CSE 241 Programming Assignment 3

DUE

April 24, 2022, 23:55

Description

- This is an individual assignment. Please do not collaborate
- If you think that this document does not clearly describes the assignment, ask questions before its too late.

This assignment is about implementing and testing classes which use dynamic memory. You are going to write create an interactive program. This program will not read and write any files. All of the operations will be through `stdin` and `stdout`

Put all of class and function definitions under the namespace `PA3`

Your submission will have the following files:

- Student.h
- Student.cpp
- Course.h
- Course.cpp
- SchoolManagerSystem.h
- SchoolManagerSystem.cpp
- main.cpp

Design and implement the following classes for your program(Feel free to add any other things if you need to):

- `Student`:
  - Members: `name`, `ID`, course references (this is going to be a dynamic array)
  - Constructors: the default constructor, a constructor which takes `name` and `ID` info...
  - Destructor
  - get/set functions for members.
  - ...
- `Course`:
  - Members: `name`, `code`, student references (this is going to be a dynamic array)
  - Constructors: the default constructor, a constructor which takes `name` and `code` info...
  - Destructor
  - get/set functions for members.
  - ...
- `SchoolManagementSystem`:
  - Members: course references (this is going to be a dynamic array), student references (this is going to be a dynamic array), any other member you want to (you cannot user `STL` vectors, variable length arrays, local arrays, global arrays)
  - Constructors
  - Members related with menu functionality...
  - ...

Menu of School Management System

The explanations inside the parentheses are not printed.

```
Main_menu
  0 exit
  1 student
    0 up
    1 add_student (the user enters student name and ID. A student object is created)
    2 select_student (the user enters student name and ID)
      0 up
      1 delete_student (after delete, print student menu)
      3 add_selected_student_to_a_course (this will list all of the courses except
                                  the ones which are already taken by the student)
        0 up
        1 code course name
        2 code course name
        3 code course name
        ...
      4 drop_selected_student_from_a_course (this will list the courses which are taken by the student)
        0 up
        1 code course name
        2 code course name
        3 code course name
        ...
  2 course
    0 up
    1 add_course (the user enters course code and name. A course object is created)
    2 select_course (the user enters course code and name)
      0 up
      1 delete_course (after delete, print course menu)
      2 list_students_registered_to_the_selected_course
  3 list_all_students
  4 list_all_courses
```

Example

Main function will create a `SchoolManagementSystem` object and run the menu.

Below is an example run of the program. Not all of the functionality is tested.

```
Main_menu
0 exit
1 student
2 course
3 list_all_students
4 list_all_courses
>> 1
0 up
1 add_student
2 select_student
>> 1
>> James Webb 123456
0 up
1 add_student
2 select_student
>> 1
>> Joe Satriani 0000
0 up
```

```
1 add_student
2 select_student
>> 0
0 exit
1 student
2 course
3 list_all_students
4 list_all_courses
>> 2
0 up
1 add_course
2 select_course
>> 1
>> CSE241 Object Oriented Programming
0 up
1 add_course
2 select_course
>> 0
0 exit
1 student
2 course
3 list_all_students
4 list_all_courses
>> 1
0 up
1 add_student
2 select_student
>> 2
>> Joe Satriani 0000
0 up
1 delete_student (after delete, print student menu)
3 add_selected_student_to_a_course
4 drop_selected_student_from_a_course
>> 3
0 up
1 CSE241 Object Oriented Programming
>> 1
0 up
1 delete_student (after delete, print student menu)
3 add_selected_student_to_a_course
4 drop_selected_student_from_a_course
>> 0
0 up
1 add_student
2 select_student
>> 0
0 exit
1 student
2 course
3 list_all_students
4 list_all_courses
>> 0
```

Remarks

- Don't forget to delete any dynamic allocation which is not needed anymore.
- With any user input error, program prints the same menu again.

- The program does not print any error messages.
- The order of the printed lists is not important.
- **You are not allowed to create copies of the same object permanently**
- All instances of Course and Student classes will be created dynamically.
- The program will not print anything other than the menu items.
- You cannot user STL vectors, variable length arrays, local arrays, global arrays
- Write comments in your code.
- If your code does not compile you will get `0`
- Do not share your code with your classmates.
- **Remove any print statements which you use for debug purposes.**

Turn in:

- You are going to create a **zip** archive which includes the following files:
  - Student.h
  - Student.cpp
  - Course.h
  - Course.cpp
  - SchoolManagerSystem.h
  - SchoolManagerSystem.cpp
  - main.cpp
- Name of the file should be in this format: **<full_name>_<id>.zip**. If you do not follow this naming convention you will loose -10 points.
- The archive type should be **zip**. The archive should be flat. When extracted, the files **should not** be placed in a subdirectory.
- Your code will be compiled according to a makefile which is something similar to the following GNU make script.

```
SRC_DIR := .
OBJ_DIR := .
SRC_FILES := $(wildcard $(SRC_DIR)/*.cpp)
OBJ_FILES := $(patsubst $(SRC_DIR)/%.cpp,$(OBJ_DIR)/%.o,$(SRC_FILES))
LDFLAGS := ...
CPPFLAGS += -std=c++11
CXXFLAGS += -MMD
-include $(OBJ_FILES:.o=.d)

main.out: $(OBJ_FILES)
    g++ $(LDFLAGS) -o $@ $^

$(OBJ_DIR)/%.o: $(SRC_DIR)/%.cpp
    g++ $(CPPFLAGS) $(CXXFLAGS) -c -o $@ $<
```

- You don't need to use an IDE for this assignment. Your code will be compiled and run in a command window.
- Your code will be compiled and tested on a Linux machine(Ubuntu). GCC will be used.
- A script will be used in order to check the correctness of your results. So, be careful not to violate the expected output format.
- Provide comments unless you are not interested in partial credit. (If I cannot easily understand your design, you may loose points.)
- You may not get full credit if your implementation contradicts with the statements in this document.

Late Submission

- Not accepted

Grading (Tentative)

- `Max Grade` : `100`.
- Multiple tests(at least `5`) will be performed.

All of the followings are possible deductions from `Max Grade`.

- Do **NOT** use hard-coded values. If you use you will loose `10pts`.
- No submission: `-100`. (be consistent in doing this and your overall grade will converge to `N/A`) (To be specific: if you miss `3` assignments you'll get `N/A`)
- Compile errors: `-100`.
- Irrelevant code: `-100`.
- Major parts are missing: `-100`.
- Unnecessarily long code: `-30`.
- Inefficient implementation: `-20`.
- Using language elements and libraries which are not allowed: `-100`.
- Not caring about the structure and efficiency: `-30`. (avoid using hard-coded values, avoid hard-to-follow expressions, avoid code repetition, avoid unnecessary loops).
- Significant number of compiler warnings: `-10`.
- Not commented enough: `-10`. (Comments are in English. Turkish comments are not accepted).
- Source code encoding is not `UTF-8` and characters are not properly displayed: `-5`. (You can use 'Visual Studio Code', 'Sublime Text', 'Atom' etc... Check the character encoding of your text editor and set it to `UTF-8`).
- Missing or wrong output values: `Fails the test`.
- Output format is wrong: `-30`.
- Infinite loop: `Fails the test`.
- Segmentation fault: `Fails the test`.
- Fails 5 or more random tests: `-100`.
- Fails the test: `deduction up to 20`.
- Prints anything extra: `-30`.
- Unwanted chars and spaces in output: `-30`.
- Submission includes files other than the expected: `-10`.
- Submission does not follow the file naming convention: `-10`.
- Sharing or inheriting code: `-200`.