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

/\*\*

\* Sergio Gutierrez

\* CIS 22C

\* List.h

\*/

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

#ifndef LIST\_H\_

#define LIST\_H\_

#include <cstddef> //for NULL

#include <iostream>

#include <assert.h>

using namespace std;

template <class listitem>

class List

{

private:

struct Node

{

listitem data;

Node\* next;

Node\* previous;

Node(listitem data): next(NULL), previous(NULL), data(data){}

};

typedef struct Node\* NodePtr;

NodePtr start;

NodePtr end;

NodePtr cursor;

int length;

public:

/\*\*Constructors and Destructors\*/

List <listitem>();

//Default constructor; initializes and empty list

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

~List<listitem>();

//Destructor. Frees memory allocated to the list

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

List<listitem>(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\*/

listitem get\_start();

//Returns the first element in the list

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

//can be accessed.

listitem get\_end();

//Returns the last element in the list

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

//can be accessed.

listitem get\_cursor();

//Returns the element pointed to by the iterator

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

listitem is\_empty();

//Determines whether a list is empty.

listitem 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(listitem 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(listitem 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(listitem 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

/\*Additional List Operations\*/

bool operator==(const List &list);

//Tests two lists to determine whether their contents are equal

//Postcondition: returns true if lists are equal and false otherwise

};

//Default Constructor

template<class listitem>

List<listitem>::List(): start(NULL), end(NULL), cursor(NULL), length(0) {}

//Destructor

template<class listitem>

List<listitem>::~List()

{

cursor = start;

NodePtr temp;

while(cursor != NULL)

{

temp = cursor->next;

delete cursor;

cursor = temp;

}

}

template<class listitem>

List<listitem>::List(const List &list): length(list.length)

{

if(list.start == NULL) //If the original list is empty, make an empty list for this list

{

start = end = cursor = NULL;

}

else

{

start = new Node(list.start->data); //using second Node constructor

NodePtr temp = list.start; //set a temporary node pointer to point at the start of the original list

cursor = start; //set iterator to point to start of the new list

while(temp->next != NULL)

{

temp = temp->next; //advance up to the next node in the original list

cursor->next = new Node(temp->data); //using node constructor to create a new node with copy of data

cursor = cursor->next; //advance to this new node

}

end = cursor; //Why do I need this line of code?

cursor = NULL;

}

}

template <class listitem>

void List<listitem>::add\_start(listitem data)

{

if (length==0)

{

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->previous = N;

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

}

length++;

}

template<class listitem>

void List<listitem>::add\_end(listitem data)

{

if (length==0)

{

end = new Node(data);

start = end;

}

else

{

NodePtr N = new Node(data);

end->next = N;

N->previous = end;

end = N;

}

length++;

}

template<class listitem>

void List<listitem>::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.

}

template<class listitem>

void List<listitem>::insert\_cursor(listitem data)

{

if(cursor == NULL)

{

cout << "The cursor is off the end of the list." << endl;

}

else if (length == 1)

{

NodePtr N = new Node(data);

N->next = NULL;

N->previous = cursor;

cursor->next = N;

length++;

}

else

{

NodePtr N = new Node(data);

N->next = cursor->next;

N->previous = cursor;

cursor->next->previous = N;

cursor->next = N;

length++;

}

}

template<class listitem>

listitem List<listitem>::is\_empty()

{

return (length==0);

}

template<class listitem>

int List<listitem>::get\_length()

{

return length;

}

template<class listitem>

listitem List<listitem>::get\_start()

{

assert (start!=NULL);

return start -> data;

}

template<class listitem>

listitem List<listitem>::get\_end()

{

assert (end!=NULL);

return end -> data;

}

template<class listitem>

listitem List<listitem>::get\_cursor()

{

assert (cursor!=NULL);

return cursor->data;

}

template<class listitem>

listitem List<listitem>:: off\_end()

{

if (cursor == NULL )

return true;

else

return false;

}

template<class listitem>

void List<listitem>::remove\_start()

{

assert(length != 0);

if (length==1)

{

delete start;

start = end = cursor = NULL;

length = 0;

}

else

{

if (cursor == start)

cursor = NULL;

NodePtr temp = start; //store original start node in a temporary variable

start->next->previous = NULL;

start = start->next; //make the start pointer point to the second node in the List

delete temp; //delete the original start

length--;

}

}

template<class listitem>

void List<listitem>::remove\_end()

{

assert(length != 0);

if (length==1)

{

delete end;

end = start = cursor = NULL;

length = 0;

}

else

{

if (cursor == end)

cursor = NULL;

NodePtr temp = end; //store original start node in a temporary variable

end->previous->next = NULL;

end = end->previous; //make the end pointer point to the previous node in the List

delete temp; //delete the original end

length--;

}

}

template<class listitem>

void List<listitem>::begin\_cursor()

{

assert(length != 0);

cursor = start;

}

template<class listitem>

void List<listitem>::move\_cursor()

{

assert(length != 0);

if(cursor == NULL)

{

cout << "The iterator is off the end." << endl;

}

else if(cursor != NULL)

{

cursor = cursor->next;

}

else

{

while(cursor->next != NULL)

{

cursor = cursor->next;

}

}

}

template<class listitem>

void List<listitem>::remove\_cursor()

{

assert(length != 0);

if(length == 1)

{

delete cursor;

cursor = end = start = NULL;

length = 0;

}

else

{

cursor->previous->next = cursor->next;

cursor->next->previous = cursor->previous;

delete cursor;

cursor = NULL;

length--;

}

}

template <class listitem>

bool List<listitem>::operator==(const List& list)

{

if(length != list.length)

return false;

NodePtr temp1 = start;

NodePtr temp2 = list.start;

while(temp1 != NULL)

{

if(temp1->data != temp2->data)

return false;

temp1 = temp1->next;

temp2 = temp2->next;

}

return true;

}

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