COMP 2012 Object-Oriented Programming and Data Structures

Lab 2 Constructors and Destructors

France winning the 2018 FIFA World Cup in Russia.
Image Source: Russian Presidential Press and Information Office, retrieved here.

Football Match Record System

Introduction

The 2022 FIFA World Cup is coming soon! For those people who do not have time to watch the game due to timezone difference, they can read the timeline summary on the next day. An example of such summary can be found HERE.

In this lab, we will implement a simplified Football Match Record System. You will practice constructors and destructors in this lab exercise. The source file can be found HERE.

Overview

This is a sample output of the system:

Page maintained by

Ivan Choi
Last Modified: 09/27/2022 16:38:17

Homepage

Course Homepage

```
---SYSTEM PRINTING INFO---
--------------------------
MATCH 1
---LISTING PLAYERS---
FIRST COUNTRY: FRANCE
Lloris, Pavard, Varane, Umtiti, Hernandez, Kante, Matuidi, Pogba, Griezmann,
Mbappe, Giroud

SECOND COUNTRY: CROATIA
Subasic, Vrsaljko, Strinic, Lovren, Vida, Rakitic, Modric, Brozovic, Perisic,
Mandzukic, Rebic

THE GAME START!!
At minute 18, Mandzukic from CROATIA scores 1 points for FRANCE!!!
At minute 27, Kante from FRANCE received a yellow card.
At minute 28, Perisic from CROATIA scores 1 points for CROATIA!!!
At minute 38, Griezmann from FRANCE scores 1 points for FRANCE!!!
At minute 27, Hernandez from FRANCE received a yellow card.
At minute 59, Pogba from FRANCE scores 1 points for FRANCE!!!
At minute 65, Mbappe from FRANCE scores 1 points for FRANCE!!!
At minute 69, Mandzukic from CROATIA scores 1 points for CROATIA!!!
At minute 92, Vrsaljko from CROATIA received a yellow card.
THE GAME ENDS!!

--------------------------
MATCH 2
---LISTING PLAYERS---
FIRST COUNTRY: GERMANY
Manuel Neuer, Benedikt Höwedes, Mats Hummels, Jérôme Boateng, Philipp Lahm, Toni
Kroos, Bastian Schweinsteiger, Sami Khedira, Mesut Özil, Miroslav Klose, Thomas
Müller

SECOND COUNTRY: ARGENTINA
Sergio Romero, Marcos Rojo, Ezequiel Garay, Martín Demichelis, Pablo Zabaleta,
Javier Mascherano, Lucas Biglia, Enzo Pérez, Ezequiel Lavezzi, Lionel Messi,
Gonzalo Higuaín

THE GAME START!!
At minute 29, Bastian Schweinsteiger from GERMANY received a yellow card.
At minute 33, Benedikt Höwedes from GERMANY received a yellow card.
At minute 64, Javier Mascherano from ARGENTINA received a yellow card.
At minute 65, Sergio Agüero from ARGENTINA received a yellow card.
At minute 113, Mario Götze from GERMANY scores 1 points for GERMANY!!!
THE GAME ENDS!!

--------------------------
MATCH 3
---LISTING PLAYERS---
FIRST COUNTRY: Netherlands
Maarten Stekelenburg, Giovanni van Bronckhorst, Joris Mathijsen, John Heitinga,
Gregory van der Wiel, Nigel de Jong, Mark vam Bommel, Dirk Kuyt, Wesley Sneijder,
Arjen Robben, Robin van Persie

SECOND COUNTRY: Spain
Iker Casillas, Joan Capdevila, Carles Puyol, Gerard Piqué, Sergio Ramos, Sergio
Busquets, Xabi Alonso, Xavi Hernández, Andres Iniesta, Pedro, David Villa

THE GAME START!!
At minute 15, Robin van Persie from Netherlands received a yellow card.
At minute 17, Carles Puyol from Spain received a yellow card.
At minute 22, Mark vam Bommel from Netherlands received a yellow card.
At minute 23, Sergio Ramos from Spain received a yellow card.
At minute 28, Nigel de Jong from Netherlands received a yellow card.
At minute 54, Giovanni van Bronckhorst from Netherlands received a yellow card.
At minute 57, John Heitinga from Netherlands received a yellow card.
At minute 67, Joan Capdevila from Spain received a yellow card.
At minute 84, Arjen Robben from Netherlands received a yellow card.
At minute 109, John Heitinga from Netherlands received a red card!!
At minute 111, Gregory van der Wiel from Netherlands received a yellow card.
At minute 116, Andres Iniesta from Spain scores 1 points for Spain!!!
```

```
At minute 117, Joris Mathijsen from Netherlands received a yellow card.
At minute 118, Andres Iniesta from Spain received a yellow card.
At minute 120, Xavi Hernández from Spain received a yellow card.
THE GAME ENDS!!

---SYSTEM END PRINTING---
```

Note: *These are real matches.*

You can see the system is able to record multiple matches, and each match records the information of:

- Player line up for both countries.
- Events in the game sorted in time. The system record events like: Goals, Yellow Card and Red Card.

Enums and Classes

We will explain each class in detail. Since a football match involves 2 countries, to distinguish them, we will call them the "`first country`" and the "`second country`". For the varible names that involve "`first`" or "`second`", they refers to "`first country`" or "`second country`" respectively.

Enum Event_Type

```
enum Event_Type {SCORE_FIRST = 0, SCORE_SECOND, YELLOW_CARD, RED_CARD};
```

In a match, we will record different kinds of event. This enum is used to distinguish the nature of the event.

- `SCORE_FIRST`: a player puts the ball over the goal line of the first country.
- `SCORE_SECOND`: a player puts the ball over the goal line of the second country.
- `YELLOW_CARD`: a player is given a yellow card.
- `RED_CARD`: a player is given a red card.

Enum Team_Type

```
enum Team_Type {FIRST = 0, SECOND};
```

This enum is used to distinguish the team of the player involved in an event.

- `FIRST`: the player belongs to the first country.
- `SECOND`: the player belongs to the second country.

With both `Event_Type` and `Team_Type`, we can represent all the possible goals and cards given in a match. For example:

- `Event_Type = SCORE_FIRST` and `Team_Type = FIRST`: a player in the first country scores an own goal and give 1 point for the second country.
- `Event_Type = SCORE_SECOND` and `Team_Type = FIRST`: a player in the first country scores a goal and get 1 point for the first country.
- `Event_Type = YELLOW_CARD` and `Team_Type = SECOND`: a player in the second country is given a yellow card.

Class Event

```
class Event{
  public:
  Event_Type event_type;
  Team_Type team_type;
  char* name;
  int minute;

  Event(Event_Type event_type, Team_Type team_type, const char* name, int
minute);
  Event(const Event& event);
  ~Event();

  void print_info(const char* first_country_name, const char*
second_country_name) const;
};
```

The `Event` class will record an event (like goals, cards given) happening in the match at certain time. It has already been implemented.

- `event_type`: It represents the type of the event.
- `team_type`: It represents the team that the player belongs to.
- `name`: The name of the player in the event. It is a pointer to a dynamically allocated array of type `char`
- `minute`: The time where the event happens.

```
void Event::print_info(const char* first_country_name, const char*
second_country_name) const;
```

This is a function for printing the sample output. It has already been implemented.

For the constructors and destructor in this class, we will introduce them in the Task section later.

Class Match

```
class Match{
  public:
  Event* events[30];
  int num_event = 0;

  char* first_country_name;
  char** first_players;
  int num_first_players;

  char* second_country_name;
  char** second_players;
  int num_second_players;

  Match(const char* first_country_name, const char* second_country_name, const
char** first_players, int num_first_players, const char** second_players, int
num_second_players);
  Match(const Match& match);
  ~Match();

  void create_event(Event_Type event_type, Team_Type team_type, const char* name,
int minute);
  void print_info() const;
};
```

The `Match` class will record all the information in a match, including players from both side, country names and a list of events.

- `event`: It is an array of `Event*`. Each pointer points to a dynamically allocated `Event` object. You can assume there will not be more than 30 events in 1 match.
- `num_event`: It represents the number of events stored in class data member `event`.

- `first_country_name`: The name of the first country. It is a pointer to a dynamically allocated `char` array.
- `first_players`: The name of players in the first country. It is a pointer to a dynamically allocated `char*` array, where each pointer points to another dynamically allocated `char` array.
- `num_first_players`: The number of players in the first country.
- `second_country_name, second_players, num_second_players`: They are similar to the ones for the first country one.

```
void Match::create_event(Event_Type event_type, Team_Type team_type, const char*
name, int minute);
```

This is a function to create an event in the match. It has already been implemented.

```
void Match::print_info() const;
```

This is a function for printing the sample output. It has already been implemented.

For the constructors and destructor in this class, we will introduce them in the Task section later.

Class System

```
class System{
  public:
  Match* matches[10];
  int num_matches = 0;

  System();
  System(const System& system);
  ~System();

  void add_match(Match* match);
  void print_info() const;
};
```

The `System` class will record matches. It has already been implemented.

- `matches`: It is an array of `Match*`, where each pointer points to a dynamically allocated `Match` object. You can assume a system will not store more than 10 matches.
- `num_matches`: It represent the number of matches stored in class data member `matches`.

```
void System::add_match(Match* match);
```

This is a function to add a match into the system.

```
void System::print_info(const char* first_country_name, const char*
second_country_name) const;
```

This is a function for printing the sample output.

For the constructors and destructor in this class, we will introduce them in the Task section later.

Lab tasks

You can download the source file HERE.

The zip file contains: `system.h`, `system.cpp`, `main.cpp`, `Makefile` and `sample-out.txt`. All the class implementation resides in `system.cpp`, and all testing resides in `main.cpp`. You may check the sample output messages in `sample-out.txt`.

You will implement all constructors and destructor of class `Match` in this lab exercise. Before you implement them, you should have a quick look of how different functions and constructors are called in both `main.cpp` and `system.cpp`. Reading the functions/constructors/destructors that are already implemented by us may be helpful for better understanding the structure of this lab exercise.

Task1: Implement Constructor of class Match

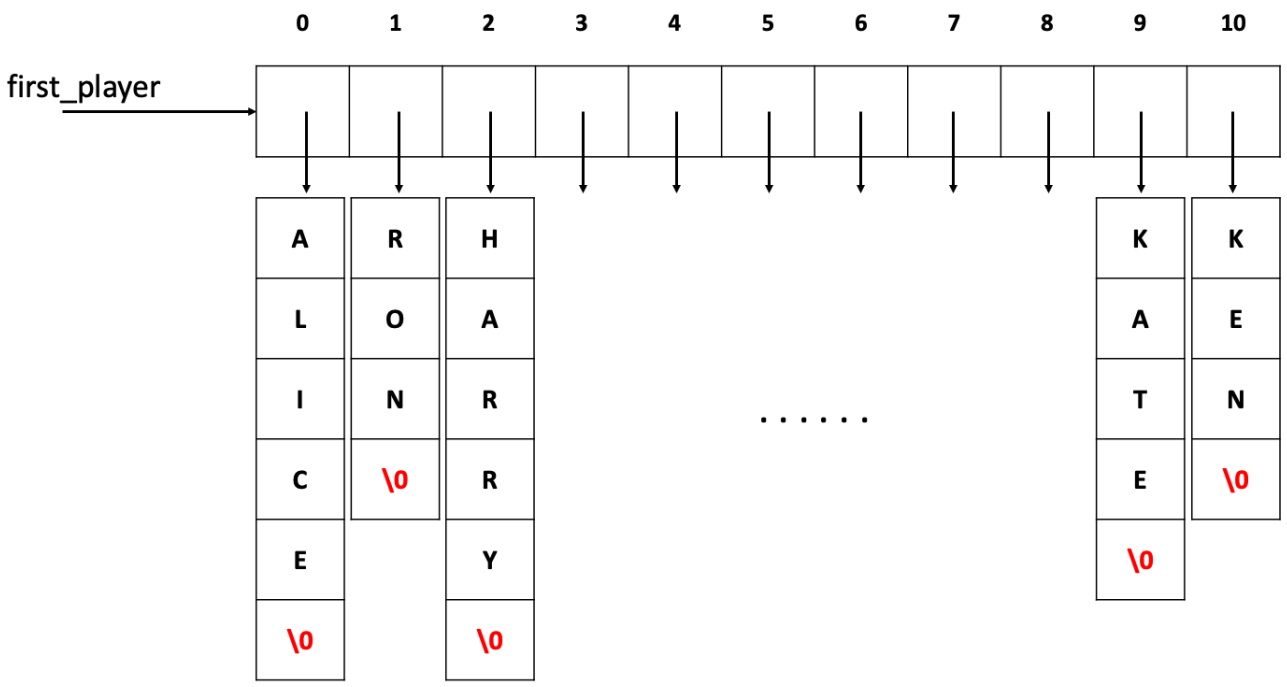
```
Match::Match(const char* first_country_name, const char* second_country_name,
const char** first_players, int num_first_players, const char** second_players,
int num_second_players);
```

This constructor is quite complicated and takes a lot of argument. You should perform the followings in this constructor:

- Since there is no any event in a match when a match is first constructed, assign `nullptr` to every entry of `event`. (How can we do this quickly?)
- Copy assignment of `num_first_players` and `num_second_players`.
- Perform deep copy of `first_country_name`, `second_country_name`, `first_players` and `second_players`.

When performing a deep copy of a C-string object (`const char*`) like like `country_names` and `players`, you should allocate just enough memory to hold the characters and the `'\0'` null character. You may find P.11 of the Constructors & Destructor lecture notes useful.

To better understand the class data member `first_players` and `second_players`, you may refer to the following figure.



This figure assume we perform deep copy to the class data member `first_players` from the following input:

```
const char* somebody[11] = {"Alice", "Ron", "Harry", ..., "Kate", "Ken"}
```

To create a deep copy of this parameter, you need to first create a dynamic array of type `char*` with size 11. Then perform deep copy for every player's name.

Task2: Implement Copy Constructor of class Match

```
Match::Match(const Match& match);
```

This copy constructor should perform a deep copy of the parameter `match`. You may find copy constructor of class `Event` useful when performing deep copy of the `events`.

Task3: Implement Destructor of class Match

```
Match::~Match();
```

Clean up all dynamic objects owned by this class to prevent memory leak. Make sure you take care of the following class data members which points to an array of dynamic objects:

- first_country_name
- second_country_name
- events
- first_players
- second_players

Note: pay extra attention to `first_players` and `second_players` as they point to 2 dimension dynamic arrays.

Compile and Test

After you finish all tasks, you can compile the files by typing `make all` in the terminal to produce executable `lab2.exe`. Your program should be able to regenerate the sample output in overview section in all test cases. You may also find the sample output in the source file. You should read the `main.cpp` file to understand how the sample output is generated. Since this lab involve a lot of dynamic memory usage, you may also want to check for memory leak using `valgrind` or `leak sanitizer`.

Test Cases

We have 4 test cases on Zinc for this lab exercise. You can find them in `main.cpp`.

1. Create `system` and check whether calling `print_info` from `system` matched with the standard output. Testing the correctness of constructor.
2. Create `system2` as a copy of `system` and check whether calling `print_info` from `system2` matched with the standard output. Testing the correctness of copy constructor.
3. Test Case 1 but with memory leak check. Testing the correctness of destructor.
4. Test Case 2 but with memory leak check. Testing the correctness of copy constructor and destructor.

You can run the first two test cases by typing `./lab2.exe` in the terminal and input 1 or 2. To run test case 3 or 4, you need to run lab2_leak.exe which checks memory leakage.

Note: All the 4 test cases should produce output same as the one in `sample-out.txt`.

Note 2: You can observe that in test case 1 and 2, we do not delete the dynamically allocated `system` or `system2`. That is because we try to avoid testing the correctness of destructor in the first 2 cases. They are designed for grading purpose, but do remember that it will cause memory leak!

Submission Deadline and Guidelines:

The lab assignment is due on 10 minutes after the end of your lab session.
We will use an online grading system ZINC to grade your lab work. Therefore, you are supposed to upload the following files in a zip file to ZINC:

```
- system.cpp
```

You can submit your codes multiple times to ZINC before the deadline. Only the last submission will be graded.