**Izak Bunda - UID 305783387**

**Project 2 - Sequence.h, Sequence.cpp, and report.docx**

**Due Tuesday, April 19, 2022 - 11:00PM**

**Circular Doubly Linked List:**

I chose to use a circular doubly linked list because it greatly simplified all of the necessary implementations for this project. I did not have to check if the list was empty, I never had to check if for a nullptr, and it was easier to traverse through the linked list simply by checking if the prev or next was pointing to the dummy head. Furthermore, I only had to create one dummy head.

| **Sequence**  Each item points to one another. There is a prev pointer which points to the previous item and a next pointer which points to the next item |  |  |
| --- | --- | --- |
| **Sequence with one item**  Each item points to one another. There is a prev pointer which points to the previous item and a next pointer which points to the next item |  |  |
| **Empty Sequence**  The dummy head points its prev pointer to itself and also points its next pointer to itself |  |  |

**Implementation/Pseudo Code**

**Sequence();**

* Creates a dummy head node
* Then points the prev to itself
* And points the next to itself

**~Sequence();**

* Initializes two pointers - one to the prev of head and one to the prev prev of head
* This moves backwards, one trailing after the other
* While they haven’t reached the head, then they keep deleting the value that they just left
* And then at the end, the two pointers are also deleted

**Sequence(const Sequence& other);**

* Loops through the length of other
* Copy the size into other
* Get then insert the current value into other

**Sequence& operator=(const Sequence& rhs);**

* Checks if rhs and the sequence calling it is the same thing
* Loops through the length of this sequence
  + Calls erase on the whole thing
* Then performs the same function as the copy constructor into this sequence

**int insert(int pos, const ItemType& value);**

* Checks if the given parameters and valid
* Creates a node
  + Copies the value into that node
  + Then points the pointers into the appropriate locations
  + Increases the size variable
* Returns the current position of the newly inserted node

**int insert(const ItemType& value);**

* Almost the same as the one above, but no pos parameter
* Creates a node
  + Loops through and checks if a value is greater than the value of the value parameter
  + If it is, then:
    - Copies the value into that node
    - Then points the pointers into the appropriate locations
    - Increases the size variable
* Returns the current position of the newly inserted node

**bool erase(int pos);**

* Loops through the entire sequence
* Uses a variable counter to keep track of the index of the linked list item
* When it reaches that pos
  + Then a “to be deleted” pointer is pointed to that spot
  + The pointers are adjusted to skip over this “to be deleted” location
  + Then it is deleted
  + The size is reduced
  + And the function returns true

**int remove(const ItemType& value);**

* Loops through the entire sequence
* Starts from the back
* If it encounters a match
  + Then it calls erase on that
  + Increases the value of the removed variable
  + Then moves on to the next item
* If it does not encounter a match
  + Then it just keeps moving on the next item

**bool get(int pos, ItemType& value) const;**

* This loops through the entire sequence
* A for loop is used to keep loop through the sequence and keep track of the “index” of the linked list
* When this matches the pos parameter
  + Then the value at that position is copied into the value parameter
  + And returns true

**bool set(int pos, const ItemType& value);**

* This loops through the entire sequence
* A for loop is used to keep loop through the sequence and keep track of the “index” of the linked list
* When this matches the pos parameter
  + Then the value at the position is set to the parameter value
  + And returns true

**int find(const ItemType& value) const;**

* Loops through the entire sequence
* As soon as it finds a match to its value
  + Then it returns the index (or however many times the loops runs)
* If it doesn’t then it returns -1

**void swap(Sequence& other);**

* Copies the size into a temp variable
* Copies the other size into this size
* Then copies the temp size into the other size
* Creates a copy pointer to this sequence
* Points this sequence to the head of the other sequence
* Then points the other sequence to the head of this sequence

**Others:**

**Struct node {};**

* Initializes a node
* That stores a value
* And a prev pointer and a next pointer

**int subsequence(const Sequence& seq1, const Sequence& seq2);**

* Check if seq1 and seq2 are proper sequences that we can apply this function to
* Special case: if seq2 has only one item
  + Find the first instance of this item in seq1
  + Return that position
* Otherwise
  + Loop through seq1 starting at the first instance of the first item of seq2
  + Then loop through seq2 and try to match the rest of seq2 with the following seq1 items
  + If they don’t match, then restart the outer loop again

**void interleave(const Sequence& seq1, const Sequence& seq2, Sequence& result);**

* Create a temp Sequence variable to apply all this function to to avoid aliasing
* If seq2 is greater than seq1 then nothing happens
* If seq1 is empty, then set temp to seq2 (and vice versa)
* If neither are empty, then we can begin
  + If seq1 is larger
    - Then insert a value from seq1
    - Then insert a value from seq2
    - Repeat until the end of seq2
    - Then insert the rest of seq1
  + If seq2 is larger
    - Then insert a value from seq1
    - Then insert a value from seq2
    - Repeat until the end of seq1
    - Then insert the rest of seq2
* Then finally set result to temp

**Test cases:**

**// Tests for empty ( and insert(2) )**

Sequence s1, s2, s5;

s1.insert(0, "a"); s1.insert(1, "b"); s1.insert(2, "c"); s1.insert(3, "d");

s2.insert(0, "w"); s2.insert(1, "x"); s2.insert(2, "y"); s2.insert(3, "z");

assert( (s1.empty() == false) && (s2.empty() == false) && (s5.empty() == true) );

**// Tests for size**

Sequence s6, s7;

s6.insert(0, "a"); s6.insert(1, "b"); s6.insert(2, "c"); s6.insert(3, "d");

s7.insert(0, "w"); s7.insert(1, "x"); s7.insert(2, "y"); s7.insert(3, "z");

assert ( ( s6.size() == 4 ) && ( s7.size() == 4 ) );

s6.erase(0); s6.erase(0);

assert( s6.size() == 2 );

s7.erase(0); s7.erase(0); s7.erase(0); s7.erase(0);

assert( s7.size() == 0);

**// Tests for erase**

Sequence s3, s4, s8;

s3.insert(0, "red"); s3.insert(1, "orange"); s3.insert(2, "yellow"); s3.insert(3, "green");

s4.insert(0, "phoebe"); s4.insert(1, "red"); s4.insert(2, "lucille"); s4.insert(3, "bridgers");

s3.erase(0); string x;

assert( s3.get(0, x) && x == "orange" && s3.size() == 3);

s3.erase(0); string y;

assert( s3.get(0, y) && y == "yellow" && s3.size() == 2);

s3.erase(0); string z;

assert( s3.get(0, z) && z == "green" && s3.size() == 1);

s3.erase(0); string a;

assert( !s3.get(0, a) && s3.size() == 0);

s4.erase(3); string xx;

assert( s4.get(2, xx) && xx == "lucille" && s4.size() == 3 );

s4.erase(0); string yy;

assert( s4.get(0, yy) && yy == "red" && s4.size() == 2 );

assert( !s8.erase(0) && !s8.erase(99) && s8.size() == 0);

**// Tests for remove**

Sequence s9, s10, s11, s12;

s9.insert(0, "I"); s9.insert(1, "I"); s9.insert(2, "know"); s9.insert(3, "I");

s10.insert(0, "dont"); s10.insert(1, "know"); s10.insert(2, "howtokeep"); s10.insert(3, "lovingyou");

s11.insert(0, "RESPONDPLS");

string aa;

assert( s9.remove("I") == 3 && s9.get(0, aa) && aa == "know" && s9.size() == 1 );

assert( s9.remove("know") == 1 );

assert( s10.remove("howdy") == 0 ); string bb;

assert( s10.remove("dont") == 1 && s10.get(1, bb) && bb == "howtokeep" && s10.size() == 3);

assert( s11.remove("RESPONDPLS") == 1 && s11.size() == 0 );

assert( s12.remove("getspotify") == 0 );

**// Tests for set**

Sequence s13, s14, s15, s16;

s13.insert(0, "I"); s13.insert(1, "I"); s13.insert(2, "I"); s13.insert(3, "I");

s14.insert(0, "a"); s14.insert(1, "b"); s14.insert(2, "c"); s14.insert(3, "d");

s15.insert(0, "one");

string h, i, j, k;

assert( s13.set(0, "a") && s13.set(1, "b") && s13.set(2, "c") && s13.set(3, "d") && s13.get(0, h ) && h == "a" && s13.get(1, i ) && i == "b" && s13.get(2, j ) && j == "c" && s13.get(3, k ) && k == "d");

string l, m;

assert( s14.set(0, "hey") && s14.get(0, l ) && l == "hey" && s14.set(3, "HOWDY") && s14.get(3, m) && m == "HOWDY" );

string n;

assert( s15.set(0, "hey") && s15.get(0, n ) && n == "hey");

assert( !s16.set(0, "hohoho") );

**// Tests for find**

Sequence s17, s18, s19, s20;

s17.insert(0, "red"); s17.insert(1, "orange"); s17.insert(2, "yellow"); s17.insert(3, "green");

s18.insert(0, "phoebe"); s18.insert(1, "hey"); s18.insert(2, "bridgers"); s18.insert(3, "phoebe");

s19.insert(0, "phoebebridgersussy");

assert( s17.find("red") == 0 && s17.find("orange") == 1 && s17.find("yellow") == 2 && s17.find("green") == 3);

assert( s18.find("phoebe") == 0 );

assert( s19.find("phoebe") == -1 && s19.find("phoebebridgersussy") == 0 );

assert( s20.find("tratorjoeortraderjoes?") == -1 );

**// Tests for swap**

Sequence s21, s22;

s21.insert(0, "paratha");

s21.insert(0, "focaccia");

s22.insert(0, "roti");

s21.swap(s22);

assert(s21.size() == 1 && s21.find("roti") == 0 && s22.size() == 2 && s22.find("focaccia") == 0 && s22.find("paratha") == 1);

Sequence s23, s24;

s23.insert(0, "loveinthetimeofsocialism");

s23.swap(s24);

assert( s23.size() == 0 && s24.size() == 1 && s24.find("loveinthetimeofsocialism") == 0 );

**// Tests for subsequence**

Sequence s25, s26, s27, s28, s282, s283, s284, s29;

s25.insert(0, "1"); s25.insert(1, "a"); s25.insert(2, "b"); s25.insert(3, "c");

s26.insert(0, "a"); s26.insert(1, "b"); s26.insert(2, "c");

s27.insert(0, "w"); s27.insert(1, "x"); s27.insert(2, "y"); s27.insert(3, "z");

s28.insert(0, "w");

s282.insert(0, "x");

s283.insert(0, "y");

s284.insert(0, "z");

assert( subsequence(s25, s26) == 1 );

assert( (subsequence(s27, s28) == 0) && (subsequence(s27, s282) == 1) && (subsequence(s27, s283) == 2) && (subsequence(s27, s284) == 3) );

assert( (subsequence(s27, s29) == -1) && (subsequence(s28, s29) == -1) );

**// Tests for interleave**

Sequence s30, s31, s32, s33, s34, s35;

s30.insert(0, "1"); s30.insert(1, "a"); s30.insert(2, "b"); s30.insert(3, "c");

s31.insert(0, "1"); s31.insert(1, "a"); s31.insert(2, "b"); s31.insert(3, "c");

s33.insert(0, "youaresick");

s34.insert(0, "andyouaremarried");

interleave( s30, s31, s32);

interleave( s33, s34, s35);

string test1, test2, test3, test4, test5;

assert( s32.get(0, test1) && test1 == "1" && s32.get(2, test2) && test2 == "a" && s32.get(4, test4) && test4 == "b" && s32.get(6, test5) && test5 == "c" );

string test6, test7;

assert( s35.get(0, test6) && test6 == "youaresick" && s35.get(1, test7) && test7 == "andyouaremarried" );