**Regis University CC&IS**

**CS362 Data Structures**

**Programming Assignment #5** (covers classes and objects)

# Due: midnight Sunday of week 6

A college wants to you to write a test program for tracking their course sections.

Classroom capacities on campus range from a minimum of 1 student to a maximum of 30 students.

Data stored about each course includes: the course number, the course title, the capacity (number of seats) for the course, the number of students enrolled in the course, and a list of student ids for students enrolled in the course.

## *Class Details*

The following **data members** must be declared as part of the new class, ***course***.

|  |  |  |
| --- | --- | --- |
| Data Field |  | Data Type |
| course number |  | string (2 uppercased letters followed by 3 digits) |
| course title |  | string |
| capacity |  | integer (between 1 and 30) |

number students enrolled integer (between 0 and capacity)

student id list array of 5-digit student ids (size = maximum course capacity)

The following **member functions** must be declared as part of the new class, ***course***.

Two constructor functions (one without parameters, and one with parameters)

Constructor without parameters will set the initial values to:

course number to “ “

course title to “ “

capacity to minimum capacity of a classroom

number students enrolled to 0

Constructor with parameters will provide the following parameters and default values:

course number (no default – must be supplied) course title (no default – must be supplied) capacity = (default = maximum capacity)

The number students enrolled will not be a parameter, and this constructor will set it to 0.

Mutator functions (assign values to data members) setCourseNum setCourseTitle

setCapacity

Accessor functions (access values from data members) getCourseNum getCourseTitle getCapacity getNumEnrolled

|  |  |
| --- | --- |
| Facilitator functions: |  |
| printCourse | (display course data) |
| printStudentIds | (display all student ids on student id list) |
| addOneStudent | (add one student to course) |
| dropOneStudent | (drop one student from course) |

Operator overloading functions:

operator++ (increment capacity by one)

operator<< (displays object data)

NOTE: Data members within the course type class may only be accessed via the above member functions.

You will also need to define several functions ***outside of the class***, for reading and error checking user input and managing the courses.

## *Program Details*

When the program begins, it will create three course type objects as follows:

course1

Create using the constructor with parameters to set the course number and course title to CS361, Control Structures (will use the ***default*** capacity).

course2

Create using the constructor with parameters to set the course number and course title to CS362, Data Structures, and set the capacity to 10. course3

Create using the constructor ***without*** parameters.

***After*** course3 is created, prompt the user for the course number and course title, and assign them to the object data fields.

When all three courses objects have been created, the program will allow the user to manipulate the objects.

The program will access the course number and course name in each of the course objects to present a **Choose Course** menu to the user for choosing a course to manage.

The **Choose Course** menu must be implemented so that the user enters either a course number or E to exit.

Example:

**Choose a course to manage:**

**CS361 – Control Structures**

**CS362 – Data Structures**

**MT415 – Linear Algebra Enter the course number (e.g. CS200) or E to exit:**

After the user chooses a course to manage, the program will display a **Course Management** menu of action choices, as detailed on the next page.

The actions will be applied to the course that was chosen to be managed.

**Course Management** menu choices:

## P - Print course data (to the screen)

Use the overloaded **<<** operator to display the course data as follows:

**CS361 - Control Structures:**

**1 seats taken, out of 15 total seats**

## N – Modify course Number

Call a function to read, error check, and return a valid course number.

Course numbers must be 2 letters followed by 3 digits. If a course number entered by the user is invalid, issue an error message ***stating why it is invalid***. Loop and re-prompt for the course number, until it is valid.

The function should uppercase the letters, if user enters them in lowercase, before returning the value.

Then use **setCourseNum** to store the new value, and then ***confirm*** the change. Example:

**New course number is CS430**

## T - Modify course Title

Call a function to read, error check, format, and return a valid course title (i.e. course name).

Course titles can contain only letters and spaces. If a title entered by the user is invalid, issue an error message ***stating why it is invalid***. Loop and re-prompt for the title, until it is valid.

NOTE: In order to read the title, you will need to use **getline** (covered in text Chapter 3) because the extraction operator (>>) will only read strings up to the first whitespace.

After error checking the title characters, format the title. The first letter and each letter following a space will be uppercased, and the rest will be lowercased. Return the valid formatted title.

Then use **setCourseTitle** to store the new value, and ***confirm*** the change. Example:

**New course title is Operating Systems**

### C - Modify course Capacity

Call a function to read, error check, and return a valid course capacity. The function should insure that the new capacity is within the minimum and maximum allowable capacity.

Using the returned value, check to insure the new capacity is not less than the current student enrollment (use **getNumEnrolled** to check the current enrollment).

If the new value is below the current number of students enrolled, issue a message saying that students must be dropped from the course first. Example:

**Error - cannot change capacity to 20 unless 2 students are dropped, because 22 students are already registered.**

Otherwise, use **setCapacity** to store the new value and ***confirm*** the change.

Example:

**Capacity is now 11**

### I - Increment course Capacity

Use the overloaded ++ operator to increment the course capacity by one, as long as it does not exceed the maximum capacity allowed, and then confirm change. Example:

**Capacity is now 12**

## S – Student Management

Display a **Student Management sub-menu** (display student ids, add one student, drop one student, and return back to the **Course Management** menu).

The options should be implemented as follows:

## P – Print Student IDs (to the screen)

Use **printStudentIds** to display the ids of students enrolled in the course, one student per line.

Example:

**2 students enrolled in CS361**

**77777**

**87228**

## A - Add one student

If course is full, issue an error message.

Example:  **Course is full. Cannot add any more students.**

Otherwise, use **addOneStudent** to add one student to the course, as follows:

Call an input function to read, error check, and return a valid student ID (must be a 5digit integer, no leading zeros allowed).

Then store the student id number in the list, increment the student enrollment, and ***confirm*** the change. Example:

**Added student id 34567**

### D - Drop one student

If the course is empty, simply display a message stating there are no students enrolled.

Example:  **Cannot drop a student -- no students are enrolled.**

Otherwise, use **dropOneStudent** to drop one student from the course, as follows:

1. Display the student ids
2. Ask the user for the **student id number** of the student to drop
3. Find the **student id number** in the list
4. If the **student id number** is found, delete the id from the list and decrement the enrollment.

***Reminder:*** Deleting from unordered arrays was covered in online content sections 3.4.3 and 3.4.4.

1. Display a message confirming which **student id number** was dropped, or stating that the **student id number** could not be found.

## B – Back to Course Management menu

Exit the **Student Management** menu and return to the **Course Management** menu.

Loop around the **Student Management** menu options, until the user chooses B.

[END of Student Management menu options]

## B - Back to Choose Course menu

Exit the **Course Management** menu and return to the **Choose Course** menu.

Loop around the **Course Management** menu options, until the user chooses B.

The program will loop around the **Choose Course** menu, until the user chooses **Exit**.

NOTE: For each menu, when a menu choice is entered, the program will error check the user has entered a valid choice, and will issue an error message if the choice is not valid.

**Before exiting the program**, use the overloaded << operator again to display the course data for all three courses.

***Program Notes****:*

1. Your program must conform to all **CS362 Coding Standards** specified in Content section 1.7. - The program should include a **file header** at the top of each program file
   * **Each** function should include a **function header**.
2. This program should be of modular design (**minimum** of **FIFTEEN** functions/methods).
   * The breakdown of the code must be logical, not arbitrary!
   * The **main** function should do little more than call other functions.
   * The other functions/methods should each perform ONE well-defined task.
3. Your program should be thoroughly tested, and test data files should be submitted.

# Submission

Submit:

* your **program code** file(s)
* a description of how you tested your code

to the ***Wk 6 – Assn 5*** folder in the ***Dropbox*** area of the online course, by **midnight, Sunday**.

WARNING:

Your program will **NOT** be accepted if it:

* + does not compile
  + uses **any** global variables
  + is not adequately modular (i.e. contains fewer than the required number of functions)

Before attaching your files to the assignment submission, please NAME them as follows:

LastnameAssn#.cpp

LastnameTestDescription.docx (or .doc, or .txt)

For example: SmithAssn5.cpp

SmithTestDescription.docx

## Grading

The rubric that will be used to grade your program is included on the next pages.

**CS 362 Data Structures**

## Programming Assignment #5 Rubric

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rating Rating Category** | **Exemplary** | **Partially Proficient** | **Basic (needs work)** | **Not Demonstrated** |
| **Documentation 8%** | Documentation includes complete and accurate **file** and **function** headers, as detailed in the CS362 coding standards, along with additional in-line comments at appropriate locations. | Documentation includes all required headers, but lacks some clarity or details, OR violated minor details of CS362 coding standards for comments, OR code is over commented. | Documentation is incomplete and/or incorrect and/or formatted incorrectly and/or violated multiple rules within the CS362 coding standards for comments. | Only a few (or no) comments in program. |
| **Data Storage 8%** | Constants used for all fixed values. Classes members and objects declared as specified. All constants and variables defined using correct data types, within correct program scope, following CS362 coding standards. | Minor errors in data declaration scope, data typing or assignment, OR missing a couple of necessary constants, OR some identifier names not descriptive enough, OR violated minor details of CS362 coding standards for data. | Many errors in data declaration scope, data typing or assignment, AND/OR many missing constants, AND/OR violated multiple rules within the CS362 coding standards for data. | Major errors in most or all of the constant/variable definitions or assignments AND/OR did not follow most of the CS362 coding standards for data. |
| **Object Creation**  **/ Constructors**  **8%** | Both constructors defined as specified. Three objects instantiated as specified, and filled with required data. | Both constructors defined, but minor issues in definitions and/or three objects instantiated, but not as specified. | Multiple problems with constructor definitions and/or object creation. | No constructors defined. |
| **Accessors /**  **Mutators**  **8%** | All accessor and mutator functions defined as specified. | All accessor and mutator functions defined, but minor issues in definitions. | Multiple problems with accessor and mutator functions OR not all required functions defined. | No accessor or mutator functions defined. |
| **Operator**  **Overloading**  **8%** | Both the << and ++ operators were successfully overloaded for use with the class as specified. | Both the << and ++ operators were overloaded, but minor issues in definitions. | Multiple problems with operator overloading functions OR only one operator was overloaded. | No operator overloading functions defined. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Menus 8%** | Displays menus options for choosing a courses, course management, and student management as specified. Prompts for user choice. Error checks choice, loops correctly, and exits correctly via menu choice. | Minor issues with menu display, prompting, choice error checking, or looping. | Many minor issues or one or more major issue(s) with menu display, prompting, choice error checking or looping. | Many major issue(s) with menu display, prompting, choice error checking or looping, which prevent correct usage. |
| **Modifying**  **Functions / User**  **Input Functions**  **9%** | Prompts for all other (nonmenu) user input are clear. Error checks all user input, issues descriptive error messages, and re-prompts when necessary. Returns valid, correctly formatted values. Modifies data correctly. | Minor issues with display, prompts, user input, error checking user input, error messages, formatting, or data modification. | Many errors with display, prompting, user input, error checking user input, error messages, formatting, or data modification. | Program was not able to correctly read any user input or modify any data fields. |
| **Facilitator:**  **Adding One**  **Student**  **7%** | Checks if class is full before adding. Adds student data and increments count. | Minor issues with checking if full or adding student. | Many errors with checking if full and/or adding student OR not implemented as a member function. | Student could not be added. |
| **Facilitator:**  **Dropping One**  **Student**  **8%** | Displays roster. Unless empty, prompts for id and verifies id entered is valid format. IF id found, deletes student from roster, decrements count, and displays confirmation message. Error message given if id is not in roster or course is empty. | Choices not displayed, OR format not validated, OR confirmation message not given, OR error message not given, OR other minor error. | Multiple problems with dropping a student OR not implemented as a member function. | Students could not be dropped. |
| **Facilitator:**  **Display**  **Functions**  **8%** | Functions to display course data and to display student ids implemented as specified. | Minor issues with display functions. | Major issues with display functions OR not implemented as member functions. | Program does not produce any display from display functions. |
| **Modularity (functional**  **breakdown)**  **7%** | Program is efficient, modular (at least 15 functions/methods), and logically organized. Each module performs ONE welldefined task and is defined and called correctly. main function does little more than call other functions. | Program is mostly efficient and modular in design, with most of the code logically organized. Contains minimum required number of functions, but may have an unnecessary function, a function that performs too many tasks, and/or a function that is incorrectly defined or called. | Multiple modules are unnecessary or perform too many tasks, OR parts of the program are not efficient and/or logically organized, OR one required function is missing, OR multiple problems with module definitions/calls. | Program is somewhat modular, but the module breakdown is not at all efficient and/or logically organized. |
| **C++ Constructs /**  **Readability /**  **Miscellaneous**  **8%** | Demonstrates understanding of control, data, and file structures, and OOP classes and objects. All parameters passed correctly. Code is exceptionally well organized, easy to follow, and adheres to all CS362 coding standards for code usage (e.g. no use of **break/return** to exit a loop) | A parameter or two was passed incorrectly (value vs.  reference), OR unnecessary parameter(s) was passed, OR minor improper control or data structure or object usage, AND/OR some occasional  spacing, indentation, and/or other minor organizational issues. | One or more major error(s), to include lots of incorrect parameter definition and usage, improper control or data structure or object usage, AND/OR substantial spacing, indentation, and/or other organizational issues. | Demonstrates minimal understanding of object/control/data structures and/or proper parameter passing AND/OR major coding standard violations throughout, AND/OR other major organization, construct, or readability issues. |
| **Testing**  **5%** | Test description and data files allow for complete validation of the program code. | Missing a couple of necessary tests. | Missing many necessary tests. | Only used test data from assignment requirements, or no tests submitted. |

NOTE: Deductions for late submissions will be applied (-2% per day late)

But programs will **not be accepted** if they are submitted **more than 5 days** late (0).