Technion

מבוא לתכנות מערכות

02340124

ש.ב 3

חלק יבש

על ידי

אדוארד איוב - 327661294

מהדי הוואש - 325394740

3.1.1

In the generic code of Sortlist.h, the necessary requirements that type-object T have to achieve are the following:

1. **The Operator >:** we use this operator in our code to guarantee that the SortedlList class stay sorted, as we you use it in the insert function to insert the given (entered) value of T is in the correct placement in the list. <= >= == != are optional operators that we can add but aren't necessary requirement.
2. **Copy Constructor:** In the generic code that we have implemented there's a necessity for the object T to copyable. As we pass object T as an argument in many functions, an action in which a copy of the object is created.
3. **Default Constructor:** T requires a defined default constructor, because it is used in the initialization of the SortedList. If not implemented this will result in a compilation error.

3.1.2

Changing the Const Iterator that we implemented in the Sortedlist.h to a Non- Const Iterator can lead to the following issue in the code:

Changing the sorted order and Inconsistency: When a non-const iterator returns a reference (&T) to the values in the list, the user (which can be unaware of the sorting mechanism used in the Sortlist.h and how it was implemented) can modify the pointed-to values directly. Changing the values will break the sorted order of the list, and by that ruining the entire concept of defining this class. If we change the value of a certain element in the list, the sorted order that we created using the insert function (that we implemented the sorting algorithm in) will be broken.

Also, Iterators created before the values are modified may point to incorrect positions in the list after the change. This can cause software errors and undefined behavior of the program.

3.1.3

We are given a list of integers.

First of all, we insert the number in a sorted list using the function “insert” that we have implemented in the sortedlist.h, so we get a sorted list.

After we sorted the given list. We should check which number in the sorted list are divisible by the Input Devisor, that’s only known during running time (and not during compilation time). To do that, we can use the filter function that we have implemented previously in sortedlist.h, which gets as an argument, a predicate which can be **lambda**, like the following:

[devisor] (int number) {return number %==0}

- Note that, before running the lambda we should check if the devisor is zero, and if so we should terminate the program!

Pseudo Code:

```
//the given list
Int givenList={…};

//creating the sorted list
Sortedlist Sorted;
For (each item in givenList){
Sorted.insert(givenlist[i]);
}

//inputing from the user the devisor number
Cin>> devisor;
If(devisor==0){
Return 0;// do nothing
}

//creating only a divisible by "devisor" list
Sortedlist OnlyDivisible=Sorted.filter([devisor] (int num) {return number %==0});
```