Bonus assignment 9: ordering matters

Course 'Imperative Programming' (IPC031)

1 Background

When sorting data we quickly run into the situation where we want to sort data in different orders. Taking the music database as an example, we may want to sort tracks based on their track title, album title, or running time. As we implemented our sorting functions in the mandatory assignment based on the `==` and `<` operators on `Track`, changing the sorting order would require us to change the implementation of these operators. Since we cannot change these operators once our program is compiled, it is clear we need a different strategy to allow changing the sorting order as our program is running.

A common solution to this problem is to use parametric sorting functions, where we provide a function parameter to test the ordering of tracks, rather than using the `<` operator[1]. The value of a function parameter can be any function that has the correct type signature. As such, the types of these function parameters represent the set of all possible functions with the correct type signature, much like a parameter of type `int` represents the set of all integer values. This allows us to write one version of the sorting algorithm, which can be called with any possible sorting order as a parameter, effectively telling the algorithm which `<` operator implementation to use. Instead of using `<` in our code however, we call the function parameter to perform this ordering test.

In our code the type of this function parameter is defined as `LessThan`, which can be any function of type signature `bool foo(const El& a, const El& b)`. As an example, we may wish to implement a parametric `min` function:

```
El min (const El& a, const El& b, LessThan less_than)
{
  if (less_than (a, b))
    return a;
  else
    return b;
}
```

We can then call this function as follows:

```
// Assume these functions exist and order based on different criteria.
// For example order by track title, or running time.
bool foo (const El& a, const El& b);
bool bar (const El& a, const El& b);

El a = ...;
El b = ...;

El c = min(a, b, foo);
El d = min(a, b, bar);
```

Note that we cannot write `min(a, b, <)` as `<` is not a function object according to C++. We can however use a utility function to convert it into one by writing `min(a, b, less<El>())` which allows us to use the `<` operator as the ordering function[2].

2 Assignment

Part 1: Parameterizing sorting functions

Adapt your implementation of the sorting algorithms insertion sort, selection sort, bubble sort, and heap sort to make use of an additional `LessThan less_than` parameter. Instead of comparing two elements of type `El` as `a < b`, use `less_than(a, b)`. Remember not to use any of the `==` and `<`, or derived `!=`, `<=`, `>` and `>=` operators on `El` types, only use the new `less_than` parameter.

[1] This converts our first-order function into a higher-order function, which is often seen in functional programming and covered in much more detail during the course IBC040 "Functional Programming".

[2] `https://en.cppreference.com/w/cpp/utility/functional/less`

Part 2: Comparing tracks by running time

Implement the `<` operator for `Length` values such that running times are correctly compared, *e.g.*, length 2:05 is smaller than length 10:00. Use the `Length` `<` and `==` operators when implementing `less_than_time` to comparing `Track` values such that a pair of tracks is compared on the basis of their members (length is interpreted as a running time, artist as a string, track title as a string, title of CD as a string) following this ordering. As a result, shorter tracks will precede longer tracks. Use all sorting algorithms and check that, after sorting, the ordered music database starts with the shortest track and ends with the longest track.

Part 3: Comparing tracks by track title

Define the comparison on `Track` values in `less_than_title` such that first the track title as a string is compared, followed by artist as a string, and title of CD as a string. However, the comparison on strings must be case-insensitive. This is not the case in ASCII ordering, in which all capital characters precede all lower case characters (`'A' < 'a'`). As a result, a track with a track title that occurs earlier alphabetically will precede tracks with track titles that occur later alphabetically. Use all sorting algorithms and check that, after sorting, the ordered music database starts with the track that has the alphabetically first track title and ends with the track that has the alphabetically last track title.

3 Products

As product-to-deliver you only need to upload to Brightspace “`main.cpp`” that you have created with your solution regarding the bonus assignment.

Deadline

Bonus assignment: Monday November 20, 2023, 15:30h