Aaron Williams

Assignment 2 (Homework Unit 3)

30 August 2017

Assignment Description

The purpose of this assignment is to begin with source code for a single linked-list (SLLlist.cpp) and make the following changes: 1) Complete an additional method called push_back(int) that will add an integer node to the end of the linked-list and 2) Modify the Node class and LinkedList class so that the parent node can be accessed (create a double linked-list). Additionally, some minor changes were implemented as necessary.

Logic and Outputs

1) The first change made to the source code was the completion of the push_back(int) method. In order to accomplish this, the logic employed including first checking if the list was empty. If the list was empty, a node was created based on the int input value and was assigned as both the head and tail node. If the list already contained nodes, a new node was added by changing the value of the tail node next field. Once the node was added, the tail pointer was adjusted to the new node.

push_back(int) Method:

```
113  void LinkedList::push_back(int val){
114      /*Your code here*/
115
116      /*If list is empty, create a new node. */
117      if (pHead == NULL) {
118          pHead = new Node(val);
119          pTail = pHead;
120          return;
121      }
122
123      /*Add new node to back of the list. */
124      pTail->pNext = new Node(val);
125      /*Change tail pointer to newly added node. */
126      pTail = pTail->pNext;
127  }
```

Current Main Output:

```
[Ace:Unit 3 A$ g++ SLList.cp -o 5
[Ace:Unit 3 A$ ./5
Created an empty list named list1.
list1:
The list is empty
Created a list named list2 with only one node.
list2:
LinkedList: 10
LinkedList: 100123456789
Ace:Unit 3 A$
```

Next, the traverse_and_print() was updated to provide a cleaner output by adding square brackets around each node in a linked-list.

traverse_and_print() Method Changes:

```
    /* output the value */
    /*Change from source code made here. */
    cout << " [" << p->value << "] ";
```

Current main() Output:

```
[Ace:Unit 3 A$ g++ SLList.cp -o 5
[Ace:Unit 3 A$ ./5
Created an empty list named list1.
list1:
The list is empty
Created a list named list2 with only one node.
list2:
LinkedList:  [10]
LinkedList:  [10]  [0]  [1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9]
Ace:Unit 3 A$
```

2) Secondly, the Node and LinkedList classes were updated so that the parent node of each node could be accessed, essentially creating a double linked-list. The Node class was edited first to include a pPrev pointer for each node. The

necessary changes were also made to the constructors and getters.

Updated Node Class:

```
20  class Node
21  {
22      friend class LinkedList;
23  private:
24      int value;
25      Node *pNext;
26      Node *pPrev; //Added
27
28  public:
29      /* Constructors with No Arguments */
30      Node(void)
31      : pNext(NULL), pPrev(NULL) //Added
32      { }
33
34      /* Constructors with a given value */
35      Node(int val)
36      : value(val), pNext(NULL), pPrev(NULL) //Added
37      { }
38
39      /* Constructors with a given value and links for next and previous nodes */
40      Node(int val, Node* next, Node* previous)
41      : value(val), pNext(next), pPrev(previous) //Added
42      { }
43
44      /* Getters */
45      int getValue(void)
46      { return value; }
47
48      Node* getNext(void)
49      { return pNext; }
50
51      Node* getPrev(void) //Change
52      { return pPrev; }
53
54  };
```

The LinkedList class methods traverse_and_print() and push_back() were then edited. The push_back() method was edited first to account for the assignment of the pPrev pointer to each newly created noted. This was accomplished by changing the constructor used from part one.

Updated push_back() Method:

```
121 void LinkedList::push_back(int val){
122     /*Your code here*/
123
124     /*If list is empty, create a new node. */
125     if (pHead == NULL) {
126         pHead = new Node(val);
127         pTail = pHead;
128         return;
129     }
130
131
132     /*Add new node to back of the list. */
133     pTail->pNext = new Node(val, NULL, pTail); //New constructor used
134     /*Change tail pointer to newly added node. */
135     pTail = pTail->pNext;
136
137 }
```

Next, the traverse_and_print() method was updated to show the next and previous links for each node.

Updated traverse_and_print() Method:

```
97  void LinkedList::traverse_and_print()
98  {
99      Node *p = pHead;
100
101
102     /* The list is empty? */
103     if (pHead == NULL) {
104         cout << "The list is empty.\n" << endl;
105         return;
106     }
107
108     cout << "LinkedList: " << endl;
109     /* A basic way of traversing a linked list */
110     while (p != NULL) { /* while there are some more nodes left */
111         /* output the value */
112         /*Change from source code made here. */
113         cout << "<-" << p->getPrev();
114         cout << " [" << "Node Value: " << p->getValue() << ", " << p << "] ";
115         cout << "->" << p->getNext() << endl;
116
117         /* The pointer moves along to the next one */
118         p = p->pNext;
119     }
120     cout << endl;
121 }
```

Lastly, the main function was updated to show the reflected changes and results of the final code.

Final main() Output:

```
Ace:Unit 3 A$ g++ SLList.cp -o 5
Ace:Unit 3 A$ ./5
Created an empty list named list1.
list1:
The list is empty.

Created a list named list2 with only one node.
list2:
LinkedList:
<-0x0 [Node Value: 10, 0x7f8f6cc02650] ->0x0

Added 10 nodes to list2.
list2:
LinkedList:
<-0x0 [Node Value: 10, 0x7f8f6cc02650] ->0x7f8f6cc02670
<-0x7f8f6cc02650 [Node Value: 0, 0x7f8f6cc02670] ->0x7f8f6cc02690
<-0x7f8f6cc02670 [Node Value: 1, 0x7f8f6cc02690] ->0x7f8f6cc026b0
<-0x7f8f6cc02690 [Node Value: 2, 0x7f8f6cc026b0] ->0x7f8f6cc026d0
<-0x7f8f6cc026b0 [Node Value: 3, 0x7f8f6cc026d0] ->0x7f8f6cc026f0
<-0x7f8f6cc026d0 [Node Value: 4, 0x7f8f6cc026f0] ->0x7f8f6cc02710
<-0x7f8f6cc026f0 [Node Value: 5, 0x7f8f6cc02710] ->0x7f8f6cc02730
<-0x7f8f6cc02710 [Node Value: 6, 0x7f8f6cc02730] ->0x7f8f6cc02750
<-0x7f8f6cc02730 [Node Value: 7, 0x7f8f6cc02750] ->0x7f8f6cc02770
<-0x7f8f6cc02750 [Node Value: 8, 0x7f8f6cc02770] ->0x7f8f6cc02790
<-0x7f8f6cc02770 [Node Value: 9, 0x7f8f6cc02790] ->0x0
```