A Generic Linked List Implementation using Templates

Generated by Doxygen 1.9.7

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 LinkedList< T > Class Template Reference

Public Member Functions

- LinkedList ()
- ~LinkedList ()

  Deconstructor.
- void clear ()

  This function removes all allocated memory used byLinked List.
- void insert (T data)
- void add (T data)
- void remove (T data)
- Node< T > * search (T data)
- std::string toString ()
- void mergeSort ()
- void bubbleSort ()
- void addFromFile (std::string fileName)
- void mergeLists (const LinkedList< T > *listTwo)
- void print ()

  This function prints each Node's data in Linked List to console.
- Node< T > * binarySearch (T target)

3.1.1 Detailed Description

template<class T>
class LinkedList< T >

Definition at line 23 of file LinkedList.hpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 LinkedList()

```
template<class T >
LinkedList< T >::LinkedList
```

Constructor

Definition at line 27 of file LinkedList.cpp.

```
00027                                {
00028     this->head = nullptr;
00029     this->tail = nullptr;
00030 }
```

3.1.2.2 ~LinkedList()

```
template<class T >
LinkedList< T >::~LinkedList
```

Deconstructor.

Definition at line 33 of file LinkedList.cpp.

```
00033                                {
00034     clear();
00035 }
```

3.1.3 Member Function Documentation

3.1.3.1 add()

```
template<class T >
void LinkedList< T >::add (
            T data )
```

This function adds a Node containing the data passed

Parameters

data	- the data a Node contains

Definition at line 80 of file LinkedList.cpp.

```
00080                                    {
00081     Node<T>* newNode = new Node<T>(data);
00082
00083     if (this->head == nullptr) {
00084         this->head = newNode;
00085         this->tail = newNode;
00086     }
00087     else {
00088         this->tail->setNextNode(newNode);
00089         newNode->setPrevNode(tail);
00090         this->tail = newNode;
00091     }
00092 }
```

3.1.3.2 addFromFile()

```
template<class T >
void LinkedList< T >::addFromFile (
            std::string fileName )
```

This function adds data from a txt file into Linked List

Parameters

fileName	- name of the txt file. File name must include .txt extension

Definition at line 282 of file LinkedList.cpp.

```
00282                                                   {
00283     std::ifstream file;
00284     T data;
00285
00286     file.open(fileName);
00287
00288     if (file.peek() == std::ifstream::traits_type::eof()) {
00289         std::cerr « "Error: File is empty" « std::endl;
00290         exit(1);
00291     }
00292
00293     if (!file.is_open()) {
00294         std::cerr « "Error opening file" « std::endl;
00295         exit(1);
00296     }
00297
00298     while (file » data) {
00299         if (file.eof()) {
00300             break;
00301         }
00302
00303         this->add(data);
00304     }
00305
00306     if (!file.eof()) {
00307         std::cerr « "Error reaching end of file" « std::endl;
00308         exit(1);
00309     }
00310
00311     file.close();
00312 }
```

3.1.3.3 binarySearch()

```
template<class T >
Node< T > * LinkedList< T >::binarySearch (
            T target )
```

Searches for a value using binary search. Requires the list to be sorted to work.

Parameters

target	The value to look for.

Definition at line 359 of file LinkedList.cpp.

```
00359                                                   {
00360     Node<T>* searchHead = this->head;
00361     Node<T>* searchTail = this->tail;
00362     Node<T>* searchMid = findMid(searchHead, searchTail);
00363     if (searchHead) {
00364         while (searchHead->getData() <= searchTail->getData()) {
00365             if (target == searchHead->getData()) {
00366                 return searchHead;
```

```
00367             }
00368             else if (target == searchMid->getData()) {
00369                 return searchMid;
00370             }
00371             else if (target == searchTail->getData()) {
00372                 return searchTail;
00373             }
00374 
00375             if (target < searchMid->getData()) {
00376                 searchHead = searchHead->getNextNode();
00377                 searchTail = searchMid->getPrevNode();
00378                 searchMid = findMid(searchHead, searchTail);
00379             }
00380             else if (target < searchMid->getData()) {
00381                 searchHead = searchMid->getNextNode();
00382                 searchTail = searchTail->getPrevNode();
00383                 searchMid = findMid(searchHead, searchTail);
00384             }
00385         }
00386     }
00387 
00388     return nullptr;
00389 }
```

3.1.3.4 bubbleSort()

```
template<class T >
void LinkedList< T >::bubbleSort
```

Sorts the LinkedList using the bubble sort algorithm.

Definition at line 253 of file LinkedList.cpp.

```
00253                                {
00254     // do not sort if empty or one
00255     if (this->head == nullptr || this->head->getNextNode() == nullptr)
00256         return;
00257 
00258     bool swap;
00259     Node<T>* current = this->head;
00260     Node<T>* sorttail = nullptr;
00261 
00262     while (current != sorttail){
00263         swap = false;
00264         Node <T>* current2 = this->head;
00265 
00266         while (current2->getNextNode () != sorttail){
00267             if (current2->getData() > current2->getNextNode()->getData()){
00268                 T temp = current2 -> getData();
00269                 current2->setData(current2->getNextNode()->getData());
00270                 current2->getNextNode()->setData(temp);
00271                 swap = true;
00272             }
00273             current2 = current2->getNextNode();
00274         }
00275         sorttail = current2; // update tail to last swap
00276         if (!swap)
00277             break; // if no swap the list is already sorted
00278     }
00279 }
```

3.1.3.5 clear()

```
template<class T >
void LinkedList< T >::clear
```

This function removes all allocated memory used byLinked List.

Definition at line 38 of file LinkedList.cpp.

```
00038                           {
00039     Node<T>* nodeToDelete = head;
00040     while (head != nullptr) {
00041         head = head->getNextNode();
00042         delete nodeToDelete;
00043         nodeToDelete = head;
00044     }
00045 }
```

3.1.3.6 insert()

```
template<class T >
void LinkedList< T >::insert (
            T data )
```

This function inserts a Node that contains the specific data in Linked List in order

Parameters

data	- the data a Node contains

Definition at line 48 of file LinkedList.cpp.

```
00048                                   {
00049     Node<T>* newNode = new Node<T>(data);
00050     if (this->head == nullptr) {
00051         this->head = newNode;
00052         this->tail = newNode;
00053         return;
00054     }
00055     if (data <= this->head->getData()) {
00056         newNode->setNextNode(head);
00057         this->head->setPrevNode(newNode);
00058         this->head = newNode;
00059         return;
00060     }
00061     if (data >= tail->getData()) {
00062         newNode->setPrevNode(tail);
00063         this->tail->setNextNode(newNode);
00064         this->tail = newNode;
00065         return;
00066     }
00067     Node<T>* temp = this->head;
00068     while (temp->getNextNode() != nullptr && temp->getNextNode()->getData() < data) {
00069         temp = temp->getNextNode();
00070
00071     }
00072
00073     newNode->setNextNode(temp->getNextNode());
00074     temp->setNextNode(newNode);
00075     newNode->setPrevNode(temp);
00076     newNode->getNextNode()->setPrevNode(newNode);
00077 }
```

3.1.3.7 mergeLists()

```
template<class T >
void LinkedList< T >::mergeLists (
            const LinkedList< T > * listTwo )
```

Modified the LinkedList from which it was called. Calling LinkedList will be modified and sorted.

Parameters

listTwo	Does not need to be sorted and remains unchanged.

Definition at line 321 of file LinkedList.cpp.

```
00321                                                   {
00322     if (!this->head && !listTwo->head) {
00323         return;
00324     }
00325     else {
00326         this->mergeSort();
00327         Node<T>* temp = listTwo->head;
00328         while (temp != nullptr) {
00329             this->insert(temp->getData());
```

```
00330             temp = temp->getNextNode();
00331         }
00332     }
00333 }
```

3.1.3.8 mergeSort()

```
template<class T >
void LinkedList< T >::mergeSort
```

Sorts the LinkedList using the merge sort algorithm.

Definition at line 156 of file LinkedList.cpp.

```
00156                                {
00157     // base case: 1 or 0 Nodes
00158     if (this->head == nullptr || this->head->getNextNode() == nullptr) {
00159         return;
00160     }
00161
00162     // split the LinkedList in half
00163     Node<T>* subHead1 = this->head;
00164     Node<T>* subHead2 = findMid(this->head, this->tail);
00165     Node<T>* subTail1 = subHead2->getPrevNode();
00166     Node<T>* subTail2 = this->tail;
00167
00168     // detach the two halves
00169     subTail1->setNextNodeNull();
00170     subHead2->setPrevNodeNull();
00171
00172     // recurse first half
00173     this->head = subHead1;
00174     this->tail = subTail1;
00175     mergeSort();
00176     subHead1 = this->head;
00177     subTail1 = this->tail;
00178
00179     // recurse second half
00180     this->head = subHead2;
00181     this->tail = subTail2;
00182     mergeSort();
00183     subHead2 = this->head;
00184     subTail2 = this->tail;
00185
00186     // merge both halves
00187     this->head = nullptr;
00188     Node<T>* nodeptr = nullptr;
00189
00190     // compare head of both halves
00191     while (subHead1 != nullptr && subHead2 != nullptr) {
00192         if (subHead1->getData() < subHead2->getData()) {
00193             if (this->head == nullptr) {
00194                 this->head = subHead1;
00195                 nodeptr = subHead1;
00196             }
00197             else {
00198                 nodeptr->setNextNode(subHead1);
00199                 subHead1->setPrevNode(nodeptr);
00200                 nodeptr = subHead1;
00201             }
00202             subHead1 = subHead1->getNextNode();
00203         }
00204         else {
00205             if (this->head == nullptr) {
00206                 this->head = subHead2;
00207                 nodeptr = subHead2;
00208             }
00209             else {
00210                 nodeptr->setNextNode(subHead2);
00211                 subHead2->setPrevNode(nodeptr);
00212                 nodeptr = subHead2;
00213             }
00214             subHead2 = subHead2->getNextNode();
00215         }
00216     }
00217
00218     // add the rest of first half to the main LinkedList
00219     while (subHead1 != nullptr) {
00220         if (this->head == nullptr) {
00221             this->head = subHead1;
00222             nodeptr = subHead1;
```

```
00223         }
00224         else {
00225             nodeptr->setNextNode(subHead1);
00226             subHead1->setPrevNode(nodeptr);
00227             nodeptr = subHead1;
00228         }
00229         subHead1 = subHead1->getNextNode();
00230     }
00231
00232     // add the rest of second half to the main LinkedList
00233     while (subHead2 != nullptr) {
00234         if (this->head == nullptr) {
00235             this->head = subHead2;
00236             nodeptr = subHead2;
00237         }
00238         else {
00239             nodeptr->setNextNode(subHead2);
00240             subHead2->setPrevNode(nodeptr);
00241             nodeptr = subHead2;
00242         }
00243         subHead2 = subHead2->getNextNode();
00244     }
00245
00246     this->tail = nodeptr;
00247 }
```

3.1.3.9 print()

```
template<class T >
void LinkedList< T >::print
```

This function prints each Node's data in Linked List to console.

Definition at line 336 of file LinkedList.cpp.

```
00336                           {
00337     Node<T>* temp = this->head;
00338
00339     if (this->head == nullptr) {
00340         std::cout « "The linked list is empty." « std::endl;
00341         return;
00342     }
00343
00344     while (temp != nullptr) {
00345         std:: cout « temp->getData() « " ";
00346         temp = temp->getNextNode();
00347     }
00348
00349     std::cout « std::endl;
00350 }
```

3.1.3.10 remove()

```
template<class T >
void LinkedList< T >::remove (
            T data )
```

This function removes a Node from the Linked List containing the data passed

Parameters

data	- the data a Node contains

Definition at line 95 of file LinkedList.cpp.

```
00095                                 {
00096     Node<T>* nodeToDelete = search(data);
00097     if (nodeToDelete != nullptr) {
00098         if (nodeToDelete == this->head) {
00099             this->head = head->getNextNode();
```

```
00100             this->head->setPrevNodeNull();
00101         }
00102         else if (nodeToDelete == this->tail) {
00103             this->tail = tail->getPrevNode();
00104             this->tail->setNextNodeNull();
00105         }
00106         else {
00107             Node<T>* prevNode = nodeToDelete->getPrevNode();
00108             Node<T>* nextNode = nodeToDelete->getNextNode();
00109             prevNode->setNextNode(nextNode);
00110             nextNode->setPrevNode(prevNode);
00111         }
00112         delete nodeToDelete;
00113     }
00114 }
```

3.1.3.11 search()

```
template<class T >
Node< T > * LinkedList< T >::search (
             T data )
```

This function searches for a Node containing the data passed and returns a reference to Node

Parameters

data	- the data a Node contains

Returns

Node reference to Node that contains data passed, if not found, returns nullptr

Definition at line 117 of file LinkedList.cpp.

```
00117                                     {
00118     Node<T>* temp = this->head;
00119
00120     while (temp) {
00121         if (temp->getData() == data) {
00122             return temp;
00123         }
00124
00125         temp = temp->getNextNode();
00126     }
00127
00128     return nullptr;
00129 }
```

3.1.3.12 toString()

```
template<class T >
std::string LinkedList< T >::toString
```

Returns the string representation of the LinkedList in the form of an array.

Definition at line 136 of file LinkedList.cpp.

```
00136                                     {
00137     Node<T>* temp = this->head;
00138     std::string output = "";
00139     std::string quote = (typeid(T).name() == typeid(std::string("")).name()) ? "\"" : "";
00140
00141     while (temp != nullptr) {
00142         T val = temp->getData();
00143         output += quote + to_string(val) + quote;
00144         temp = temp->getNextNode();
00145         if (temp != nullptr) {
```

```
00146             output += ", ";
00147         }
00148     }
00149     return "{" + output + "}";
00150 }
```

The documentation for this class was generated from the following files:

- LinkedList.hpp
- LinkedList.cpp

3.2 Node< T > Class Template Reference

Public Member Functions

- Node ()
- Node (T data)
- ~Node ()
- T getData ()
- Node * getNextNode ()
- Node * getPrevNode ()
- void setData (T data)
- void setNextNode (Node *next)
- void setPrevNode (Node *prev)
- void setNextNodeNull ()
- void setPrevNodeNull ()

3.2.1 Detailed Description

template<class T>
class Node< T >

Definition at line 19 of file Node.hpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Node() [1/2]

```
template<class T >
Node< T >::Node
```

Node constructor function. Initializes the next and prev to nullptr.

Definition at line 31 of file Node.cpp.

```
00031                 {
00032     this->data = "";
00033     this->next = nullptr;
00034     this->prev = nullptr;
00035 }
```

3.2.2.2 Node() [2/2]

```
template<class T >
Node< T >::Node (
            T data )
```

Node constructor function

Parameters

data	- the data that the node will hold.

Definition at line 42 of file Node.cpp.

```
00042                     {
00043     this->data = data;
00044     this->next = nullptr;
00045     this->prev = nullptr;
00046 }
```

3.2.2.3 ~Node()

```
template<class T >
Node< T >::~Node
```

Node deconstructor function. Resets the next and prev to nullptr.

Definition at line 53 of file Node.cpp.

```
00053                {
00054     this->next = nullptr;
00055     this->prev = nullptr;
00056 }
```

3.2.3 Member Function Documentation

3.2.3.1 getData()

```
template<class T >
T Node< T >::getData
```

Definition at line 60 of file Node.cpp.

```
00060 {return this->data;}
```

3.2.3.2 getNextNode()

```
template<class T >
Node< T > * Node< T >::getNextNode
```

Definition at line 63 of file Node.cpp.

```
00063 {return this->next;}
```

3.2.3.3 getPrevNode()

```
template<class T >
Node< T > * Node< T >::getPrevNode
```

Definition at line 66 of file Node.cpp.

```
00066 {return this->prev;}
```

3.2.3.4 setData()

```
template<class T >
void Node< T >::setData (
             T data )
```

Definition at line 69 of file Node.cpp.
```
00069 {this->data = data;}
```

3.2.3.5 setNextNode()

```
template<class T >
void Node< T >::setNextNode (
             Node< T > * next )
```

Definition at line 72 of file Node.cpp.
```
00072 {this->next = next;}
```

3.2.3.6 setNextNodeNull()

```
template<class T >
void Node< T >::setNextNodeNull
```

Definition at line 78 of file Node.cpp.
```
00078 {this->next = nullptr;}
```

3.2.3.7 setPrevNode()

```
template<class T >
void Node< T >::setPrevNode (
             Node< T > * prev )
```

Definition at line 75 of file Node.cpp.
```
00075 {this->prev = prev;}
```

3.2.3.8 setPrevNodeNull()

```
template<class T >
void Node< T >::setPrevNodeNull
```

Definition at line 81 of file Node.cpp.
```
00081 {this->prev = nullptr;}
```

The documentation for this class was generated from the following files:

- Node.hpp
- Node.cpp

3.3 Vault Class Reference

Public Member Functions

- Vault (int startBal)
- bool operator== (const Vault &r)
- bool operator!= (const Vault &r)
- bool operator< (const Vault &r)
- bool operator> (const Vault &r)
- bool operator<= (const Vault &r)
- bool operator>= (const Vault &r)

Friends

- std::ostream & operator<< (std::ostream &os, const Vault &v)

3.3.1 Detailed Description

Definition at line 18 of file Vault.hpp.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 Vault() [1/2]

```
Vault::Vault ( )
```

Definition at line 20 of file Vault.cpp.

```
00020             {
00021     this->balance = 0;
00022 }
```

3.3.2.2 Vault() [2/2]

```
Vault::Vault (
             int startBal )
```

Definition at line 24 of file Vault.cpp.

```
00024                         {
00025     this->balance = startBal;
00026 }
```

3.3.2.3 ~Vault()

```
Vault::~Vault ( )
```

Definition at line 28 of file Vault.cpp.

```
00028 {}
```

3.3.3 Member Function Documentation

3.3.3.1 operator"!=()

```
bool Vault::operator!= (
            const Vault & r )
```

Definition at line 34 of file Vault.cpp.

```
00034                                    {
00035     return this->balance != r.balance;
00036 }
```

3.3.3.2 operator<()

```
bool Vault::operator< (
            const Vault & r )
```

Definition at line 38 of file Vault.cpp.

```
00038                                   {
00039     return this->balance < r.balance;
00040 }
```

3.3.3.3 operator<=()

```
bool Vault::operator<= (
            const Vault & r )
```

Definition at line 46 of file Vault.cpp.

```
00046                                    {
00047     return this->balance <= r.balance;
00048 }
```

3.3.3.4 operator==()

```
bool Vault::operator== (
            const Vault & r )
```

Definition at line 30 of file Vault.cpp.

```
00030                                    {
00031     return this->balance == r.balance;
00032 }
```

3.3.3.5 operator>()

```
bool Vault::operator> (
            const Vault & r )
```

Definition at line 42 of file Vault.cpp.

```
00042                                   {
00043     return this->balance > r.balance;
00044 }
```

3.3.3.6 operator>=()

```
bool Vault::operator>= (
            const Vault & r )
```

Definition at line 50 of file Vault.cpp.

```
00050                                       {
00051     return this->balance >= r.balance;
00052 };
```

3.3.4 Friends And Related Symbol Documentation

3.3.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & os,
            const Vault & v )  [friend]
```

Definition at line 33 of file Vault.hpp.

```
00033                                                                    {
00034         os « "Vault with balance: " « v.balance;
00035         return os;
00036     }
```

The documentation for this class was generated from the following files:

- Vault.hpp
- Vault.cpp

Chapter 4

File Documentation

4.1 FunctionTests.hpp

```
00001 /*********************************************
00002  *  @file FunctionTests.hpp
00003  *  @brief Spring 2023 - CSC340.05 Final Project
00004  *
00005  *  This is the final project to make a custom
00006  *  LinkedList and Node class project for
00007  *  Spring 2023 - CSC340.05
00008  *
00009  *  @author Ashley Ching
00010  *  @author Charlene Breanne Calderon
00011  *  @author Eduardo Loza
00012  *  @author Lennart Richter
00013  *********************************************/
00014
00015 #ifndef FunctionTests_hpp
00016 #define FunctionTests_hpp
00017
00018 #include "Vault.hpp"
00019
00020 void testAddRemove();
00021 void testSearch();
00022 void emptyMerge();
00023 void callingListEmptyMerge();
00024 void paramterListEmptyMerge();
00025 void twoNonEmptyMerge();
00026 void testMerge();
00027 void testMergeSort();
00028 void testAddFromFile();
00029 void testBubbleSort();
00030 void testInsert();
00031 void testBinarySearch();
00032 void testLinkedList();
00033 void demo();
00034 template<class T>
00035 bool assertion(T actual, T expected);
00036 template<class T>
00037 bool assertion(T actual, T expected, bool message);
00038
00042 void testAddRemove() {
00043     unsigned int i;
00044     LinkedList<int> intList = LinkedList<int>();
00045     LinkedList<std::string> stringList = LinkedList<std::string>();
00046
00047     for (i = 1; i <= 20; i++) {
00048         intList.add(i);
00049         stringList.add("Number " + std::to_string(i));
00050     }
00051
00052     assertion(intList.toString(), std::string("{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
      17, 18, 19, 20}"));
00053     std::cout « "Added: ";
00054     std::cout « intList.toString() « std::endl;
00055
00056     assertion(stringList.toString(), std::string("{\"Number 1\", \"Number 2\", \"Number 3\", \"Number
      4\", \"Number 5\", "
00057                                                  "\"Number 6\", \"Number 7\", \"Number 8\", \"Number
      9\", \"Number 10\", \"Number 11\", \"Number 12\", \"Number 13\", "
```

```
00058                                                 "\"Number 14\", \"Number 15\", \"Number 16\",
      \"Number 17\", \"Number 18\", \"Number 19\", \"Number 20\"}"));
00059     std::cout « "Added: ";
00060     std::cout « stringList.toString() « std::endl;
00061 
00062     for (i = 1; i <= 20; i += 2) {
00063         intList.remove(i);
00064         stringList.remove("Number " + std::to_string(i));
00065     }
00066 
00067     assertion(intList.toString(), std::string("{2, 4, 6, 8, 10, 12, 14, 16, 18, 20}"));
00068     std::cout « "Removed: ";
00069     std::cout « intList.toString() « std::endl;
00070 
00071     assertion(stringList.toString(), std::string("{\"Number 2\", \"Number 4\", \"Number 6\", \"Number
      8\", \"Number 10\", "
00072                                                 "\"Number 12\", \"Number 14\", \"Number 16\",
      \"Number 18\", \"Number 20\"}"));
00073     std::cout « "Removed: ";
00074     std::cout « stringList.toString() « std::endl;
00075 }
00076 
00080 void testSearch() {
00081     LinkedList<int> intListTest = LinkedList<int>();
00082     Node<int>* result;
00083 
00084     std::cout « "[\033[0;36m----\033[0m] ";
00085     intListTest.print();
00086     result = intListTest.search(3);
00087     assertion(result, static_cast<Node<int>*>(nullptr));
00088     std::cout « "Empty Search -> ";
00089     std::cout « "Searching for 3: ";
00090     if (result) {
00091         std::cout « "Node was found" « std::endl;
00092     }
00093     else {
00094         std::cout « "Node not found" « std::endl;
00095     }
00096 
00097     intListTest.add(1);
00098     std::cout « "[\033[0;36m----\033[0m] ";
00099     intListTest.print();
00100     result = intListTest.search(1);
00101     assertion(result->getData(), 1);
00102     std::cout « "One Item Search -> ";
00103     std::cout « "Searching for 1: ";
00104     if (result) {
00105         std::cout « "Node was found" « std::endl;
00106     }
00107     else {
00108         std::cout « "Node not found" « std::endl;
00109     }
00110 
00111     std::cout « "[\033[0;36m----\033[0m] ";
00112     intListTest.print();
00113     result = intListTest.search(2);
00114     assertion(result, static_cast<Node<int>*>(nullptr));
00115     std::cout « "One Item Search -> ";
00116     std::cout « "Searching for 2: ";
00117     if (result) {
00118         std::cout « "Node was found" « std::endl;
00119     }
00120     else {
00121         std::cout « "Node not found" « std::endl;
00122     }
00123 
00124     intListTest.add(10);
00125     intListTest.add(25);
00126     intListTest.add(30);
00127 
00128     std::cout « "[\033[0;36m----\033[0m] ";
00129     intListTest.print();
00130     result = intListTest.search(30);
00131     assertion(result->getData(), 30);
00132     std::cout « "Multiple Item Search -> ";
00133     std::cout « "Searching for 30: ";
00134     if (result) {
00135         std::cout « "Node was found" « std::endl;
00136     }
00137     else {
00138         std::cout « "Node not found" « std::endl;
00139     }
00140 
00141     std::cout « "[\033[0;36m----\033[0m] ";
00142     intListTest.print();
00143     result = intListTest.search(10000);
00144     assertion(result, static_cast<Node<int>*>(nullptr));
```

```
00145     std::cout « "Multiple Item Search -> ";
00146     std::cout « "Searching for 10000: ";
00147     if (result) {
00148         std::cout « "Node was found" « std::endl;
00149     }
00150     else {
00151         std::cout « "Node not found" « std::endl;
00152     }
00153 }
00154
00155
00159 void emptyMerge() {
00160     LinkedList<int> list1 = LinkedList<int>();
00161     LinkedList<int> list2 = LinkedList<int>();
00162     list1.mergeLists(&list2);
00163     std::string expect_output = "{}";
00164     std::string output = list1.toString();
00165     assertion(output, expect_output);
00166     std::cout « "Empty Merge Output: " « output « std::endl;
00167 }
00168
00172 void callingListEmptyMerge() {
00173     LinkedList<int> list1 = LinkedList<int>();
00174     LinkedList<int> list2 = LinkedList<int>();
00175     list2.add(25);
00176     list1.mergeLists(&list2);
00177     std::string expect_output = "{25}";
00178     std::string output = list1.toString();
00179     assertion(output, expect_output);
00180     std::cout « "Calling List Empty Output: " « output « std::endl;
00181 }
00182
00183
00187 void paramterListEmptyMerge() {
00188     LinkedList<int> list1 = LinkedList<int>();
00189     LinkedList<int> list2 = LinkedList<int>();
00190     list1.add(25);
00191     list1.mergeLists(&list2);
00192     std::string expect_output = "{25}";
00193     std::string output = list1.toString();
00194     assertion(output, expect_output);
00195     std::cout « "Parameter List Empty Output: " « output « std::endl;
00196 }
00197
00201 void twoNonEmptyMerge() {
00202     LinkedList<int> list1 = LinkedList<int>();
00203     LinkedList<int> list2 = LinkedList<int>();
00204     list1.add(65);
00205     list1.add(25);
00206     list1.add(35);
00207     list2.add(45);
00208     list2.add(90);
00209     list2.add(10);
00210     list1.mergeLists(&list2);
00211     std::string expect_output = "{10, 25, 35, 45, 65, 90}";
00212     std::string output = list1.toString();
00213     assertion(output, expect_output);
00214     std::cout « "Two Non Empty List Output: " « output « std::endl;
00215 }
00216
00220 void testMerge() {
00221     emptyMerge();
00222     callingListEmptyMerge();
00223     paramterListEmptyMerge();
00224     twoNonEmptyMerge();
00225 }
00226
00230 void testMergeSort() {
00231     std::string stringItems[26] = {
00232         "Quebec", "Victor", "November", "Mike", "Charlie", "X-Ray",
00233         "Zulu", "Yankee", "Juliett", "Uniform", "Oscar", "Lima", "Romeo",
00234         "Bravo", "Tango", "Kilo", "Foxtrot", "India", "Delta", "Sierra",
00235         "Golf", "Alpha", "Papa", "Echo", "Hotel", "Whiskey"
00236     };
00237     LinkedList<std::string> stringList = LinkedList<std::string>();
00238
00239     for (unsigned int i = 0; i < 26; ++i) {
00240         stringList.add(stringItems[i]);
00241     }
00242
00243     int intItems[26] = {23, 1, 21, 5, 4, 17, 15, 13, 3, 2, 12, 19, 6, 10, 20, 26, 18, 9, 25, 24, 16,
    14, 11, 22, 8, 7};
00244     LinkedList<int> intList = LinkedList<int>();
00245
00246     for (unsigned int i = 0; i < 26; ++i) {
00247         intList.add(intItems[i]);
00248     }
```

```
00249
00250     std::cout « "[\033[0;36m----\033[0m] ";
00251     std::cout « "Before Sort: ";
00252     stringList.print();
00253
00254     stringList.mergeSort();
00255
00256     assertion(stringList.toString(), std::string("{\"Alpha\", \"Bravo\", \"Charlie\", \"Delta\",
      \"Echo\", \"Foxtrot\", \"Golf\", "
00257                                                   "\"Hotel\", \"India\", \"Juliett\", \"Kilo\",
      \"Lima\", \"Mike\", \"November\", \"Oscar\", \"Papa\", \"Quebec\", \"Romeo\", "
00258                                                   "\"Sierra\", \"Tango\", \"Uniform\", \"Victor\",
      \"Whiskey\", \"X-Ray\", \"Yankee\", \"Zulu\"}"));
00259     std::cout « "After Sort: ";
00260     stringList.print();
00261
00262     std::cout « std::endl;
00263
00264     std::cout « "[\033[0;36m----\033[0m] ";
00265     std::cout « "Before Sort: ";
00266     intList.print();
00267
00268     intList.mergeSort();
00269
00270     assertion(intList.toString(), std::string("{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
      17, 18, 19, 20, 21, 22, 23, 24, 25, 26}"));
00271     std::cout « "After Sort: ";
00272     intList.print();
00273 }
00274
00275
00276 void testAddFromFile() {
00277     std::string fileName = "file333.txt";
00278     LinkedList<std::string>* list = new LinkedList<std::string>;
00279     Node<std::string>* node = new Node<std::string>;
00280
00281     list->add("The");
00282     list->add("best");
00283     list->add("team");
00284     list->add("in");
00285     list->add("CSC340");
00286
00287     std::cout « "Before adding data from file: " « list->toString() « std::endl;
00288
00289     list->addFromFile(fileName);
00290
00291     std::cout « "After adding data from file: " « list->toString() « std::endl;
00292
00293     node = list->search("Dummy");
00294
00295     if (node->getNextNode()->getData() == "Text") {
00296         std::cout « "Test passed" « std::endl;
00297     }
00298     else {
00299         std::cerr « "Test failed" « std::endl;
00300     }
00301 }
00302
00303 std::string read_file() {
00304     std::string data = "";
00305     std::ifstream file;
00306     std::string line = "";
00307     file.open("file333.txt");
00308
00309     if (!file.is_open()) {
00310         std::cerr « "Error opening file" « std::endl;
00311     }
00312
00313     while (file » line) {
00314         data += line;
00315     }
00316
00317     file.close();
00318
00319     return data;
00320 }
00321
00325 void testBubbleSort() {
00326     LinkedList<int> intList;
00327     LinkedList<std::string> stringList;
00328
00329     int intItems[10] = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3};
00330     for (unsigned int i = 0; i < 10; ++i) {
00331         intList.add(intItems[i]);
00332     }
00333
00334     std::cout « "[\033[0;36m----\033[0m] ";
```

```
00335     std::cout « "Before Sorting: " « intList.toString() « std::endl;
00336     intList.bubbleSort();
00337     assertion(intList.toString(), std::string("{1, 1, 2, 3, 3, 4, 5, 5, 6, 9}"));
00338     std::cout « "After Sorting: " « intList.toString() « std::endl;
00339 
00340     std::string stringItems[4] = {"zzz", "bbb", "eee", "ddd"};
00341     for (unsigned int i = 0; i < 4; ++i) {
00342         stringList.add(stringItems[i]);
00343     }
00344 
00345     std::cout « "[\033[0;36m----\033[0m] ";
00346     std::cout « "Before Sorting: " « stringList.toString() « std::endl;
00347     stringList.bubbleSort();
00348     assertion(stringList.toString(), std::string("{\"bbb\", \"ddd\", \"eee\", \"zzz\"}"));
00349     std::cout « "After Sorting: " « stringList.toString() « std::endl;
00350 }
00351 
00359 template<class T>
00360 bool assertion(T actual, T expected) {
00361     if (actual == expected) {
00362         std::cout « "[\033[0;32mPass\033[0m] ";
00363         return true;
00364     }
00365     else {
00366         std::cout « "[\033[0;31mFail\033[0m] ";
00367         std::cout « "Expected: " « expected « std::endl;
00368         std::cout « "[\033[0;31m--->\033[0m] ";
00369         std::cout « "  Actual: " « actual « std::endl;
00370         std::cout « "[\033[0;36m----\033[0m] ";
00371         return false;
00372     }
00373 }
00374 
00375 template<class T>
00376 bool assertion(T actual, T expected, bool message) {
00377     if (message) {
00378         return assertion(actual, expected);
00379     }
00380     else {
00381         return actual == expected;
00382     }
00383 }
00384 
00388 void testInsert() {
00389     LinkedList<int> intList;
00390     LinkedList <std::string> stringList;
00391 
00392     // pi
00393     int intItems[9] = {3, 1, 4, 1, 5, 9, 2, 6, 5};
00394     for (unsigned int i = 0 ; i < 9; ++i) {
00395         intList.insert(intItems[i]);
00396         std::cout « "[\033[0;36m----\033[0m] ";
00397         intList.print();
00398     }
00399 
00400     intList.insert(3);
00401     assertion(intList.toString(), std::string("{1, 1, 2, 3, 3, 4, 5, 5, 6, 9}"));
00402     intList.print();
00403 
00404     std::string stringItems[2] = {"pug", "bear"};
00405     for (unsigned int i = 0; i < 2; ++i) {
00406         stringList.insert(stringItems[i]);
00407         std::cout « "[\033[0;36m----\033[0m] ";
00408         stringList.print();
00409     }
00410 
00411     stringList.insert("zebra");
00412     assertion(stringList.toString(), std::string("{\"bear\", \"pug\", \"zebra\"}"));
00413     stringList.print();
00414 }
00415 
00419 void testBinarySearch() {
00420     LinkedList<int> listForSearch = LinkedList<int>();
00421     Node<int>* result;
00422 
00423     std::cout « "[\033[0;36m----\033[0m] ";
00424     listForSearch.print();
00425     result = listForSearch.binarySearch(2);
00426     assertion(result, static_cast<Node<int>*>(nullptr));
00427     std::cout « "Empty Binary Search -> ";
00428     std::cout « "Searching for 2: ";
00429     if (result) {
00430         std::cout « "Node was found" « std::endl;
00431     }
00432     else {
00433         std::cout « "Node not found" « std::endl;
00434     }
```

```
00435
00436     listForSearch.add(2);
00437     std::cout « "[\033[0;36m----\033[0m] ";
00438     listForSearch.print();
00439     result = listForSearch.binarySearch(2);
00440     assertion(result->getData(), 2);
00441     std::cout « "One Item Binary Search -> ";
00442     std::cout « "Searching for 2: ";
00443     if (result) {
00444         std::cout « "Node was found" « std::endl;
00445     }
00446     else {
00447         std::cout « "Node not found" « std::endl;
00448     }
00449
00450     listForSearch.insert(1);
00451     std::cout « "[\033[0;36m----\033[0m] ";
00452     listForSearch.print();
00453     result = listForSearch.binarySearch(1);
00454     assertion(result->getData(), 1);
00455     std::cout « "Two Item List Binary Search - First Node -> ";
00456     std::cout « "Searching for 1: ";
00457     if (result) {
00458         std::cout « "Node was found" « std::endl;
00459     }
00460     else {
00461         std::cout « "Node not found" « std::endl;
00462     }
00463
00464     listForSearch.insert(10);
00465     std::cout « "[\033[0;36m----\033[0m] ";
00466     listForSearch.print();
00467     result = listForSearch.binarySearch(10);
00468     assertion(result->getData(), 10);
00469     std::cout « "Tree Item List Binary Search - Last Node -> ";
00470     std::cout « "Searching for 10: ";
00471     if (result) {
00472         std::cout « "Node was found with a " « result->getData() « std::endl;
00473     }
00474     else {
00475         std::cout « "Node not found" « std::endl;
00476     }
00477 }
00478
00482 void testLinkedList() {
00483     std::cout « " -- Add and Remove Node Test -- " « std::endl;
00484     testAddRemove();
00485     std::cout « std::endl;
00486
00487     std::cout « " -- Insert Node Test -- " « std::endl;
00488     testInsert();
00489     std::cout « std::endl;
00490
00491     std::cout « " -- Search For Node Test -- " « std::endl;
00492     testSearch();
00493     std::cout « std::endl;
00494
00495     std::cout « " -- Binary Search Test -- " « std::endl;
00496     testBinarySearch();
00497     std::cout « std::endl;
00498
00499     std::cout « " -- Merge LinkedLists Test -- " « std::endl;
00500     testMerge();
00501     std::cout « std::endl;
00502
00503     std::cout « " -- Bubble Sort Test -- " « std::endl;
00504     testBubbleSort();
00505     std:: cout « std::endl;
00506
00507     std::cout « " -- Merge Sort Test -- " « std::endl;
00508     testMergeSort();
00509     std:: cout « std::endl;
00510 }
00511
00512 void demo() {
00513
00514     LinkedList<Vault> BankSystem = LinkedList<Vault>();
00515     LinkedList<Vault> BankSystem2 = LinkedList<Vault>();
00516     BankSystem2.add(Vault(100));
00517     BankSystem2.add(Vault(100000));
00518     BankSystem2.add(Vault(13));
00519     int choice = 0;
00520     while (choice != -1) {
00521         std::cout « "Banking System Main Menu: " « std::endl;
00522         std::cout « "Enter '1' to print the current Bank System" « std::endl;
00523         std::cout « "Enter '2' to add a vault to the current system" « std::endl;
00524         std::cout « "Enter '3' to search for a vault" « std::endl;
```

```
00525         std::cout « "Enter '4' to do a binary search must be sorted" « std::endl;
00526         std::cout « "Enter '5' to bubble sort the Bank System" « std::endl;
00527         std::cout « "Enter '6' to merge sort the Bank System" « std::endl;
00528         std::cout « "Enter '7' to merge another Bank System into this one" « std::endl;
00529         std::cout « "Enter '8' to remove a vault from the Bank System" « std::endl;
00530         std::cout « "Enter '-1' to exit the management system" « std::endl;
00531         std::cout « "Console: ";
00532         std::cin » choice;
00533         if (system("cls")) system("clear");
00534         switch (choice) {
00535             case 1:
00536                 std::cout « "Current Bank System: ";
00537                 BankSystem.print();
00538                 std::cout « std::endl;
00539                 break;
00540             case 2:
00541                 int toAdd;
00542                 std::cout « "Enter the value of the new vault to add to the system: ";
00543                 std::cin » toAdd;
00544                 BankSystem.add(Vault(toAdd));
00545                 std::cout « "Added" « std::endl;
00546                 break;
00547             case 3:
00548                 int searchTarget;
00549                 std::cout « "Enter the value of the target vault: ";
00550                 std::cin » searchTarget;
00551                 if (BankSystem.search(searchTarget)) {
00552                     std::cout « "Vault located" « std::endl;
00553                 }
00554                 else {
00555                     std::cout « "Vault not located" « std::endl;
00556                 }
00557                 break;
00558             case 4:
00559                 int binaryTarget;
00560                 std::cout « "Enter the value of the target vault: ";
00561                 std::cin » binaryTarget;
00562                 if (BankSystem.binarySearch(binaryTarget)) {
00563                     std::cout « "Vault located" « std::endl;
00564                 }
00565                 else {
00566                     std::cout « "Vault not located" « std::endl;
00567                 }
00568                 break;
00569             case 5:
00570                 std::cout « "Bubble Sorting" « std::endl;
00571                 BankSystem.bubbleSort();
00572                 std::cout « "Sorted!" « std::endl;
00573                 break;
00574             case 6:
00575                 std::cout « "Merge Sorting" « std::endl;
00576                 BankSystem.mergeSort();
00577                 std::cout « "Sorted!" « std::endl;
00578                 break;
00579             case 7:
00580                 std::cout « "Merging other Bank System" « std::endl;
00581                 std::cout « "Other system: ";
00582                 BankSystem2.print();
00583                 BankSystem.mergeLists(&BankSystem2);
00584                 std::cout « "New Merge System: ";
00585                 BankSystem.print();
00586                 break;
00587             case 8:
00588                 std::cout « "Enter the value of the vault to be removed: ";
00589                 int vaultVal;
00590                 std::cin » vaultVal;
00591                 if (BankSystem.search(vaultVal)) {
00592                     BankSystem.remove(vaultVal);
00593                     std::cout « "Removed Vault with value " « vaultVal « std::endl;
00594                 }
00595                 else {
00596                     std::cout « "That vault does not exist" « std::endl;
00597                 }
00598 
00599                 break;
00600             case -1:
00601                 std::cout « "Thank you for using the Banking System" « std::endl;
00602                 break;
00603             default:
00604                 std::cout « "Enter a value between 1 and 6 or -1";
00605         }
00606     }
00607 }
00608 
00609 #endif /* FunctionTests_hpp */
```

4.2 LinkedList.cpp

```
00001 /*******************************************
00002  *  @file LinkedList.cpp
00003  *  @brief Spring 2023 - CSC340.05 Final Project
00004  *
00005  *  This is the final project to make a custom
00006  *  LinkedList and Node class project for
00007  *  Spring 2023 - CSC340.05
00008  *
00009  *  @author Ashley Ching
00010  *  @author Charlene Breanne Calderon
00011  *  @author Eduardo Loza
00012  *  @author Lennart Richter
00013  *******************************************/
00014
00021 #ifndef LINKEDLIST_CPP
00022 #define LINKEDLIST_CPP
00023 #include <fstream>
00024 #include "LinkedList.hpp"
00025
00026 template<class T>
00027 LinkedList<T>::LinkedList(){
00028     this->head = nullptr;
00029     this->tail = nullptr;
00030 }
00031
00032 template<class T>
00033 LinkedList<T>::~LinkedList() {
00034     clear();
00035 }
00036
00037 template<class T>
00038 void LinkedList<T>::clear() {
00039     Node<T>* nodeToDelete = head;
00040     while (head != nullptr) {
00041         head = head->getNextNode();
00042         delete nodeToDelete;
00043         nodeToDelete = head;
00044     }
00045 }
00046
00047 template<class T>
00048 void LinkedList<T>::insert(T data) {
00049     Node<T>* newNode = new Node<T>(data);
00050     if (this->head == nullptr) {
00051         this->head = newNode;
00052         this->tail = newNode;
00053         return;
00054     }
00055     if (data <= this->head->getData()) {
00056         newNode->setNextNode(head);
00057         this->head->setPrevNode(newNode);
00058         this->head = newNode;
00059         return;
00060     }
00061     if (data >= tail->getData()) {
00062         newNode->setPrevNode(tail);
00063         this->tail->setNextNode(newNode);
00064         this->tail = newNode;
00065         return;
00066     }
00067     Node<T>* temp = this->head;
00068     while (temp->getNextNode() != nullptr && temp->getNextNode()->getData() < data) {
00069         temp = temp->getNextNode();
00070
00071     }
00072
00073     newNode->setNextNode(temp->getNextNode());
00074     temp->setNextNode(newNode);
00075     newNode->setPrevNode(temp);
00076     newNode->getNextNode()->setPrevNode(newNode);
00077 }
00078
00079 template<class T>
00080 void LinkedList<T>::add(T data) {
00081     Node<T>* newNode = new Node<T>(data);
00082
00083     if (this->head == nullptr) {
00084         this->head = newNode;
00085         this->tail = newNode;
00086     }
00087     else {
00088         this->tail->setNextNode(newNode);
00089         newNode->setPrevNode(tail);
00090         this->tail = newNode;
00091     }
```

```
00092 }
00093
00094 template<class T>
00095 void LinkedList<T>::remove(T data) {
00096     Node<T>* nodeToDelete = search(data);
00097     if (nodeToDelete != nullptr) {
00098         if (nodeToDelete == this->head) {
00099             this->head = head->getNextNode();
00100             this->head->setPrevNodeNull();
00101         }
00102         else if (nodeToDelete == this->tail) {
00103             this->tail = tail->getPrevNode();
00104             this->tail->setNextNodeNull();
00105         }
00106         else {
00107             Node<T>* prevNode = nodeToDelete->getPrevNode();
00108             Node<T>* nextNode = nodeToDelete->getNextNode();
00109             prevNode->setNextNode(nextNode);
00110             nextNode->setPrevNode(prevNode);
00111         }
00112         delete nodeToDelete;
00113     }
00114 }
00115
00116 template<class T>
00117 Node<T>* LinkedList<T>::search(T data) {
00118     Node<T>* temp = this->head;
00119
00120     while (temp) {
00121         if (temp->getData() == data) {
00122             return temp;
00123         }
00124
00125         temp = temp->getNextNode();
00126     }
00127
00128     return nullptr;
00129 }
00130
00135 template<class T>
00136 std::string LinkedList<T>::toString() {
00137     Node<T>* temp = this->head;
00138     std::string output = "";
00139     std::string quote = (typeid(T).name() == typeid(std::string("")).name()) ? "\"" : "";
00140
00141     while (temp != nullptr) {
00142         T val = temp->getData();
00143         output += quote + to_string(val) + quote;
00144         temp = temp->getNextNode();
00145         if (temp != nullptr) {
00146             output += ", ";
00147         }
00148     }
00149     return "{" + output + "}";
00150 }
00151
00155 template<class T>
00156 void LinkedList<T>::mergeSort() {
00157     // base case: 1 or 0 Nodes
00158     if (this->head == nullptr || this->head->getNextNode() == nullptr) {
00159         return;
00160     }
00161
00162     // split the LinkedList in half
00163     Node<T>* subHead1 = this->head;
00164     Node<T>* subHead2 = findMid(this->head, this->tail);
00165     Node<T>* subTail1 = subHead2->getPrevNode();
00166     Node<T>* subTail2 = this->tail;
00167
00168     // detach the two halves
00169     subTail1->setNextNodeNull();
00170     subHead2->setPrevNodeNull();
00171
00172     // recurse first half
00173     this->head = subHead1;
00174     this->tail = subTail1;
00175     mergeSort();
00176     subHead1 = this->head;
00177     subTail1 = this->tail;
00178
00179     // recurse second half
00180     this->head = subHead2;
00181     this->tail = subTail2;
00182     mergeSort();
00183     subHead2 = this->head;
00184     subTail2 = this->tail;
00185
```

```
00186     // merge both halves
00187     this->head = nullptr;
00188     Node<T>* nodeptr = nullptr;
00189 
00190     // compare head of both halves
00191     while (subHead1 != nullptr && subHead2 != nullptr) {
00192         if (subHead1->getData() < subHead2->getData()) {
00193             if (this->head == nullptr) {
00194                 this->head = subHead1;
00195                 nodeptr = subHead1;
00196             }
00197             else {
00198                 nodeptr->setNextNode(subHead1);
00199                 subHead1->setPrevNode(nodeptr);
00200                 nodeptr = subHead1;
00201             }
00202             subHead1 = subHead1->getNextNode();
00203         }
00204         else {
00205             if (this->head == nullptr) {
00206                 this->head = subHead2;
00207                 nodeptr = subHead2;
00208             }
00209             else {
00210                 nodeptr->setNextNode(subHead2);
00211                 subHead2->setPrevNode(nodeptr);
00212                 nodeptr = subHead2;
00213             }
00214             subHead2 = subHead2->getNextNode();
00215         }
00216     }
00217 
00218     // add the rest of first half to the main LinkedList
00219     while (subHead1 != nullptr) {
00220         if (this->head == nullptr) {
00221             this->head = subHead1;
00222             nodeptr = subHead1;
00223         }
00224         else {
00225             nodeptr->setNextNode(subHead1);
00226             subHead1->setPrevNode(nodeptr);
00227             nodeptr = subHead1;
00228         }
00229         subHead1 = subHead1->getNextNode();
00230     }
00231 
00232     // add the rest of second half to the main LinkedList
00233     while (subHead2 != nullptr) {
00234         if (this->head == nullptr) {
00235             this->head = subHead2;
00236             nodeptr = subHead2;
00237         }
00238         else {
00239             nodeptr->setNextNode(subHead2);
00240             subHead2->setPrevNode(nodeptr);
00241             nodeptr = subHead2;
00242         }
00243         subHead2 = subHead2->getNextNode();
00244     }
00245 
00246     this->tail = nodeptr;
00247 }
00248 
00252 template<class T>
00253 void LinkedList<T>::bubbleSort() {
00254     // do not sort if empty or one
00255     if (this->head == nullptr || this->head->getNextNode() == nullptr)
00256         return;
00257 
00258     bool swap;
00259     Node<T>* current = this->head;
00260     Node<T>* sorttail = nullptr;
00261 
00262     while (current != sorttail){
00263         swap = false;
00264         Node <T>* current2 = this->head;
00265 
00266         while (current2->getNextNode () != sorttail){
00267             if (current2->getData() > current2->getNextNode()->getData()){
00268                 T temp = current2 -> getData();
00269                 current2->setData(current2->getNextNode()->getData());
00270                 current2->getNextNode()->setData(temp);
00271                 swap = true;
00272             }
00273             current2 = current2->getNextNode();
00274         }
00275         sorttail = current2; // update tail to last swap
```

```
00276         if (!swap)
00277             break; // if no swap the list is already sorted
00278     }
00279 }
00280
00281 template<class T>
00282 void LinkedList<T>::addFromFile (std::string fileName) {
00283     std::ifstream file;
00284     T data;
00285
00286     file.open(fileName);
00287
00288     if (file.peek() == std::ifstream::traits_type::eof()) {
00289         std::cerr « "Error: File is empty" « std::endl;
00290         exit(1);
00291     }
00292
00293     if (!file.is_open()) {
00294         std::cerr « "Error opening file" « std::endl;
00295         exit(1);
00296     }
00297
00298     while (file » data) {
00299         if (file.eof()) {
00300             break;
00301         }
00302
00303         this->add(data);
00304     }
00305
00306     if (!file.eof()) {
00307         std::cerr « "Error reaching end of file" « std::endl;
00308         exit(1);
00309     }
00310
00311     file.close();
00312 }
00313
00320 template<class T>
00321 void LinkedList<T>::mergeLists(const LinkedList<T>* listTwo) {
00322     if (!this->head && !listTwo->head) {
00323         return;
00324     }
00325     else {
00326         this->mergeSort();
00327         Node<T>* temp = listTwo->head;
00328         while (temp != nullptr) {
00329             this->insert(temp->getData());
00330             temp = temp->getNextNode();
00331         }
00332     }
00333 }
00334
00335 template<class T>
00336 void LinkedList<T>::print() {
00337     Node<T>* temp = this->head;
00338
00339     if (this->head == nullptr) {
00340         std::cout « "The linked list is empty." « std::endl;
00341         return;
00342     }
00343
00344     while (temp != nullptr) {
00345         std:: cout « temp->getData() « " ";
00346         temp = temp->getNextNode();
00347     }
00348
00349     std::cout « std::endl;
00350 }
00351
00358 template<class T>
00359 Node<T>* LinkedList<T>::binarySearch(T target) {
00360     Node<T>* searchHead = this->head;
00361     Node<T>* searchTail = this->tail;
00362     Node<T>* searchMid = findMid(searchHead, searchTail);
00363     if (searchHead) {
00364         while (searchHead->getData() <= searchTail->getData()) {
00365             if (target == searchHead->getData()) {
00366                 return searchHead;
00367             }
00368             else if (target == searchMid->getData()) {
00369                 return searchMid;
00370             }
00371             else if (target == searchTail->getData()) {
00372                 return searchTail;
00373             }
00374
```

```
00375             if (target < searchMid->getData()) {
00376                 searchHead = searchHead->getNextNode();
00377                 searchTail = searchMid->getPrevNode();
00378                 searchMid = findMid(searchHead, searchTail);
00379             }
00380             else if (target < searchMid->getData()) {
00381                 searchHead = searchMid->getNextNode();
00382                 searchTail = searchTail->getPrevNode();
00383                 searchMid = findMid(searchHead, searchTail);
00384             }
00385         }
00386     }
00387
00388     return nullptr;
00389 }
00390
00391 #endif
```

4.3 LinkedList.hpp

```
00001 /*********************************************
00002  *  @file LinkedList.hpp
00003  *  @brief Spring 2023 - CSC340.05 Final Project
00004  *
00005  *  This is the final project to make a custom
00006  *  LinkedList and Node class project for
00007  *  Spring 2023 - CSC340.05
00008  *
00009  *  @author Ashley Ching
00010  *  @author Charlene Breanne Calderon
00011  *  @author Eduardo Loza
00012  *  @author Lennart Richter
00013  *********************************************/
00014
00015 #ifndef LINKEDLIST_HPP
00016 #define LINKEDLIST_HPP
00017 #include "Node.hpp"
00018 #include <iostream>
00019 #include <string>
00020 #include <sstream>
00021
00022 template<class T>
00023 class LinkedList {
00024 public:
00027     LinkedList();
00029     ~LinkedList();
00031     void clear();
00034     void insert(T data);
00037     void add(T data);
00040     void remove(T data);
00044     Node<T>* search(T data);
00045     std::string toString();
00046     void mergeSort();
00047     void bubbleSort();
00050     void addFromFile(std::string fileName);
00051     void mergeLists(const LinkedList<T>* listTwo);
00053     void print();
00054     Node<T>* binarySearch(T target);
00055
00056 private:
00057     Node<T>* head;
00058     Node<T>* tail;
00059
00064     std::string to_string(const T& obj) {
00065         std::ostringstream oss{};
00066         oss << obj;
00067         return oss.str();
00068     }
00069
00078     static Node<T>* findMid(Node<T>* start, Node<T>* end) {
00079         bool flip = true;
00080         while (start != end) {
00081             if (flip) {
00082                 start = start->getNextNode();
00083             }
00084             else {
00085                 end = end->getPrevNode();
00086             }
00087             flip = flip ? false : true;
00088         }
00089
00090         return start;
00091     }
```

```
00092 };
00093 #include "LinkedList.cpp"
00094 #endif /* LinkedList_hpp */
```

4.4 main.cpp

```
00001 /*********************************************
00002  *  @file main.cpp
00003  *  @brief Spring 2023 - CSC340.05 Final Project
00004  *
00005  *  This is the final project to make a custom
00006  *  LinkedList and Node class project for
00007  *  Spring 2023 - CSC340.05
00008  *
00009  *  @author Ashley Ching
00010  *  @author Charlene Breanne Calderon
00011  *  @author Eduardo Loza
00012  *  @author Lennart Richter
00013  *********************************************/
00014
00015 #include "LinkedList.hpp"
00016 #include "FunctionTests.hpp"
00017
00018 int main(int argc, const char* argv[]) {
00019
00020     if (argc == 2 && (std::string(argv[1]) == "-t" || std::string(argv[1]) == "--test")) {
00021         testLinkedList();
00022     }
00023     else {
00024         demo();
00025     }
00026     return 0;
00027 }
```

4.5 Node.cpp

```
00001 /*********************************************
00002  *  @file Node.cpp
00003  *  @brief Spring 2023 - CSC340.05 Final Project
00004  *
00005  *  This is the final project to make a custom
00006  *  LinkedList and Node class project for
00007  *  Spring 2023 - CSC340.05
00008  *
00009  *  @author Ashley Ching
00010  *  @author Charlene Breanne Calderon
00011  *  @author Eduardo Loza
00012  *  @author Lennart Richter
00013  *********************************************/
00014
00019 #ifndef NODE_CPP
00020 #define NODE_CPP
00021
00022 #include "Node.hpp"
00023
00030 template<class T>
00031 Node<T>::Node() {
00032     this->data = "";
00033     this->next = nullptr;
00034     this->prev = nullptr;
00035 }
00036
00041 template<class T>
00042 Node<T>::Node(T data) {
00043     this->data = data;
00044     this->next = nullptr;
00045     this->prev = nullptr;
00046 }
00047
00052 template<class T>
00053 Node<T>::~Node() {
00054     this->next = nullptr;
00055     this->prev = nullptr;
00056 }
00057
00058 // getters
00059 template<class T>
00060 T Node<T>:: getData() {return this->data;}
00061
```

```
00062 template<class T>
00063 Node<T>* Node<T>::getNextNode() {return this->next;}
00064
00065 template<class T>
00066 Node<T>* Node<T>::getPrevNode() {return this->prev;}
00067
00068 template<class T>
00069 void Node<T>::setData(T data) {this->data = data;}
00070
00071 template<class T>
00072 void Node<T>::setNextNode(Node* next) {this->next = next;}
00073
00074 template<class T>
00075 void Node<T>::setPrevNode(Node* prev) {this->prev = prev;}
00076
00077 template<class T>
00078 void Node<T>::setNextNodeNull() {this->next = nullptr;}
00079
00080 template<class T>
00081 void Node<T>::setPrevNodeNull() {this->prev = nullptr;}
00082 #endif
```

4.6 Node.hpp

```
00001 /*********************************************
00002  *  @file Node.hpp
00003  *  @brief Spring 2023 - CSC340.05 Final Project
00004  *
00005  *  This is the final project to make a custom
00006  *  LinkedList and Node class project for
00007  *  Spring 2023 - CSC340.05
00008  *
00009  *  @author Ashley Ching
00010  *  @author Charlene Breanne Calderon
00011  *  @author Eduardo Loza
00012  *  @author Lennart Richter
00013  *********************************************/
00014
00015 #ifndef NODE_HPP
00016 #define NODE_HPP
00017
00018 template<class T>
00019 class Node {
00020 public:
00021     Node();
00022     Node(T data);
00023     ~Node();
00024     T getData();
00025
00026     Node* getNextNode();
00027     Node* getPrevNode();
00028
00029     void setData(T data);
00030     void setNextNode(Node* next);
00031     void setPrevNode(Node* prev);
00032     void setNextNodeNull();
00033     void setPrevNodeNull();
00034
00035 private:
00036     T data;
00037     Node<T>* next;
00038     Node<T>* prev;
00039 };
00040 #include "Node.cpp"
00041 #endif /* Node_hpp */
```

4.7 Vault.cpp

```
00001 /*********************************************
00002  *  @file Vault.cpp
00003  *  @brief Spring 2023 - CSC340.05 Final Project
00004  *
00005  *  This is the final project to make a custom
00006  *  LinkedList and Node class project for
00007  *  Spring 2023 - CSC340.05
00008  *
00009  *  @author Ashley Ching
00010  *  @author Charlene Breanne Calderon
00011  *  @author Eduardo Loza
```

```
00012  *  @author Lennart Richter
00013  ********************************************/
00014
00015 #ifndef Vault_cpp
00016 #define Vault_cpp
00017
00018 #include "Vault.hpp"
00019
00020 Vault::Vault() {
00021     this->balance = 0;
00022 }
00023
00024 Vault::Vault(int startBal) {
00025     this->balance = startBal;
00026 }
00027
00028 Vault::~Vault() {}
00029
00030 bool Vault::operator==(const Vault& r) {
00031     return this->balance == r.balance;
00032 }
00033
00034 bool Vault::operator!=(const Vault& r) {
00035     return this->balance != r.balance;
00036 }
00037
00038 bool Vault::operator<(const Vault& r) {
00039     return this->balance < r.balance;
00040 }
00041
00042 bool Vault::operator>(const Vault& r) {
00043     return this->balance > r.balance;
00044 }
00045
00046 bool Vault::operator<=(const Vault& r) {
00047     return this->balance <= r.balance;
00048 }
00049
00050 bool Vault::operator>=(const Vault& r) {
00051     return this->balance >= r.balance;
00052 };
00053 #endif
```

4.8 Vault.hpp

```
00001 /*******************************************
00002  *  @file Vault.hpp
00003  *  @brief Spring 2023 - CSC340.05 Final Project
00004  *
00005  *  This is the final project to make a custom
00006  *  LinkedList and Node class project for
00007  *  Spring 2023 - CSC340.05
00008  *
00009  *  @author Ashley Ching
00010  *  @author Charlene Breanne Calderon
00011  *  @author Eduardo Loza
00012  *  @author Lennart Richter
00013  ********************************************/
00014
00015 #ifndef Vault_hpp
00016 #define Vault_hpp
00017
00018 class Vault {
00019 public:
00020     Vault();
00021     Vault(int startBal);
00022     ~Vault();
00023
00024     bool operator==(const Vault& r);
00025     bool operator!=(const Vault& r);
00026     bool operator<(const Vault& r);
00027     bool operator>(const Vault& r);
00028     bool operator<=(const Vault& r);
00029     bool operator>=(const Vault& r);
00030
00031 private:
00032     int balance;
00033     friend std::ostream& operator«(std::ostream& os, const Vault& v) {
00034         os « "Vault with balance: " « v.balance;
00035         return os;
00036     }
00037 };
00038 #include "Vault.cpp"
00039 #endif /* Vault_hpp */
```

Index