

lab 08: Linked Lists

Instructions: For this lab we will complete our implementation of a Singly Linked List and a Doubly Linked List

Implement the following classes and functions:

```
1 /* SLL = Singly Linked List */
2 template<class T>
3 class SLList {
4     private:
5         /* Class exercise to fill in. */
6     public:
7         /* Empty constructor shall create an empty Linked List! */
8         SLList();
9
10        /* Do a deep copy of sll into the this.
11         * Note: This one uses a reference to a Singly Linked List!
12         */
13        SLList(const SLList<T> &sll);
14
15        /* Destructor shall free up memory */
16        ~SLList();
17
18        /* Return the current length of the Singly Linked List */
19        int getLength() const;
20
21        /* Insert at the end of the list.*/
22        bool append(const T &val);
23
24        /* Insert val at position pos.
25         * Return true if successful (it can be placed.)
26         * Otherwise return false.
27         */
28        bool insert(const int pos, const T &val);
29
30        /* Print out the Singly Linked List */
31        void print() const;
32
33        /* Remove the first instance of val
34         * Return true if found and removed.
35         * Otherwise return false.
36         */
37        bool remove(const T &val);
38
39        /* Retrieves the element at position pos */
40        T& operator[](const int pos);
```

```

41
42     /* Returns if the two lists contain the same elements in the
43      * same order.
44      */
45     bool operator==(const SLList<T> &list) const;
46 };
47
48
49 /* DLL = Doubly Linked List */
50 template<class T>
51 class DLList {
52     private:
53         /* Class exercise to fill in. */
54     public:
55         /* Empty constructor shall create an empty Linked List! */
56         DLList();
57
58         /* Do a deep copy of dll into the this.
59          * Note: This one uses a reference to a Singly Linked List!
60          */
61         DLList(const DLList<T> &dll);
62
63         /* Deconstructor shall free up memory */
64         ~DLList();
65
66         /* Return the current length of the Singly Linked List */
67         int getLength() const;
68
69         /* Insert at the end of the list.*/
70         bool append(const T &val);
71
72         /* Insert val at position pos.
73          * Return true if successful (it can be placed.)
74          * Otherwise return false.
75          */
76         bool insert(const int pos, const T &val);
77
78         /* Print out the Singly Linked List */
79         void print() const;
80
81         /* Remove the first instance of val
82          * Return true if found and removed.
83          * Otherwise return false.
84          */
85         bool remove(const T &val);
86
87         /* Retrieves the element at position pos */

```

```

88     T& operator[] (const int pos);
89
90     /* Returns if the two lists contain the same elements in the
91      * same order.
92      */
93     bool operator==(const DLLList<T> &list) const;
94 };

```

Some Questions to Answer:

- 1) What is the performance difference for append between an Array, SLList, and DLLList?
 - 2) What is the performance difference for insert between an Array, SLList, and DLLList?
 - 3) What is the performance difference for operator[] between an Array, SLList, and DLLList?
 - 4) What is the performance difference for remove between an Array, SLList, and DLLList?
 - 5) What is the performance difference for search between an Array, SLList, and DLLList?
- You may answer in the comments of your code or in a separate .txt/.doc document.

Write some test cases:

Create some test cases, using cxxtestgen, that you believe would cover all aspects of your code. We will be creating some of these test cases in class.

Memory Management:

Now that are using new, we must ensure that there is a corresponding delete to free the memory. Ensure there are no memory leaks in your code!

How to turn in:

Turn in via GitHub. Ensure the file(s) are in your directory and then:

- \$ git add <files>
- \$ git commit
- \$ git push

Due Date: September 30, 2019 2359

Teamwork: No teamwork, your work must be your own.