Due Date: Mar 13, 2020 @ 11:59:59

Object oriented programming – Department database

This assignment will focus on the concepts of Object-oriented programming in C++. In this assignment, we create 5 classes with member variables and functions. We reuse the code better by inheriting from the base class and performing slight modifications on the derived class. We also demonstrate multiple inheritance and write a program that creates a vector of type base pointer, which can hold the object to any derived class instance. Here, an object can be a Faculty, Student or a Teaching assistant. The respective display function is determined at run-time exhibiting polymorphism. You will also implement some display utility functions to display only the persons/students associated with a specific course.

You will submit your work to Gradescope and it will be both auto graded and human graded. The autograder will compile your code with g++-7.4.0. Be sure to watch Piazza for any clarifications or updates to the assignment.

Don't forget the Assignment 4 Canvas Quiz! Your assignment submission is not complete without the quiz.

Your submission must consist of the following files: **Person.h**, **Person.cpp, Employee.cpp, Employee.h, Student.cpp, Student.h, Faculty.cpp, Faculty.h, TeachingAssistant.cpp, TeachingAssistant.h, DisplayUtility.h, DisplayUtility.cpp, demo.cpp, README.md** and a **Makefile**. There are 5 classes to be implemented in total and, header and source file are required for each class. **DisplayUtility.cpp** defines the required display functions that are called from **main()** and as usual, **demo.cpp** implements the **main()** function.

For header files, in general *(*.h):*

- Be sure to have include guards.
- Don't import anything that is unnecessary.
- Include file header comments that have your name, student id, a brief description of the functionality, and be sure to cite any resource (site or person) that you used in your implementation.
- Each function should have appropriate function header comments.
  - You can use javadoc style (see doxygen) or another style but be consistent.
  - Be complete: good description of the function, parameters, and return values.
  - In your class declarations (and header files in general), only inline appropriate functions.

For source files, in general *(*.cpp):*

- Don't clutter your code with useless inline comments (it is a code smell [Inline Comments in the Code is a Smell, but Document the Why], unless it is the why).

- Follow normal programming practices for spacing, naming, etc.: Be reasonable and consistent.
- Be sure to avoid redundant code.

For classes, in general:

- Remember the rule of 3: Explicitly define the destructor, copy operator=, and copy constructor if needed.
- Reuse the code better by calling base class functions and constructors from derived classes.
- Do not make a member variable public or protected, unless it makes sense to do so.

For *Person.h:*

- classType enumeration with the following fields: PER, EMP, STU, FAC, TA to differentiate between the objects of different classes
- Class Person is the base class and has the following data member variables
    1. string name
    2. int ID
    3. classType type
    4. vector <int> * courseId
- Class Person has the following member functions
    1. Person parametrized constructor with the following values as default arguments:
        - Name : Unknown person
        - ID : -1
        - courseId : nullptr
        - type : PER
    2. void displayDetails() as a pure virtual function to display the respective class member variables (displays name, ID, type)
    3. vector <int>* getCourseId() as a getter function to return the courseId member variable
    4. classType getClassType() as a getter function to return the type member variable
    5. string getName() as a getter function to return the name member variable

For *Employee.h:*

- Class Employee, inherits from class Person and has the following data member variables
    1. int officeNum
- Class Employee has the following member functions
    1. Person parametrized constructor with the following values as default arguments:
        - Name : Unknown employee
        - ID : -1
        - courseId : nullptr
        - type : EMP
        - officeNum : 0

2. void displayDetails() as a pure virtual function to display the respective class member variables (displays name, ID, type, officeNum)

For *Student.h:*

- Class Student, inherits from class Person and has the following data member variables
    1. vector <float>* grade
    2. float average
- Class Employee has the following member functions
    1. Person parametrized constructor with the following values as default arguments:
        - Name : Unknown student
        - ID : -1
        - courseId : nullptr
        - type : STU
        - grade : nullptr
    2. void displayDetails() to display the respective class member variables (displays name, ID, type, courses, grades, average)
    3. void calcAverage() as a private function to calculate the average of the grades

For *Faculty.h:*

- Class Faculty, inherits from class Employee and has the following data member variables
    1. int publications
- Class Faculty has the following member functions
    1. Person parametrized constructor with the following values as default arguments:
        - Name : Unknown faculty
        - ID : -1
        - courseId : nullptr
        - type : FAC
        - officeNum : 0
        - publications : 0
    2. void displayDetails() to display the respective class member variables (displays name, ID, type, courses, officeNum, publications)

For *TeachingAssistant.h:*

- Class TeachingAssistant, inherits from class Employee and Student, and has the following data member variables
    1. int officeHours
    2. int TAcourse
- Class TeachingAssistant has the following member functions
    1. Person parametrized constructor with the following values as default arguments:
        - Name : Unknown TA

- ID : -1
- courseId : nullptr
- type : TA
- grade : nullptr
- officeNum : 0
- officeHours : 0
- TAcourse : 0

2. void displayDetails() to display the respective class member variables (displays name, ID, type, courses and grade, average, officeNum, officeHours, TAcourse)

FUNCTIONS TO IMPLEMENT in *DisplayUtility.cpp:*

You have to implement the following two functions to display either all persons or only students who are associated with a given courseID.

*void **displayAllStudentNames** (vector <Person *> v, int courseID)*

This function takes the reference to a vector of Person * type and iterates over it. It displays only the name of the students who have taken the courseID (argument).

*void **displayAllPersonNames** (vector <Person *> v, int courseID)*

This function takes the reference to a vector of Person * type and iterates over it. It displays the name of the students, TA and faculty associated with the courseID (argument).

For *DisplayUtility.h:*

Prototype the functions that are implemented in *DisplayUtility.cpp.*

- *void **displayAllStudentNames** (vector <Person *> v, int courseID)*
- *void **displayAllPersonNames** (vector <Person *> v, int courseID)*

OUTPUT format for display functions, in general:

Have a new line after printing every member variable, with the exception being courses and grades for Student type (course:grade). Take a look at the A4_output_format.txt to check the display output format for objects of different data types.

demo.cpp

You must create a program (place it in demo.cpp) that tests all the functions you implemented in DisplayUtility.cpp and various member functions defined in the classes. Your demo.cpp should have:

- A main function.

- There must be at least 1 call to each of the functions described above.
- This file won't be used as a part of auto grading and hence no fixed output is required.

Makefile

Your submission must include a *Makefile*. The minimal requirements are:

- Compile using the demo executable with the *make* command.
- Use appropriate variables for the compiler and compiler options.
- Have a rule to build the object file, with appropriate dependencies, for each module (header and source pairs).
- It must contain a clean rule.
- The demo executable must be named something meaningful (not *a.out*).

README.md

Your submission must include a README.md file (see https://guides.github.com/features/mastering-markdown/). The file should give a brief description of the program, the organization of the code, how to compile the code, and how to run the program. Using markdown, make a section for each item required in your README.md.

Submission

You will upload all the required files to Gradescope. Reminder to complete the Assignment 4 Canvas quiz. There are no limits on the number of submissions. See the syllabus for the details about late submissions.

Grading

For this assignment, half of the marks will come from the auto grader. For this assignment, none of the test details have been hidden, but there may be hidden tests in future assignments.

The other half of the marks will come from human grading of the submission. Your files will be checked for style and to ensure that they meet the requirements described above. In addition, all submitted files should contain a header (comments or otherwise) that, at a minimum, give a brief description of the file, your name and wisc id.

HAPPY CODING!