|  |  |  |
| --- | --- | --- |
| American University of Sharjah  College of Engineering  Department of Computer Science & Engineering  P. O. Box 26666, Sharjah, UAE |  | **Instructors:** Dr. Michel Pasquier  **Lab Instructor:** Praveena Kolli  **Office:** EB2-126  **Phone**: 971-6-5152352  **e-mail**: pkolli@aus.edu  **Semester**: Spring 2021 |

**CMP 305 L – Data Structures and Algorithms**

**Lab. Assignment 5 – List class**

***Objectives:***

* Understand the working principles and make use of the list class.
* Develop (part of) a sorted list class that extends the list class.

***Note:***

***Lab:*** Exercises 1 (10 marks)

***Bonus*:** Exercise 2 (1 mark)

**Exercise1:**

**Part a:**

Complete the implementation of the given **SortedList**.h class that extends the **list** class (List.h is provided). Assume the data is always in ascending order. The following functions must be implemented:

1. **Override** iterator insert( iterator itr, const Object & x ) that inserts x before position itr. Check that x is larger than the previous item and less or equal to the next item. If not, the function should throw an exception SortedOrderMismatchException. The function returns the position of the inserted item.
2. **Add** iterator insert( const Object & x ) that inserts x at the appropriate position in the list i.e., keeping it sorted. The function returns the position of the inserted item.
3. **Add** iterator erase( const Object & x ) that removes x from the list, keeping it sorted. The function returns the position of the next item, if there is one, or the list end-marker.

You are encouraged to examine the other **list** functions inherited by **SortedList** to verify that they will still work as is (hence that no further change is required, other than indicated above).

**SortList.h:**

List<Object>::iterator insert(const Object& x)

{

typename List<Object>::iterator itr = List<Object>::begin();

while (itr != List<Object>::end() && \*itr < x) {

++itr;

}

//Now we have found the node bigger than x.

//X needs to be inserted behind that node.

List<Object>::insert(itr, x);

return --itr;

}

List<Object>::iterator erase(const Object & x)

{

typename List<Object>::iterator itr = List<Object>::begin();

while (itr != List<Object>::end() && \*itr < x) {

++itr;

}

//Now we have found the node bigger than x.

//X needs to be inserted behind that node.

List<Object>::erase(itr);

if (itr == List<Object>::end()) {

return itr;

}

else {

return ++itr;

}

}

List<Object>::iterator insert(List<Object>::iterator itr, const Object & x)

{

if (itr == List<Object>::begin()){ //insert at first node

if (x < \*(++itr)) {

List<Object>::insert(itr, x);

return --itr;

}

else {

throw SortedOrderMismatchException{};

}

}

else if (x > \*(--itr) && x <= \*(++itr)) {//insert at any other node

List<Object>::insert(itr, x);

return --itr;

}

else {

throw SortedOrderMismatchException{};

}

}

**Main.cpp:**

#include<iostream>

#include"SortedList.h"

using namespace std;

int main() {

SortedClass<int> arr;

for (int i = 1; i <= 5; i++) {

arr.insert(i);

}

cout << "Initial array:" << endl;

arr.insert(7);

for (auto i = arr.begin(); i != arr.end(); i++) { cout << \*i << " "; }

cout << endl;

cout << "Insert in the middle: " << endl;

arr.insert(6);// Insert in the middle

for (auto i = arr.begin(); i != arr.end(); i++) { cout << \*i << " "; }

cout << endl;

cout << "Insert at the start: " << endl;

arr.insert(0);// Insert at the start

for (auto i = arr.begin(); i != arr.end(); i++) { cout << \*i << " "; }

cout << endl;

cout << "Erase element 2:" << endl;

arr.erase(2);

for (auto i = arr.begin(); i != arr.end(); i++) { cout << \*i << " "; }

cout << endl << "Add element 2 back using the location:" << endl;

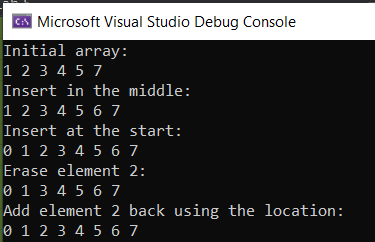
arr.insert(++(++arr.begin()),2);

for (auto i = arr.begin(); i != arr.end(); i++) { cout << \*i << " "; }

return 0;

}

**Screenshot:**



**Part b:**

1. **Add** the following friend function that computes the intersection of two given sorted lists using only the basic list operations, in linear time (i.e. O(N)).

friend sorted\_list<Object>\* intersection( const sorted\_list<Object>& list1,

const sorted\_list<Object> & list2);

1. **Add** the following friend function that computes the union of two given sorted lists using only the basic list operations, in linear time (i.e. O(N)).

friend sorted\_list<Object>\* Union( const sorted\_list<Object>& list1,

const sorted\_list<Object> & list2);

**Bonus:**

**Exercise2:**

\_

Implement the member function below that returns true if the given value is in the container, and false if not. Your code should use iterators so it works with any type of container and data.

template< typename Value, class Container >

bool member( const Value& val, const Container& cont );

You must use thetestprogram given below. Note that it uses 7 different containers: (1) vector of int, (2) array of float, (3) string object, (4) deque of char, (5) list of bool, (6) list of lists(!), (7) set of strings. There are two tests of membership for each: one true and the other false.

#include <iostream>

#include <array>

#include <deque>

#include <list>

#include <set>

#include <string>

#include <vector>

using namespace std;

#define test(result) { cout << (result ? "yes" : "no") << endl; }

template< typename Value, class Container >

bool member( const Value& val, const Container& cont )

{

***// your code goes here***

}

int main() {

vector<int> numbers{ 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 };

array<float, 6> roots{ 1.0, 1.41, 1.73, 2.0, 2.24, 2.45 };

string letters = "abracadabra";

deque<char> chars(letters.begin(), letters.end());

list<bool> flags{ true, true, true, true, true };

list<list<int>> lists{ {0,1}, {2,3,5}, {8,13,21,34,55} };

set<string> words{ "hello", "hi", "ta-ta", "bye" };

test( member( 21, numbers )); // vector of integers

test (member( 42, numbers ));

test( member( 2.0, roots )); // array of decimal roots

test( member( 3.14, roots ));

test( member( 'c', letters )); // string object

test( member( 'k', letters ));

test( member( 'd', chars )); // deque of characters

test( member( 't', chars ));

test( member( true, flags )); // list of booleans

test( member( false, flags ));

test( member( list<int>{2, 3, 5}, lists )); // list of lists!

test( member( list<int>{2, 3, 4}, lists ));

test( member( "hi", words )); // set of strings

test( member( "no", words ));

system("pause");

}