Assignment 1

CSE 503 – Summer 2024

Richard Douglas

1 Assignment Description

We were asked to edit code in a file called sll.cpp containing an incomplete implementation of singly linked list (with a typo). Specifically, we were asked to do:

> Starting with my code for the single linked list in SLList.cpp, do the following:
> 1. Write an additional method called push_back(int) that will add an integer to the end of the list. You can modify the provided code.
> 2. Modify the Node class and LinkedList class so that you can access your parent node (double linked-list).

2 Logic Employed

Linked lists are a collection of nodes, stored at random in memory, which point to one another's memory addresses. To keep the list in sorted order, two special nodes' addresses are stored as variables in the list object- the starting node (head) and ending node (tail). These special nodes often only contain pointers, and no actual data. The head contains a pointer to the next node (the first with any actual data); this first data containing node contains a pointer to the previous node (head) and the next node. The same structure is repeated for each node until the tail, which contains no data- only a pointer to the final data containing node. To complete the tasks, I implemented this fairly standard list construction.

For the push_back(int) method, the int argument is the data. Simply put, we need to insert this node in between the tail and current final data containing node. The following code is used and descriptions are given in the comments:

```
void LinkedList::push_back(int val){
/*Your code here*/
Node *newNode = new Node(val); /*create a node to store the int in this temporary variable*/
newNode->pPrev = pTail→pPrev; /*point the new nodes previous pointer to the previous final data
containing node*/
newNode->pNext = pTail; /*point the new nodes next pointer to the tail*/
pTail->pPrev->pNext = newNode; /*point the previous final data containing node's next pointer to the newly
created node*/
pTail->pPrev = newNode; /*point the tail's previous pointer to the new node*/
}
```

This function successfully inserts a new node at the last data containing position of the list, passing in the only argument (int val) as the data contained.

For step two (already implemented in the above code, some modifications needed to be made to the original source. First of all, the node implementation needed to be modified to contain pointers to the previous and next nodes ad follows:

```
private:
int value;
Node *pNext;
Node *pPrev;
```

next, although I didn't use it in this assignment, I thought it might be useful to create another node constructor which included both pointers.

```
Node(int val, Node* prev, Node* next)
: value(val), pPrev(prev), pNext(next)
{}
```

In the List class, I modified the constructor which actually constructs a list containing nodes with values in the following commented code:

```
LinkedList::LinkedList(int val)
{
/* Instead (like in the lectures and example code) initialize head and tail as empty nodes */
/* then create a new node with the first value of list. Set pointers to the empty, head and tail*/
/* finally, reset the head pNext pointer to the first real Node and the tail's pPrev pointer to the same*/
pHead = new Node();
pTail = new Node();
Node *firstReal = new Node(val, pHead, pTail);
pHead->pNext = firstReal;
pTail->pPrev = firstReal;
}
```

Finally, the last bit of logic is in the aforementioned push_back(int val) function (not push_pack by the way).

3 I/O

The user is not prompted for input. All args passed into the push_back function simply come from variables inside the main() function. I did edit the output though. I edited the code that output the list so that it was a little prettier – wrapping the list in square brackets and separating each node's value by a comma- sufficient for this list of integers, although other print formatting might make sense for a list of other class data types. Also, I added a function to print out every node including its value, pointers and memory address, to prove that there is an empty head and tail with data containing nodes in sorted order between.

4 Screenshots

The following is the output run in my terminal.

```
richard@richard-IdeaPad:~/Desktop/uofl/2024_summer/cse503$ ls
 a.out   dll_solution.cpp   sll.cpp  'Untitled 2.odt'  'Week 2 Assignment.docx'
richard@richard-IdeaPad:~/Desktop/uofl/2024_summer/cse503$ ./a.out
Created an empty list named list1.
list1:
The list is empty
The list is empty
Created a list named list2 with only one node.
list2:
LinkedList: [ 10 ]
LinkedList: [
        value: 0
        pPrev: 0
        pNext: 0x6124e5098300
        memLocation: 0x6124e50982c0

        value: 10
        pPrev: 0x6124e50982c0
        pNext: 0x6124e50982e0
        memLocation: 0x6124e5098300

        value: 0
        pPrev: 0x6124e5098300
        pNext: 0
        memLocation: 0x6124e50982e0

 ]
```

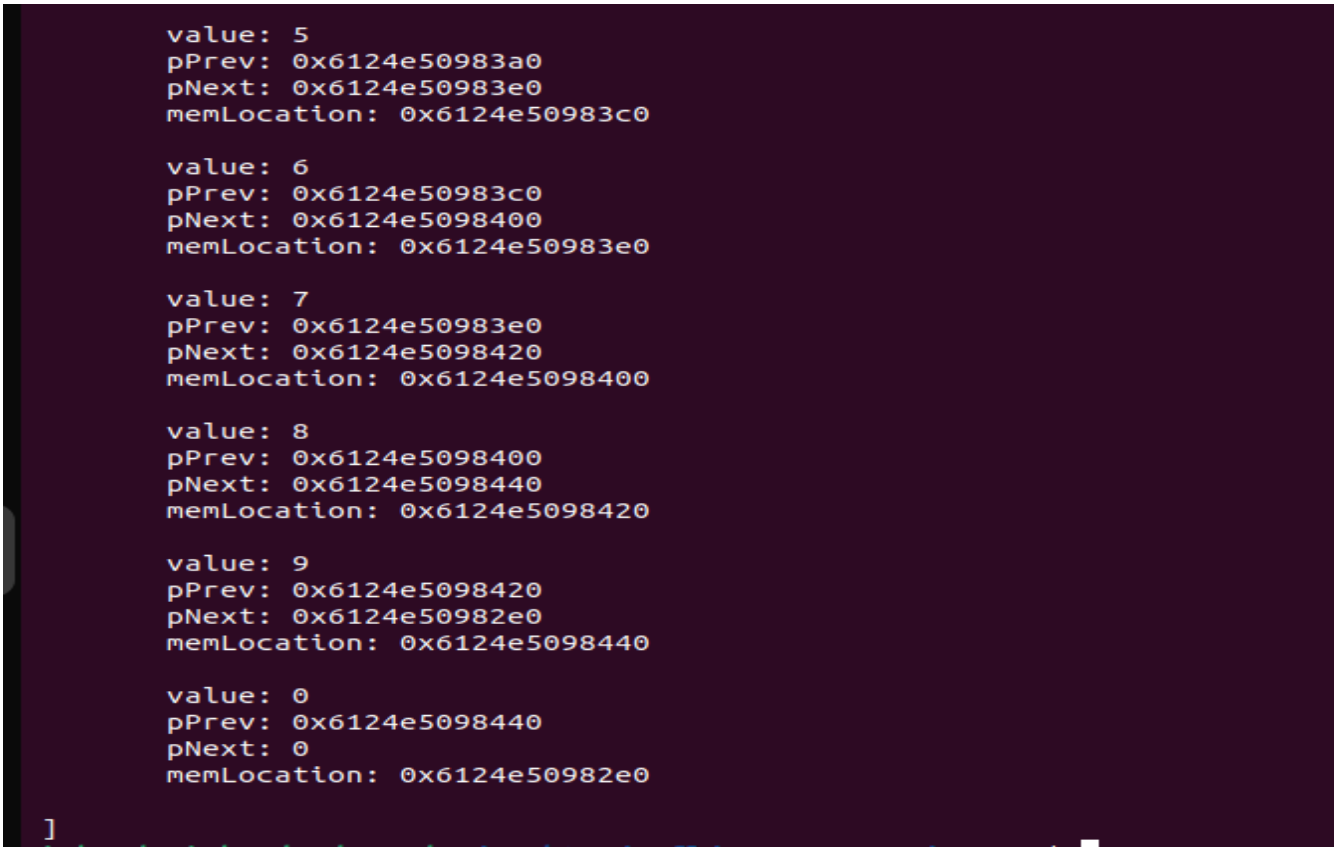
```
LinkedList: [ 10, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
LinkedList: [
        value: 0
        pPrev: 0
        pNext: 0x6124e5098300
        memLocation: 0x6124e50982c0

        value: 10
        pPrev: 0x6124e50982c0
        pNext: 0x6124e5098320
        memLocation: 0x6124e5098300

        value: 0
        pPrev: 0x6124e5098300
        pNext: 0x6124e5098340
        memLocation: 0x6124e5098320

        value: 1
        pPrev: 0x6124e5098320
        pNext: 0x6124e5098360
        memLocation: 0x6124e5098340

        value: 2
        pPrev: 0x6124e5098340
        pNext: 0x6124e5098380
        memLocation: 0x6124e5098360
```



5 Conclusions

This was a fairly easy assignment. The logic of linked lists is very clear and easy to follow. I'm excited to move on to trees.