Project 5

Design Document

Introduction

The goal of this project is to implement a linked list ADT with recursive functions using a menu in the client program. The linked list is located in a class, and is called from the main program. The data type of the ADT will be integers. The client program makes use of a menu, which makes performing actions on the list ADT easier because after every command, the program will wait for the next command. This is easier than having to run the program every time you want to perform an operation.

Data Structures

The data structure we are concerned with in this project is the simple singly linked list ADT. A linked list is a collection of nodes that all include a piece of data and a pointer that points to the next

Functions

The main program consists of only one function, which displays a menu with the possible operations listed. Within the class, there are many functions that can: check if a list is empty, initialize a list to be empty (default constructor), make a list empty, re-initialize an existing list to be empty, insert a value into a List, remove a value from the list, find the length of a list, check if a given value is in the list, return the kth element of a list, and display the values of a list to an output stream.

Main Program

The main program basically consists of several variable declarations and then it displays a menu. There is only one function in the main program. This function displays a menu and then passes given input into a switch statement.

Project Code

Here are the contents of the class header file *p5.h*:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

typedef statement: this is the data type of the items in the list. In our program it is

integer. Data items are represented by the word 'Item'

CONSTRUCTOR

Precondition: none

Postcondition: List is initialized to its null state. In our case this is an empty list

List()

COPY CONSTRUCTOR

Precondition: none

Postcondition: The new list has been initialized as a copy of the initial list

List(const List& source)

DESTRUCTOR

Never needs to be explicitly called. Destroys Lists when the user/program is done with them

to free up memory

Precondition: none

Postcondition: none

~List()

MUTATOR FUNCTIONS

void make\_empty()

Precondition: none

Postcondition: The list becomes empty

void insert(const Item& entry)

Precondition: entry is not already in the list

Postcondition: the value entry has been inserted into the correct place in the List

void remove(const Item& target)

Precondition: target is actually in the List

Postcondition: the value target is removed from the List

void operator = (const List& source)

Precondition: The given list has been assigned a copy of the source list

CONSTANT MEMBER FUNCTIONS

bool empty() const

Precondition: none

Postcondition: Return TRUE/1 if the given List is empty, return FALSE/0 if the given List

is not empty

size\_t length() const

Precondition: none

Postcondition: The total number of items in the list is returned

bool present(const Item& target) const

Precondition: none

Postcondition: return TRUE/1 if target is actually in the list, return FALSE/0 if target is

not in the list

int kth (int k) const

Precondition: The list cannot be empty. the value k must be at least 1 and smaller than the

total number of items in the list

FRIEND FUNCTION

friend ostream& operator << (ostream& out\_s, const List& b)

Precondition: none

Postcondition: All of the values from List b is written to the output stream

PRIVATE FUNCTION

Node\* get\_node(const Item& entry, Node\* link);

Precondition: The number of the entry must exist

Postcondition: a pointer to a Node is returned

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#ifndef P5\_h

#define P5\_h

namespace csci301\_P5\_1

{

class List

{

public:

//this is the data type of the items in the list - int. Data items are

// represented by the word Item

typedef int Item;

//Constructors

List() {first = NULL;}

List(const List& source);

//Destructor

~List();

// Mutator member functions

void make\_empty();

void insert(const Item& entry);

void remove(const Item& target);

void operator = (const List& source);

// Constant member functions

bool empty() const {return first == NULL;}

std::size\_t length() const;

bool present(const Item& target) const;

Item kth(std::size\_t k) const;

// Friend function for the List class

friend std::ostream& operator << (std::ostream& out\_s, const List& l);

private:

// Private data members:

struct Node

{

Item data;

Node \*next;

};

Node \*first;

//Private function:

Node\* get\_node(const Item& entry, Node\* link);

};

}

#endif

.cpp file for the class:

/\*

Invariant:

- The items in the List are stored in a linked list in ascending order

- The member variable called 'first' is a pointer that points to the first node in the

Linked List. If it is an empty List, first points to the nullptr.

\*/

#include <cassert> // Provides assert()

#include <cstdlib> // Provides size\_t

#include <iostream>

#include <iomanip> // Provides setw()

#include <fstream>

#include "p5.h"

using namespace std;

namespace csci301\_P5\_1

{

//Copy constructor

List::List(const List& source)

{

Node\* p;

Node\* last;

if(source.first == NULL)

first = NULL;

else

{

first = get\_node(source.first -> data, NULL); //copy the first node

last = first;

p = source.first -> next;

while(p != NULL) //copy remaining Nodes

{

last -> next = get\_node(p -> data, NULL);

last = last -> next;

p = p -> next;

}

}

}

//Destructor

List::~List()

{

Node\* temp;

while(first != NULL)

{

temp = first;

first = first -> next;

delete temp;

}

}

//Mutator member functions

void List::make\_empty()

{

Node\* temp;

while(first != NULL)

{

temp = first;

first = first -> next;

delete temp;

}

}

void List::insert(const Item& entry)

{

Node \*prev;

//assert(!present(entry));

if(first == NULL || entry < first -> data)

first = get\_node(entry, first);

else

{

prev = first;

while(prev->next != NULL && prev->next->data < entry)

prev = prev->next;

prev->next = get\_node(entry, prev->next);

}

}

void List::remove(const Item& target)

{

Node \*temp;

Node \*prev;

//assert(present(target));

prev = first;

if(prev->data == target)

{

first = first->next;

delete prev;

}

else

{

while(prev->next != NULL && prev->next->data < target)

prev = prev->next;

temp = prev->next;

prev->next = temp->next;

delete temp;

}

}

void List::operator = (const List& source)

{

Node\* p;

Node\* last;

if(&source != this)

{

make\_empty();

if(source.first != NULL)

{

first = get\_node(source.first->data, NULL); //copy the first Node

last = first;

p = source.first->next;

while(p != NULL)

{

last->next = get\_node(p->data, NULL);

last = last->next;

p = p->next;

}

}

}

}

// Constant member functions

size\_t List::length() const

{

Node \*cursor;

size\_t count;

count = 0;

for(cursor = first; cursor != NULL; cursor = cursor->next)

count++;

return count;

}

bool List::present(const Item& target) const

{

Node\* cursor;

for(cursor = first;

cursor != NULL && cursor->data != target;

cursor = cursor->next)

;

return(cursor != NULL);

}

List::Item List::kth(size\_t k) const

{

Node \*cursor;

size\_t count;

//assert(1 <= k && k <= length());

cursor = first;

for(count = 1;count < k; count++)

cursor = cursor->next;

return cursor->data;

}

//Friend Function

ostream& operator << (ostream& out\_s, const List& l)

{

List::Node \*cursor;

out\_s << '(';

for(cursor = l.first;

cursor != NULL && cursor->next != NULL;

cursor = cursor->next)

out\_s << cursor->data;

if(cursor != NULL)

out\_s << cursor->data;

out\_s << ')';

return out\_s;

}

//Private Function:

List::Node\* List::get\_node(const Item& entry, Node\* link)

{

Node \*temp;

temp = new Node;

temp -> data = entry;

temp -> next = link;

return temp;

}

}

Main Program:

// Created By: John Blee

// Due 15 February 2015

// CSCI 301, Dr. Julstrom

// Project 5

/\* This program utilizes a menu to implement a linked list. The List is defined in a class

that is included in this main program. When the class object is defined, the program uses

a do-while loop and switch statement to power the menu. The loop waits for input, and the

switch statement carries out the correct operation based on the user input.

\*/

#include <cassert> // Provides assert()

#include <cstdlib> // Provides size\_t

#include <iostream>

#include <iomanip> // Provides setw()

#include <fstream>

#include "p5.h"

using namespace std;

using namespace csci301\_P5\_1;

void displayMenu();

int main()

{

char letter;

int value, numElements, present, kth;

bool empty;

List listOne;

displayMenu(); //Call the function to display the menu

do

{

cout << "--> "; //Issue a prompt

cin >> letter; //Read a command letter

switch (letter) //Carry out the command

{

case 'e':

cout << "The list has been reinitialized to empty.\n";

listOne.make\_empty();

break;

case 'i': cin >> value;

cout << value << " has been inserted into the list.\n";

listOne.insert(value);

break;

case 'r': cin >> value;

cout << value << " has been removed from the list.\n";

listOne.remove(value);

break;

case 'm':

empty = listOne.empty();

if(empty == 1)

cout << "The list is empty.\n";

else

cout << "The list is NOT empty.\n";

break;

case 'l':

numElements = listOne.length();

cout << "There are " << numElements << " elements in the list.\n";

break;

case 'p': cin >> value;

present = listOne.present(value);

if(present == 1)

cout << value << " is in the list.\n";

else

cout << value << " is NOT in the list.\n";

break;

case 'k': cin >> value;

cout << "code to display the " << value << "th value in the list\n";

kth = listOne.kth(value);

cout << "The " << value << "th value of the list is " << kth << ".\n";

break;

case 'w':

cout << "Here are the contents of the list:\n";

cout << listOne << endl;

break;

case 'h': displayMenu();

break;

}

}while(letter != 'q'); // quit program when 'q' is selected

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Precondition: none

//Postcondition: the following code is displayed in the terminal

void displayMenu()

{

cout << endl;

cout << "This program responds to commands the user enters to" << endl;

cout << "manipulate an ordered list of integers, which is" << endl;

cout << "initially empty. In the following commands, k1 is a" << endl;

cout << "position in the list, and v is an integer." << endl;

cout << " e -- Re-initialize the list to be empty." << endl;

cout << " i v -- Insert the value v into the list." << endl;

cout << " r v -- Remove the value v from the list." << endl;

cout << " m -- Is the list empty?" << endl;

cout << " l -- Report the length of the list." << endl;

cout << " p v -- Is the value v present in the list?" << endl;

cout << " k k1 -- Report the k1th value in the list." << endl;

cout << " w -- Write out the list." << endl;

cout << " h -- See this menu." << endl;

cout << " q -- Quit." << endl << endl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

User Document

This program can store a set of integers and perform several operations on the set. The user will enter the proper input and the desired operation will be executed. The user interacts with a menu to manipulate the set of integers.

The program is contained within three files named *p5.h, p5.cpp,* and *p5main.cpp*. To compile these files begin with the command:

*g++ -c p5.cpp*

This will create a file named *p5.o* that contains all the information we need from *p5.h* and *p5.cpp*. Next, link this file with the main client program by entering:

*g++ p5main.cpp p5.o*

Finally, execute the program by entering:

*a.out* or *./a.out*

Once the program is running, you will immediately see the menu and all of the options that you have. The menu looks like this:

This program responds to commands the user enters to

manipulate an ordered list of integers, which is

initially empty. In the following commands, k1 is a

position in the list, and v is an integer.

e -- Re-initialize the list to be empty.

i v -- Insert the value v into the list.

r v -- Remove the value v from the list.

m -- Is the list empty?

l -- Report the length of the list.

p v -- Is the value v present in the list?

k k1 -- Report the k1th value in the list.

w -- Write out the list.

h -- See this menu.

q -- Quit.

--> q

Most commands only require entering a single character, but there are four that also need an integer value following the character. Here are the possible operations to be performed and additional info for each.

* e -- Re-initialize the list to be empty.
  + This will take all values that are currently in the list and remove them. The result will always be an empty list
* i v -- Insert the value v into the list.
  + v should be an integer. If v is already in the list, it will not be added again.
  + Ex: ‘i 32’ would insert the value 32 into the List
* r v -- Remove the value v from the list.
  + v is an integer. If v is not in the list, no change will occur.
  + Ex: ‘r 600’ would remove the value 600 from the List.
* m -- Is the list empty?
* l -- Report the length of the list.
  + The length of the list will be returned as an integer
* p v -- Is the value v present in the list?
  + The program will check if the value v is currently in the list
  + Ex: ‘p 11’ would check if the value 11 is in the list.
* k k1 -- Report the k1th value in the list.
  + k1 must be greater than 0 but less than or equal to the total number of items in the list
  + Ex: ‘k 7’ would return the value located in the 7th position of the list
* w -- Write out the list.
  + The list will be displayed to the terminal. It will not be stored or saved in a file.

Tests

Script started on Thu Feb 15 09:09:09 2018

[?1034hbash-3.2$ pwd

/Users/johnblee/Desktop/CSCI/CSCI 301/Project\_5

bash-3.2$ ls

Project 5 Report.docx p5.o

a.out p5main.cpp

p5.cpp typescript

p5.h ~$oject 5 Report.docx

bash-3.2$ ls

Project 5 Report.docx p5.o

a.out p5main.cpp

p5.cpp typescript

p5.h ~$oject 5 Report.docx

bash-3.2$ cat p5.h

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

typedef statement: this is the data type of the items in the list. In our program it is

integer. Data items are represented by the word 'Item'

CONSTRUCTOR

Precondition: none

Postcondition: List is initialized to its null state. In our case this is an empty list

List()

COPY CONSTRUCTOR

Precondition: none

Postcondition: The new list has been initialized as a copy of the initial list

List(const List& source)

DESTRUCTOR

Never needs to be explicitly called. Destroys Lists when the user/program is done with them

to free up memory

Precondition: none

Postcondition: none

~List()

MUTATOR FUNCTIONS

void make\_empty()

Precondition: none

Postcondition: The list becomes empty

void insert(const Item& entry)

Precondition: entry is not already in the list

Postcondition: the value entry has been inserted into the correct place in the List

void remove(const Item& target)

Precondition: target is actually in the List

Postcondition: the value target is removed from the List

void operator = (const List& source)

Precondition: The given list has been assigned a copy of the source list

CONSTANT MEMBER FUNCTIONS

bool empty() const

Precondition: none

Postcondition: Return TRUE/1 if the given List is empty, return FALSE/0 if the given List

is not empty

size\_t length() const

Precondition: none

Postcondition: The total number of items in the list is returned

bool present(const Item& target) const

Precondition: none

Postcondition: return TRUE/1 if target is actually in the list, return FALSE/0 if target is

not in the list

int kth (int k) const

Precondition: The list cannot be empty. the value k must be at least 1 and smaller than the

total number of items in the list

FRIEND FUNCTION

friend ostream& operator << (ostream& out\_s, const List& b)

Precondition: none

Postcondition: All of the values from List b is written to the output stream

PRIVATE FUNCTION

Node\* get\_node(const Item& entry, Node\* link);

Precondition: The number of the entry must exist

Postcondition: a pointer to a Node is returned

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <string>

#include <cstdlib>

#ifndef P5\_h

#define P5\_h

namespace csci301\_P5\_1

{

class List

{

public:

//this is the data type of the items in the list - int. Data items are

// represented by the word Item

typedef int Item;

//Constructors

List() {first = NULL;}

List(const List& source);

//Destructor

~List();

// Mutator member functions

void make\_empty();

void insert(const Item& entry);

void remove(const Item& target);

void operator = (const List& source);

// Constant member functions

bool empty() const {return first == NULL;}

std::size\_t length() const;

bool present(const Item& target) const;

Item kth(std::size\_t k) const;

// Friend function for the List class

friend std::ostream& operator << (std::ostream& out\_s, const List& l);

private:

// Private data members:

struct Node

{

Item data;

Node \*next;

};

Node \*first;

//Private function:

Node\* get\_node(const Item& entry, Node\* link);

};

}

#endifbash-3.2$ ls

Project 5 Report.docx p5.o

a.out p5main.cpp

p5.cpp typescript

p5.h ~$oject 5 Report.docx

bash-3.2$ cat p5.cpp

/\*

Invariant:

- The items in the List are stored in a linked list in ascending order

- The member variable called 'first' is a pointer that points to the first node in the

Linked List. If it is an empty List, first points to the nullptr.

\*/

#include <cassert> // Provides assert()

#include <cstdlib> // Provides size\_t

#include <iostream>

#include <iomanip> // Provides setw()

#include <fstream>

#include "p5.h"

using namespace std;

namespace csci301\_P5\_1

{

//Copy constructor

List::List(const List& source)

{

Node\* p;

Node\* last;

if(source.first == NULL)

first = NULL;

else

{

first = get\_node(source.first -> data, NULL); //copy the first node

last = first;

p = source.first -> next;

while(p != NULL) //copy remaining Nodes

{

last -> next = get\_node(p -> data, NULL);

last = last -> next;

p = p -> next;

}

}

}

//Destructor

List::~List()

{

Node\* temp;

while(first != NULL)

{

temp = first;

first = first -> next;

delete temp;

}

}

//Mutator member functions

void List::make\_empty()

{

Node\* temp;

while(first != NULL)

{

temp = first;

first = first -> next;

delete temp;

}

}

void List::insert(const Item& entry)

{

Node \*prev;

//assert(!present(entry));

if(first == NULL || entry < first -> data)

first = get\_node(entry, first);

else

{

prev = first;

while(prev->next != NULL && prev->next->data < entry)

prev = prev->next;

prev->next = get\_node(entry, prev->next);

}

}

void List::remove(const Item& target)

{

Node \*temp;

Node \*prev;

//assert(present(target));

prev = first;

if(prev->data == target)

{

first = first->next;

delete prev;

}

else

{

while(prev->next != NULL && prev->next->data < target)

prev = prev->next;

temp = prev->next;

prev->next = temp->next;

delete temp;

}

}

void List::operator = (const List& source)

{

Node\* p;

Node\* last;

if(&source != this)

{

make\_empty();

if(source.first != NULL)

{

first = get\_node(source.first->data, NULL); //copy the first Node

last = first;

p = source.first->next;

while(p != NULL)

{

last->next = get\_node(p->data, NULL);

last = last->next;

p = p->next;

}

}

}

}

// Constant member functions

size\_t List::length() const

{

Node \*cursor;

size\_t count;

count = 0;

for(cursor = first; cursor != NULL; cursor = cursor->next)

count++;

return count;

}

bool List::present(const Item& target) const

{

Node\* cursor;

for(cursor = first;

cursor != NULL && cursor->data != target;

cursor = cursor->next)

;

return(cursor != NULL);

}

List::Item List::kth(size\_t k) const

{

Node \*cursor;

size\_t count;

//assert(1 <= k && k <= length());

cursor = first;

for(count = 1;count < k; count++)

cursor = cursor->next;

return cursor->data;

}

//Friend Function

ostream& operator << (ostream& out\_s, const List& l)

{

List::Node \*cursor;

out\_s << '(';

for(cursor = l.first;

cursor != NULL && cursor->next != NULL;

cursor = cursor->next)

out\_s << cursor->data;

if(cursor != NULL)

out\_s << cursor->data;

out\_s << ')';

return out\_s;

}

//Private Function:

List::Node\* List::get\_node(const Item& entry, Node\* link)

{

Node \*temp;

temp = new Node;

temp -> data = entry;

temp -> next = link;

return temp;

}

}bash-3.2$ ls

Project 5 Report.docx p5.o

a.out p5main.cpp

p5.cpp typescript

p5.h ~$oject 5 Report.docx

bash-3.2$ cat pt[K5main.cpp

// Created By: John Blee

// Due 15 February 2015

// CSCI 301, Dr. Julstrom

// Project 5

/\* This program utilizes a menu to implement a linked list. The List is defined in a class

that is included in this main program. When the class object is defined, the program uses

a do-while loop and switch statement to power the menu. The loop waits for input, and the

switch statement carries out the correct operation based on the user input.

\*/

#include <cassert> // Provides assert()

#include <cstdlib> // Provides size\_t

#include <iostream>

#include <iomanip> // Provides setw()

#include <fstream>

#include "p5.h"

using namespace std;

using namespace csci301\_P5\_1;

void displayMenu();

int main()

{

char letter;

int value, numElements, present, kth;

bool empty;

List listOne;

displayMenu(); //Call the function to display the menu

do

{

cout << "--> "; //Issue a prompt

cin >> letter; //Read a command letter

switch (letter) //Carry out the command

{

case 'e':

cout << "The list has been reinitialized to empty.\n";

listOne.make\_empty();

break;

case 'i': cin >> value;

cout << value << " has been inserted into the list.\n";

listOne.insert(value);

break;

case 'r': cin >> value;

cout << value << " has been removed from the list.\n";

listOne.remove(value);

break;

case 'm':

empty = listOne.empty();

if(empty == 1)

cout << "The list is empty.\n";

else

cout << "The list is NOT empty.\n";

break;

case 'l':

numElements = listOne.length();

cout << "There are " << numElements << " elements in the list.\n";

break;

case 'p': cin >> value;

present = listOne.present(value);

if(present == 1)

cout << value << " is in the list.\n";

else

cout << value << " is NOT in the list.\n";

break;

case 'k': cin >> value;

cout << "code to display the " << value << "th value in the list\n";

kth = listOne.kth(value);

cout << "The " << value << "th value of the list is " << kth << ".\n";

break;

case 'w':

cout << "Here are the contents of the list:\n";

cout << listOne << endl;

break;

case 'h': displayMenu();

break;

}

}while(letter != 'q'); // quit program when 'q' is selected

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Precondition: none

//Postcondition: the following code is displayed in the terminal

void displayMenu()

{

cout << endl;

cout << "This program responds to commands the user enters to" << endl;

cout << "manipulate an ordered list of integers, which is" << endl;

cout << "initially empty. In the following commands, k1 is a" << endl;

cout << "position in the list, and v is an integer." << endl;

cout << " e -- Re-initialize the list to be empty." << endl;

cout << " i v -- Insert the value v into the list." << endl;

cout << " r v -- Remove the value v from the list." << endl;

cout << " m -- Is the list empty?" << endl;

cout << " l -- Report the length of the list." << endl;

cout << " p v -- Is the value v present in the list?" << endl;

cout << " k k1 -- Report the k1th value in the list." << endl;

cout << " w -- Write out the list." << endl;

cout << " h -- See this menu." << endl;

cout << " q -- Quit." << endl << endl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bash-3.2$ ls

Project 5 Report.docx p5.o

a.out p5main.cpp

p5.cpp typescript

p5.h ~$oject 5 Report.docx

bash-3.2$ g++[K[K[Krm[K[Krm p5.o

bash-3.2$ ls

Project 5 Report.docx p5main.cpp

a.out typescript

p5.cpp ~$oject 5 Report.docx

p5.h

bash-3.2$ g++ -c p5.cpp

bash-3.2$ g[Kls

Project 5 Report.docx p5.o

a.out p5main.cpp

p5.cpp typescript

p5.h ~$oject 5 Report.docx

bash-3.2$ g++ p5.s[K[Kmain.cpp p5.o

bash-3.2$ ./a.out

This program responds to commands the user enters to

manipulate an ordered list of integers, which is

initially empty. In the following commands, k1 is a

position in the list, and v is an integer.

e -- Re-initialize the list to be empty.

i v -- Insert the value v into the list.

r v -- Remove the value v from the list.

m -- Is the list empty?

l -- Report the length of the list.

p v -- Is the value v present in the list?

k k1 -- Report the k1th value in the list.

w -- Write out the list.

h -- See this menu.

q -- Quit.

--> w

Here are the contents of the list:

()

--> m

The list is empty.

--> l

There are 0 elements in the list.

--> h

This program responds to commands the user enters to

manipulate an ordered list of integers, which is

initially empty. In the following commands, k1 is a

position in the list, and v is an integer.

e -- Re-initialize the list to be empty.

i v -- Insert the value v into the list.

r v -- Remove the value v from the list.

m -- Is the list empty?

l -- Report the length of the list.

p v -- Is the value v present in the list?

k k1 -- Report the k1th value in the list.

w -- Write out the list.

h -- See this menu.

q -- Quit.

--> i 2

2 has been inserted into the list.

--> w

Here are the contents of the list:

(2)

-->

i 3434

3434 has been inserted into the list.

--> w

Here are the contents of the list:

(2, 3434)

--> i r 3434

3434 has been removed from the list.

--> w

Here are the contents of the list:

(2)

--> i 5

5 has been inserted into the list.

--> i 8

8 has been inserted into the list.

--> i 9

9 has been inserted into the list.

--> i 12

12 has been inserted into the list.

--> w

Here are the contents of the list:

(2, 5, 8, 9, 12)

--> m

The list is NOT empty.

--> l

There are 5 elements in the list.

--> p 2

2 is in the list.

--> p 8

8 is in the list.

--> p 9

9 is in the list.

--> p 12

12 is in the list.

--> k 1

code to display the 1th value in the list

The 1th value of the list is 2.

--> k 2

code to display the 2th value in the list

The 2th value of the list is 5.

--> k 3

code to display the 3th value in the list

The 3th value of the list is 8.

--> k 4

code to display the 4th value in the list

The 4th value of the list is 9.

--> k 5

code to display the 5th value in the list

The 5th value of the list is 12.

--> p r 5

5 has been removed from the list.

--> w

Here are the contents of the list:

(2, 8, 9, 12)

--> r 12

12 has been removed from the list.

--> w

Here are the contents of the list:

(2, 8, 9)

--> r 9

9 has been removed from the list.

--> m

The list is NOT empty.

--> l

There are 2 elements in the list.

--> i e

The list has been reinitialized to empty.

--> w

Here are the contents of the list:

()

--> m

The list is empty.

--> l

There are 0 elements in the list.

--> i -23

-23 has been inserted into the list.

--> w

Here are the contents of the list:

(-23)

--> - i -1

-1 has been inserted into the list.

--> w

Here are the contents of the list:

(-23, -1)

--> - i 0

0 has been inserted into the list.

--> i 100

100 has been inserted into the list.

--> i 9999

9999 has been inserted into the list.

--> w

Here are the contents of the list:

(-23, -1, 0, 100, 9999)

--> m

The list is NOT empty.

--> p 3

3 is NOT in the list.

--> p 23

23 is NOT in the list.

--> p -45

-45 is NOT in the list.

--> l

There are 5 elements in the list.

--> k 1

code to display the 1th value in the list

The 1th value of the list is -23.

--> k 2

code to display the 2th value in the list

The 2th value of the list is -1.

--> k 3

code to display the 3th value in the list

The 3th value of the list is 0.

--> k 4

code to display the 4th value in the list

The 4th value of the list is 100.

--> k 5

code to display the 5th value in the list

The 5th value of the list is 9999.

--> r 0

0 has been removed from the list.

--> w

Here are the contents of the list:

(-23, -1, 100, 9999)

--> l

There are 4 elements in the list.

--> r 9999

9999 has been removed from the list.

--> w

Here are the contents of the list:

(-23, -1, 100)

--> l

There are 3 elements in the list.

--> r -1

-1 has been removed from the list.

--> w

Here are the contents of the list:

(-23, 100)

--> m

The list is NOT empty.

--> l

There are 2 elements in the list.

--> e

The list has been reinitialized to empty.

--> w

Here are the contents of the list:

()

--> h

This program responds to commands the user enters to

manipulate an ordered list of integers, which is

initially empty. In the following commands, k1 is a

position in the list, and v is an integer.

e -- Re-initialize the list to be empty.

i v -- Insert the value v into the list.

r v -- Remove the value v from the list.

m -- Is the list empty?

l -- Report the length of the list.

p v -- Is the value v present in the list?

k k1 -- Report the k1th value in the list.

w -- Write out the list.

h -- See this menu.

q -- Quit.

--> q

bash-3.2$ exit

exit

Script done on Thu Feb 15 09:16:32 2018

Summary

Project dealt with simple singly linked lists again, but this time implemented with a menu. This one was similar to project 4. The structures of the class and main program were intuitive to create because for each operation that the project required there would be one case in the switch statement in the main program, and there would be a function in the class. This made it easy to search and find things in each of the files because they followed the same order of functions/operations.

What other ordered list operations might a class like this one reasonably provide?

Ideas for some other list operations that a class like this might provide are maybe some mathematical operations that could potentially be useful depending on how someone is using the list. These all assume that the data type of the list is int or double or some other numerical data type.

* Find the sum of all of the items in the list
* Find the sum of a given number of items in the list or a given range of items
* Find the difference/product/quotient of given items in the list
* Determine if any of the items in the list are prime
* Find the size of the distance between the values of two items in the list
* Add or remove a value in the kth spot in the list