Marie Biernacki

COMP-SCI 303 Data Structures

Professor Shah

Assignment 02 Report

**PROGRAM EXECUTION:**

* **IN MAIN.CPP**
  + include Linked\_List.h and Stack.h
  + create and initialize code that tests Linked List and Stack functionality
    - should test the following functions for Linked\_List:
      * push\_front()
      * push\_back()
      * pop\_front()
      * pop\_back()
      * insert()
      * remove()
      * find()
      * empty()
    - should test the following functions for Stack
      * insert()
      * top()
      * remove()
      * average()
      * empty()
    - incorporate printList(), getNumItems() and printStack() to verify the functions are working properly
* **IN LINKED\_LIST.H**
  + **create the following template structs**
    - Item\_Type
      * define data variable, create the Item\_Type constructor, and define the operator== for use in the find() function
    - Node
      * define nodeData, pointer to the next node and the Node constructor
  + **create the Linked class as a template class**
    - **protected members:** 
      * head pointer, tail pointer, and num\_items
    - **public members:**
      * default constructor, constructor passing Item\_Type, and destructor
      * **declare and define the following functions:**
        + void push\_front(const Item\_Type<T>& item);

inserts a copy of item as the first element of the list

* + - * + void push\_back(const Item\_Type<T>& item);

adds copy of item to end of the list

* + - * + void pop\_front();

removes first item from list

* + - * + void pop\_back()

removes last item from list

* + - * + Item\_Type<T>& front();

returns first element in the list

* + - * + Item\_Type<T>& back();

returns last element in the list

* + - * + bool empty();

returns true if list is empty

* + - * + void insert(size\_t index, const Item\_Type<T>& item);

insert item at position index (starting at 0), insert at the end if the index is beyond the end of the list

* + - * + bool remove(size\_t index);

removes item at index, returns true if successful, returns false if the index is beyond the end of the list

* + - * + size\_t find(const Item\_Type<T>& item);

return the position of the first occurrence of the item if it is found, return the size of list if not found

* + - * + void printList();

iterates through list and prints data at each node

* + - * + int getNumItems() const;

returns the value of num\_items

* **IN STACK.H**
  + include standard vector library to use for functionality
  + create the Stack class
    - **protected members:**
      * create vector of integers
    - **public members:** 
      * **declare the following functions:**
        + bool empty();

returns true or false based on whether the stack is empty

* + - * + void insert(int number);

adds a number to the top of the stack

* + - * + void remove();

removes the top number from the stack

* + - * + int& top();

returns the value of the top of the stack

* + - * + double average();

finds and returns the average of the numbers in the stack

* + - * + void printStack();

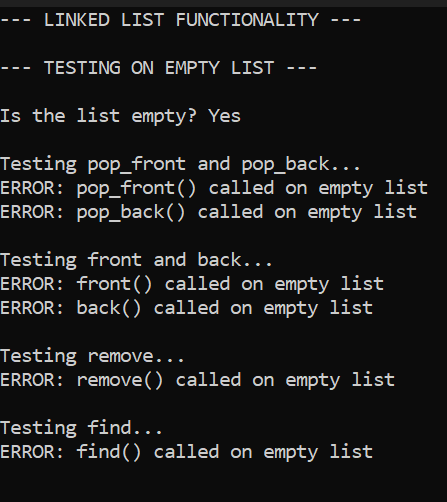
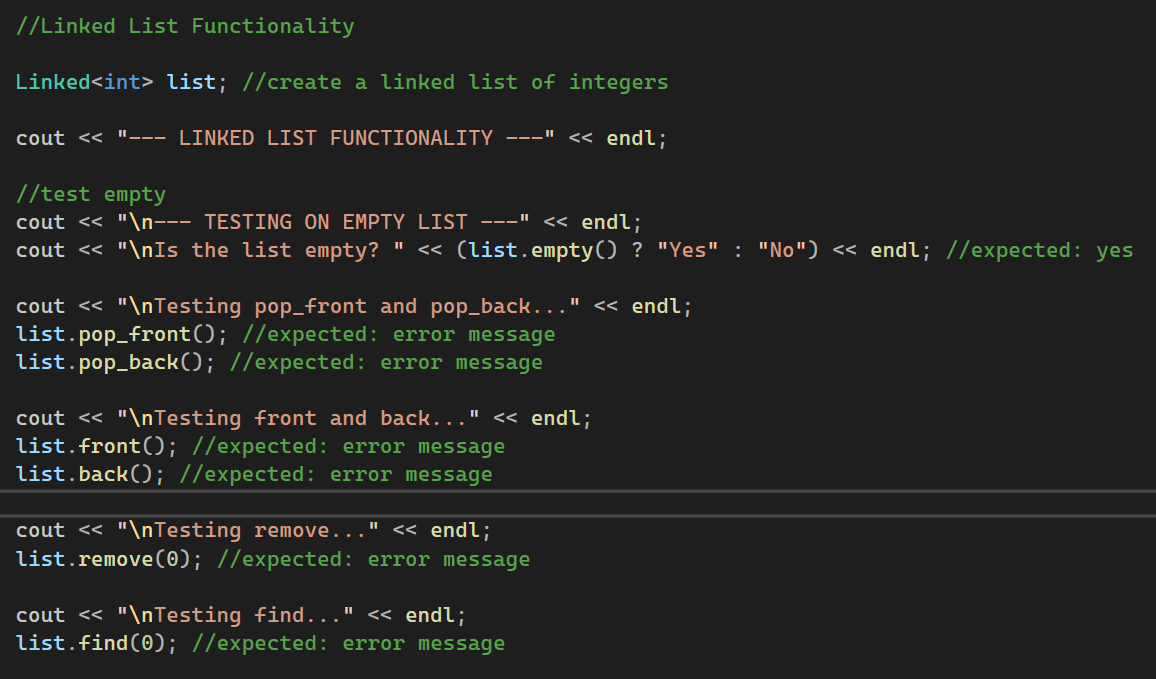
print function to showcase stack functionality

* **IN STACK.CPP**
  + **define the following functions:**
    - bool empty();
      * call empty()
    - void insert(int number);
      * call push\_back(number)
    - void remove();
      * call pop\_back()
    - int& top();
      * call back()
    - double average();
      * check if stack is empty
      * iterate through stack using for loop
      * calculate the average using static\_cast
    - void printStack();
      * iterate through stack and print the values

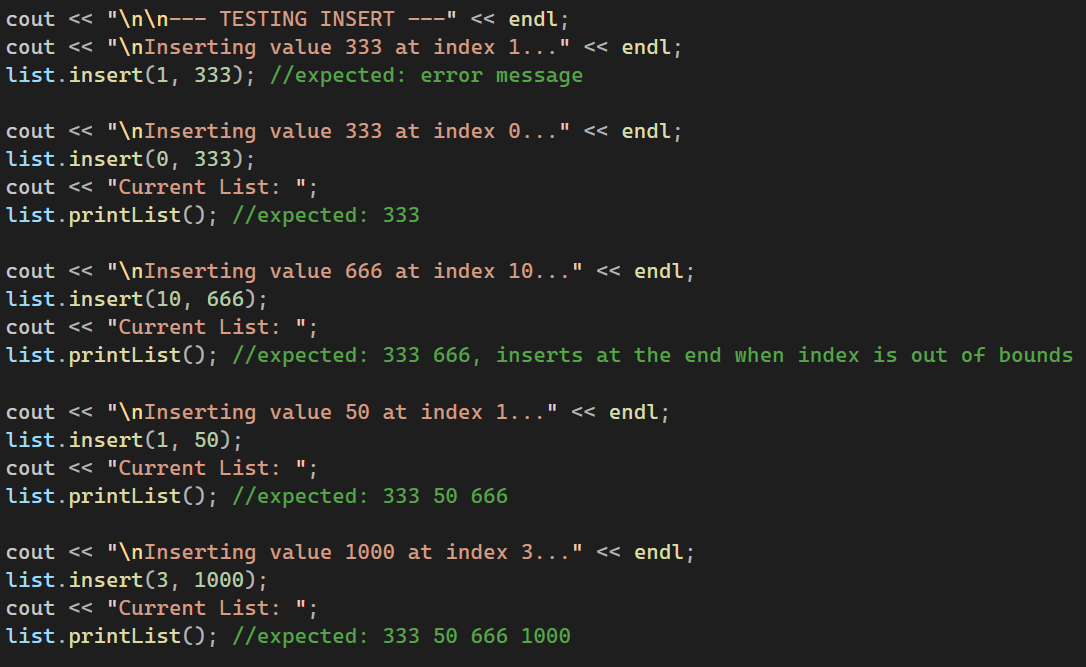
**OUTPUT AND CODE SCREENSHOTS:**

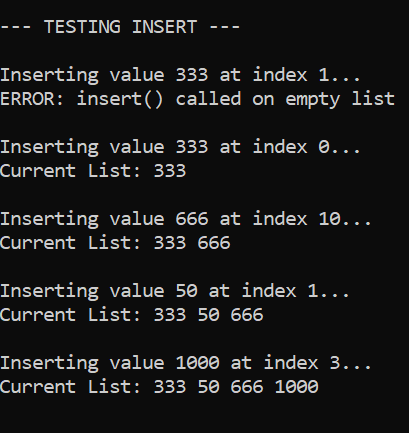
*Linked List*

Testing on Empty List

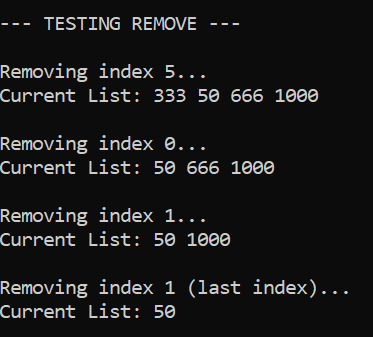
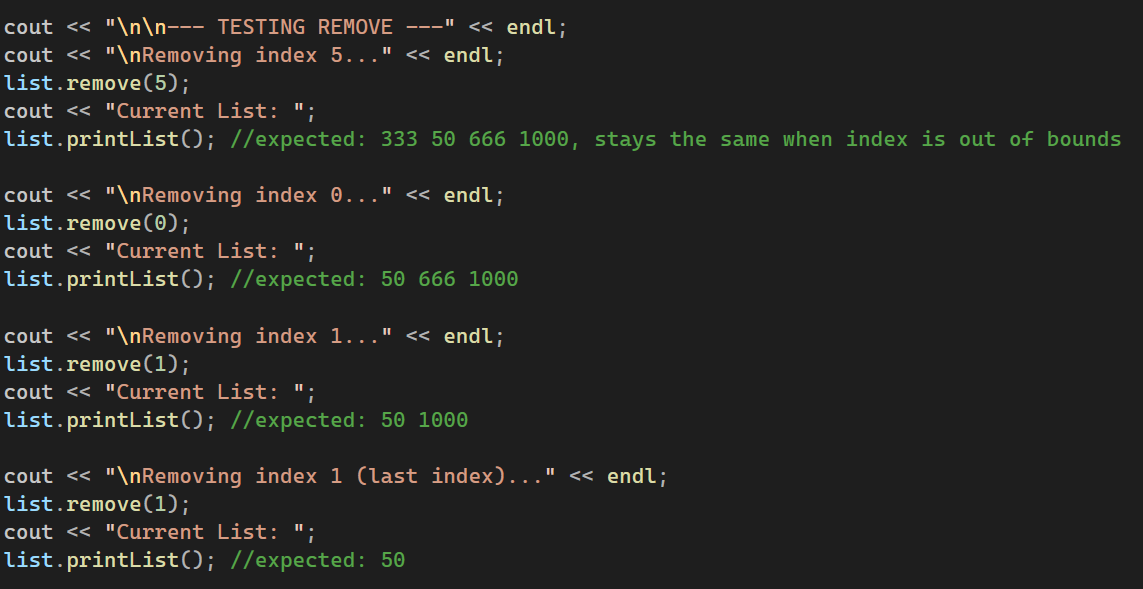


Testing Insert

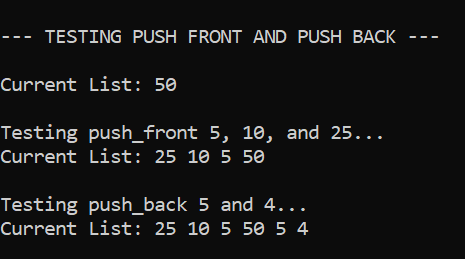
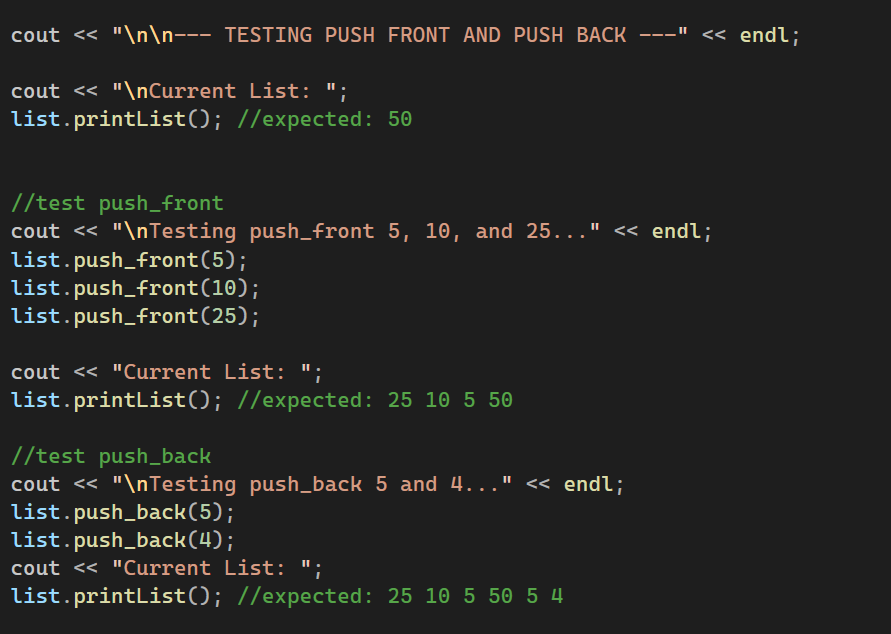




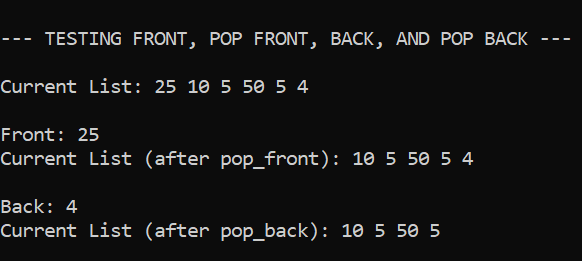
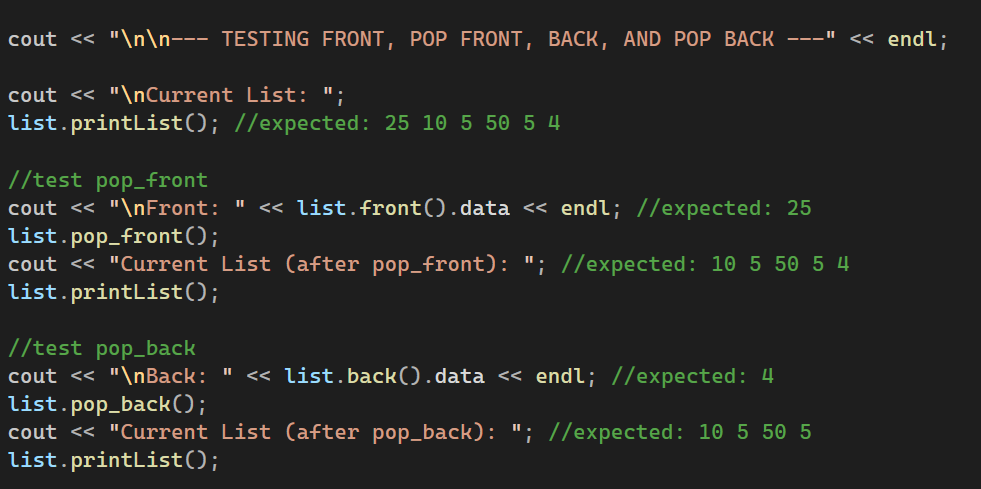
Testing Remove



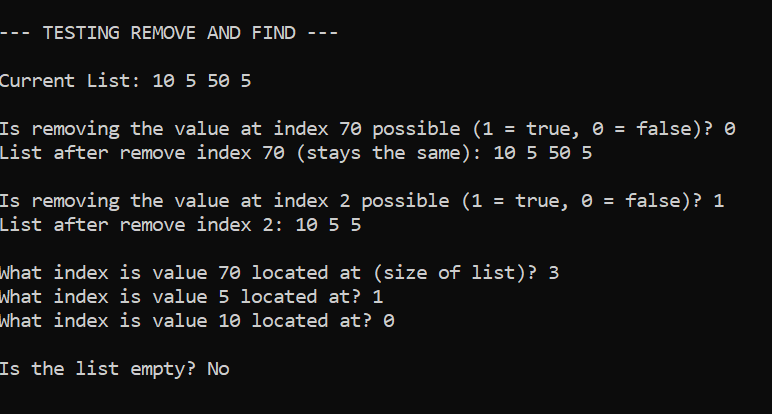
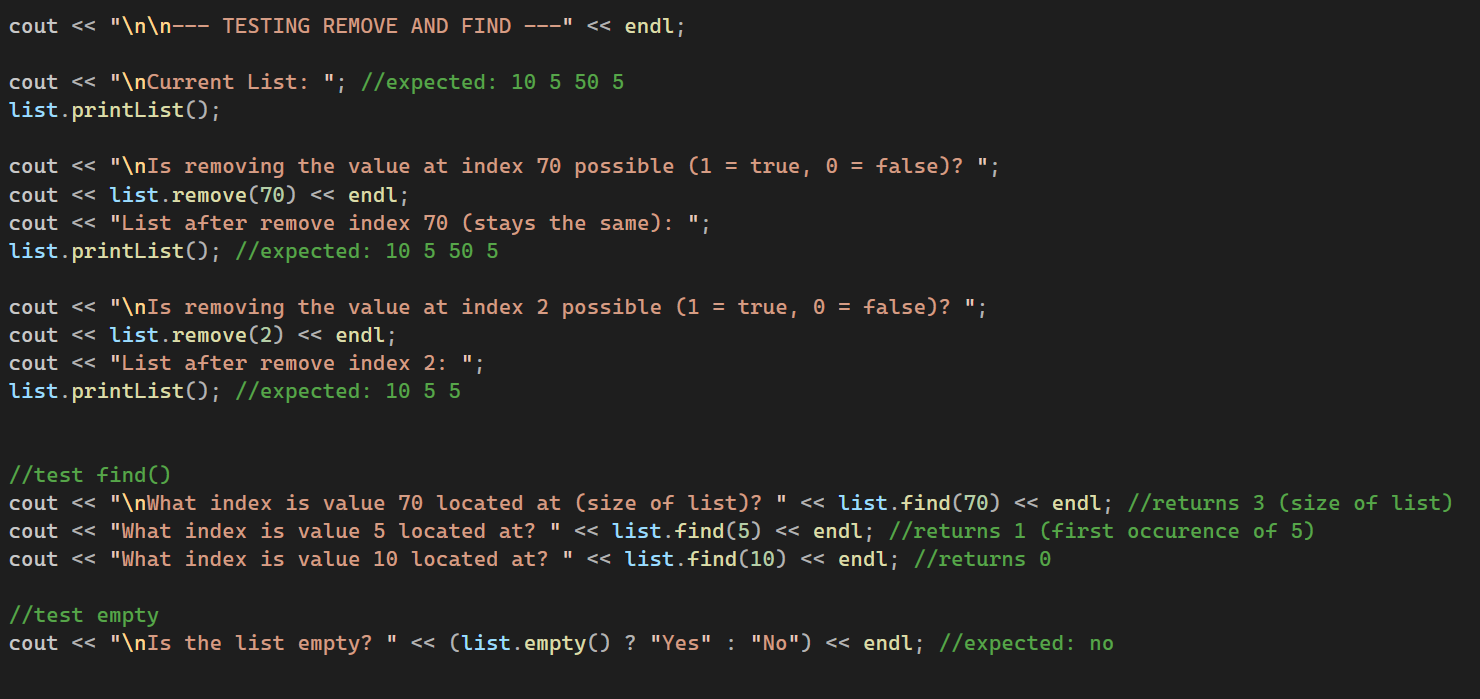
Testing Push Front and Push Back



Testing Front, Pop Front, Back, and Pop Back



Testing Remove and Find



*Stack*

