Implementing a Task List as a Priority Queue

Programming Assignment# 3

- **Objective:** Learn about the Priority Queue ADT (Abstract Data Type). Implement Priority Queue ADT using a traditional linked list. Design an object oriented implementation in C++.
- **Points**: 50
- **Team-based**? You have the option to either work individually or work in pair. It is highly suggested that you work in pair.
- **Due**: Friday, 2015-May-08 @ 11pm
- **What to hand in:** Refer to the section on "what to hand in" of this document.
- **Note:** Late assignment submitted within 24 hrs. from the deadline shall be penalized 10%. No late assignment shall be accepted after 24 hours from the deadline.

You may lose points if you don't pay special attention to the following in your program:

- *Your code should be properly indented. Separate various sections of your code with a blank line, like variable declaration, taking input, computation, printing, etc.*
- *Use appropriate data types, e.g., use integer data types where there will be no need to store floating point numbers.*
- *Avoid magic numbers, i.e., use constants instead of numeric/string literals and write a short comment that explains its purpose (it is okay to have string literals in cout for displaying messages or numeric/string literals when initializing variables).*

  E.g., `const double PI = 3.14; //value of pi`

  `area = PI * radius * radius;` *instead of* `area = 3.14 * radius * radius;`
- *Variables and constants should be named according to their purpose. E.g., if you need to store the number of cars then* `numCars` *is more descriptive then saying* `x`*.*
- *Pay attention to naming convention for variables and constants. Constants are as a practice declared in upper case; multiple words are joined by underscore. Variables are declared in lower case, multiple words are joined by underscore or first letter is capitalized for each word except for the first word.*
- ***Document your code by writing comments where appropriate to explain what the various pieces of your code might do. Refer "additional notes" section of this document for more details.***

- *Each line should not have too much code, as a matter of practice any line longer than 80 columns should be broken down into multiple lines of code.*

Introduction

A priority queue is a queue where items are associated with a priority value, and are removed based on priority: the item with highest priority is removed first (in contrast to standard queue, where FIFO order is enforced). Priority could be determined by integers where 0 is the highest priority (most favored) and higher numbers have lower priority (less favored). This style of queue is used when some items are deemed more important than others and items have to wait in line. Phone calls to 911 for instance are more important than routine calls between users of telephone network, and should be given priority in the phone network if too many calls are coming through a phone service at a time. A business might decide that internet traffic associated with online meeting or credit card processing software is more important than web surfing when network demand is high. A printer might prioritize short documents first so fewer people wait around a high-traffic printer.

One of the ways to become more productive with your everyday tasks is to be able to prioritize them. In this assignment, you will implement a Priority Queue ADT as a variation on a singly linked queue with a head node. Nodes in the queue will have three fields: task, priority, and next. You will insert tasks by priority so higher priority tasks are at the front of the queue and lower priority tasks are at the back of the queue. If there is already a task with the same priority in the queue, the new task goes after it, this will ensure removal of equal priority tasks follow the FIFO ordering.

Getting started

Download the file Assignment3.cpp from Titanium, in this file you will find the code for your Priority Queue class with some of the function prototypes that you will implement. Download the file input.txt, your program should read in the tasks listed in this file and create a priority queue. Note that there are a lot of details to this assignment; too many to simply start programming without any forethought. For your own benefit you would want to plan your solution in pseudo code before you start to write your C++ code.

Operations to implement

(Note: To improve readability of your C++ code, you should define your member functions outside the class using scope resolution operator.)

❖ You will be coding your own Priority Queue class implemented as a singly linked list. Do **NOT** use the STL Priority Queue or Queue classes. Below are the functions to write, additionally you are free to write helper functions that you may need.

- `~PriorityQueue();` destructor to delete all tasks from the queue.
- `void goNext(NodeType * & curr) const;` moves the pointer to point to the next task in the queue.
- `bool isEmpty() const;` checks for the queue being empty.
- `int size() const;` returns the number of tasks in the queue.
- `void insert(string taskName, int priority);` This function takes two parameters. One is the priority of the task being added and the other is the task itself. The function needs to item to the queue so that it is:
    - the last task with its priority
    - after any tasks with a higher priority (lower number)
    - before any tasks with a lower priority (higher number)
- `void min() const;` prints the task and its priority (the one with the highest priority).
- `void removeMin();` removes the highest priority task from the queue. If the queue is empty, it simply prints a message saying the queue is empty.

You should include in your assignment report screen shots for below input and output. First read the tasks listed from input.txt (download from Titanium) and insert each task in your priority queue. Next remove each task in order of priority and print it out as below:

```
Removing all tasks in order of priority and printing...
0 Fix broken sink
1 Order cleaning supplies
2 Shampoo carpets
2 Replace light bulb
3 Pet the dog
4 Take a nap
```

```
5 Empty trash
5 Clean coffee maker
5 Water plants
6 Remove pencil sharpener shavings
```

What to hand in

A) Submit your C++ source code (cpp file). If you choose to work in pair, **both students must individually** upload the cpp file via Titanium. Write *yours and your partner's name, section number, and "Assignment 3"* at the top of your ***cpp*** file as comments.

B) Submit a written report electronically as a PDF or MS Word doc file through Titanium. Again, if you worked in pair then each student uploads the report individually. Write *yours and your partner's name, section number, and "Assignment 3" on the first page of your report*. **Your report should include:**

1. Give pseudo code for each of those functions as previously asked in this document. (Refer "additional notes" section of this document for tips on how to write good pseudo code).
2. Screen shots of your input and output for each of those functions as previously asked in this document. **Note that typed or hand written input/output will not be considered. You must provide screen shots of your actual program run.**
3. Do not include any C++ code in the report. It is to be uploaded separately as a .cpp file as asked in A).

Additional Notes

- **Pseudo code**

  Word of caution: It may be tempting to jump into C++ coding and later convert it to some sort of pseudo code. This is contrary to the purpose of writing pseudo code. If you find yourself in such a situation then you are missing out on the benefits that result from detailed planning.

  The idea behind pseudo code is that it is easier for people to understand than conventional programming language code, and that it is an efficient and environment-independent description of the key principles of your solution.

  Another advantage is that pseudo code is a useful tool for planning your program design. It allows you to sketch out the structure of your program and perform stepwise refinement before the actual coding takes place.

  Below is a sample pseudo code for the game of fizz buzz. It contains some programming language elements augmented with high level English description. The goal is to have enough useful details while leaving out elements not essential for human understanding of your program, like, variable declaration and system specific code.

```
fizzbuzz()
Input: none
Return: none
   for i <- 1 to 100 do
       print_number <- true
       if i is divisible by 3
           print "Fizz"
           print_number <- false
       if i is divisible by 5
           print "Buzz"
           print_number <- false
       if print_number == true
           print i
       print a newline
```

  Pseudo code style that you follow for your program does not have to be exactly as above. You may choose the style described in your textbook. In addition, there are numerous resources available on the internet that describe various popular styles, feel free to look up and choose one.

How detailed? Check for balance. If the pseudo code is hard for a person to read (too close to a particular programming language's syntax) or difficult to translate into working code, i.e., too vague, (or worse yet, both!), then something is wrong with the level of detail you have chosen to use.

- **Code Writing:** A bad approach is to write entire code all inside one function and later perform a massive refactoring, i.e., split your code into multiple cohesive functions. It leads to increased effort in integration, testing, and debugging. Good approach is to plan ahead and start by writing small cohesive functions (up to a maximum of 30 lines or what might easily fit on one screen) with reasonable refactoring of code as you go along.

- **Choice of loops:** Choose appropriate loop type (for, while or do while) in a way that it makes your program logic simpler and code more readable.

- **Testing:** You should test each function as you code it. Use of big bang approach to testing, i.e., postponing to test after you have written all functions would make it very difficult to isolate errors in your code.

- **Commenting your code**
  1. **File comments**: Start your cpp file with a description of your program, names of authors, date authored, and version. Add other comments as asked in the "what to hand in" section of this document.

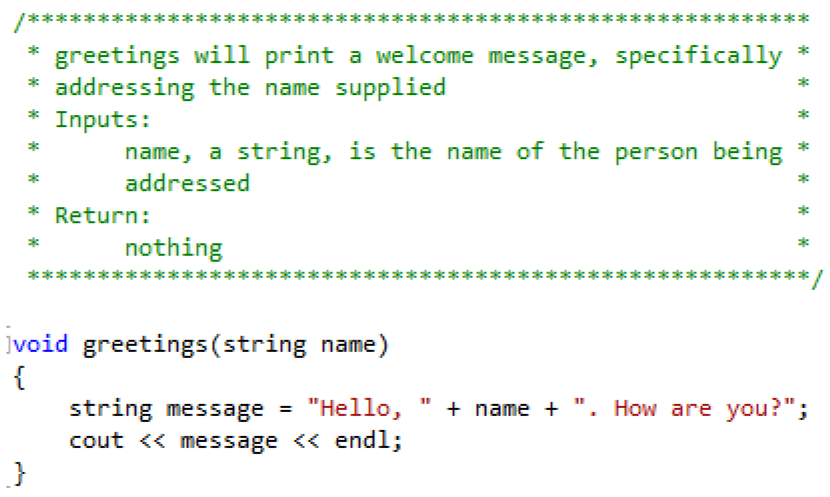
  2. **Commenting function header:** Right above the header of each of your function you should have a comment. The comment will have three pieces to it:

     Describe what the function accomplishes.

     Inputs: Indicates what the required parameters are for the function. "nothing" for no parameters.

     Return: Clearly states what the function computes and returns. "nothing" for void functions.

Here is an example:



```
/*********************************************************
 * greetings will print a welcome message, specifically *
 * addressing the name supplied                          *
 * Inputs:                                               *
 *      name, a string, is the name of the person being *
 *      addressed                                        *
 * Return:                                               *
 *      nothing                                          *
 *********************************************************/

void greetings(string name)
{
    string message = "Hello, " + name + ". How are you?";
    cout << message << endl;
}
```

3. **Section comments:** Add comments for each section of your code within your function like variable declaration, taking input, computation, display output, etc.

4. **Explanatory comments** should be added for tricky or complicated or important code blocks.

5. **Line comments:** Lines that are non-obvious should get a comment at the end of the line. These end-of-line comments should be separated from the code by 2 spaces. Break long comments into multiple lines.