Summer 2016
OOP244 Final Project (v. 1.5)

Seneca Course Management Tool (SCM)

You are asked to develop a C++ application that manages Seneca courses taken by the students at the School of Information and Communications Technology (ICT). In this proof of concept application, the ICT students can take two types of courses, namely ICT-related courses (e.g. OOP244, BTP200) and general education courses (e.g. EAC150, BTC140). The application will provide a console-based menu system to manage these courses.

You will develop this application *incrementally* as new user requirements emerge during the project development process. As you work on the project, you will build up your conceptual and practical knowledge of object-oriented programming in C++. You will learn to apply three principles of object-oriented programming (i.e. encapsulation, inheritance and polymorphism). In particular, you will create five classes as new user requirements emerge. The application class will be the point of integrating four classes.

Marking Scheme

This project contains 4 milestones, each worth 5% of your final mark (20% total of final mark). All milestones must be submitted, compile with no errors or warnings, and be working for the project to be considered complete. Each milestone will be submitted individually through the same submission system used for the workshops. Submission instructions are found at the end of this document.

Late submissions are allowed. There will be a late penalty of 10% per day late, weekends included, up to a maximum of 50%. Submissions will not be accepted after the 15th of August.

A. THE USER INTERFACE

```
Seneca Course Management Tool
1- List courses.
2- Display the details of a course.
3- Add a course.
4- Change the study load of a course.
5- Load courses from a file.
6- Save courses to a file.
0- Exit program.
> _
```

B. CLASSES TO BE DEVELOPED

You will develop the following five classes for this application:

Streamable

This interface (a class with "only" pure virtual functions) enforces that the classes inherited from it are to be "*streamable*". In the first half of the course we define streams to be sequence of characters. IO streams allow program data to be exchanged with peripherals such as computer consoles or keyboards. Any class derived from "Streamable" can read from or write to streams.

Using this class, a list of courses can be saved to and retrieved from a file. Individual course details can be displayed on screen or read from keyboard.

Course

A class that encapsulates information about courses.
As the project develops, this class will acquire *streamable* characteristics as new user requirements emerge.

ICTCourse

A class that encapsulates information about ICT-related courses (e.g. OOP244, BTP200). It is derived from the **Course** class.

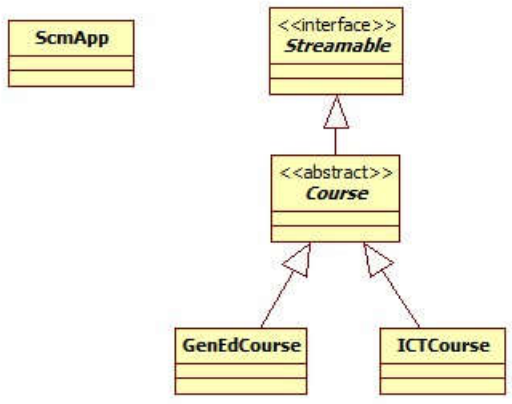
GenEdCourse

A class that encapsulates information about general education courses (e.g. EAC150, BTC140). It is derived the **Course** class.

ScmApp

An application class that provides a console-based menu system to manage The courses. It will use the other four classes in order to provide the system's functionality. You will develop this application *incrementally* as new user requirements emerge.

C. CLASS DIAGRAM



D. PROJECT DEVELOPMENT PROCESS

You have **four weeks** to complete the project. The project is divided into 4 milestones and thus four deliverables. Each milestone has a clear learning focus and specific learning tasks. The approximate schedule for the deliverables is as follows:

- **Final Project Due date: Sunday August 7, 2016 by 11:59:00pm.**
  - MS1: The Course and ScmApp classes. Due: July 22, 2016 by 11:59:00pm
  - MS2: The ICTCourse, GenEdCourse and ScmApp classes. Due: July 29, 2016 by 11:59:00pm
  - MS3: The Streamable , Course, ITCourse and GenEdCourse classes. Due: August 5, 2016 by 11:59:00pm
  - MS4: Completion of the ScmApp class. Due: **August 12, 2016 by 11:59:00pm**

E. FILE STRUCTURE OF THE PROJECT

Each class will have its own header file and .cpp file. The names of these files should be the same as the class names.
Example: Class **Course** has two files: **Course.h** and **Course.cpp**

In addition to header files for each class, create a header file called **"general.h"** that will hold the general defined values for the project, such as:

```
MAX_COURSECODE_LEN (6)  The maximum length of a course code.

DISPLAY_LINES (10)      The maximum number of lines used to display course details
                        before each pause.

MAX_NO_RECS (2000)      The maximum number of records in the data file.
```

This header file should get included where these values are used.

Notes

1. All the code developed for this application should be in the ***sict*** namespace.
2. All the code must be properly documented and indented. Ask your instructor for details.

MILESTONE 1: TWO CLASSES (Course and ScmApp).

Learning Focus: Dynamic Memory Allocation; Operator Overloading.

Learning Tasks:

1. Create the Course and ScmApp classes according to user requirements.

2. Test the Course and ScmApp classes.

A. User Requirements

Create a class called **Course**. The class **Course** is responsible for encapsulating information about general courses at Seneca College.

At Milestone 1, this class is a concrete class. It will become an abstract base class at Milestone 3. The class is implemented under the ***sict*** namespace.

Code the **Course** class in Course.h and Course.cpp.

1. The Course Class Specs.

Private Member Variables.

courseCode_: Character array, MAX_COURSECODE_LEN + 1 characters long.
This character array holds the course code as a string.

courseTitle_: Character pointer
This character pointer points to a dynamic string that holds the title of the course.

credits_: Integer
It holds the number of credits of the course.

studyLoad_: Integer
It hold the amount of study load, defined by the number of assignments.

Public Member Functions and Constructors

No Argument Constructor

This constructor sets the **Course** object to a safe recognizable empty state. All number values are set to zero in this state.

Four Argument Constructor

The Course object is constructed by passing 4 values to the constructor:
the course code, the course title, the number of credits and the amount of study load.
The constructor:

- Copy the course code into the corresponding member variable up to MAX_COURSECODE_LEN characters.

- Allocate enough memory to hold the name as pointed by the pointer *courseTitle_*. Then copy the name into the allocated memory pointed to by the member variable *courseTitle_*.
- Set the rest of the member variables to the corresponding values received by the arguments.

Copy Constructor

See below.

Dynamic Memory Allocation Requirements

Implement the copy constructor and the operator= so the **Course** object is copied from and assigned to another **Course** object safely and without any memory leak. (Also implement a destructor to make sure that the memory allocated by the pointer ***courseTitle_*** is freed when the **Course** object is destroyed

Setters:

Create the following setter functions to set the corresponding member variables:

- **courseCode_**
- **courseTitle_**
- **credits_**
- **studyLoad_**

All the above setters return void.

Getters (Queries):

Create the following constant getter functions to return the values or addresses of the member variables: (these getter functions do not receive any arguments)

- **courseCode,** `returns constant character pointer`
- **courseTitle_,** `returns constant character pointer`
- **credits_,** `returns integer`
- **studyLoad_,** `returns integer`

Also:

- **isEmpty** `returns bool`
  isEmpty returns true if the Course object is in a safe empty state, false otherwise

All the above getters are constant functions, which means that they CANNOT modify the current object.

Overload Member Operators.

Operator== : `receives a constant character pointer and returns a boolean.`
This operator will compare the received constant character pointer to the **course code** of the course, if they are the same, it will return true. Otherwise it will return false.

Operator+= : `receives an integer and returns an integer.`
This operator will change the **study load** of the current object by the received integer value, returning the new **study load** value. Note: A negative integer is used to reduce the study load.

Overload the ostream operator << (a FREE helper function).

After implementing the **Course** class, overload the ostream operator<< as a **free** helper function. Hint: You should implement a public member function called display( ).

Make sure that the prototypes of the functions are in **Course.h**.

2. The ScmApp Class Specs.

The **ScmApp** class provides a console-based menu system to manage the courses. Code the class in **ScmApp.h** and **ScmApp.cpp**.

Private member variables.

`Course* courseList_[MAX_NO_RECS];`

An array of `Course` pointers. The size of this static array is MAX_NO_RECS.

Note: Each element of this array is a `Course` pointer.

Reminder: MAX_NO_RECS is defined in in the header file **general.h**.

`int noOfCourses;`

The number of courses (ICTCourse or GenEdCourse) that are currently pointed by the array ***courseList_***.

The Constructor.

The no-argument ScmApp constructor does the following initialization:
1- Set all the courseList_ elements to nullptr.
2- Set noOfCourses_ to zero.

Private member functions.

The Copy Constructor and Assignment Operator.

Make sure that an **ScmApp** object cannot get copied or assigned to another **ScmApp** object.

`void pause() const;`

It prints: `"Press Enter to continue..."<NEWLINE>` and then waits for the user to hit enter. If user hits any other key, the key in ignored. Only ENTER will terminate this function.

`int menu();`

It displays the menu as follows and waits for the user to select an option.

```
Seneca Course Management Tool
1- List courses.
2- Display the details of a course.
3- Add a course.
4- Change the study load of a course.
0- Exit program
> _
  ^ here is where the cursor stands when menu is printed
```

- If the selection number is valid, the member function *menu( )* will return the selection. Otherwise it will return -1.
- This function makes sure that there are no characters left in the keyboard buffer and wipes it clean before exit.

`void listCourses()const;`

- It displays the details of all courses. First, it displays the following title :

```
 Row | Code | Course Title  | Credits | Study Load| System | Lang. Req.|
-----|------|---------------|---------|-----------|--------|-----------|
```

- Then it iterates through the array courseList_ up to noOfCourses_ and displays the following for each course

- Row number in four spaces right justified
- a Bar character (|) surrounded by two spaces.

Then it displays the current Course in the iteration followed by a newline
- If the Row number reaches to 10, the program will pause.
- At the end of the iteration, it will close the list with the following dotted line:

```
---------------------------------------------------------------------------
```

`int searchACourse(const char* courseCode)const;`

Iterates through the array courseList_ up to noOfProducts_ and checks each array element for the same course code as the incoming argument using the operator== implemented for the Course class. If a match is found, it will return the index of the found Course in the array courseList_. Otherwise it will return -1.

`void changeStudyLoad(const char* courseCode);`

It changes the study load of a course whose course code matches courseCode as the incoming argument. If not found it will display:

```
"Not found!"<NEWLINE>
```

If found, it will ask for an integer for the amount of study load:

```
"Please enter the amount of the study load: "
```

It uses the operator += (overloaded in the Course class) to change the study load of the course. Make sure that the keyboard is flushed after the data entry.

`void addACourse();`

It gets the data values **from the user** to initialize the object. Finally it adds the object address to *the end of the array* courseList_.

Public member functions.

`int run();`

It displays the menu and processes user requests. Depending on the user's selection number, it performs the action as requested and pauses. (Use the pause function.) Then it redisplays the menu until the user selects zero to exit.

```
1- List courses.
      List all the courses.
2- Display the details of a course.
      Ask for a course code using this prompt
      "Please enter the course code: "
      and get it from the console. Then search for
      it. If found, display the course details.
      Otherwise display:
      "Not found!"
3- Add a course.
      It calls the addACourse() function.

4- Change the study load of a course.
      Ask for a course code using this prompt
      "Please enter the course code: "
      and get it from the console. Then call the
      changeStudyLoad() function.
0- Exit program
      The program will terminate printing:
      "Goodbye!!"

In case of an invalid menu selection the program will
print: "===Invalid Selection, try again.===".
Then it will pause before redisplaying the menu.

The function run() returns 0 at the end.
```

B. Testing and Submission Requirements (MILESTONE 1).

1. If not on matrix already, upload **general.h**, **Course.h, ScmApp.h**, **Course.cpp, ScmApp.cpp** and the tester to your matrix account. Compile and run your code and make sure that everything works properly. You will be able to submit your milestone even if the outputs do not match 100% (with a penalty).

   To test your submission, compare your output and to submit your milestone, run the following command on your matrix account. If the outputs match, you will be able to submit your milestone.

   ```
   ~bradly.hoover/submit ms1-tester
   ```

   If your output does not match with the test 100%, run the following command from your matrix account. This will allow you to submit without the outputs matching perfectly. :

   ```
   ~bradly.hoover/submit ms1
   ```

Sample Testing Data

Course Code	Course Title	Credits	Study Load
OOP244	Object-Oriented Programming in C++	1	4
BTP200	The Object-Oriented Paradigm Using C++	1	4
IPC144	Introduction to Programming Using C	1	4
BTP100	Programming Fundamentals Using C	1	4
EAC150	College English	1	4
BTP140	Critical Thinking and Writing	1	4
COM480	The Art of Storytelling	1	3
CUL485	Movies and Moraltiy	1	3
PHL105	Introduction to Philosophy	1	3
PSY141	Social Psychology	1	3
HIS201	World War II	1	3
SOC600	Introduction to Sociology	1	3

THE END OF MILESTONE 1