AST20105 Data Structures & Algorithms

Lab 5 – Array and Linked List

A. Submission Details

In this lab, you are required to submit **ONE** C++ program to solve the given problem shown in the section "Exercise". To make the program implementation easier, you are suggested to use Visual Studio 2012 or later version instead of doing it using pencil and paper. After you have completed the task, submit your program files (MySinglyList.cpp for this lab) via Canvas by **Oct 21, 2016**. For details, please refer to the following:

You are reminded to double-check your solutions to verify everything is correct before submission. You are also required to put your name, student ID, and lab number at the beginning of your source files.

Important: You are only able to submit your work once.

B. Objective

The objective of this lab is to give you a revision on a couple of basic data structures, namely array and singly linked list introduced last week. Please note that all these concepts are not only useful for the concepts to be introduced later in the course, but also for your further study. Apart from this, you will be asked to work on a lab question with the help of the provided singly linked list source code written in C++. The following gives greater details about this lab.

Part 1: A quick review of concepts related to the lab topics.

1. Array: Features and its pros and cons
2. Singly Linked List: Features and its pros and cons

Part 2: Lab question about further extension of Singly Linked List.

C. Review of concepts

1. Array: Features and its Pros and Cons

Recall, an array can be used to store a list of values / objects and it has the following features:

- Size is fixed once it is created no matter you create it using new or without new in C++
- Cells in array are in contiguous positions in the memory

Pros:

i. Can efficiently return the value of element at certain position using subscript operator [] in C++
ii. Can search for value / object with binary search if the list of values are sorted.

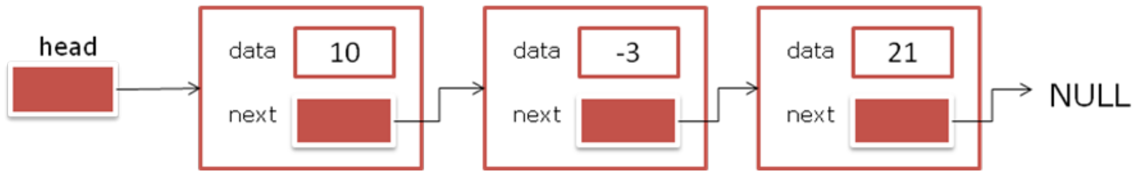
Cons:

i. Memory space is wasted if it cannot be fully utilized
ii. Not possible to expand or shrink the size during run time
iii. Insertion and deletion of element is slow. Since all the elements in the list may need to be moved.

2. Singly Linked List: Features and its Pros and Cons

A Singly Linked List is another way of storing a list of values and its has the following features:

- Node is the basic element of a linked list with data and a variable storing the address to next node
- Unlike array, nodes are not stored contiguously in the memory
- The first node is pointed by head, a pointer variable stores its address
- The last node stores address NULL to indicate no more node



Node class:

```
class Node
{
  public:                      // Making all the data members
    // data                    // as public is very exceptional.
    double data;               // Since this facilitates rapid
    // pointer to next node    // access of data members
    Node* next;
};
```

Singly Linked List class:

```
class SinglyList
{
   private:
     Node* head; // a pointer to the first node in the list
   public:
      SinglyList();    // constructor
     ~SinglyList();    // destructor
     // isEmpty determines whether the list is empty or not
     bool isEmpty();
     // insertNode inserts a new node at position "index"
     Node* insertNode(int index, double x);
     // findNode finds the position of the node with a given value
     int findNode(double x);
     // deleteNode deletes a node with a given value
     int deleteNode(double x);
     // displayList prints all the nodes in the list
     void displayList() const;
};
```

Pros:

i. No fixed size and it grows one node at a time. So it only uses as much space as in needed for the objects in the list
ii. The insertion and deletion of a node is relatively easier than array

Cons:

i. Coding is more complex than array
ii. With space overhead, since each node stores the address of the next node
iii. Not easy to access the data in a node at certain position. We need go through all the nodes from the beginning to that object

D. Exercise

1. Create a New Project and give your project the name Lab5.

2. Download the given source files Node.h, SinglyList.h, SinglyList.cpp, MySinglyList.h and main.cpp from Canvas and save them to your Lab5 folder. Also import it to your project.

3. The given MySinglyList.h for a class as indicated in the UML class diagram below.

MySinglyList
- sLL : SinglyList
+ MySinglyList() + checkSortedOrder() : bool + addValue(value: double) : void + insertNodeWith888(value: double) : void + moveLastNodeToBegin() : Node* + displayList() : void

4. Add a source file to your project, called MySinglyList.cpp and implement all the member functions according to the following description:

   i. sLL is an object of type **`SinglyList`**.

   ii. MySinglyList() is a default constructor to initialize sLL by inserting a number of nodes into it as follows:
      1. Insert value 3.8 to position 0. (Also initial *head* as the first node of the linked list)
      2. Insert value 4.6 to position 1.
      3. Insert value 4.6 to position 2.
      4. Insert value 0.2 to position 3.
      5. Insert value 9.1 to position 4

      After this, call local displayList function to display the list.

   iii. checkSortedOrder() is a constant member function that checks whether the data in the sLL are in sorted order (increasing order). If so, it returns true, false otherwise.

   iv. addValue() is a member function that adds a given value to every node data in sLL.

   v. insertNodeWith888() is a member function that inserts a new node containing the value 888 after every node in sLL that contains the given value.

   vi. moveLastNodeToBegin() is a member function that returns a pointer to a linked list which is the result of moving the last node in sLL to the begin of sLL. For instance, if sLL contains 2.1, 3.2, 5.3, 1.4, 7.5, then when the function is called, a pointer to the list that contains 7.5, 2.1, 3.2, 5.3, and 1.4 should be returned. If sLL contains zero or one node, the function returns a pointer to the list without having to make any change.

   vii. displayList() is a member function that display all node in sLL.

5. Use the provided main.cpp to test your program and you should get the following output if your implementation of MySinglyList is correct.

   Output of the program:

```
---------------------
Construction of mySLL
---------------------
3.8
4.6
4.6
0.2
9.1
Number of nodes in the list: 5

------------
addValue(10)
------------
13.8
14.6
14.6
10.2
19.1
Number of nodes in the list: 5

------------------
checkSortedOrder()
------------------
The list is in unsorted.

-------------------------
Move Last Node To Begin()
-------------------------
19.1
13.8
14.6
14.6
10.2
Number of nodes in the list: 5


-----------------------
insertNodeWith888(14.6)
-----------------------
19.1
13.8
14.6
888
14.6
888
10.2
Number of nodes in the list: 7
```

Important:

In order to test your ability to program in linked list, you ARE ONLY ALLOWED to use *isEmpty()*, *setHead(…)*, *getHead()* functions provided in SinglyList class. There are new functions that require you going through basic operations of linked list. The SinglyList could offers you a reference of how pointers are passed.

You will receive ZERO points if you adopt functions other than mentioned above in SinglyList.

Useful code:

Note: #include <climits> should be put for the following piece of code to work

```
bool compareDoubleEqual(double val1, double val2)
{
    if(abs(val1 - val2)<2.0*std::numeric_limits<double>::epsilon())
        return true;
    else
        return false;
}
```

Program Submission Checklist

Before submitting your work, please check the following items to see you have done a decent job.

Items to be checked	☑ / ☒
1. Did I put my name and student ID at the beginning of all the source files?	☐
2. Did I put reasonable amount of comments to describe my program?	☐
3. Are they all in .cpp extension and named according to the specification?	☐
4. Have I checked that all the submitted code are compliable and run without any errors?	☐
5. Did I zip my source files using Winzip / zip provided by Microsoft Windows? Also, did I check the zip file and see if it could be opened? ***(Only applicable if the work has to be submitted in zip format.)***	☐
6. Did I submit my lab assignment to Canvas?	☐

Marking scheme:

Graded items	Weighting
1. Correctness of program	30%
2. Readability	15%
3. Re-usability	10%
4. Documentation	15%
5. Delivery	20%
6. Efficiency	10%
Total:	100%

Grading rubric:

Area of Assessment	**Exceptional**	**Acceptable**	**Amateur**	**Unsatisfactory**
Correctness	The program works and meets all the specifications.	The program works and produces the correct results and displays them correctly. It also meets most of the other specifications.	The program produces correct results but does not display them correctly.	The program is producing incorrect results.
Readability	The code is exceptionally well organized and very easy to follow.	The code is fairly easy to read.	The code is readable only by someone who knows what it is supposed to be doing.	The code is poorly organized and very difficult to read.
Re-usability	The code could be re-used as a whole or each routine could be re-used.	Most of the code could be re-used in other programs.	Some parts of the code could be re-used in other programs.	The code is not organized for re-usability.
Documentation	The documentation is well written and clearly explains what the code is accomplishing and how.	The document consists of embedded comment and some simple header documentation that is somewhat useful in understanding the code.	The documentation is simply comments embedded in the code with some simple header comments separating routines.	The documentation is simply comments embedded in the code and does not help the reader understand the code.
Delivery	The program was submitted on time.	The program was submitted within 1 day of the due date.	The code was submitted within 2 days of the due date.	The code was submitted more than 2 days overdue.
Efficiency	The code is extremely efficient without sacrificing readability and understanding.	The code is fairly efficient without sacrificing readability and understanding.	The code is brute force and un-necessarily long.	The code is huge and appears to be patched together.

Adopted from California State University (Long Beach)

http://www.csulb.edu/colleges/coe/cecs/views/programs/undergrad/grade_prog.shtml

-End-