**Implementation Details**

Sequence uses a circular doubly-linked list with a dummy node. Each list node is a struct (a member variable of Sequence,) that contains pointers to its successor (m\_next) and predecessor (m\_prev) as well as a variable, data, of the type specified by ItemType. When a Sequence is initialized, its size is set to zero and the head pointer points to a dynamically allocated dummy node, which has m\_prev and m\_next pointing to itself.

**Nontrivial Algorithms**

int Sequence::insert(int pos, const ItemType& value)

if 0 <= pos <= size of the list

create a new Node, newNode

have p point to the Node previous the Node at pos

store value in newNode

link newNode with p and the node after p

make the node after p point to newNode

make p point to newNode

increment the size

return pos

else

return -1

bool Sequence::erase(int pos)

if 0 <= pos < size of the list

make p point to the Node at p

make the Node after p point to the Node before p

make the Node before p point to the Node after p

delete p

decrement the size

return true

else

return false

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

if seq1 is an alias of seq2

return 0

if both seq1 and seq2 have at least one item

initialize index variable of subsequence to -1

initialize index variable of seq2 to 0

for each element of index i in seq1

if the item at i in seq1 is the same as the item at the index of seq2 in seq2

increment the index of seq2 to examine next item

if the index of subsequence is -1

set index of subsequence to i

else

reset the index of subsequence to -1

reset the index of seq2 to 0

if index variable of seq2 equals the size of seq2

all items in seq2 have been matched with consecutive items in seq1, return index of subsequence

else

return -1

void concatReverse(const Sequence& seq1, const Sequence& seq2, Sequence& result)

copy seq1 into s1

copy seq2 into s2

erase all items in result

for each item in s1, traversing right to left

append the current item onto result

for each item in in s2, traversing right to left

append the current item onto result after the position where the previous loop finished

**Test Cases/Testing Routine –** *(tests performed with ItemType = unsigned long)*

| #include <iostream> #include <cassert> #include "Sequence.h" #include <iostream>  using namespace std;  bool checkEqual(Sequence& a, Sequence& b) {  ItemType aa;  ItemType bb;  if (a.size() != b.size())  return false;  for (int i = 0; i < a.size(); i++) {  a.get(i, aa);  a.get(i, bb);  if (aa != bb)  return false;  }  return true; }  void test() {  Sequence s;  assert(s.insert(0, 10) == 0); *// test insert first item*  assert(s.insert(0, 20) == 0); *// test insert at beginning of list*  assert(s.insert(30) == 2); *// test one-argument insert*   assert(s.insert(25) == 2); *// ...*  assert(s.insert(21) == 2); *// ...*  assert(s.insert(-2, 10) == -1); *// test invalid argument for insert*  assert(s.insert(7, 50) == -1); *// ...*  assert(s.insert(4, 40) == 4); *// test insert at end of list*  assert(s.insert(4, 1000) == 4); *// test insert in middle of list*  assert(s.size() == 7); *// test size()*  s.dump();  ItemType x = 999;  assert(s.get(1, x) && x == 10); *// test get()*  assert(s.get(0, x) && x == 20); *// ...*  Sequence t;  assert(t.empty()); *// test empty()*  t = s;   t.dump();  assert(checkEqual(t, s)); *// test assignment operator*  Sequence c(s);   c.dump();  assert(checkEqual(c, s)); *// test copy constructor*  c.insert(20);  assert(c.remove(20) == 2); *// test remove() for items in list*  assert(c.remove(-10) == 0); *// test remove() for item not in list*  c.dump();  assert(c.find(1000) == 3); *// test find() for item in list*  assert(c.find(-10) == -1); *// test find() for item not in list*  *// s and t are currently the same*  Sequence a(c);  c.swap(s);  assert(checkEqual(c, t)); *// test swap()*  assert(checkEqual(s, a)); *// test swap()*    Sequence seq1;  Sequence seq2;  seq1.insert(0, 30);  seq1.insert(1, 21);  seq1.insert(2, 63);  seq1.insert(3, 42);  seq1.insert(4, 17);  seq1.insert(5, 63);  seq1.insert(6, 17);  seq1.insert(7, 29);  seq1.insert(8, 8);  seq1.insert(9, 32);  seq2.insert(0, 63);  seq2.insert(1, 17);  seq2.insert(2, 29);  seq1.dump();  seq2.dump();  assert(subsequence(seq1, seq2) == 5); *// test subsequence() found successfully*  Sequence seq3;  seq3.insert(0, 17);  seq3.insert(1, 63);  seq3.insert(2, 29);  assert(subsequence(seq1, seq3) == -1); *// test subsequence() failed to find*  assert(subsequence(seq1, seq1) == 0); *// test subsequence() with alias*  Sequence empty;  assert(subsequence(seq1, empty) == -1); *// test subsequence() with empty Sequence*  assert(subsequence(empty, seq1) == -1); *// test subsequence() with empty Sequence*  *// (different params)*   Sequence seq4;  Sequence seq5;  Sequence r;  seq4.insert(0, 3);  seq4.insert(1, 2);  seq4.insert(2, 1);  seq5.insert(0, 7);  seq5.insert(1, 6);  seq5.insert(2, 5);  seq5.insert(3, 4);  r.insert(9999);  r.insert(10000);  seq4.dump();  seq5.dump();  r.dump();  concatReverse(seq4, seq5, r); *// test concatReverse() with different*  *// Sequences*  r.dump();  concatReverse(seq4, seq4, seq4); *// test concatReverse() with alias*  seq4.dump();  concatReverse(empty, empty, r); *// test concatReverse() with two empty*  *// params*  r.dump();  assert(r.size() == 0);  concatReverse(empty, seq5, r); *// test concatReverse() with one empty param*  r.dump();  concatReverse(seq5, empty, r);  r.dump();  }  int main() {  test();  cout << "Passed all tests" << endl; } |
| --- |

**Console Output:**

| {size: 7, [20, 10, 21, 25, 1000, 40, 30]}  {size: 7, [20, 10, 21, 25, 1000, 40, 30]}  {size: 7, [20, 10, 21, 25, 1000, 40, 30]}  {size: 6, [10, 21, 25, 1000, 40, 30]}  {size: 10, [30, 21, 63, 42, 17, 63, 17, 29, 8, 32]}  {size: 3, [63, 17, 29]}  {size: 3, [3, 2, 1]}  {size: 4, [7, 6, 5, 4]}  {size: 2, [9999, 10000]}  {size: 7, [1, 2, 3, 4, 5, 6, 7]}  {size: 6, [1, 2, 3, 1, 2, 3]}  {size: 0, []}  {size: 4, [4, 5, 6, 7]}  {size: 4, [4, 5, 6, 7]}  Passed all tests |
| --- |