Part C - Encapsulation   
  
**Constructors**

Workshop 4

In this workshop, you are to code a class that represents an enrollment and processes transactions on that enrollment.

**LEARNING OUTCOMES**

Upon successful completion of this workshop, you will have demonstrated the abilities to

* code a class for simple objects
* define a constructor that sets an object to a safe empty state
* overload a constructor to receive information from a client

**SUBMISSION POLICY**

The “in-lab” section is to be completed **during your assigned lab section**. It is to be completed and submitted by the end of the workshop. If you do not attend the workshop, you can submit the “in-lab” section along with your “at-home” section (a 20% late deduction will be assessed). The “at-home” portion of the lab is **due the day before you next scheduled workshop**

**Enrollment Class**

Download or clone workshop 4 from <https://github.com/Seneca-OOP244/Workshop4.git>

Open Workshop4/in\_lab directory and complete the code in **Enrollment.h** and **Enrollment.cpp.**

Design and code a class named **Enrollment** that holds information about a student’s enrollment in course sections at a college. Place your class definition in a header file named **Enrollment.h** and your function definitions in an implementation file named **Enrollment.cpp**. Include in your coding all of the statements necessary for your code to compile under a standard C++ compiler, and within the **sict** namespace.

The class **Enrollment** has the following six data members:

\_name: A C-style string holding the name of the course in which the student is enrolled. You may assume that the string is no longer than 30 characters, excluding the null terminator.

\_code: A C-style string holding the alphanumeric course code for the course in which the student is enrolled. You may assume that the string is no longer than 10 characters, excluding the null terminator.

* \_year: An integer indicating the year in which the course is being offered **(>= 2015**).
* \_semester: An integer indicating the semester in which the course is being offered **(> 0 and < 4**).
* \_slot: An integer indicating the time-slot this course occupies **(> 0 and < 6**).
* \_enrolled A boolean value to indicate whether the student is currently enrolled (a value of **false** indicates the student has been withdrawn from the course).

Enrollment has the following member functions (methods) that are already implemented in **Enrollment.cpp.**

* **void display(bool nameOnly=false) const** - a query that displays the course name and code as well as the year, term and time-slot when nameOnly is false. If nameOnly is true, it will display the name only. This function displays Invalid enrollment! if the object is in a safe empty state (when it is invalid);

*Enrollment is in a safe empty state when the char strings are empty and other values are set to zero or false.*

* **bool valid()const** - returns **true** when the object is NOT in a safe empty state.
* **void setEmpty()**- sets the object to a safe empty state.

**IN-LAB section (80%)**

Implement the following member functions: (make sure you reuse your code and DO NOT re-implement the logics alrady implemented)

* **void set(const char\* name, const char\* code, int year, int time, int semester, bool enrolled = false) -** a setter function that stores valid data in an **Enrollment** object. The first parameter receives the address of a C-style string containing the course name, the second parameter receives the address of a C-style string containing the course code, followed by 3 integers receiving year, term, time-slot and enrolled respectively. If the last argument is not provided, \_enrolled is set to false.   
  If any of the data is invalid, this function will set the object to a safe empty state.
* **bool hasConflict(const Enrollment &other) const -** a query that accepts an unmodifiable reference to the other **Enrollment** object and returns whether two course sections have a conflict.

*Two sections have a conflict if they have the same year, semester, and time-slot. If any of the objects are invalid this function returns false.*

Include two constructors in your definition: a no-argument constructor that sets the object to a safe empty state and a five-argument constructor that receives the address of a C-style string containing the course name, a second address of a C-style string containing the course code, 3 integers indicating year, semester, and time-slot respectively. The five-argument constructor should set the value indicating that the student is enrolled to **false**.

Note that you must check the validity of users’ input. If null pointer arguments are provided, or validation fails, the object is set to a ***safe empty state****.*

**CLIENT MODULE**

Here is a sample of implementation file for the **w4\_in\_lab.cpp** main module that you should use to test your implementation:

// OOP244 Workshop 4: Constructors

// File: w4\_in\_lab.cpp

// Version: 1.0

// Date: 2015/10/4

// Author: Fardad Soleimanloo

// Description:

// This file tests in-lab section of your workshp

/////////////////////////////////////////////

#include <iostream>

#include "Enrollment.h"

using namespace std;

using namespace sict;

void displayEnrollments(const Enrollment& e1, const Enrollment& e2);

int main(){

// constructors

Enrollment e1("College English", "EAC150", 2015, 1, 5);

Enrollment e2, e3;

Enrollment BadE[9] = {

Enrollment("Intro to Programming in C", "IPC144", 2015, 1, 5),

Enrollment((char\*)0, "IPC144", 2015, 1, 5),

Enrollment("Intro to Programming in C", (char\*)0, 2015, 1, 5),

Enrollment("", "IPC144", 2015, 1, 5),

Enrollment("Intro to Programming in C", "", 2015, 1, 5),

Enrollment("Intro to Programming in C", "IPC144", 2014, 1, 5),

Enrollment("Intro to Programming in C", "IPC144", 2015, 0, 5),

Enrollment("Intro to Programming in C", "IPC144", 2015, 3, 5),

Enrollment("Intro to Programming in C", "IPC144", 2015, 1, 0)

};

cout << "Testing Enrollment objects" << endl << endl;

// testing two invalid Enrollments

displayEnrollments(e3, e2);

// testing valid and invalid Enrollments

displayEnrollments(e1, e2);

// setting the second one to OOP244

e2.set("Object Oriented Programming", "OOP244", 2015, 1, 5);

displayEnrollments(e1, e2);

// setting the enrollment to true;

e2.set("Object Oriented Programming", "OOP244", 2015, 1, 5, true);

displayEnrollments(e1, e2);

// removing conflict;

e2.set("Object Oriented Programming", "OOP244", 2015, 1, 6);

displayEnrollments(e1, e2);

// making sure all the conditions are met for an invalid Enrollment.

cout << "Only index 0 should be valid:" << endl;

for (int i = 0; i < 9; i++){

cout <<"index: " << i << " - "<< (BadE[i].valid() ? "V" : "Not v") << "alid." << endl;

}

return 0;

}

void displayEnrollments(const Enrollment& e1, const Enrollment& e2){

cout << "----------------------------------------------" << endl;

e1.display();

cout << (e1.isEnrolled() ? "E" : "Not e") << "nrolled" << endl;

e2.display();

cout << (e2.isEnrolled() ? "E" : "Not e") << "nrolled" << endl << endl;

cout << "There is " << (e1.hasConflict(e2)? "" : "not ") << "a conflict!" << endl;

}

Output Example:  
(Your output should exactly match the following)

Testing Enrollment objects

----------------------------------------------

Invalid enrollment!

Not enrolled

Invalid enrollment!

There is not a conflict!

----------------------------------------------

College English

EAC150, Year: 2015 semester: 1 Slot: 5

Status: Not enrolled.

Not enrolled

Invalid enrollment!

There is not a conflict!

----------------------------------------------

College English

EAC150, Year: 2015 semester: 1 Slot: 5

Status: Not enrolled.

Not enrolled

Object Oriented Programming

OOP244, Year: 2015 semester: 1 Slot: 5

Status: Not enrolled.

There is a conflict!

----------------------------------------------

College English

EAC150, Year: 2015 semester: 1 Slot: 5

Status: Not enrolled.

Not enrolled

Object Oriented Programming

OOP244, Year: 2015 semester: 1 Slot: 5

Status: Enrolled.

There is a conflict!

----------------------------------------------

College English

EAC150, Year: 2015 semester: 1 Slot: 5

Status: Not enrolled.

Not enrolled

Invalid enrollment!

There is not a conflict!

Only index 0 should be valid:

index: 0 - Valid.

index: 1 - Not valid.

index: 2 - Not valid.

index: 3 - Not valid.

index: 4 - Not valid.

index: 5 - Not valid.

index: 6 - Not valid.

index: 7 - Not valid.

index: 8 - Not valid.

**In-Lab SUBMISSION**

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload your **Enrollment.h** and **Enrollment.cpp** and **w3\_in\_lab.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account:

**Sections SAA and SBB:  
~fardad.soleimanloo/submit w4\_in\_lab <ENTER>   
Section SCC and SDD:  
~ronald.burton/submit\_w4\_in\_lab <ENTER>**

and follow the instructions.

**AT-HOME Section: to Enrol**

Add the following member functions to the Enrollment class:

* **void withdraw() -** a modifier that withdraws the student from the course, and displays an appropriate message.
* **int enrol(const Enrollment\* enrollments, int size);** - this modifier sets the \_enrolled flag to true and returns 0, unless there is a conflict between this class and one the elements of enrollments array. In later case, it will return the sequence number of the element in conflict (i.e. returns index+1).

At this stage you can test your program with w4\_at\_home.cpp to make sure your enroll and withdraw functions work correctly.

**Dynamic array of Enrollment**

Change Enrollment slots[3]; array to an Enrollment pointer called slots.

Modify w4\_at\_home.cpp and remove the limit of 3 course sections and ask the user to enter the number of course sections before starting the process.

Then allocated the user specified number of course sections dynamically pointed by slots pointer.

Change the “3” course sections limits throughout the code to the specified number by the user.

At the end complete your dynamic memory allocation conversion by deallocating the memory allocated in slots.

// OOP244 Workshop 4: Constructors

// File: w4\_at\_home.cpp

// Version: 1.0

// Date: 2015/10/4

// Author: Fardad Soleimanloo

// Description:

// This file tests at-home section of your workshp

/////////////////////////////////////////////

#include <iostream>

#include "Enrollment.h"

using namespace std;

using namespace sict;

void displayEnrollments(const Enrollment& e1, const Enrollment& e2);

int main(){

// change this slot array to an Enrollment pointer dynamic memory allocation.

Enrollment slots[3];

Enrollment current;

char name[31];

char code[11];

int year;

int semester;

int slot;

int i=0;

// uncomment the following comment to dynamically allocate memory for slots.

// cout << "Please enter the number of course sections: ";

cout << "Please enter the following " << 3 << " course sections for enrollment:" << endl;

while (i < 3){

cout << "Course section " << (i + 1) << ":" << endl

<< "Subject Name: ";

cin.getline(name, 30, '\n');

cout << "Subject Code: ";

cin.getline(code, 10, '\n');

cout << "Year: ";

cin >> year;

cout << "Semester: ";

cin >> semester;

cout << "Slot: ";

cin >> slot;

cin.ignore(1000, '\n');

current.set(name, code, year, semester, slot);

if (current.valid()){

int c;

if ((c = current.enrol(slots, 3)) == 0){

char res;

slots[i] = current;

cout << "Enrolled!" << endl;

i++;

cout << "Continue? (y/n)";

res = cin.get();

if (res != 'y' && res != 'Y'){

i = 3;

}

cin.ignore(1000, '\n');

}

else{

cout << "There is a conflict with the following, please change enrollment time: " << endl;

slots[c - 1].display();

}

}

else{

cout << "Invalid information entered, please redo: " << endl;

}

}

for (i = 0; i < 3; i++){

cout << "------------------------------" << endl;

slots[i].display();

}

// testing withdraw();

slots[0].withdraw();

slots[0].display();

// deallocate the slots memory here;

return 0;

}

Sample output:

Please enter the number of course sections: 5

Please enter the following 5 course sections for enrollment:

Course section 1:

Subject Name: aaa aaa

Subject Code: aa

Year: 2015

Semester: 3

Slot: 3

Enrolled!

Continue? (y/n)y

Course section 2:

Subject Name: bbb bbb

Subject Code: bb

Year: 2015

Semester: 2

Slot: 2

Enrolled!

Continue? (y/n)y

Course section 3:

Subject Name: ccc ccc

Subject Code: cc

Year: 2015

Semester: 2

Slot: 2

There is a conflict with the following, please change enrollment time:

bbb bbb

bb, Year: 2015 semester: 2 Slot: 2

Status: Enrolled.

Course section 3:

Subject Name: ccc ccc

Subject Code: cc

Year: 2015

Semester: 1

Slot: 1

Enrolled!

Continue? (y/n)n

------------------------------

aaa aaa

aa, Year: 2015 semester: 3 Slot: 3

Status: Enrolled.

------------------------------

bbb bbb

bb, Year: 2015 semester: 2 Slot: 2

Status: Enrolled.

------------------------------

ccc ccc

cc, Year: 2015 semester: 1 Slot: 1

Status: Enrolled.

------------------------------

Invalid enrollment!

------------------------------

Invalid enrollment!

aaa aaa

aa, Year: 2015 semester: 3 Slot: 3

Status: Not enrolled.

**AT-HOME SUBMISSION: REFLECTION (20%)**

Please provide brief answers to the following questions in a text file named **reflect.txt.**

1. What is meant by a **safe empty state**?
   1. What is the safe empty state of the **Enrollment** class? Please describe in English words, not code.
   2. Is this a realistic empty state? Explain why or why not.
2. Describe a case where having multiple **Enrollment** constructors would simplify the code of clients that use it.

**SUBMISSION**

To test and demonstrate execution of your program use the same data as the sample output above.

If not on matrix already, upload your **Enrollment.h**, **Enrollment.cpp, reflect.txt**  and **w4\_at\_home.cpp** to your matrix account. Compile and run your code and make sure everything works properly.

Then run the following script from your account:

**Sections SAA and SBB:  
~fardad.soleimanloo/submit w4\_at\_home <ENTER>   
Section SCC and SDD:  
~ronald.burton/submit\_w4\_at\_home <ENTER>**

and follow the instructions.