New Mexico Tech
Department of Computer Science

# Programming Assignment 2

CS122 Algorithms and Data Structures

**Due 11:00AM, Wednesday, October 13th, 2004**

**Objectives:**

1. Gain familiarity with Object-Oriented programming and design by implementing commonly-used data structures as C++ style classes.

2. Learn about some of the most common data structures used in programming, what their applications are, and why they are useful.

**Specifications:**

1. For the first part of this assignment, you are required to create a "Node" class which is suitable for being used to implement a linked-list. This class is required to meet the following specifications:

    1a. There must be at least two private data members, the first is a pointer to the next node in the list (i.e. "Node* nextNode;") and the second may be an int, float, double, long, or string, whichever you prefer (or, see the "Extensions" section if you'd like to create something a little more useful). This member is that actual data that a single Node in the list is meant to store.

    1b. There must be, at minimum, public member functions that get and set both of these data members. You may add additional member functions if you wish, provided that they make sense in the context of a Node (for example, a print() function that displays the Node's data value would be a good idea, a sqrt(int n) function that computes the square root of the given integer would not).

    1c. Appropriate constructors need to be defined as well. Specifically, there should be the constructors (where <data> indicates the datatype of your choosing) "Node()", "Node(<data> val)", and "Node (Node* next, <data> val)". The first two constructors should

set the next node pointer to NULL, and the first constructor should also set your data value to something that is appropriate for the datatype you've chosen (for example, "" if using strings, -1 if using int's, and so on).

2. Once your node class is completed, use it to implement a "List" class that meets the following specifications:

   2a. There must again be at least two private data members, the first being a pointer to the first Node in the List, the second being an integer (or size_t if you prefer) which keeps track of the number of Nodes in the List. You may also wish to keep a pointer to the last Node so that append operations are more efficient (and also so that the code becomes simpler).

   2b. The class is required to contain implementations of the following public member functions:

```
//appends the specified node to the end of the list
void append(Node n)

//inserts a Node into the list at the specified position
//returns 0 if no errors occur, -1 if the desired position is outside
//    of the bounds of the list
//counting starts at 0, which corresponds to the first Node in the
//    list
int insert(Node n, int location)

//returns the Node at the specified location in the list, or NULL
//    if the index specified is invalid
//again, counting begins a 0
Node* getNodeAtIndex(int index)

//returns the number of nodes currently in the list
int getSize()

//removes the node at the specified index, and updates the pointers
//    of the nodes before and after the deleted node so that the list
//    remains intact
//returns 0 if no errors occur, -1 if the specified index is outside the
//    bounds of the list
int deleteNodeAtIndex(int index)

//prints the value stored in all Nodes in the list, beginning at
//    Node 0 and terminating at the last Node in the list
```

**void print()**

Note that there should not be public methods that allow the modification of the class's private data members in this case (i.e. there should not be a setSize() function, or a setFirstNode() function). Additionally, there should not be a getFirstNode() function.

2c. Your default constructor for this class should set the first node pointer to NULL, and should initialize size to 0. No other constructors should be defined.

2d. The List class is responsible for garbage collection, so it must include a destructor that deallocates all the Nodes in the List, and the deleteNodeAtIndex() function must deallocate the node that is being removed from the list.

3. Using your List class, implement a "Queue" class and a "Stack" class. These classes must follow the typical rules associated with stacks and queues (i.e. a stack is a "last in, first out" structure, meaning that the last element added to the stack is the first thing that comes off the stack, and a queue is a "first in, first out" data structure, meaning that the first element added to the queue is the first thing that comes off of the queue). Specifically:

    3a. The Stack class should have push() and pop() methods. The push method should take as a parameter a variable of whatever type you chose in step 1a. The pop method should return a variable of the same type, and remove the Node at the top of the Stack. There should also be a size() function that returns the current number of elements contained by the Stack, and a print() function that prints the current contents of the stack.

    3b. The Queue class should have enqueue() and dequeue() methods. The enqueue() method should take as a parameter a variable of whatever type you chose in step 1a. The dequeue method should return a variable of the same type, and remove the Node at the beginning of the Queue. There should also be a size() function that returns the current number of elements in the queue, and a print() function that prints the current contents of the queue.

4. Write a simple driver program that tests the Stack, Queue, and List objects by adding several values to each one. In the case of the List, be sure that your

driver invokes the insert(), deleteNodeAtIndex(), and getNodeAtIndex() functions in order to ensure that these functions are working properly. It does not matter how you generate/gather your test data, you may hard-code it into your driver program, or you can get it from user-input if you wish. It does matter that your driver program invokes all the public methods of every class being tested **at least** a few times in order to ensure that they work properly, and also that the driver program generates sufficient output to make it clear that everything is indeed working (for example, add several entries to a List, and then invoke print(), and then perform some deletions and some insertions, and invoke print() again. At every step along the way, the program should generate **clear, understandable** output describing what is being done to which data structure.

5. As always, coding and documentation guidelines must be followed, and a README file must be submitted along with your source code. At a minimum, the README file must include compilation instructions, a description of what your program is and how to use it, a discussion of any design issues you ran into while implementing this project, and a description of how the List, Stack, and Queue objects work. If there are any parts of the project that don't work quite right, this should be noted in the README file as well. Any additional information relevant to this project is also fair game for inclusion in the README file if you wish to include it.

6. Note that the user-friendliness of your program, which includes things like displaying clear, concise instructions to the user detailing what the program is and how it should be used when your program starts, and the formatting applied to input prompts and output data, **is** important and **does** count. Having an impossible to figure out interface, or nearly impossible to decipher output data, will result in points being deducted. In the case of this project, it is entirely possible that your program will never display a prompt to the user and read input, but the program should still print instructions describing what it is, and output should still be well formatted.

**Submission Guidelines:**

Use whatever archiving software you prefer to archive your source and README files together. The archive should be called '<your name>.Homework2.<ext>'. You may include a directory structure in your archive if desired, but it is not required. E-mail your archive to aroth@nmt.edu with the subject line of "[CS 122] Programming Assignment 2" by the due-date posted on the website. In the body of your e-mail, be sure to include your name and student-id number

**Grading Criteria:**

A fully working program with correct class implementations is worth 70 points.

A useful and descriptive README file is worth 15 points.

Adhering to coding style and documentation guidelines is worth 15 points.

**Extensions:**

For a maximum of an additional 15 bonus points, complete the following enhancement to this project:

Use templates when implementing your classes so that you may have a Node/List/Stack/Queue that contains any data-type. If your implementations of these classes place restrictions on the types of data that can be used (for example, maybe one of your functions for some reason does '+=' on a data member…this would mean that only data-types with a '+=' operator defined can be used with your template classes), be sure to note it in your class documentation and/or README file. Modify your driver program so that it tests all the classes with a few different data-types (or maybe add a menu that lets the user select what data-type to use when running the tests). If you attempt/complete this extension, be sure to make note if it in your README file.