Lab 10. Classes, enums, and movies

Introduction

Classes in C++ are *user-defined* compound datatypes that can be used for grouping together several related variables. These 'variables' inside a class are called fields (or members) of the classes. Variables of of these class types are refered to as **objects**.

Let's define a datatype (class) for storing time HH:MM

We want to create this new class/datatype for representing time in the 24-hour format. It is reasonable to define it to have two integer fields:

- `h` for the number of hours, and
- `m` for the number of minutes.

Notice the semicolon after the closing `}`, it is necessary:

```
class Time {
public:
    int h;
    int m;
};
```

Alright, let's try to use this new fancy type. There are several ways for creating objects (or variables) of this type `Time`. The most explicit one is to create an object (a variable) and then update its fields individually:

```
Time now;     // creates a new variable (or object)
now.h = 17;   // assigns its hours field
now.m = 45;   // assigns its minutes field
```

The object/variable is called `now`, and it is storing 5:45 PM. The fields of `now` can be accessed as `now.h` ans `now.m`.

Alternatively, we can create an object and immediately initialize all its fields with the following literal syntax (the order of values in curly braces is important):

```
Time t = { 17, 45 };
```

Let's write a printer function that prints a `Time` value on the screen in HOURS:MINUTES format:

```
void printTime(Time time) {
    cout << time.h << ":" << time.m;
}
```

One convenient feature, which you might not need but should be aware of, is that the assignment operator works for objects, it copies all the elements of the object, field by field:

```
Time morningLecture = {8, 10};

Time myAlarm;                 // make another variable
```

```
myAlarm = morningLecture;    // copying

printTime(morningLecture);   // will print 8:10
printTime(myAlarm);        // will print 8:10 as well
                           // You may be late for the class tho
```

Task A. Simple functions for time

Create a new program `time.cpp`. (Copy the class `Time` declaration in your program, it should be placed before `main()` function.)

Implement two new functions:

```
int minutesSinceMidnight(Time time);
int minutesUntil(Time earlier, Time later);
```

The first function should return the number of minutes from 0:00AM until `time`.

The second function should receive two `Time` arguments `earlier` and `later` and report how many minutes separate the two moments. For example, when passing 10:30AM and 1:40PM:

```
minutesUntil( {10, 30}, {13, 40} )
// ==> should return 190 minutes
```

(A caveat: If the `earlier` moment of time happens to be after the `later` moment of time, report a negative number of minutes. Although it’s not difficult to achieve this if your implementation for the first function is correct.)

For testing purposes, implement a simple user interface:

```
$ ./time
Enter first time:  10 30
Enter second time: 13 40
```

```
These moments of time are X and Y minutes after midnight.
The interval between them is Z minutes.
```

Task B. Making it more interesting

Add a new function to your program `time.cpp`:

```
Time addMinutes(Time time0, int min);
```

The function should create and return a new moment of time that is `min` minutes after `time0`. Example:

```
addMinutes({8, 10}, 75)
// ==> should return {9, 25}
```

(We will not test how your function behaves if the new returned time will be on the next day, feel free to assume that it will remain withing the same day, **≤ 23:59**.)

Adjust the `main` function for testing this function. Feel free to add additional tests to check the correctness of your code.

Scheduling time slots for a movie theater

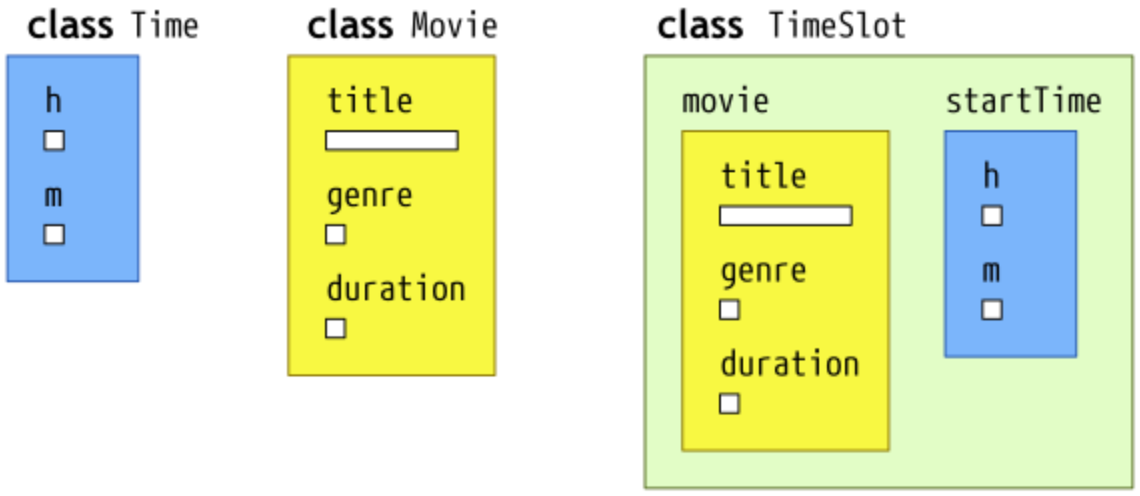
Let's add a few more datatypes:

```
enum Genre {ACTION, COMEDY, DRAMA, ROMANCE, THRILLER};
```

(Enum types work as sets of named values. A variable of type `Genre` can assume any of the values listed in the curly braces, example: `Genre myFavorite = COMEDY;`)

```
class Movie {
public:
    string title;
    Genre genre;     // only one genre per movie
    int duration;    // in minutes
};

class TimeSlot {
public:
    Movie movie;     // what movie
    Time startTime;  // when it starts
};
```



A usage example:

```
Movie movie1 = {"Back to the Future", COMEDY, 116};
Movie movie2 = {"Black Panther", ACTION, 134};

TimeSlot morning = {movie1, {9, 15}};
TimeSlot daytime = {movie2, {12, 15}};
TimeSlot evening = {movie2, {16, 45}};
```

For testing purposes, we would want to make functions `printMovie(Movie m)` and `printTimeSlot(Timeslot ts)`. Let’s write the first of them:

```
void printMovie(Movie mv){
    string g;
```

```
        switch (mv.genre) {
            case ACTION   : g = "ACTION"; break;
            case COMEDY   : g = "COMEDY"; break;
            case DRAMA    : g = "DRAMA";  break;
            case ROMANCE  : g = "ROMANCE"; break;
            case THRILLER : g = "THRILLER"; break;
        }
        cout << mv.title << " " << g << " (" << mv.duration << " min)";
    }
```

(It does not print `endl` at the end, and `printTime`, which we defined in the beginning, did not print `endl` either. Thanks to that, both functions can be uses as part of `printTimeSlot` function.)

Task C. TimeSlot ending time and printTimeSlot

In the same program `time.cpp`, implement your own printing function `printTimeSlot(TimeSlot ts)`. It should make output in the following format:

```
Back to the Future COMEDY (116 min) [starts at 9:15, ends by 11:11]
```

The ending time is the starting time + movie duration.

Write main function that defines at least five time slots

- `morning`, `daytime`, and `evening` defined previously,
- plus add a couple of your own time slots with some of your favorite movies (their duration time can be found in IMDB).

The program output should look like:

```
$ ./time
Back to the Future COMEDY (116 min) [starts at 9:15, ends by 11:11]
Black Panther ACTION (134 min) [starts at 12:15, ends by 14:29]
Black Panther ACTION (134 min) [starts at 16:45, ends by 18:59]
 -- your time slot #1 --
 -- your time slot #2 --
 --        ...        --
```

When defining your own time slots, please make sure they *end before midnight*, 23:59, so the ending time does not show the next day.

Task D. Scheduling X after Y?

Add a new function

```
TimeSlot scheduleAfter(TimeSlot ts, Movie nextMovie);
```

The function should produce and return a new `TimeSlot` for the movie `nextMovie`, scheduled immediately after the time slot `ts`.

For example, if the movie scheduled in `ts` starts at 14:10 and lasts 120 minutes, then the time slot for the next movie should start at exactly 16:10.

Modify `main` function to test your code.

Task E. Overlapping time slots?

Add a new function

```
bool timeOverlap(TimeSlot ts1, TimeSlot ts2);
```

The function should return `true` if the two time slots overlap, otherwise return `false`. (Take into account the starting times of the time slots and the duration of the scheduled movies.)

Modify `main` function to test your code.

Hint: You may use `minutesUntil` to check which time slot is earlier, then find the how long is the interval between their starting times. They overlap if the movie duration is *greater* than the interval between the time slots' starting times. Alternatively, converting times into minutes since midnight can be a good idea as well.

(By the way, if you want to be accurate, if one movie starts at 10:00 and lasts 90 minutes until 11:30, then it *does not* overlap with a movie that starts exactly at 11:30. However, they would overlap if the latter movie started one minute earlier, at 11:29.)

How to submit your programs

Each program should be submitted through Gradescope

Write separate programs for each part of the assignment.
Submit only the source code (.cpp) files, not the compiled executables.
Each program should start with a comment that contains your name and a short program description, for example:

```
/*
Author: your name
Course: CSCI-136
Instructor: their name
Assignment: title, e.g., Lab1A

Here, briefly, at least in one or a few sentences
describe what the program does.
*/
```