Final Project

‘Vuelta a España’ – Piotr Skibiński

Warsaw, 6.06.2020

Table of Contents:

1 General overview

1.1 Short Introduction
1.2 Target group
1.3 Brief look on the Classes
1.4 Explanation on used containers
1.5 Limitations

2 Memory Map

3 Testing

3.1 Test Aim
3.2 Discussion
3.3 Tests by Classes
3.4 Examples

4 Declaration

1 General overview

1.1 Short Introduction

The aim of the project is to develop an application which is going to store necessary data behind cycling races. Program is going to store all of the data about the **c**yclists, stages and general tournament. In addition, it will be capable of storing different stages of the race, from the registration until all of the stages are completed. Furthermore, it is equipped with a simple UI system indicating access to the crucial information and the possibility of saving various types of data in ".txt" files.

1.2 Target group

The main advantage of the application is the storage and easy access to a large amount of information about the race regardless of its stage. Saving data in text files provides a banal informational path between the application and various systems.

- for tournament organizers. Due to the systems of quick data entry and execution of complex calculations related to points.
- for television and sports services. Due to the quick and easy to use ".txt" file generating system that can easily be synchronized with the graphical interface
- for website manufacturers. Quick access associated with the extreme amount of data that can certainly fill a web page.

1.3 Brief look on the Classes

Project is concentrated around 6 classes:

<u>Cyclist</u> – a class with main information about the cyclists.

<u>Team</u> – a class with information about the teams of the **c**yclists. It also includes list of sponsors of the team.

<u>Phase</u> – a class with most of the information about the phase of the tournament. Phase is defined by the status. Status describes the current stage of the phase. Project is capable of running tournaments, so Phase can be Undone, Done and Locked.

<u>Result</u> – a small class with information about the results of the **c**yclist at the current <u>Phase</u>. It could be used for general results as well.

<u>Prize</u> – a class with information of the prizes for the tournament and the <u>Result</u> of particular winner. According to the rules there could be more than one.

<u>Tournament</u> – a main class which uses each of the above classes. It handles calculations related to the points, results, winners and leaders. It is capable of several ways of filing the data to the containers and returning them back.

Finally we have two containers list and set. They will be responsible for storing data in our project as these are the methods implemented in the STL there will be no need to test them. Code is going to use several containers. Two of them inside Tour: list<**C**yclists> and list<Phases> and one inside of Phase: set<Results> and in Team: list<Sponsors>. Each class is designed to support the containers.

1.4 Explanation on used containers

To avoid additional testing I am going to use Standard Template Library containers. All elements of the library are tested and proven mechanisms that will ensure access to and handling of data placed in the tournament. For the sake of the project I would like to highlight two containers: List and Set.

- List – It is a double linked list. Iterators provide access to the specific memory in addition to that the list has implemented sort mechanism with a n log n complexity which is especially efficient when it comes to sorting large amounts of data. List will be the main container of out project because:
    1. We will not need to access random elements of the list that frequently
    2. Most of the containers do not need to be sorted. There is no need of sorting when inserting an item
    3. We will need to be able to add data at the beginning and at the end

*I will have to implement a mechanism that will check the uniqueness of the variables inside of tournament class (n complexity)

- Set – Container can be really slow in some cases but it is going to be useful inside of phase results because:
    1. All of the set elements are sorted when they are accessed by iterators
    2. Set stores only unique values in this case unique means with a different player's ID

1.5 Limitations

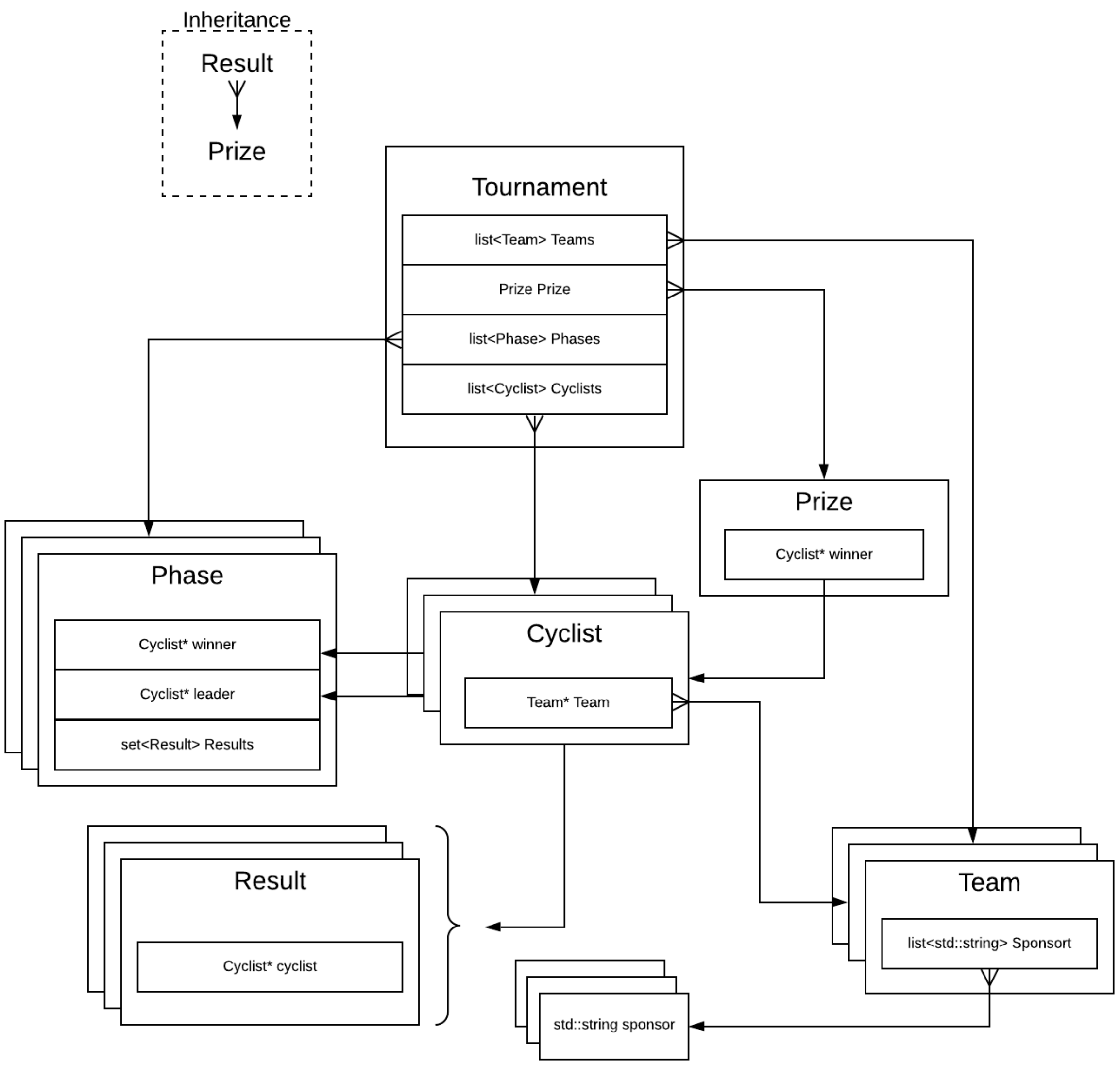
Due to the complexity of the project, some limitations have occurred.

- It is not possible to change any sort of data inside Phase when it is locked.

- It is not possible to delete individual **c**yclist because of disqualification.

- Tournament is based on Vuelta a España rules. It is not possible to change the number of categories of the winners. Currently we have Green, Blue and Red winner of the race.

2 Memory Map


Inheritance
Result
Prize
Tournament
list<Team> Teams
Prize Prize
list<Phase> Phases
list<Cyclist> Cyclists
Prize
Cyclist* winner
Phase
Cyclist* winner
Cyclist* leader
set<Result> Results
Cyclist
Team* Team
Result
Cyclist* cyclist
Team
list<std::string> Sponsort
std::string sponsor

3 Testing

3.1 Tests aim

The aim of the tests will be to:

- get familiar with the functions of the program.
- present cases of incorrect use of code.

3.2 Discussion

To test the program, “tests_file.cpp” will be implemented. Test file will include tests for all of the functions and functionalities of the code. The goal is to implement a code which is going to throw an exception whenever any method or Class will be used incorrectly.

3.3 Tests by Classes

- Tournament – the tests will check all the ways of entering data, the programme will be given various cases of erroneous data so that the operation of the errors can be presented. Finally, all data recording methods will be checked.
- Cyclist/Result/Prize - the tests will check the correctness of all functions and operators.
- Phase - the tests will check the operation of all functions, memory allocation and mechanisms blocking access to data editing when the class is "Locked".

3.4 Examples

I am going to present some examples of the test. I will not list all of the tests, because some of them are rather analogous.

Classes	Action	Data	Expected result	Result
Tournament.hpp Cyclist.hpp Phase.hpp	Adding a new cyclist	**C**yclist ID is unique	**C**yclist added to the list of **c**yclist inside tournament Class	**C**yclist added to the list
		Cyclist ID is not unique		Error generated inside of Tournament.hpp “Value is not unique”
Tournament.hpp Phase.hpp	Locking in the phase	Results of all **c**yclists are filled	Closing current phase of the tournament	Current phase receives Locked status
		Some of the results are still waiting to be filled		Error generated inside Phase.hpp “Results are not Filled!”
Tournament.hpp Phase.hpp	Trying to change the data of the phase	Phase status is Done or Undone	Change the data	Data is changed
		Phase status is Locked		Error generated inside of Phase.hpp “Phase is locked!”
Tournament.hpp Cyclist.hpp Phase.hpp	Trying to add a list of cyclists from .txt file	Text file is made in a correct way	Data added to the list.	Data is added to the list
		Text file does not exist		Error generated inside Tournament.hpp “File does not exit”
		Text file is broken		Error generated inside Tournament.hpp “Text file is broken”
		Data inside of the text file is not unique		Error generated inside Tournament.hpp “Value is not unique”

Those are only examples of tests, all of them will be placed in the file , "tests_file.cpp"

4 Declaration

Class Tournament

Members	
list<**C**yclist> Cyclists;	list of the members
list<Phase> Phases;	list of the phases
list<Team> Teams;	list of the teams
Stage_Of_Tournament Stage;	stage of the tournament
Prize* Winner_Green;	general winner from points classification
Prize* Winner_Red;	winner of team classification
Prize* Winner_Blue;	winner of the mountain classification
Public Methods	

Tournament([..]);
~Tournament();
	Constructor and Destructor

add(const enum What& what, void (Tournament::*f)(const std::string& line), Tournament& x, const std::string& path);	Addition **enum What** type of data from the file with path **path** with the useage of ***f** function to the **x** tournament.
void add_general_info(const std::string& text);	Addition of the general tournament information from the file.
void add_cyclist(const Cyclist& cyclist);	Addition of the unique cyclist when the tournament stage is Registration.
void add_team(const Team& team);	Addition of the unique team when the tournament stage is Registration
void add_results(const std::string& loc, const std::string& name);	Addition of the results when the tournament stage is In_pregress
void calculate();	Proceeding all the necessary calculations.
void lock_phases();	Locking in the phases.
void Show(const enum What& w, const enum Operations& o = OP_NONE, const enum Oper& op = O_NONE, const std::string& arg = "");	Showing **Operations** type of data from **What** class with the **Oper** symbol and the **arg** limiter.
void Save(const std::string& path, const enum What& w, const enum Operations& o = OP_NONE, const enum Oper& op = O_NONE, const std::string& arg = "");	Saving **Operations** type of data from **What** class with the **Oper** symbol and the **arg** limiter the the **path** location.
void info(std::ostream& out) const;	Returning all of the data to ostream variable.
Operators	
<<	Returning all of the data to ostream variable.
Enums	
enum What	GENERAL, PHASE, CYCLIST, TEAM
enum Oper	O_NONE, LOWER, GRATER, EQUAL, GRATER_EQUAL, LOWER_EQUAL
enum Operations	OP_NONE, ID, AGE, FIRSTNAME, LASTNAME, NATION, TEAM_ID, TEAM_NAME, TEAM_SHORT_NAME, PHASE_TYPE, PHASE_ID_ADVANCED, PHASE_STATUS
enum Stage_Of_Tournament	Registration, In_progress, Tournament_Done

Class Phase

Members	
unsigned int stage_ID;	Unique ID of the phase
string Course;	Course of the phase.
float distance;	Distance of the phase.
const Type type;	Type of the phase. Possibilities: Mountain_Stage, Hilly_Stage, Individual_Time_Trial, Flat_Stage
Status status;	Status of the phase. Possibilities: Undone, Done, Locked
Cyclist * Winner;	Winner of a phase.
Cyclist * Leader;	Leader is general classification after the phase.
set<Result> Results;	Results of all of the cyclist on phase.
Public Methods	
Phase([…]);	Constructor and destructor
~Phase();	
std::string course()const	Function returning phase course.
unsigned int phase_ID() const	Function returning phase ID.
void Finish_Registration(std::list<Cyclist>& cyclists);	Function finishing the registration for the phase.
void Enter_Results(const unsigned int id, const int Time, const int Additional_Time);	Function entering the phase results.
void Enter_Results(const unsigned int id, int const Time);	Function entering the phase results.
void Enter_Results(const std::string loc);	Function entering the phase results.
void Set_Led(Cyclist* Leader);	Changing the leader.
std::set<Result> Results_Data()	returning results data.
Cyclist* Winner_Data()	Returning winner data
Cyclist* Leader_Data()	Returning leader data
void lock();	Locking the phase
bool is_done() const;	Informing about the Status
bool is_locked() const;	Informing about the Status
Type Phase_Type() const	Returning type.
Status Phase_Status() const	Returning status.
int player_points(const unsigned int ID);	Returning player's ID points.
void info_basic(std::ostream &out) const;	Returning basic info.
void info_advanced(std::ostream &out) const;	Returning advanced phase info.
Operators	
==, !=	Comparison between stage_ID

Class Cyclist

Members	
unsigned int Unique_ID;	Unique id of the cyclist
int Age;	Age of the cyclist
string FirstName;	First name of the cyclist
string LastName;	Last name of the cyclist
string Nation;	Nation of the cyclist
string Story;	Story of the cyclist
int general_time;	General time. Is updated each time Tournament::calculate() is executed
Team* cyclist _team;	Team of the cyclist. Nullptr is the cyclist have no team
Public Methods	
Cyclist([…]);	Constructor and destructor
~Cyclist();	
unsigned int Cyclist_Team_ID() const	Returning Cyclist's team ID
std::string Cyclist_Team_Name() const	Returning Cyclist's team name
std::string Cyclist_Team_Short_Name() const	Returning Cyclist's team short name
unsigned int Cyclist _ID() const;	Returning **C**yclist's ID
string Cyclist_Name() const;	Returning **C**yclist's First Name
void Change_Cyclist_Name(const string name);	Changing **C**yclist's First Name
string Cyclist_SecondName() const;	Returning **C**yclist's First Name
void Change_Cyclist_SecondName(const string sname);	Changing **C**yclist's First Name
int Cyclist_Age() const;	Returning **C**yclist's Age
void Change_Cyclist_Age(const string age);	Changing **C**yclist's Age
string Cyclist_Nation() const;	Returning **C**yclist's Nation
void Change_Cyclist _Nation(const string nation);	Changing **C**yclist's Nation
string Cyclist _Story() const;	Returning **C**yclist's Story
void Change_Cyclist_Story(const string story);	Changing **C**yclist's Story
Team assign_team() const;	Returning **C**yclist's team or a pointer to it
void change_team(Team* team);	Changing **C**yclist's team.
void basic_info(std::ostream &out) const;	Returning basic info.
void info(std::ostream &out) const;	Returning advanced Cyclists and Team info.
Operators	
<<	Returning all of the data to ostream variable

Class Team

Members	
const unsigned int team_ID;	Unique id of the team
string name;	Name of the team
string short_name;	Short name of the team
List<string> Sponsors;	Sponsors list of the team
Public Methods	
void fill_list(std::initializer_list<string> sponsors);	Private functions which is filling the sponsors list with the content
Team([…]);	Constructor and Destructor
~Team([…]);	
unsigned int Team_ID() const;	Function returning team’s ID
std::string Team_Name() const	Function returning team’s name
std::string Team_SName() const	Function returning team’s short name
void Add_sponsor(std::initializer_list <string> sponsors);	Functions adding new sponsors
void Add_sponsor(string sponsor);	
Operators	
==, !=	Comparison between Team’s ID
<<	Returning all of the data to ostream variable

Class Result

Members	
Cyclist* cyclist;	Pointer to the cyclist
unsigned int Time;	Time on the phase in [s]
int Additional_Time;	Additional time of the phase [s]
int General_Time;	General time after the phase [s]
int Points;	Number of points granted on the phase.
Public Methods	
Results([…]);	Constructor and Destructor
~Results();	
unsigned int Cyclist _ID() const;	Function returning cyclist’s ID
void Set_Cyclist(Cyclist* cyc);	Function setting a new Cyclist
Cyclist* Cyclist_Data()const	Function returning Cyclist
int Points_Data()const	Function returning cyclists points.
void Points_Change(int points)	Function changing cyclists points.
void basic_info(std::ostream &out)	Returning all of the data to ostream variable.
Operators	
<,	Comparisons between Points
<<	Returning all of the data to ostream variable.

Class Prize – inheritance of Results

Members	
flaot Cash	Winning in cash
Public Methods	
Prize();	Constructor
float cash() const;	Returning cash
void cash_change(const float &cash);	Changing cash