Lab 4 Algorithms

Task: use the STL (Standard Template Library) for implementing some algorithms by using the ready-made algorithms and boxed container terminal in STL.
Only a score of G.

Tips:

In several of the data below needed test data is unordered. There are two ways to produce these:

- You can generate random data-easy in the case of integers, harder when we sort People.
- You can find a list of people and then "mix" them – this can be done with all the items that can be sorted. There is a standard feature for this: random_shuffle

Some useful std features and the libraries they are in:

Random numbers: C++ library is awkward; the old C variant is simple:

```
#include <time.h>
#include <stdlib.h>

srand ((unsigned) time (NULL));
    int r = RAND();
```

and now r is a random positive integer (0 < = r < = RAND_MAX == 32767)

rand () gives a predictable series of random numbers if one knows what the seed is (srand sets the seed). Sometimes it can be useful when testing use this to get a different random sequence. Now we will at every test run obtain exactly the same random numbers (which simplifies troubleshooting).

Fill a container with something:

```
iota (v. begin (), v. end (), 101); //fill the v with 101, 102, 103, etc.
```

Mix v

```
random_shuffle (v. begin (), v. end ());
```

Lambdafunktioner is anonymous functions e.g.:

```
    [](int in) {return in % 10 = = 7;};
```

provides true for all numbers that have 7 as the final figure.

In <vector>, see vector class. have the methods:

- push_back
- erase

In <algorithm>

- sort
- stable_sort
- remove

To write std:: begin(v) and std:: end(v) is more general than to write v. begin (). For example, this can handle a c-array declared as int arr[10];

Tasks:

All exercises provided below to do according to the template:

1. Create a "container" in random order
2. print out the container.
3. Change it in some way (different in each task)
4. Write out the container.

What is different is the "container" and how it is changing

Task 1: Sorting

Task 1a: Sort a std::vector <int> using std:: sort

Task 1b: Sort int [] with std:: sort

Task 1 c: Sort a std::vector <int> using std:: sort
but sort it in descending order by using the rbegin, rend

Task 1 d: Sort a int [] using std:: sort
but sort it in descending order by using a lambda expression as the comparison operator and third arguments to the std:: sort.

Task 2: change the contents of a container.

A problem with algorithm library use of iterators and not the actual containers is that an iterator cannot remove elements in a container. Make a program that removes all the even numbers from a container. Use std:: remove_if to move the even numbers to the end of the container and erase to delete them. The condition to remove_if is best written as a lambda function.

Task 3: sorting of forward_list

All iterators can not do everything. For example, you can in a single list only go in one direction, for example. std:: forward_list has only forward links can just go one step at a time. I.e. operations--, - and + are missing and the standard std:: sort function does not work. In this task, you should write a ForwardSort in the same style as STL.

```
template < class ForwardIterator>
void ForwardSort(ForwardIterator begin, ForwardIterator end);
```

It should only use the forward iterator functions (i.e. you can do * it, + + it, it1! = it2 and not much more).
Test it with a forward_list.

1 Comment

Of course, do not have memory leaks in this lab (but it should not be any).
You will reuse the parts of this lab in a later lab, you will sort your mm. using iterator that you made to the dual-link list and String.