DM2126 Assignment 2

A framework has already been provided for you, it consists of 4 header and 3 cpp files. You are to use the framework to work on your submission.

main.cpp - Do not modify the function prototypes. You can modify the parameters for the values for your own testing purposes. In your submission, this file need not be reverted to the original form. In fact, it is encouraged to leave your test cases in the code (even if it is commented out) so that the tutors can have a rough idea of how you tested your code.

ListQueueStack.cpp - Implement the 3 data structures as outlined in this brief.

mission.cpp - Fill in the function definitions so that they fulfil the assignment requirements.

There are 5 exercises in total. Finish all of them. Even if you do not have solutions to all possible test cases, it is good to have a sub-optimal solution where some of the test cases will pass the test. Remember to comment your code.

Code will be compiled with Visual Studio 2013.

Late submission

Late submissions will be penalized at 5 points per day.

Due Date

Due date is on the 14-Feb-2016 2359hrs (11:59pm) Any submissions after this time will be considered as a late submission. Blackboard has this tendency to go down at the most inopportune time, so submit early and submit often. Only the latest submission will be graded.

Tips

Write your test cases or code in main.cpp.

Remember to remove whatever debug code (e.g. cout) in `mission.cpp` and `ListQueueStack.cpp` before submission, recompile and test again.

Be sure to compile in both **Debug** and **Release** modes and check that your code still works.

Warnings will be treated as errors, you have to resolve the warnings.

Download your submitted assignment from Blackboard and check again.

Seek help if you are stuck.

Don’t submit any work that is not produced with your own effort.

For parts 1 to 3, you are supposed to write your code in `ListQueueStack.cpp` file. The respective header files are given to you, and you are not supposed to modify the header files in any way. Doing so will make your code incompatible.

The node structure is given to you, you are supposed to use this node struct in the implementation of the following data structures.

Do not forget about memory management. Memory leaks will be penalized.

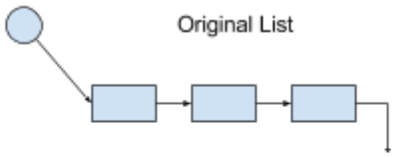
Section A

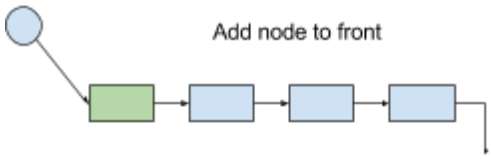
Part 1 - Linked List

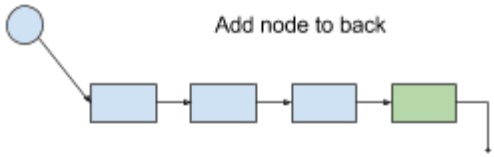
Implement the Linked List according to the LinkedList.h file given.

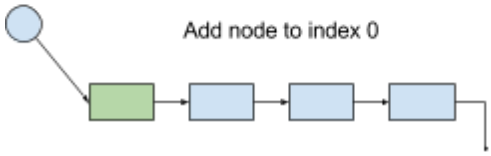
1. `head`
    a. This is the node pointer that points to the head of the linked list.
2. `push_front(int data)`
    a. A new node containing this value will be pushed to the front of the linked list.
3. `push_back(int data)`
    a. A new node containing this value will be pushed to the end of the linked list.
4. `pop_front()`
    a. The first node will be removed from the linked list and the value in that node returned from this call. If there are no nodes in the linked list, return a 0.
5. `pop_back()`
    a. The last node will be removed from the linked list and the value in that node will be returned from this call. If there are no nodes in the linked list, return a 0.
6. `insert_at(int pos, int data)`
    a. Insert a node with this value at this position. If the position is not available, insert the node at the end of the linked list. Negative positions should be treated as 0.
7. `pop_at(int pos)`
    a. Pop the data contained in the node at this position, and delete the node. If the location is not available, do not delete any nodes and return a 0. Negative positions should be treated as position 0.
8. `size()`
    a. Returns the size of the linked list. i.e. How many nodes are there in your linked list.

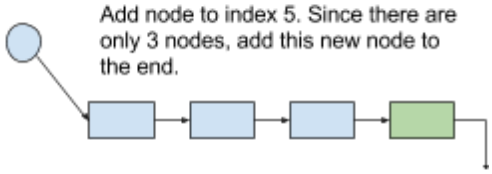
Graphical representation of the operations on the following page.


Original List


Add node to front


Add node to back


Add node to index 0


Add node to index 5. Since there are only 3 nodes, add this new node to the end.


Po node at index 1

Part 2 - Queue

Implement the Queue according to the `Queue.h` file given.

- `front`
  - This is the node pointer that points to the front of the queue.
- `back`
  - This is the node pointer that points to the back of the queue.
- `enqueue(int data)`
  - Put this value in a node and insert it at the end of the queue.
- `dequeue()`
  - Remove the node from the front of the queue, and return its value. Return 0 if there are no nodes in the queue.
- `size()`
  - Returns the size of the queue. i.e. How many nodes are there in the queue.

Part 3 - Stack

Implement the Stack according to the `Stack.h` file given.

- `top`
  - This is the top node of the stack.
- `push(int data)`
  - Push a node with this value to the top of the stack.
- `pop()`
  - Delete the topmost node and return its value. If there are no nodes, return 0.
- `size()`
  - Returns the size of the stack. i.e. How many nodes are there in the stack.

Section B

For this section, you are allowed to use the STL containers. (vector, list, stack, queue or map) Your code should be in `mission.cpp`.

Part 4 - Brackets

A twist on the brackets assignment that you had done earlier.

In this case, there is not 1, not 2, not 3, but 4 different kinds of brackets for you to manage.

You will be given a string containing brackets, and you should determine if the brackets are matched.

Definition

Method:	Brackets
Parameter:	const string&
Returns:	bool
Method Signature:	`bool Brackets(const string& input)`

Constraints

There will only be 8 types of characters in the input.
The string will be null terminated.
The valid characters are `(){}[]<>`

Examples

```
Brackets("()") returns true
Brackets("(<)>") returns false, each open bracket should be closed with its type
Brackets("(<))") returns false, unmatched bracket type
Brackets("{<[()]>}") returns true
Brackets("()()[]<>{{[{}]}}") returns true
```

Hints

Every open bracket should be matched with a matching closed brackets.
Can you exploit any data structure in this case?

Part 5 - Query machine

Given a list of numbers as the data, and a list of numbers as the queries, return the list of how many times each query has appeared in the data.

Definition

Method:	QueryMachine
Parameter:	vector<int>&, vector<int>&, vector<int>&
Returns:	void
Method Signature:	`void QueryMachine(vector<int>& data, vector<int>& queries, vector<int>& results)`

Constraints

The numbers given in data, and queries will be a 32 bit number between -2,147,483,648 to 2,147,483,647
The results vector can be assumed to be empty.
The populated results vector should be of the same length as the queries vector.

Examples

```
data = {1, 2, 3, 4, 3, 2, 2, 6}

QueryMachine(data, {1, 2, 3}, results)
returns {1, 3, 2}
1 occurred 1 time
2 occurred 3 times
3 occurred 2 times

QueryMachine(data, {3, 4, 5}, results)
returns {2, 1, 0}
3 occurred 2 times
4 occurred 1 time
5 occurred 0 times


QueryMachine(data, {6, 5, 7}, results)
returns {1, 0, 0}
6 occurred 1 time
5 occurred 0 times
7 occurred 0 times
```

Hints

Can you exploit any data structure in this case?
Is your initial solution optimized?

Grading

A basic test will be provided at a later date. There will be additional coding style checks conducted on your submission.

It is expected that you adhere to good coding practices. You may be penalized for poor coding practices.

You should read **https://google.github.io/styleguide/cppguide.html**

Extra Credit - Version control
If you are using version control for your project. Good for you.
Get a log of your Git commits and attach to the submission with the following command.

```
git log --pretty=format:'%cn : %cd : %s' --graph > gitlog.log
```

Submission Guide

Due date 14 Feb 2016 Sunday 11:59hrs

- Submit in a single zip file
- Use the standard naming convention
  - `Assignment01_<AdminNo>_<Name>.zip` e.g. `Assignment01_141592Z_JustinBieber.zip`
  - Uppercase for the Admin number checksum. E.g. 141592Z
  - Uppercase for the first letter of your name. e.g. JustinBieber
  - No spaces in the filename.
- Only `*.h, *.cpp, *.sln, *.vcxproj, *.vcxproj.filters` files are allowed.
- No `*.suo, *.sdf, *.exe, *.pdb, *.idb, *.ilk`, etc.
- No Debug, Release, ipch folders.
- Remember to test your submission to see if it can be opened by Visual Studio 2013 and build properly.
- The zip file should contain the solution file at the root level.
- Non compliance may result in deduction of marks.