VE 280 Lab 7

Out: 00:01 am,July 1st, 2020; **Due**: 11:59 pm, July 7, 2021.

You are going to become a junior student next semester and you expect your workload to be high as you plan to take some upper level technical electives. To better organize the tasks in all your future courses, you decide to use the skills you learned in VE280 and plan to write a program that can store, update and print all unfinished tasks in a course. Before starting to code, you need to choose a representation for tasks and courses.

For tasks, you decide to use the `struct` type. Tasks are identified by their types and indices, which means two different tasks will not have both the same type and the same index. A due date is also specified in a task.

```
struct Task{
    std::string type;
    int index;
    int dueMonth;
    int dueDay;
};
```

where `type` could be "Lab", "Project", "Assignment" and so on, `index` is the number of the task for that `type`. For example, Lab 7 due on July 7 is represented as `Task lab7 = {"Lab", 7, 7, 7};`. A task is submitted via the online judgement system, canvas or GitHub. Each course has a certain number of unfinished tasks at a given time.

For courses, you decide to use an ADT `Course` with three operations:

- `updateTask` adds a new task or updates the due date of an existing task;
- `finishTask` removes a task from the unfinished tasks of a course;
- `print` prints the unfinished tasks of a course.

For implementation details, please refer to the exercises below.

Ex.1

Related Topics: *virtual function, interface*

First of all, in order to hide all implementation details, you decide to realize the ADT `Course` as a homonym virtual base class in `course.h`. Regarding the implementation of this base class, you decide that the unfinished tasks are managed with an array `tasks`.

The three operations of this ADT are defined as follows in the base class:

```
void updateTask(const std::string &type, int index, int dueMonth, int dueDay);
// REQUIRES: dueMonth and dueDay are in normal range.
// MODIFIES: this
// EFFECTS: adds/updates Task index of type; throw exception if fails to add Task
void finishTask(const std::string &type, int index, int finishMonth, int finishDay);
// REQUIRES: Task index of type exists in tasks. finishMonth and finishDay are in normal range.
// MODIFIES: this
// EFFECTS: removes Task index of type
void print();
// EFFECTS: prints all unfinished tasks of this Course
```

"in normal range" means that there is no date like `dueMonth=14` or `dueMonth=4, dueDay=31`.
In `updateTask`, if array `tasks` is full, then you need to throw an exception of the `tooManyTasks` type, which is a class defined in `course.h`.
`print` will print the course code and all elements in `tasks` in order. The implementation of `print` is already given to you, please do not modify it.

This ADT is implemented as a derived class `TechnicalCourse` in `course.cpp`. It has four protected data members:

```
Task *tasks;                // Array of current unfinished tasks
int sizeTasks;              // Maximum number of unfinished tasks in the array
int numTasks;               // Number of current unfinished tasks in the array
std::string courseCode;     // Course code, e.g. "VE280"
```

This derived class represents courses like VE280, which require to submit labs/projects via the online judgement system and submit other work via canvas.

When you create a new `TechnicalCourse`, `numTasks` should be initialized to 0, `courseCode` should be initialized according to the input. If the input argument `size` is specified, `sizeTasks` should be initialized as `size`; otherwise, it should be initialized as the default value `MAXTASKS`, which is 4.

```
TechnicalCourse(const std::string &courseCode, int size = MAXTASKS);
```

As for the three methods:

- `updateTask` takes the `type`, the `index`, the `dueMonth` and the `dueDay` of the task to be updated as inputs. If the task already exists in the array `tasks`, you should update its `dueMonth` and `dueDay`. If it is a new task, inserts it at the end of `tasks` and throws an exception of type `tooManyTasks` if `tasks` is full.

  After inserting tasks whose `type` is "Lab" or "Project", you need to print a message:

  ```
  <courseCode> <type> <index> is released! Submit it via oj!
  ```

  After inserting tasks of type other than "Lab" and "Project", you need to print another message:

```
<courseCode> <type> <index> is released! Submit it via canvas!
```

Example:

```
// ve281 is a pointer to an instance of TechnicalCourse with courseCode "VE281"
ve281->updateTask("Assignment", 1, 5, 10);
ve281->updateTask("Lab", 1, 5, 20);
ve281->updateTask("Project", 1, 5, 30);
ve281->updateTask("Lab", 1, 5, 15); // no message is printed since it already exists in tasks
```

Example output:

```
VE281 Assignment 1 is released! Submit it via canvas!
VE281 Lab 1 is released! Submit it via oj!
VE281 Project 1 is released! Submit it via oj!
```

If you print the tasks in `ve281` after updating the three tasks like above,

```
ve281->print();
```

the output will be:

```
VE281
Assignment 1: 5/10
Lab 1: 5/15
Project 1: 5/30
```

- `finishTask` takes the `type` and the `index` of the newly finished task as well as the time when you finish it, `finishMonth` and `finishDay` as inputs. It will search the corresponding task in `tasks` according to `type` and `index`, and remove it from `tasks`.

  If the task is finished before/on the due date, you need to print:

  ```
  <courseCode> <type> <index> is finished!
  ```

  If you fail to finish the task before the due date, you need to print:

  ```
  <courseCode> <type> <index> is overdue!
  ```

  Example:

  ```
  // follows the code in the updateTask part
  ve281->finishTask("Assignment", 1, 5, 5);
  ve281->finishTask("Lab", 1, 5, 16);
  ```

  Example output:

  ```
  VE281 Assignment 1 is finished!
  VE281 Lab 1 is overdue!
  ```

If you print the unfinished tasks use `print` after finishing these two tasks, the output will be:

```
VE281
Project 1: 5/30
```

- `print` is already implemented.

Ex.2

Related Topics: *subclass, inheritance, virtual function*

You will start to take upper level technical courses next semester. `UpperLevelTechnicalCourse` inherits from `TechnicalCourse`, while there are two differences between them.

First, there is a new type of tasks, "Team Project". For this type of tasks, you need to cooperate with your teammates and share the codes through GitHub. So after inserting tasks of "Team Project", a message should be printed:

```
<courseCode> <type> <index> is released! Submit it via github!
```

Second, since the number of tasks in a upper level technical course are a lot larger than the number of tasks in a technical course, you can only finish all the tasks before due by following the principle of "Earliest Deadline First". So the tasks in these courses will be ordered according to their due dates, where the task with earlier due date will be put in the front.

So, you need to rewrite `updateTask` of this class. When inserting a new task, it needs to decide the position of this task in array `tasks` according to its due date. Also, when updating the due date of an existing task, it should also modify the position of this task in `tasks`. If the task exists and the due date does not change, `updateTask` will do nothing. If two tasks have the same due dates, the one that is updated earlier will be in the front.

Example:

```
// ve482 is a pointer to an instance of UpperLevelTechnicalCourse with
courseCode "VE482"
ve482->updateTask("Homework", 1, 9, 18);
ve482->updateTask("Lab", 1, 9, 16);
ve482->updateTask("Team Project", 1, 10, 6);
ve482->print();
ve482->updateTask("Homework", 1, 9, 15);
ve482->print();
ve482->updateTask("Lab", 1, 9, 15);
ve482->updateTask("Homework", 1, 9, 15); // do nothing
ve482->print();
```

Example output:

```
VE482 Homework 1 is released! Submit it via canvas!
VE482 Lab 1 is released! Submit it via oj!
VE482 Team Project 1 is released! Submit it via github!
VE482
Lab 1: 9/16
Homework 1: 9/18
```

```
Team Project 1: 10/6
VE482
Homework 1: 9/15
Lab 1: 9/16
Team Project 1: 10/6
VE482
Homework 1: 9/15
Lab 1: 9/15
Team Project 1: 10/6
```

Testing

Since more than one instance of the class is needed, a function, with overload, is provided to create them dynamically:

```
Course *create(const std::string &classType, const std::string &courseCode);
Course *create(const std::string &classType, const std::string &courseCode,
int taskSize);
```

`classType` could be either "Technical" or "Upper Level Technical". `courseCode` specifies the course code of the course you want to create. If `classType` is "Technical", then it returns a pointer to an instance of `TechnicalCourse` with `courseCode`; if `classType` is "Upper Level Technical", then it returns a pointer to an instance of `UpperLevelTechnicalCourse` with `courseCode`; if classType is not "Technical" or "Upper Level Technical", it returns a null pointer.

Besides, if `taskSize` is true, the maximum number of tasks in this course at a time is specified by `taskSize`; otherwise, the maximum number of tasks is the default value `MAXTASKS`.

Example:

```
Course *ve281 = create("Technical", "VE281");
ve281->updateTask("Project", 1, 10, 20);
ve281->updateTask("Lab", 1, 9, 20);
ve281->print();
delete ve281;
```

Example output:

```
VE281 Project 1 is released! Submit it via oj!
VE281 Lab 1 is released! Submit it via oj!
VE281
Project 1: 10/20
Lab 1: 9/20
```

`simpletest.cpp` is provided for you to test your result and the output is in `simpletest.out`. You can compile them by

```
g++ -Wall -o lab7 simpletest.cpp course.cpp -std=c++11
```

Run `./lab7 > mysimpletest.out` and use `diff` to see if your answer is correct.

This `simpletest.cpp` is just an example and you still need to write test cases yourself to get full scores.

Please make sure there is no memory leak!

You can use `valgrind --leak-check=full ./lab7` to run the program and check memory leaks. Or you can add `-fsanitize=leak -fsanitize=address` when compiling the program.

Submitting

You can find `course.h` and `course.cpp` in `lab7Starter_files.zip`. Please implement all the methods needed, compress these two files into a zip file and submit to JOJ.

Created by Yaxin Chen.

Modified by Jiayao Wu.

Last update: June 28, 2021

@UM-SJTU Joint Institute