

Dillon Li

Assignment Unit 3

Submitted: 9/4/2016

Assignment Description

Given some source code for a simple singly linked list, I was to do the following:

1. Write an additional method called `push_back(int)` that will add an integer to the end of the list.
2. Modify the `Node` class and `LinkedList` class so that you can access your parent node (double linked-list).

Logic Implemented/Expected Input/Output

I will separate this into sections detailing the logic I implemented in each function I added to the initially given source code, and then other changes I made. I wrote three functions, one “`push_back`” for the actual logic desired in the assignment, and one additional test function, “`traverse_print_reversed`”.

`Push_back(int val):`

This is the function that is asked to be implemented in the first part of the assignment. The purpose is to add an integer to the end of the linked list.

A temp variable is created for the tail node in order to set `pPrevious` pointer for the newly pushed node. The next pointer for the tail node is set to point to a newly created node with the value of the given parameter for the function. Then the tail node is updated. The previous pointer of the tail node is set to point to the temp variable created (the previous tail node).

Inputs: `val` – the integer to set the new node at

Outputs: technically none, but adds a node to the end of the list. Expected output after test code (pushing from 0 to 9) is “10, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9”.

`Traverse_print_reversed():`

This function does the same thing that “`traverse_and_print()`” does, except it starts from the tail node and traverses to the parent node. This is just to test that `pPrevious` is set properly for each node in the list, and to confirm that it is indeed a doubly linked list.

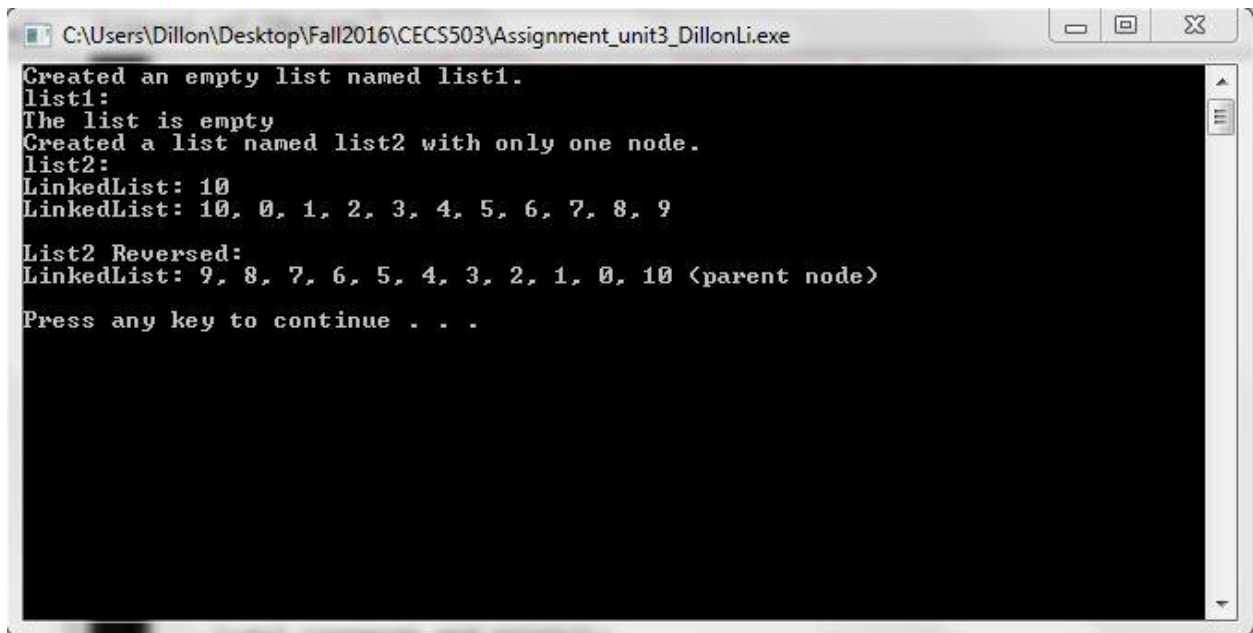
Inputs: none

Outputs: With the given list, `list2` (which `push_back()` appended nodes 0-9 to after initializing with 10), this function should print out “9, 8, 7, 6, 5, 4, 3, 2, 1, 10”

Other changes:

I modified `traverse_and_print` to print out the list a little neater with comma-separated form and added my test code to the end of the main function to print out things in addition to the already given test code to verify that the list is doubly linked.

Snapshot of Output:



```
C:\Users\Dillon\Desktop\Fall2016\CECS503\Assignment_unit3_DillonLi.exe
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

List2 Reversed:
LinkedList: 9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 10 <parent node>
Press any key to continue . . .
```

Source Code:

```
/**
 * A sample code of Creating C++ linked lists,
 * Including definitions the list node class and linked list class,
 * and how to create a blank linked list and a one-node linked list.
 *
 * Outline: understand the definition and structure of the linked
 * list and build a linked list based on it.
 */

#include <iostream>
#include <cstdint>

using std::cout;
using std::endl;

/* definition of the list node class */
class Node
{
    friend class LinkedList;
private:
    int value;
    Node *pNext;
    Node *pPrevious;

public:
    /* Constructors with No Arguments */
    Node(void)
    : pNext(NULL)
    { }

    /* Constructors with a given value */
    Node(int val)
    : value(val), pNext(NULL)
    { }

    /* Constructors with a given value and a link of the next node */
    Node(int val, Node* next)
    : value(val), pNext(next)
    { }

    /* Getters */
    int getValue(void)
    { return value; }
```

```

    Node* getNext(void)
    { return pNext; }
};

/* definition of the linked list class */
class LinkedList
{
private:
    /* pointer of head node */
    Node *pHead;
    /* pointer of tail node */
    Node *pTail;

public:
    /* Constructors with No Arguments */
    LinkedList(void);
    /* Constructors with a given value of a list node */
    LinkedList(int val);
    /* Destructor */
    ~LinkedList(void);

    /* Traversing the list and printing the value of each node */
    void traverse_and_print();

    void push_back(int val); // for assignment

    /* Prints list in reverse to show that parent node can be accessed */
    void traverse_print_reversed();
};

LinkedList::LinkedList()
{
    /* Initialize the head and tail node */
    pHead = pTail = NULL;
}

LinkedList::LinkedList(int val)
{
    /* Create a new node, acting as both the head and tail node */
    pHead = new Node(val);
    pTail = pHead;
}

LinkedList::~~LinkedList()
{
}

```

```

void LinkedList::traverse_and_print()
{
    Node *p = pHead;

    /* The list is empty? */
    if (pHead == NULL) {
        cout << "The list is empty" << endl;
        return;
    }

    cout << "LinkedList: ";
    /* A basic way of traversing a linked list */
    while (p != NULL) { /* while there are some more nodes left */
        /* output the value */
        cout << p->value;
        if (p != pTail)
            cout << ", "; // Added by me for sake of neatness
        /* The pointer moves along to the next one */
        p = p->pNext;
    }
    cout << endl;
}

void LinkedList::push_back(int val)
{
    /* Adds node to end, moves tail, sets previous tail as tail->previous */
    Node *temp_prev = pTail;
    pTail->pNext = new Node(val);
    pTail = pTail->pNext;
    pTail->pPrevious = temp_prev;
}

void LinkedList::traverse_print_reversed()
{
    Node *p = pTail;

    /* The list is empty? */
    if (pHead == NULL) {
        cout << "The list is empty" << endl;
        return;
    }

    cout << "LinkedList: ";
    while (p != NULL) { /* while there are some more nodes left */
        /* output the value */
        cout << p->value;
        if (p != pHead)
            cout << ", "; // Added by me for sake of neatness
    }
}

```

```

        else
            cout << " (parent node)";
        /* The pointer moves along to the next one */
        p = p->pPrevious;
    }
    cout << endl;
}

int main(int argc, const char * argv[])
{
    /* Create an empty list */
    LinkedList list1;
    cout << "Created an empty list named list1." << endl;
    /* output the result */
    cout << "list1:" << endl;
    list1.traverse_and_print();

    /* Create a list with only one node */
    LinkedList list2(10);
    cout << "Created a list named list2 with only one node." << endl;
    /* output the result */
    cout << "list2:" << endl;
    list2.traverse_and_print();

    /*your testing code here*/

    /* Appends to list with 10 in it: 10, 1, 2, 3, ... 9 */
    for (int i = 0 ; i < 10; i++){
        list2.push_back(i);
    }
    list2.traverse_and_print();

    /* Print out list traversing the other way */
    cout << "\n" << "List2 Reversed:\n";
    list2.traverse_print_reversed();
    cout << "\n";

    system("pause");
    return 0;
}

```