Summer 2016
OOP244 Final Project (v. 3.3)

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.

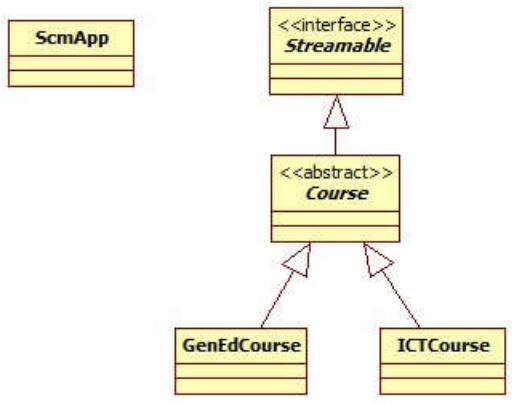
GenEdCourse

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 12, 2016 by 11:59:00pm.**
  - MS1: The Course and ScmApp classes. Due: July 24, 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 noOfCourses_ 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	OOP in C++	1	4
BTP200	OO Paradigm Using C++	1	4
IPC144	Programming Using C	1	4
BTP100	Intro C Programming	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	Intro to Philosophy	1	3
PSY141	Social Psychology	1	3
HIS201	World War II	1	3
SOC600	Intro to Sociology	1	3

THE END OF MILESTONE 1

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

Learning Focus: Inheritance and Polymorphism.

Learning Tasks

1. **Create two classes (ICTCourse, 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 ICT course or a GenEdCourse.)**
   - **Change the study load of a course.**

******Please add the following modification to ALL of your classes. In all of your classes, declare a friend class named xxxxxTester, where xxx is the name of your class.**

- **CourseTester**
- **ScmAppTester**
- **ICTCourseTester**
- **GenEdCourseTester**

*****Class setter and getters – for the various Course classes, your getters and settings must have the following function names:**

- **setCourseTitle**
- **setCourseCode**
- **setCredits**
- **setStudyLoad**
- **setComputerSystem**
- **setLangLevel**
- **getCourseTitle**
- **getCourseCode**
- **getCredits**
- **getStudyLoad**
- **getComputerSystem**
- **getLangLevel**

**** Course class – isEmpty() must return true in the following condition**

- **courseCode_ is empty “”**
- **courseTitle_ is nullptr or empty “”**
- **courseCredit_ is 0 or -1**
- **studyLoad_ is 0 or -1**

A. User Requirements.

1. The ICTCourse Class Specs.

Implement the ICTCourse class in ICTCourse.h and ICTCourse .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 ICT course (e.g. matrix, oracle, as400, win and linux).

Constructors

The ICTCourse class has two constructors. The FIVE-ARGUMENT constructor that receives four arguements for data members of the base class and a string for the data member of the the derived class. It uses the value of the string 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 ostreram operator << (a FREE helper function):

Reuse the ostream operator<< from Milestone 1. Hint: In order to have polymorphic behavior, you should implement a virtual function called display( ) and reuse the function display( ) in the base class.. The function displays the data values of an **ICTCourse** object in the following format (separated by the bar "|" character):

```
OOP244 | OOP in C++          | 1 | 4 | matrix |
```

course code: 6 Characters Wide
course title: 41 Characters Wide
credits: 9 Characters Wide
study load: 12 Characters Wide
system: **left-justified, 9 characters wide, followed by a "|"**
language: **blank right-justified, 13 characters wide, followed by a "|"**

NO NEW LINE CHARACTER

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 GendEdCourse class has one private member variable:

- **langLevel_** : Integer

  It holds the English language level requirement.

Constructor.

GenEdCourse has two constructors. The FIVE-ARGUMENT constructor that receives four arguements for data members of the base class and a integer for the data member of the derived class. 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 ostreram operator << (a FREE helper function).

Reuse the ostream operator<< from Milestone 1. Hint: In order to have polymorphic behavior, you should implement a virtual 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:

```
EAC150 | College English     |  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
system: black, left-justified, 9 characters wide, followed by a "|"

language: **right-justified, 13 characters wide, followed by a "|"**

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 ICT-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-ICT or 2-GenEd): "
     and get it from the console.
     Then call the addACourse() function.
```

b) Modify the addACourse( ) function such that it *receives an integer as a parameter*. 1 means that an **ICTCourse** object will be added. 2 means that a **GenEdCourse** object will be added.

You should use the following order to get input values for an **ICTCourse** object.

```
Course Code: OOP244<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: EAC150<ENTER>
Course Title: College English<ENTER>
Credits: 1<ENTER>
Study Load: 4<ENTER>
Language Requirement: 12<ENTER>
```

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 **ICTCourse** 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 ICTCourse Class.

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

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

Using the ostream operator << , this function first writes the 'I' character (defined by **`TYPE_ICT`**) and a comma into the argument **fileStream.** Then, without any formatting or spaces, it writes all the member variables of an ICTCourse 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:

```
I,OOP244,OOP in C++,1,4,matrix<Newline>
```

`fstream& ICTCourse::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 'I' (defined by **`TYPE_ICT`**) 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& ICTCourse::display(ostream& os) const
```

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

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

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

```
Course Code: OOP244<ENTER>
Course Title: OOP in C++<ENTER>
Credits: 1<ENTER>
Study Load: 4<ENTER>
Computer System: oracle<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: EAC150<ENTER>
Course Title: College English<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_ICT`** as 'I' and **`TYPE_GEN`** as 'G' in the header file **general.h**.

B. TESTING AND SUBMISSION REQUIREMENTS (MILESTONE 3).

1. To be released

THE END OF MILESTONE 3

MILESTONE 4: COMPLETION OF THE APPLICATION CLASSS.

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.

```
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.
> _
```

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 is not open, close the file.

Otherwise
  Deallocate the memory for all objects pointed by courseList_.

  Loop (until all records have been read.
     read one char character into a variable to identify the
     type of a course.
     If the value of objectype is 'G'
        Dynamically create a GenEdCourse object and store the object address at
```

```
        courseList_[readindex]

    If the value of objectype is 'I'

        Dynamically create an ICTCourse object and store the object address at

        courseList_[readindex]


    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.

Testing and Submission Requirements (MILESTONE 4).

1. If not on matrix already, upload **general.h**, **Course.h, ScmApp.h**, **Course.cpp, ScmApp.cpp, ICTCourse.h, ICTCourse.cpp, GenEdCourse.h, GenEdCourse.cpp, Streamable.h** to your matrix account. Compile and run your code and make sure that everything works properly. Your program must pass all the tests in order for you to submit your files.

   To test your submission, use on of the two following commands based on whether you are using the standard string library or c_style strings.

   If your program uses the string library, use the following command.

   ```
   ~bradly.hoover/submit ms4-string
   ```

   If your program is using c-style strings, us the following command:

   ```
   ~bradly.hoover/submit ms4-char
   ```