**CSE 330 Homework 3 Report**

Daniel Meyer

Data Structures

Fall 2017

**Status:** 100%

**Time Complexity:** ordered\_union - O(n)

ordered\_intersect - O(n)

unordered\_union - O(n)

unordered\_intersect - O(n^2)

**Storage Complexity:** O(n)

**Source Code:** Pages 2 - 4

**Sample Run:** Page 5

/\*

Daniel Meyer

11-13-17

CSE 330

Fall 2017

Homework 3: Ordered Lists

Problem: Create function to create new functions to use with muiltiple lists.

Algorithm: There are functions to produce the union of an ordered or unordered

list and functions for finding the intersection of an ordered or unordered list.

Status: 100%

Time Complexity: ordered\_union - O(n)

ordered\_intersect - O(n)

unordered\_union - O(n)

unordered\_intersect - O(n^2)

Storage Complexity: O(n)

\*/

#include <list>

#include <iostream>

#include <time.h>

using namespace std;

template <class T>

list<T> ordered\_union(const list<T> &a, const list<T> &b)

{

list<T> order;

typename list<T>::const\_iterator i = a.begin();

typename list<T>::const\_iterator j = b.begin();

while (i != a.end() && j != b.end()) {

if (\*i < \*j) {

order.push\_back(\*i);

i++;

}

else if (\*j < \*i) {

order.push\_back(\*j);

j++;

}

else {

order.push\_back(\*i);

order.push\_back(\*j);

i++;

j++;

}

}

return order;

}

template <class T>

list<T> ordered\_intersect(const list<T> &a, const list<T> &b)

{

list<T> intersect;

typename list<T>::const\_iterator i = a.begin();

typename list<T>::const\_iterator j = b.begin();

while (i != a.end() && j != b.end()) {

if (\*i < \*j) {

i++;

}

else if (\*j < \*i) {

j++;

}

else {

intersect.push\_back(\*i);

i++;

j++;

}

}

return intersect;

}

template <class T>

list<T> unordered\_union(const list<T> &a, const list<T> &b)

{

list<T> order = a;

typename list<T>::const\_iterator j = b.begin();

while (j != b.end()) {

order.push\_back(\*j);

j++;

}

return order;

}

template <class T>

list<T> unordered\_intersect(const list<T> &a, const list<T> &b)

{

list<T> intersect;

typename list<T>::const\_iterator i = a.begin();

typename list<T>::const\_iterator j = b.begin();

for (i = a.begin(); i != a.end(); i++) {

for (j = b.begin(); j != b.end(); j++) {

if (\*j == \*i) {

intersect.push\_back(\*j);

}

}

}

return intersect;

}

int main()

{

list<int> l1, l2, l3;

typename list<int>::const\_iterator i;

srand(time(NULL));

//Unordered list test

l1.push\_back(2);

l1.push\_back(1);

l1.push\_back(5);

l1.push\_back(7);

l1.push\_back(12);

l2.push\_back(12);

l2.push\_back(2);

l2.push\_back(9);

l2.push\_back(3);

l2.push\_back(5);

l3 = unordered\_union(l1, l2);

for (i = l3.begin(); i != l3.end(); i++) {

cout << \*i << " ";

}

cout << endl;

l3.clear();

l3 = unordered\_intersect(l1, l2);

i = l3.begin();

cout << "Intersect at: ";

for (i = l3.begin(); i != l3.end(); i++) {

cout << \*i << " ";

}

cout << endl;

//Ordered list test

l1.clear();

l2.clear();

l3.clear();

for (int i = 0; i < 20; i++) {

l1.push\_back(i);

l2.push\_back(++i);

}

l3 = ordered\_union(l1, l2);

for (i = l3.begin(); i != l3.end(); ++i) {

cout << \*i << " ";

}

cout << endl;

l3.clear();

l3 = ordered\_intersect(l1, l2);

i = l3.begin();

cout << "Intersect at: ";

for (i = l3.begin(); i != l3.end(); ++i) {

cout << \*i << " ";

}

cout << endl;

return 0;

}

**Sample Run**

Script started on Tue 14 Nov 2017 08:10:34 PM UTC

To run a command as administrator (user "root"), use "sudo <command>".

See "man sudo\_root" for details.

]0;ubuntu@ubuntu: ~[01;32mubuntu@ubuntu[00m:[01;34m~[00m$ cd Desktop

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ g++ List.h

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ g++ OrderedLists.[K\_test.cpp

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ ./a.out

2 1 5 7 12 12 2 9 3 5

Intersect at: 2 5 12

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Intersect at:

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ exit

Script done on Tue 14 Nov 2017 08:10:58 PM UTC