

Fall 2019

COMP2006 Term Project (v. 1.2)

School Course Management Tool (SCM)

In Teams of 5 students or less, you are asked to develop a C++ application that manages Georgian courses taken by the Computer Studies (CS) students. You will be required to do some of your own research to complete this Project, and leverage some of the OOP Concepts you have previously learned in other OOP Courses like Java. It is strongly suggested that you make a detailed plan for your group to *divide and conquer* all of the requirements! Do NOT UNDERESTIMATE THE AMOUNT OF WORK NEEDED TO COMPLETE THIS PROJECT. WORK AS A TEAM / COMPANY!!!!

In this proof of concept application, the CS students can take two types of courses, namely CS-related courses (e.g. COMP 1002, COMP 2006) and general education courses (e.g. PSYL1001, SOCL1000). 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.

A. THE USER INTERFACE

Georgian 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.

CSCourse

A class that encapsulates information about CS-related courses (e.g. COMP 1002, COMP 2006). It is derived from the **Course** class.

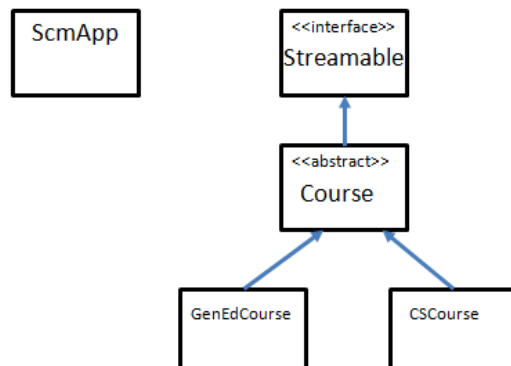
GenEdCourse

A class that encapsulates information about general education courses (e.g. PSYL1001, SOCL1000). 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: Dec 9, 2019 by 11:59:00pm.**
 - MS1: The Course and ScmApp classes. Due: Nov 26, 2019 by 11:59:00pm
 - MS2: The CSCourse, GenEdCourse and ScmApp classes. Due: Dec 3, 2019 by 11:59:00pm
 - MS3: The Streamable , Course, CSCourse and GenEdCourse classes. Due: Dec 9, 2019 by 11:59:00pm
 - MS4: Completion of the ScmApp class. Due: Dec 9, 2019 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 **cs** 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.

A. User Requirements

Create a class called **Course**. The class **Course** is responsible for encapsulating information about general courses at Georgian 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 **cs** 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 received integer value to the **study load** of the course, returning the sum. 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().

It displays the data values of a **Course** object in the following format (separated by Bar “|” characters) :

```
COMP2006 | OOP In C++ | 1 | 3 |
```

course code: left justified, MAX_COURSECODE_LEN characters

course title: left justified, 20 characters wide (truncated if longer than 20 chars)

credits: right justified, 6 characters wide

study load: right justified, 4 characters wide

NO NEW LINE CHARACTER

Note: Each Bar “|” character is surrounded by two blank spaces.
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 the header file **general.h**.

int noOfCourses;

The number of courses (CSCourse 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 is ignored. Only ENTER will terminate this function.

int menu();

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

Georgian 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	Credit	Load	System	Lang
-----	-----	-----	-----	-----	-----	-----

- 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_`.

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 redisplay 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. Submission Requirements (MILESTONE 1).

1. Provide the NAME OF YOUR GROUP (EX. "C++ Commandos") as well as the NAMES of EACH GROUP MEMBER in a Word .doc. Create a File Folder called **MS1-C++ Commandos**. Add your files, `general.h`, `Course.h`, `ScmApp.h`, `Coures.cpp`, `ScmApp.cpp`, including the word.doc, in your **MS1-C++ Commandos** file and .zip.

2. Include **SCREEN SHOTS** of the working code, in the .zip file.

3. Upload all your .zip source files to Blackboard.

4. Sample Testing Data

- COMP 1002 - HTML, CSS, and JS Fundamentals
- COMP 1030 - Programming Fundamentals
- COMP 1035 - Networking Essentials
- COMP 1045 - Internet of Things using Arduino
- MATH 1003 - Math for the Computer Industry
- COMP 1006 - Introduction to Web Programming using PHP
- COMP 1008 - Introduction to Object Oriented Programming using Java
- COMP 1098 - .NET Programming using C#
- COMP 2003 - Relational Database
- ENTR 1002 - Introduction to Entrepreneurship
- COMP 1009 - The Mainframe Environment
- COMP 1011 - Advanced Object Oriented Programming using Java

- COMP 1073 - Client~Side JavaScript
- COMP 2084 - Server~Side Scripting using ASP.NET
- COMP 3002 - Advanced Databases
- COMP 2068 - JavaScript Frameworks

GenEd

- PSYL1001 - Intro to Psychology
- SOCL1000 - Intro to Sociology

THE END OF MILESTONE 1

MILESTONE 2: TWO NEW CLASSES (CSCourse, GenEdCourse).

Learning Focus: Inheritance and Polymorphism.

Learning Tasks

1. Create two classes (CSCourse, GenEdCourse) that are derived from the class Course.
2. Modify the class Course in order to implement the inheritance hierarchy successfully.
3. Modify the application class ScmClass with the following functionality:
 - List courses.
 - Display the details of a course.
 - Add a course. (Note: Add an CS course or a GenEdCourse.)
 - Change the study load of a course.

A. User Requirements.

1. The CSCourse Class Specs.

Implement the **CSCourse** class in **CSCourse.h** and **CSCourse.cpp** as a class derived from the **Course** class.

Private member variables

```
char computerSystem__[6+1];
```

It holds the type of computer system that will be used in an CS course (e.g. matrix, oracle, as400, win and linux).

Constructors

The **CSCourse** class has two constructors. The one-argument constructor that receives a string as an argument. It uses the value of the argument to initialize the member variable

`computerSystem_`. The no-argument constructor sets this member variable to “matrix.”

Public Member Functions

```
const char* getComputersystem() const;
```

It returns a constant pointer to the `computerSystem_` member variable.

```
void setComputerSystem (const char* value);
```

It copies the incoming value string into the member variable `computerSystem_`.
Make sure that the copying does not pass the size of the array `computerSystem_`.

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

After implementing the **CSCourse** class, overload the ostream operator<< as a **free** helper function. Hints: You should implement a public member function called `display()` and reuse the function `display()` in the base class. The function displays the data values of an **CSCourse** object in the following format (separated by the bar “|” character):

```
COMP2006 | OOP in C++ | 1 | 4 | blackboard |
```

course code: as specified in Milestone 1
course title: as specified in Milestone 1
credits: as specified in Milestone 1
study load: as specified in Milestone 1
system: left-justified, 6 characters wide (truncated if longer than 6 characters)
NO NEW LINE CHARACTER

Note: Each Bar “|” character is surrounded by two blank spaces.

2. The GenEdCourse Class Specs.

Implement the **GenEdCourse** class in `GenEdCourse.h` and `GenEdCourse.cpp` as a class derived from the **Course** class.

Private Member Variables.

The `GenEdCourse` class has one private member variable:

- `langLevel_` : Integer

It holds the English language level requirement.

Constructor.

`GenEdCourse` has two constructors. The one-argument constructor that receives an integer as an argument. It uses the value of the argument to initialize the member variable `langLevel_`. The no-argument constructor sets this member variable to 0.

Public Member Functions.

```
int getLangLevel() const;
```

It returns the value of the member variable `langLevel_`.

```
void setLangLevel (int value);
```

It copies the incoming integer value into the member variable `langLevel_`.

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

After implementing the **GenEdCourse** class, overload the ostream operator << as a **free** helper function. Hints: You should implement a public member function called `display()` and reuse the function `display()` in the base class. The function displays the data values of an **GenEdCourse** object separated by the bar "|" character in following format:

```
PSYL1001 | Intro to Psychology | 1 | 3 | | 12 |
```

course code: as specified in Milestone 1

course title: as specified in Milestone 1

credits: as specified in Milestone 1

study load: as specified in Milestone 1

a field with 6 blank spaces

language requirement: right justified, 4 characters wide

NO NEW LINE CHARACTER

3. Modify the application class (ScmApp).

- a) Modify the application class such that the menu system can be used to add two types of courses, namely CS-related courses and general education courses.

3- Add a course.

Ask the user for a course type.

Ask for a course type using this prompt :

"Please enter the course type (1-CS or 2-GenEd): "

and get it from the console.

Then call the `addACourse()` function.

- b) Modify the `addACourse()` function such that it gets the data values **from the user** to initialize an **CSCourse** or **GenEdCourse** object. You should use the following order to get input values for an **CSCourse** object.

Course Code: COMP2006<ENTER>

Course Title: OOP in C++<ENTER>

Credits: 1<ENTER>

Study Load: 4<ENTER>
Computer System: oracle<ENTER>

You should also use the following order to get input values for a **GenEdCourse** object.

Course Code: PSYL1001<ENTER>
Course Title: Intro to Psychology <ENTER>
Credits: 1<ENTER>
Study Load: 4<ENTER>
Language Requirement: 12<ENTER>

B. Submission Requirements (MILESTONE 2).

1. Provide the NAME OF YOUR GROUP (EX. "C++ Commandos") as well as the NAMES of EACH GROUP MEMBER in a Word .doc. Create a File Folder called **MS2-C++ Commandos**. Add your files, [general.h](#), [Course.h](#), [ScmApp.h](#), [Cours.cpp](#), [ScmApp.cpp](#), including the word.doc, in your **MS2-C++ Commandos** file and .zip.
2. **Include SCREEN SHOTS** of the working code, in the .zip file.
3. Upload all your .zip source files to Blackboard.

CONGRATULATIONS!

**AFTER COMPLETING MILESTONES 1 AND 2, YOU WILL
HAVE A SENSE OF ACHIEVEMENT!**

TWO MORE MILESTONES TO GO...

1. **FOCUS OF MILESTONE 3: Interface, Abstract Base Class and Object Serialization.**
2. **FOCUS OF MILESTONE 4: Completion of the Application Class.**

MILESTONE 3: THE STREAMABLE INTERFACE , ABSTRACT BASE CLASS AND OBJECT SERIALIZATION.

In this milestone, the **Course** class and its derived classes will acquire *streamable* characteristics. As a result, the **Streamable** interface is provided to *enforce* that all concrete classes on the inheritance hierarchy will implement functions that work with `fstream` and `iostream` objects. In other words, the **CSCourse** and **GenEdCourse** classes have the capability of doing object serialization (i.e. writing an object to a file and reading an object from a file).

Code the **Streamable** interface in the **Streamable.h** file. It does not have an implementation file.

A. User Requirements

1. Pure virtual member functions.

The **Streamable** interface has four *pure virtual member functions*:

- 1- `fstream& store(fstream& file, bool addNewLine = true) const`
It is a constant member function (does not modify the owner) and receives and returns references of `std::fstream`.
- 2- `fstream& load(std::fstream& file)`
It receives and returns references of `std::fstream`.
- 3- `ostream& display(ostream& os) const`
It is a constant member function and returns a reference of `std::ostream`. It receives one argument: a reference of `std::ostream`.
- 4- `istream& read(istream& is)`
It returns and receives references of `std::istream`.

2. Modify the Course Class.

Derive the **Course** class from the **Streamable** interface.

3. Modify the CSCourse Class.

CSCourse implements all four pure virtual functions of the **Streamable** interface. (Note: The signatures of the functions are identical to those of **Streamable**.)

fstream& CSCourse::store(fstream& fileStream, bool addNewLine) const

Using the ostream operator <<, this function first writes the 'I' character (defined by **TYPE_CS**) and a comma into the argument **fileStream**. Then, without any formatting or spaces, it writes all the member variables of an **CSCourse** object, separated by commas, in the following order:

course code, course title, credits, study load, system

and a new line character if **addNewLine** is true. Finally it returns the argument **file**.

Example:

```
C,COMP2006,OOP in C++,1,4,matrix<Newline>
```

fstream& CSCourse::load(fstream& fileStream)

Using the istream operator >>, the ignore() and getline() functions of istream, this function reads a record from the argument **fileStream**. It uses setters to set the member variables of the current object. When reading a record, the function assumes that the record does not have the character 'C' (defined by **TYPE_CS**) at the beginning. Thus it starts reading from the field that has the course code.

No error detection is done.

At the end the argument **fileStream** is returned.

Hint: Create temporary variables of primitive data types to read the fields one by one, skipping the commas. After reading a field, use a setter to set a member variable of the current object.

ostream& CSCourse::display(ostream& os) const

Reuse the function display() that has been implemented in this class.

istream& CSCourse::read(istream& istr):

It receives data values from an istream object (the argument **istr**) in the following order:

Course Code: COMP2006<ENTER>

Course Title: OOP in C++<ENTER>

Credits: 1<ENTER>

Study Load: 4<ENTER>

Computer System: d21<ENTER>

It uses the setters to set the member variables of the current object. Finally it returns

the argument `istr`.

4. Modify the GenEdCourse Class.

`GenEdCourse` implements all four pure virtual functions of the `Streamable` interface. (Note: The signatures of the functions are identical to those of `Streamable`.)

```
fstream& GenEdCourse::store(fstream& fileStream, bool addNewLine) const
```

Using the ostream operator `<<`, this function first writes the 'G' character (defined by `TYPE_GEN`) and a comma into the argument `fileStream`. Then, without any formatting or spaces, it writes all the member variables of an `GenEdCourse` object, separated by commas, in the following order:

```
course code, course title, credits, study load, language requirement
```

and a new line character if `addNewLine` is true. Finally it returns the argument `fileStream`.

Example:

```
I,EAC150,College English,1,4,12<Newline>
```

```
fstream& GenEdCourse::load(fstream& fileStream)
```

Using the istream operator `>>`, the `ignore()` and `getline()` functions of istream, this function reads a record from the argument `fileStream`. It uses setters to set the member variables of the current object. When reading a record, the function assumes that the record does not have the character 'G' (defined by `TYPE_GEN`) at the beginning. Thus it starts reading from the field that has the course code.

No error detection is done.

At the end the argument `fileStream` is returned.

Hint: Create temporary variables of primitive data types to read the fields one by one, skipping the commas. After reading a field, use a setter to set a member variable of the current object.

```
ostream& GenEdCourse::display(ostream& os) const
```

Reuse the function `display()` that has been implemented in this class.

```
istream& GenEdCourse::read(istream& istr):
```

It receives data values from an istream object (the argument `istr`) in the following order:

```
Course Code: PSYL1001 <ENTER>
Course Title: Intro to Psychology <ENTER>
Credits: 1<ENTER>
Study Load: 4<ENTER>
Language Requirement: 12<ENTER>
```


It uses the setters to set the member variables of the current object. Finally it returns the argument `istr`.

5. Modify the header file `general.h`.

Define `TYPE_CS` as 'C' and `TYPE_GEN` as 'G' in the header file `general.h`.

B. SUBMISSION REQUIREMENTS (MILESTONE 3).

1. Provide the NAME OF YOUR GROUP (EX. "C++ Commandos") as well as the NAMES of EACH GROUP MEMBER in a Word .doc. Create a File Folder called **MS3-C++ Commandos**. Add your files, `general.h`, `Course.h`, `ScmApp.h`, `Coures.cpp`, `ScmApp.cpp`, including the word.doc, in your **MS3-C++ Commandos** file and .zip.
2. Include **SCREEN SHOTS** of the working code, in the .zip file.
3. Upload all your .zip source files to Blackboard.

THE END OF MILESTONE 3

MILESTONE 4: COMPLETION OF THE APPLICATION CLASS.

In this last milestone, you are going to implement two new functionalities of the application class (ScmApp). The application class will allow a user to save courses to a file and read courses from a file. In brief, you are going to implement selections #5 and #6 on the following menu system.

Georgian 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.
- > _

A. User Requirements

The ScmApp Class.

1. Private Member Variables.

char filename_ [256];

It holds the name of the text file that stores information about the courses.

fstream datafile_;

An fstream object used to create and access a file.

2. The Constructor.

It receives a const character string as an argument (**filename**) and does the following initialization:

- a) It copies the argument filename to the member variable **filename_**.
- b) It initializes other member variables as specified in Milestone 1.

3. Private Member Functions.

Note: You are free either to use your own logic or follow the pseudo-code or improve the pseudo-code.

void loadRecs();

It opens the file for reading the courses, If the file does not exist, it will create an empty file and exit. Otherwise it loads the records (i.e. objects) from the file to initialize new objects. These new objects will be pointed by the member variable **courseList_**.

Note: In order to make sure that there is no memory leak *before* loading the records from the file, the function must release the memory storage that has been allocated for *all* the objects that are currently pointed by the member variable **courseList_.**

Finally, it closes the file

Here is the pseudo-code:

```
Set readIndex to zero.
Open the file for reading (use ios::in).
If the file is in fail state (i.e. the files does not exist on the disk),
then
    clear the failure,
    close the file.

Otherwise
    Deallocate the memory for all objects pointed by courseList_.

until reading fails
loop
    read one char character into a variable (objectType) to identify the
    type of a course.
    If the value of objecttype is 'G'
        Dynamically create a GenEdCourse object and store the object address at
        courseList_[readindex]
    If the value of objecttype is 'C'
        Dynamically create an CSCCourse object and store the object address at
        courseList_[readindex]

    If neither 'G' nor 'C' is read
        skip the comma in the file
        load the record (i.e. the course object) from the file, using the load()
        function
        add one to read index
    continue the loop
    Set the number of courses to readIndex.
Close the file.
```

void saveRecs();

- It opens the file for writing records (i.e. course objects).
- It loops through the array `courseList_` up to `noOfCourses_` and stores the objects in the file.
- It closes the file.

B. FINAL SUBMISSION REQUIREMENTS (MILESTONE 4).

1. Provide the NAME OF YOUR GROUP (EX. "C++ Commandos") as well as the NAMES of EACH GROUP MEMBER in a Word .doc. Create a File Folder called **MS4-C++ Commandos**. Add your files, `general.h`, `Course.h`, `ScmApp.h`, `Coures.cpp`, `ScmApp.cpp`, including the word.doc, in your **MS4-C++ Commandos** file and .zip.
2. **Include SCREEN SHOTS** of the working code, in the .zip file.
3. Upload all your .zip source files to Blackboard.

THE END OF MILESTONE 4

CONGRATULAIONS! YOU HAVE COMPLETED THE FINAL PROJECT!