Golf

Conclusive document about the given assignment

Università degli Studi di Catania
Giovanni Maria Manduca | CS Student

1 : Abstract

The goal of this project is to develop an effective solution to the requests in the assignment (see the pdf).
The project, in his final version, contains classes and data structures aimed at:

- Loading data from a standardized input file (see specifications in the appendix)
- Efficiently saving said data into an appropriate data structure
- Being capable to perform a variety of operation on said data structure

At the end of the development, all the requested standards have been met.
All code shown, as well as all code in the repository, is C++, as per the specification.

2 : Introduction

One of the most important things in CS is to be practical in the solutions you develop. For this reason, one of the requirements mentioned in the specification (see specifications in the appendix) was to implement the bare minimum to make the program work as intended. For this reason, as will be described later, many possible features were not implemented unless specifically requested, although most of them are easily implemented on top of the existing framework.

3 : Problem, Approach

The problem can be subdivided into three main parts:

- Data reading
- Data organizing
- Data manipulation

In order to be able to produce an efficient solution, a simple framework has been built, keeping in mind these requests, and it consists of various classes and structs divided into four C-style header files:

- Reader.h
- List.h

- GolfList.h
- Types.h

All the external libraries used are part of the `std` namespace.

3.1: Data reading

The data reading part is handled by a `static` method in the Static[1] class Reader inside Reader.h.
This class contains only one method:
`static void populateList(GolfList* l, std::string filename)`
This method takes as input a pointer to a `GolfList` (see 3.2) and a string that contains the path to the input file.
The method is implemented in such a way that, if any malformed string is encountered while reading, the program ignores it and continues it's path.
During input parsing, instances of `Data` (see 3.2) are created and populated with the parsed output. They are then added to the `GolfList* l`, which is passed as an argument.

3.2: Data organizing

As for the organization of the data, two structures have been built:

- `List`
- `GolfList`

The data itself is organized into the `Data` struct.

`List` is an implementation of a simple dynamic list with some basic functions:

- `getSize()`
- `swap()`
- `push_back(T* elem) , push_back(T elem)`
- `pop(int pos) , pop_back()`

---

1 Although Static classes do not exist in C++, the Reader class is very similar in practice to what static classes are in other languages such as Java

- `operator[]`
- `find(T elem)`

Some of these features could also be removed from the code, as they were used primarily for debugging purposes and serve no particular use in the final product. These remained in the code because of the potential usability in expanding functionality.
A particular choice has been made regarding the "dynamicity" of the list: in particular, `List` is structured in "buckets" of fixed size, so that the reallocation of space makes the size vary each time by a constant size, defined in the class constructor and with a default value of 10.

As for the `Data` structure, it contains surname, name, club, points and year of birth of a certain player, it has a normal constructor with predefined parameters, a copy constructor and overloading of the operators == and >. In particular, the > operator compares the points of two `Data` instances, while the == returns true if and only if all members of the two `Data` instances are equal.

3.3: Data Manipulation

Though useful, the `List` class is too generic for the purposes of the project. Thus, a wrapper class `GolfList` has been implemented. This class contains all the methods that are requested in the specifications, except for the file input (see 3.1):

- `void sort()`
- `static void print(GolfList *g,bool increasing)`
- `static void infoPlayerAndSame(GolfList *g,std::string surname)`
- `void removePlayersByPoints(double points)`
- `void add(Data* d), void add(Data d)`

In particular, the `print()` function solves point b of the specification, the `infoPlayerAndSame()` function solves point c, and the `removePlayersByPoints()` function is for point d. The other functions are utility functions used by other methods/other classes. One particular choice regarding the `sort()` function was to call that function after loading the List wrapped class with all the data from the input read phase. This simplifies the `print()` function, which only needs to check whether to read and print data from `*g` in normal or reverse order.

4: Results

The code is fully capable of absolving the purposes in the specifications, and in a test of performance, has scored reasonable times.

4.1: Future work

Some future work could be done, without adding any unrequested functionality:

- implement real buckets in the `List` class, so that only one of them is reallocated when needed
- swap the `List` class for an `Hash-Map`, to guarantee a constant amortized constant time in the search of a given player
- swap the bubbleSort algorithm in the `sort()` method with a better sorting algorithm
- there is room for parallelization of parsing and data manipulation

5: Conclusions

The project aimed to represent a mock-up of a possible customer request. Although basic and expandable, the final product is fully capable of accomplishing every required task. Thus, it is safe to say that the project was successful.

6: Appendix & References

The text can be found (in Italian) in the gitHub repository, along with the code.
The input file is a simple .txt file that contains all the information of a certain player. A malformed line is one that does not have all the required information.
The link to the gitHub repository is:
https://github.com/TheRealGioviok/Golf-Tournament