Wai Yan (Will) Htun

CS 32, Nachenberg

Project 2 Specifications

**Doubly Linked List Implementation**

The doubly linked list is implemented using nodes containing two pointers, the first one pointing to the next node and the second pointing to the previous node. The linked list class which contains the Node struct will also hold two other member pointer variables – head and tail. Head will point to the first node in the list while the tail will point to the last. The linked list is not circular, meaning the next node after tail will be NULL, instead of the head node.

**Pseudocode**

1. Combine (Map1, Map2, Result)

* Delete Result to make sure it is empty before adding elements
* Traverse through Map1
  + Check if the value of the current node appears in Map2
  + If Map2 does not contain any node with the same values
    - add the node to Result using insert function
    - return true
  + If Map2 contains a node with the same value
    - check if that node contains the same value
      * if so, add the node to Result and return true
      * if not, do nothing and return false
* Traverse through Map2
  + Check if the value of the current node appears in Map1
  + If Map1 does not contain any node with the same values
    - add the node to Result using insert function
    - return true
  + If Map2 contains a node with the same value and the node is not in Result
    - check if that node contains the same value
      * if so, add the node to Result and return true
      * if not, do nothing and return false

1. Subtract (Map1, Map2, Result)

* Delete Result to make sure it is empty before adding elements
* Traverse through Map1
  + Check if the value of the current node appears in Map2
  + If Map2 does not contain any node with the same values
    - add the node to Result
  + If Map2 contains a node with the same value
    - do nothing

1. Assignment Operator = (Source)

* Declare two pointers, target and source, each pointing to the head of this Map and source Map respectively
* If source is empty
  + assign head and tail of target to NULL
* While source pointer is not pointing to NULL
  + If target is null
    - allocate a new node
    - copy the values from the node being pointed to by source
    - add the node to the back of the list
  + If target is not null
    - copy the values from the node being pointed to by source, to the node being pointed to by target
    - advance target pointer to next node
  + advance source pointer to next node
* If source is NULL, and target is not NULL
  + delete the node being pointed to by target
  + move onto the next node
  + Repeat until target is NULL

1. Insert (g\_key, g\_value)

* Declare node pointer p and set it to head
* If head is NULL
  + Allocate a new node
  + Copy values into new node
  + Set next and prev to NULL
  + Set head and tail to address of add
  + Return true
* While p is not NULL
  + If the node pointed to by p has the same key as g\_key
    - Break
  + Advance p to next node
* If p is NULL, indicating p is at the end of the list
  + Allocate a new node
  + Copy values into new node
  + Set the tail node’s next to the address of the new node
  + Make tail point to the new node
  + Return true
* Return false

1. Update (g\_key, g\_value)

* Declare node pointer p and set it to head
* While p is not NULL
  + If the node pointed to by p has the same key as g\_key
    - Break
  + Advance p to next node
* If p is NULL, indicating p is at the end of the list
  + Return false
* If not, indicating same key has found in the list
  + Set value pointed to by p to m\_value
  + Return true

1. Erase (g\_key)

* Declare node pointer p and set it to head
* If head if NULL
  + Return false
* Else if head has the same value as g\_key
  + Delete head
  + Set head to point to next node
* Else
  + While next pointer of p is not NULL
    - If value of next pointer of p is equal to g\_key
      * break
    - Advance p to next node
  + If next pointer of p is not NULL, indicating a key is found
    - Make next of p point to the node after the to-be-deleted node
    - Make the prev of the node after to-be-deleted node point to p
    - Delete next node pointed to by p
    - Return true
* Return false

1. Get (i, g\_key, g\_value)

* Declare int count and set to 0
* If i is less than the size of list
  + Declare node pointer p and set to head
  + While p is not NULL
    - If count is not equal to i
      * Increment count
      * Advance p to p next
    - Else
      * Set g\_key to the key pointed to by p
      * Set g\_value to the value pointed to by p
      * Return true
* Return false

1. Swap (otherMap)

* Declare two temporary nodes temp\_head and temp\_tail
* Set them to head and tail respectively
* Set tail and head to the tail and head of otherMap respectively
* Set the tail and head of otherMap to temp\_head and temp\_tail respectively

**Test Cases**

1. Adding elements into two Map objects and displaying them using dump()

#include <iostream>

#include "Map.h"

using namespace std;

int main() {

Map a;

Map b;

a.insert("Apple", 10);

a.insert("Watermelon", 20.0);

a.insert("Orange", 666);

a.insert("Banana", 800);

a.dump();

cout << endl;

b.insert("Toyota", 20302);

b.insert("Camry", 232);

b.insert("Boo", 130);

b.dump();

cout << endl;

}

1. Using assignment operator to copy elements of a into b while length of a is greater than that of b

#include <iostream>

#include "Map.h"

using namespace std;

int main() {

Map a;

Map b;

a.insert("Apple", 10);

a.insert("Watermelon", 20.0);

a.insert("Orange", 666);

a.insert("Banana", 800);

a.dump();

cout << endl;

b.insert("Toyota", 20302);

b.insert("Camry", 232);

b.insert("Mercedez", 130);

b.dump();

cout << endl;

b = a;

b.dump();

cout << endl;

}

1. Using assignment operator to copy elements of a into b while length of a is greater than that of b

#include <iostream>

#include "Map.h"

using namespace std;

int main() {

Map a;

Map b;

a.insert("Apple", 10);

a.insert("Watermelon", 20.0);

a.dump();

cout << endl;

b.insert("Toyota", 20302);

b.insert("Camry", 232);

b.insert("Mercedez", 130);

b.dump();

cout << endl;

b = a;

b.dump();

cout << endl;

}

1. Using assignment operator to copy an empty Map into a

#include <iostream>

#include "Map.h"

using namespace std;

int main() {

Map a;

Map b;

a.insert("Apple", 10);

a.insert("Watermelon", 20.0);

a.dump();

cout << endl;

a = b;

b.dump();

cout << endl;

}

1. Using assignment operator to copy Map a into an empty Map

#include <iostream>

#include "Map.h"

using namespace std;

int main() {

Map a;

Map b;

a.insert("Apple", 10);

a.insert("Watermelon", 20.0);

a.dump();

cout << endl;

b = a;

b.dump();

cout << endl;

}

1. Combine function

#include <iostream>

#include "Map.h"

using namespace std;

int main() {

Map a;

Map b;

Map r;

a.insert("Apple", 10);

a.insert("Watermelon", 20.0);

a.insert("Orange", 666);

a.insert("Banana", 800);

a.dump();

cout << endl;

b.insert("Apple", 20302);

b.insert("Cranberry", 232);

b.insert("Banana", 800);

b.dump();

cout << endl;

combine(a, b, r);

r.dump();

cout << endl;

}

1. Subtract function

#include <iostream>

#include "Map.h"

using namespace std;

int main() {

Map a;

Map b;

Map r;

a.insert("Apple", 10);

a.insert("Watermelon", 20.0);

a.insert("Orange", 666);

a.insert("Banana", 800);

a.dump();

cout << endl;

b.insert("Apple", 20302);

b.insert("Cranberry", 232);

b.insert("Banana", 800);

b.dump();

cout << endl;

subtract(a, b, r);

r.dump();

cout << endl;

}