's your new version of stack.h and leaves out all queue-related statements. When you are satisfied with your program, run:

```
ck_stack > 132hw07_2_out
```

... and e-mail to me your resulting files **stack.h**, **stack.template**, **ck_stack.cpp**,and **132hw07_2_out**.

3. You will probably not be shocked that we are now going to proceed with some light implementation of queues.

Consider the queue pseudo-UML provided in lecture, and **queue.h** and **queue.template**, which you copied over for the lab exercise. These are a **linked** implementation of that queue UML. Also consider your **node** template class from problem #1.

Using these as a basis, within directory **132hw07**, create a **queue.h** and **queue.template**, which are an implementation of this same **queue** UML using **linked lists** instead --- BUT it MUST use the node class from problem #1, and it cannot use the linked-list toolkit functions described in the text.

Just to make sure we're clear on this: your code should be indented and documented using the same style currently seen in the provided **queue.h** and **queue.template**. You **may not change** the queue pseudo-UML provided.

Now, adapt the lab exercise's **ck_them.cpp** into **ck_queue.cpp**, that #include's your new version of queue.h and leaves out all stack-related statements. When you are satisfied with your program, run:

```
ck_queue > 132hw07_3_out
```

... and e-mail to me your resulting files **queue.h**, **queue.template**, **ck_queue.cpp**,and **132hw07_3_out**.

4. **NOTE:** You may use **either** the provided **stack.h/stack.template** OR your **stack.h/stack.template** from problem #1 for this problem. It is YOUR CHOICE. (Since both use the same UML, either should work! BUT if you use the provided one, you'll need to put #4 in a DIFFERENT directory...)

For this problem, **postfix notation** means that an operator follows its operands --- that is, to add 3 and 5, I'd say **3 5 +**. To multiply the sum of 3 and 5 and the difference of 7 and 4 (**(3+5)*(7-4)**), I'd write **3 5 + 7 4 - ***.

It turns out (and I believe the chapter mentions this, too) that stacks are quite lovely for evaluation of expressions written in postfix notation. If something is an operand, you push it on the stack. If something is an operator, you top-then-pop the first two items on the stack, perform the given operation on the two operands items, and then push the result back onto the stack. When you run out of expression, there should be but one item left on the stack: the result of the operation. (This is if the postfix notation is well-formed, of course.)

We're going to keep this simple:

* the only **operations** will be +, -, *, /. (The / is **integer** division, note.)

* the only **operands** will be the single digits 0 - 9.

* write a function **simplePost** that returns **bool** and has two parameters:
    * the first parameter is a **string** which contains a postfix expression

    * the second parameter is a **reference parameter**, the integer result of the given postfix expression.

    * **simplePost** returns true if the given postfix expression was "well-formed", and returns false if it is not. If it returns false, the second parameter will **not** have been changed/set by simplePost. If it returns true, then the

second parameter will have been set to be the result of evaluating the postfix expression.

* for your Examples and **test_simplePost.cpp**, make sure that you include at **least** the following:
  * at least the two examples given above
  * at least one ill-formed expression
  * other examples that you realize are important to more fully-test your function.
  * (and any others you just feel like including...)

* note that:
  * text p. 762 includes note of some useful character manipulation functions; I didn't have to #include <cctype> to use them, but do so if you need to. (I had #include'd <iostream> in my playing, which might include <cctype>.)
  * isdigit(aChar) returns true if char aChar is a digit character (0-9)
  * isspace(aChar) returns true if char aChar is a blank, tab, newline, or carriage return. (We're assuming that **simplePost** takes a single string as its first parameter, though, remember --- so newlines and carriage returns should not have to be worried about.)
  * quick-n-sleazy conversion of a digit character to its int equivalent can be performed by "subtracting" the digit character '0' from a digit character ---

```
const char DIGIT_ZERO = '0';

int intVersion;
char digitChar;

if (isdigit(digitChar))
{
    intVersion = digitChar - DIGIT_ZERO;
}
```

    Why, yes, a little helper function that does this conversion on-demand *would* be a lovely little helper function... (And feel free to use other helper functions that occur to you --- a few of these can make **simplePost** much "prettier" and clearer.)

When you are satisfied with your program, run:

`test_simplePost > 132hw07_4_out`

... and submit your resulting files **simplePost.cpp**, **test_simplePost.cpp**, **132hw07_4_out**, and any helper functions that you use.

5. **NOTE:** You may use **either** the provided **queue.h/queue.template** OR your **queue.h/queue.template** from problem #2 for this problem. It is YOUR CHOICE. (Since both use the same UML, either should work! BUT if you use the provided one, you'll need to put #5 in a DIFFERENT directory...)

Did you know that the set of palindromes is really a **language**? In computer science, a **language** is any set of strings that meet some given criteria --- I can say that the language of palindromes is all strings for which the in-lecture function **isPalindrome** returns true. And then, we can call **isPalindrome** a language-recognizer for that language, because it can recognize if a string is a member of that lanuage or not.

Well, we need a simple queue problem, and this provides a simpler one than the quite-excellent car-wash simulation already given in the chapter.

Consider the following strange little language: it consists of strings that contain exactly two lower-case **a**'s, each of which come at the beginning of identical strings (the rest of which contain NO **a**'s). That is, the following are all members of this language:

"abcdabcd", "aa", "arkark", "avonavon", "a blonde doga blond dog"

These are not:

"" (the empty string), **"abca"**, **"abac"**, **"ababc"**, **"alphaalpha"** (because there can only be exactly 2 a's in strings of this language), **"baba"** (because it doesn't start with **a**)

Write a function **isMember** that uses a **queue** appropriately to determine if the string passed to it is a member of this strange little language or not. It should have only that one parameter, and should return true if the string is a member of the language, and return false otherwise. It may only make **one** pass, total, through the passed string (although it may "leave" early if it has determined that the string is indeed not a member of the language).

Write a main in **test_isMember.cpp** to run your examples and test your function.

When you are satisfied with your program, run:

```
test_isMember > 132hw07_5_out
```

... and submit your resulting files **isMember.cpp**, **test_isMember.cpp**, and **132hw07_5_out**,

And, when you are satisfied with all of the above, make sure that you have submitted (using ~st10/132submit) your versions of all of the following files:

node.h, node.template, test_node.cpp, 132hw07_1_out
stack.h, stack.template, ck_stack.cpp, 132hw7_2_out
queue.h, queue.template, ck_queue.cpp, 132hw07_3_out
simplePost.cpp, test_simplePost.cpp, 132hw07_4_out
isMember.cpp, test_isMember.cpp, 132hw07_5_out