//============================================================================

/\* List.cpp

\* Created on: Sept 30, 2016

\* Author: Sergio Gutierrez

\* Class: 22C

\* Fall Quarter 2016

\* Professor: Jennifer Parrish

\*/

//============================================================================

**#include** <iostream>

**#include** "List.h"

**using** **namespace** std;

//Default Constructor

**List::List**(): start(NULL), end(NULL), cursor(NULL), length(0) {}

//Destructor

**List::~List**()

{

cursor = start;

NodePtr temp;

**while**(cursor != NULL)

{

temp = cursor->next;

**delete** cursor;

cursor = temp;

}

}

**void** **List::add\_start**(**int** data)

{

**if** (length==0) //It tests to see if there is an already existing list.

{

start = **new** Node(data);

end = start;

}

**else**

{

NodePtr N = **new** Node(data);//create the new node by calling the node constructor

N->next = start;//set the new node's next field to point to the start

start = N;//make the start be the new node

}

length++;

}

**void** **List::add\_end**(**int** data)

{

**if** (length==0)

{

end = **new** Node(data);

start = end;

}

**else**

{

NodePtr N = **new** Node(data);

end->next = N;

end = N;

}

length++;

}

**void** **List::print**()

{

NodePtr temp = start; //create a temporary iterator

**while** (temp != NULL) {

//What two lines of code go here?

cout << temp->data << " ";

temp = temp->next;

//Hint: One statement should contain a cout

}

cout << **endl**; //This keeps your printouts well organized with a space

//between the lists.

}

**bool** **List::is\_empty**()

{

**return** (length==0);

}

**int** **List::get\_length**()

{

**return** length;

}

**int** **List::get\_start**()

{

**return** start -> data;

}

**int** **List::get\_end**()

{

**return** end -> data;

}

/\* List.h

\* Created on: Sept 30, 2016

\* Author: Sergio Gutierrez

\* Class: 22C

\* Fall Quarter 2016

\* Professor: Jennifer Parrish

\*/

**#ifndef** LIST\_H\_

**#define** LIST\_H\_

**#include** <cstddef> //for NULL

**#include** <iostream>

**class** List

{

**private**:

**struct** Node

{

**int** data;

Node\* next;

**Node**(**int** data): next(NULL), data(data){}

};

**typedef** **struct** Node\* NodePtr;

NodePtr start;

NodePtr end;

NodePtr cursor;

**int** length;

**public**:

/\*\*Constructors and Destructors\*/

**List**();

//Default constructor; initializes and empty list

//Postcondition: The list is now initialized with default values.

**~List**();

//Destructor. Frees memory allocated to the list

//Postcondition: The memory that was allocated in the list is now free

**List**(**const** List &list);

//Copy construcor. Initializes list to have the same elements as another list

//Postcondition: The new copy of the list now has a copy

// of the same elements as the original list.

/\*\*Accessors\*/

**int** **get\_start**();

//Returns the first element in the list

//Precondition: The Data which the start pointer points to

//can be accessed.

**int** **get\_end**();

//Returns the last element in the list

//Precondition: The Data which the end pointer points to

//can be accessed.

**int** **get\_cursor**();

//Returns the element pointed to by the iterator

//Precondition: The Data on which the cursor is on can be accessed.

**bool** **is\_empty**();

//Determines whether a list is empty.

**bool** **off\_end**();

//Determines if the iterator is off the end of the list (i.e. whether cursor is NULL)

**int** **get\_length**();

//Returns the length of the list

/\*\*Manipulation Procedures\*/

**void** **begin\_cursor**();

//Moves the iterator to point to the first element in the list

//If the list is empty, the iterator remains NULL

//Postcondition: The iterator is now pointing to the

// the first element in the list

**void** **insert\_cursor**(**int** data);

//Inserts a new element into the list in the position after the iterator

//Precondition: Iterator has a certain position on the list.

//Postcondition: Iterator points to new element that was inserted.

**void** **remove\_end**();

//Removes the value of the last element in the list

//Precondition: There is an element that is last on the list.

//Postcondition: The element that was last on the list is now removed.

**void** **remove\_start**();

//Removes the value of the first element in the list

//Precondition: There is an element that is first on the list.

//Postcondition: The element that was first on the list is now removed.

**void** **add\_end**(**int** data);

//Inserts a new element at the end of the list

//If the list is empty, the new element becomes both start and end

//Postcondition: A new element was inserted at the end of the list.

**void** **add\_start**(**int** data);

//Inserts a new element at the start of the list

//If the list is empty, the new element becomes both start and end

//Postcondition: A new element was inserted at

// the beginning of the list

**void** **remove\_cursor**();

//Removes the element pointed at by the iterator

//Precondition: The iterator points to an element.

//Postcondition: The iterator is no longer pointing to the element.

**void** **move\_cursor**();

//Moves the iterator forward by one element in the list

//Precondition: The iterator is pointing to an element.

//Postcondition: The iterator is now pointing to an element that

//is one element forward to the one it was originally pointing.

/\*\*Additional List Operations\*/

**void** **print**();

//Prints to the console the value of each element in the list sequentially

//and separated by a blank space

//Prints nothing if the list is empty

};

**#endif** /\* LIST\_H\_ \*/

//============================================================================

/\* ListTest.cpp

\* Name : Sergio Gutierrez

\* Created on: Sept 30, 2016

\* Author: Sergio Gutierrez

\* Class: 22C

\* Fall Quarter 2016

\* Professor: Jennifer Parrish

//============================================================================

**#include** <iostream>

**#include** "List.h"

**using** **namespace** std;

**int** **main**()

{

//creating a new list object L

List L;

L.add\_start(5);

cout << "The start of the List should be 5: " << L.get\_start() << **endl**;

cout << "The end of the List should be 5: " << L.get\_end() << **endl**;

cout << "List should contain: 5" << **endl**;

L.print();

L.add\_end(10);

cout << "The start of the List should be 5: " << L.get\_start() << **endl**;

cout << "The end of the List should be 10: " << L.get\_end() << **endl**;

cout << "List should contain: 5 10" << **endl**;

L.print();

//YOU MUST ADD MORE TESTS HERE TO RECEIVE CREDIT FOR THIS LAB!!!

//Here's my code:

cout << "-----my added code-----" << **endl**;

List X; //Initializing new X list.

X.add\_start(4);

X.add\_start(3);

X.add\_start(2);

X.add\_start(1);

X.add\_end(5);

X.add\_end(6);

X.add\_end(7);

cout << "The start of the List should be 1: " << X.get\_start() << **endl**;

cout << "The end of the List should be 7: " << X.get\_end() << **endl**;

cout << **endl**;

cout << "List X should contain 1, 2, 3, 4, 5, 6, 7:" << **endl**;

X.print();

X.add\_start(0);

X.add\_end(8);

cout << "List X should contain 0, 1, 2, 3, 4, 5, 6, 7, 8:" << **endl**;

X.print();

cout << "The length of this list should be 9: " << X.get\_length() << **endl**;

cout << **endl**;

cout << "The following X list should not be empty." << **endl**;

**if** (X.is\_empty())

cout << "The list is empty." << **endl**;

**else**

cout << "The list is not empty." << **endl**;

cout << **endl**;

List Y;

cout << "The following list Y should be empty:" << **endl**;

**if** (Y.is\_empty())

cout << "The list is empty." << **endl**;

**else**

cout << "The list is not empty." << **endl**;

//This ran perfectly with the expected output which is found below.

}

/\*

Full output was:

The start of the List should be 5: 5

The end of the List should be 5: 5

List should contain: 5

5

The start of the List should be 5: 5

The end of the List should be 10: 10

List should contain: 5 10

5 10

-----my added code-----

The start of the List should be 1: 1

The end of the List should be 7: 7

List X should contain 1, 2, 3, 4, 5, 6, 7:

1 2 3 4 5 6 7

List X should contain 0, 1, 2, 3, 4, 5, 6, 7, 8:

0 1 2 3 4 5 6 7 8

The length of this list should be 9: 9

The following X list should not be empty.

The list is not empty.

The following list Y should be empty:

The list is empty.\*/