Gabriella Ang

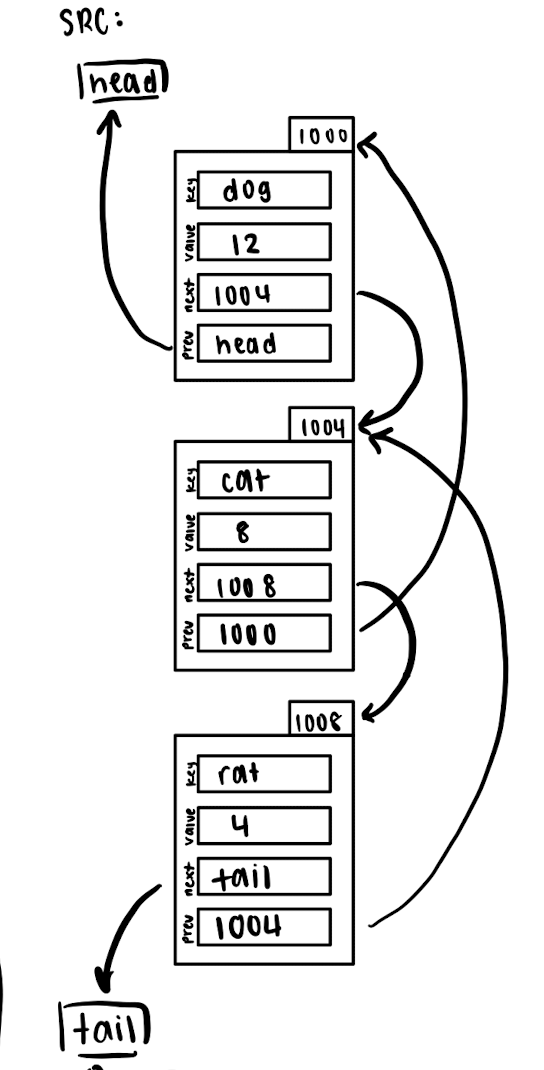
Carey Nachenberg

005677251

**Project 2 Report**

---------------------------------------------------------------------------------------------------------------------

For this project, I used a doubly linked list with both a head and tail pointer. Even though it was a doubly linked list, I ended up only really needing a singly linked list since I didn’t really have the need to traverse in both directions. I did find the tail pointer more helpful for erasing any objects at the end of the list. The objects are always added to the end of the list. In each node, is a key/value pair as well as a next and previous pointer.



---------------------------------------------------------------------------------------------------------------------

**Pseudocode**

bool Map::insert(const KeyType& key, const ValueType& value);

insert {

check for duplicates (return false if true)

create new item

add new item if no items

if (smaller than first object in list)

add to front

else if (larger than all elements)

add to end

else //adding to middle

loop through head and tail to find the objects surrounding the new object

add object

increase count

}

---------------------------------------------------------------------------------------------------------------------

bool Map::erase(const KeyType& key)

erase {

return if empty list

if (only one object in list)

head and tail = nullptr

delete object

decrease count

else if (deleted item is at the front)

move head to head->next

delete first object

decrease count

else if (deleted item is at the end)

move tail to tail->prev

delete last object

decrease count

else //in the middle of the list

loop

find the object before the deleted one

if (not nullptr)

rearrange pointers

delete object

decrease count

}

---------------------------------------------------------------------------------------------------------------------

bool merge(const Map& m1, const Map& m2, Map& result)

merge {

//clear results

if (result is not the same as m1 or m2)

loop through result

get 0th object

erase object

for (loop through m1)

get object from m1

insert object into result

for (loop through m2)

get object from m2

insert object into result

for (loop through m1)

get object at m1

if (value in m1 != value in m2)

erase object from result

}

---------------------------------------------------------------------------------------------------------------------

void reassign(const Map& m, Map& result)

reassign {

//clear results

if (result is not the same as m1 or m2)

loop through result

get 0th object

erase object

if (size == 0) return

if (size == 1)

insert object into result

return

get last key

get first value

for (loop through m.size - 1)

get key at i

get value at i + 1

insert into results

insert last key and first value

}

---------------------------------------------------------------------------------------------------------------------

**Test Cases**

Map creatures;

//checking empty and size functions

assert(creatures.empty() && creatures.size() == 0); //checking that the list is empty empty

assert(creatures.insert("Penguin", 12)); //checking insert

assert(!creatures.empty());

//checking insert function

assert(creatures.insert("Monkey", 1)); //checking insert add to front

assert(creatures.insert("Bee", 102)); //checking insert add to front

assert(creatures.insert("", 95)); //checking empty string

assert(creatures.insert("Aardvark", 2)); //checking insert add to front

assert(creatures.insert("aardvark", 403)); //checking insert adding to end

assert(creatures.insert("zebra", 403)); //checking insert adding to end

assert(creatures.insert("Narwal", 37)); //checking insert adding to middle

assert(creatures.insert("Mammoth", 98)); //checking insert adding to middle

assert(creatures.insert("Bear", 7)); //checking insert adding to middle

assert(!creatures.insert("zebra", 3)); //checking for duplicate cases

//checking size function

assert(creatures.size() == 10); //checking size function

//checking update function

assert(creatures.update("zebra", 402)); //checking for update function

assert(!creatures.update("Zebra", 402)); //checking for noexistent key

//checking insertOrUpdate function

assert(creatures.insertOrUpdate("Mammoth", 32)); //checking for update

assert(creatures.insertOrUpdate("Boar", 6)); //checking for insert

//checking erase function

assert(creatures.erase("")); //checking erase function in the front

assert(!creatures.erase("bee")); //checking nonexistent key

assert(creatures.erase("zebra")); //checking erase function at the end

assert(creatures.erase("aardvark")); //checking erase function in the middle

Map m;

m.insert("m", 1);

assert(m.erase("m")); //erasing one item

assert(m.size() == 0);

assert(m.insert("M", 1));

assert(m.size() == 1);

//checking contains function

assert(creatures.contains("Aardvark")); //checking contains

assert(!creatures.contains("")); //checking false case

//checking get function (two parameters)

**double** result;

creatures.get("Bear", result);

assert(result == 7); //checking that it works

creatures.get("Aardvark", result);

assert(result == 2); //checking that it works

assert(!creatures.get("zebra", result)); //checking for a nonexistent key

//checking get function (three parameters)

string animal;

**double** count;

assert(creatures.get(1, animal, count));

assert(animal == "Bear" && count == 7); //checking that it works

assert(creatures.get(7, animal, count));

assert(animal == "Penguin" && count == 12); //checking that it works

assert(!creatures.get(8, animal, count)); //checking out of bounds

//checking swap function

Map cats;

cats.insert("Siamese", 4);

cats.insert("Sphinx", 3);

cats.insert("Tabby", 2);

cats.insert("Persian", 5);

cats.swap(creatures);

assert(cats.contains("Aardvark"));

assert(cats.contains("Bear"));

assert(cats.contains("Bee"));

assert(cats.contains("Boar"));

assert(cats.contains("Mammoth"));

assert(cats.contains("Monkey"));

assert(cats.contains("Narwal"));

assert(cats.contains("Penguin"));

assert(cats.size() == 8);

assert(creatures.contains("Siamese"));

assert(creatures.contains("Sphinx"));

assert(creatures.contains("Tabby"));

assert(creatures.contains("Persian"));

assert(creatures.size() == 4);

//checking copy constructor

Map animals = creatures;

assert(animals.size() == 4);

assert(animals.contains("Siamese"));

assert(animals.contains("Sphinx"));

assert(animals.contains("Tabby"));

assert(animals.contains("Persian"));

assert(cats.size() == 8);

//checking assignment operator

cats = creatures;

assert(cats.contains("Siamese"));

assert(cats.contains("Sphinx"));

assert(cats.contains("Tabby"));

assert(cats.contains("Persian"));

assert(cats.size() == 4);

creatures.dump();

//checking merge function

Map results;

merge(creatures, cats, results); //checking for repeats

assert(creatures.contains("Siamese"));

assert(creatures.contains("Sphinx"));

assert(creatures.contains("Tabby"));

assert(creatures.contains("Persian"));

assert(creatures.size() == 4);

Map students = cats;

assert(students.size() == 4);

Map girls;

girls.insert("Holly", 12);

girls.insert("Belle", 12);

girls.insert("Samantha", 11);

girls.insert("Brianna", 13);

girls.insert("Chloe", 12);

Map boys;

boys.insert("Bryan", 12);

boys.insert("Emmett", 13);

boys.insert("Joel", 13);

boys.insert("Michael", 12);

boys.insert("Baron", 12);

boys.insert("Ian", 11);

merge(girls, boys, students); //checking with existing values in result

students.dump();

Map blues;

Map reds;

Map mix;

blues.insert("cyan", 1);

blues.insert("baby blue", 1);

blues.insert("navy", 1);

blues.insert("purple", 1);

blues.insert("turquoise", 1);

blues.insert("pastel", 1);

blues.insert("blue", 1);

reds.insert("red", 2);

reds.insert("pink", 2);

reds.insert("purple", 2);

reds.insert("maroon", 2);

merge(blues, reds, mix); //checking with duplicate keys, different values

mix.dump();

//multiple duplicates, different values

Map water;

Map ocean;

ocean.insert("water", 10000);

ocean.insert("fish", 200);

ocean.insert("algae", 1257);

ocean.insert("sharks", 53);

Map lake;

lake.insert("water", 5000);

lake.insert("plankton", 403);

lake.insert("people", 12);

lake.insert("algae", 32);

assert(!merge(ocean, lake, water)); //returns false bc different values for duplicate keys

water.dump();

//merging empty maps

Map z;

z.insert("stuff", 34);

Map y;

Map x;

assert(merge(x,y,z));

assert(merge(blues, water, water));

water.dump();

//checking reassign function

Map aacf;

aacf.insert("Christiana", 1);

aacf.insert("Stephanie", 3);

aacf.insert("Tiffany", 4);

aacf.insert("Elaine", 2);

aacf.dump();

Map goc;

reassign(aacf, goc);

goc.dump();

reassign(x,y); //nothing in either list

x.insert("Barb", 3);

reassign(x, y);

y.dump();

aacf.dump();

reassign(aacf, aacf);

aacf.dump();