COMP 2404 B/C – Assignment #2

Due: Thursday, February 15 at 11:59 pm

1. Goal

For this assignment, you will write a program in C++, in the Ubuntu Linux environment of the course VM, to build and display the end user’s weekly course schedule for different academic terms. The program will allow the user to change the selected term, or view the available courses for the selected term, or view the user’s weekly schedule for that term, or add a course to the user’s schedule, or clear their schedule.

2. Learning Outcomes

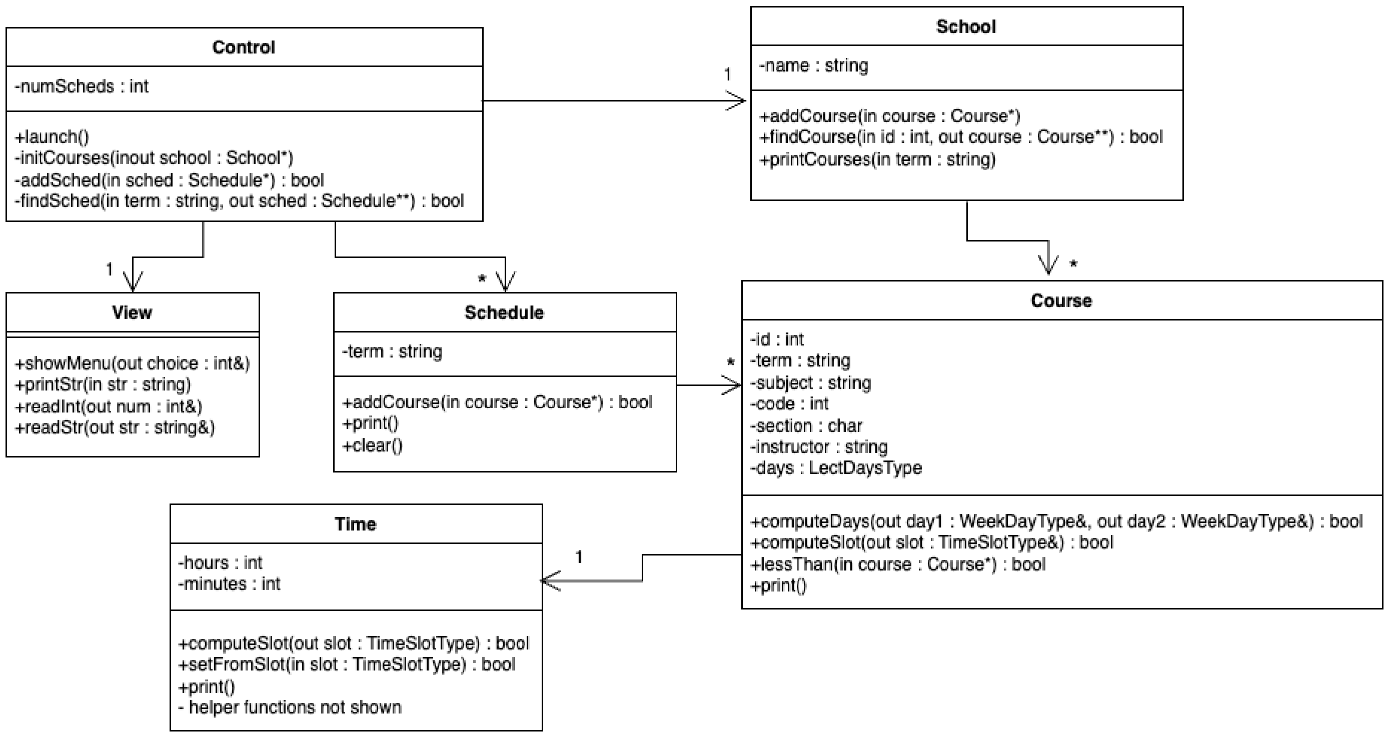
With this assignment, you will:

- practice implementing a design that is given as a UML class diagram
- implement a program separated into control, view, entity, and collection objects
- write code that uses dynamically allocated memory and follows the principle of least privilege

3. Instructions

3.1. Understand the UML class diagram

You will begin by understanding the UML class diagram below. Your program will implement the objects and class associations represented in the diagram, as they are shown. UML class diagrams are explained in the course material in section 2.3.


Control
-numScheds : int
+launch()
-initCourses(inout school : School*)
-addSched(in sched : Schedule*) : bool
-findSched(in term : string, out sched : Schedule**) : bool
School
-name : string
+addCourse(in course : Course*)
+findCourse(in id : int, out course : Course**) : bool
+printCourses(in term : string)
View
+showMenu(out choice : int&)
+printStr(in str : string)
+readInt(out num : int&)
+readStr(out str : string&)
Schedule
-term : string
+addCourse(in course : Course*) : bool
+print()
+clear()
Course
-id : int
-term : string
-subject : string
-code : int
-section : char
-instructor : string
-days : LectDaysType
+computeDays(out day1 : WeekDayType&, out day2 : WeekDayType&) : bool
+computeSlot(out slot : TimeSlotType&) : bool
+lessThan(in course : Course*) : bool
+print()
Time
-hours : int
-minutes : int
+computeSlot(out slot : TimeSlotType) : bool
+setFromSlot(in slot : TimeSlotType) : bool
+print()
- helper functions not shown
1
1
*
*
*
1

The program data is organized as follows:

3.1.1. A `School` contains the primary collection of courses that are available for student registration.

3.1.2. A `Course` represents a single course offering, with the usual information: a unique id; the academic term when the course is offered, for example "F23" or "W24"; the subject (e.g. "COMP"); the course code (e.g. 2404); the section (e.g. ’B’ or ’C’); and the instructor’s surname. In addition, a course includes the combination of days when the lectures take place (either Mon/Wed, Tue/Thu, or Wed/Fri), represented as a provided enumerated data type, as well as the start time of the course lectures.

3.1.3. The `Control` object allows the end user to manage multiple schedules, one for each academic term.

3.1.4. *Definition #1:* In a user's schedule, a *day* indicates a work-week day (Monday through Friday) when a course's lectures take place, as represented in a provided enumerated data type.

3.1.5. *Definition #2:* In a schedule, a *time slot* indicates the start time of a course's lectures, using one of seven possible options: 8:30 am, 10:00 am, 11:30 am, 1:00 pm, 2:30 pm, 4:00 pm, or 6:00 pm. Time slots are also represented using a provided enumerated data type.

3.1.6. A `Schedule` contains the courses that a user attends weekly, displayed with one column for each of the five days of the week, and one row for each time slot when a lecture can take place.

To make the schedule printing easier, each schedule contains of a 2D array of `Course` *pointers*, representing the courses that the user is taking each week. In the first dimension of the array, each element represents a single week day, and in the second dimension, each element represents a time slot on that day. Every course is offered twice a week, in the same time slot. The schedule must **not** contain duplicate course objects, only pointers to courses already stored in the School's primary collection. As enumerated values correspond to numbers, they can be used as array indexes.

3.2. Understand the provided base code

The `a2-posted.tar` file contains the `View` class that your code must use for most communications with the end user. It contains a skeleton `Control` class, with a data initialization member function that your code is required to call. It also contains a basic `Time` class that will be used as indicated in the UML class diagram above. Finally, the enumerated data types are defined in the provided `defs.h` file.

3.3. Modify the `Time` class

You will modify the `Time` class provided in the base code, as follows:

3.3.1. Implement a `bool computeSlot(TimeSlotType& slot)` member function that computes the time slot corresponding to the `Time` object, and sets the `slot` parameter to the matching enumerated data type value.

3.3.2. Implement a `bool setFromSlot(TimeSlotType slot)` member function that sets the hours and minutes of the `Time` object to the values corresponding to the given time slot. For example, if the `slot` parameter represents 14:30, then the function must set the hours to 14 and the minutes to 30.

3.4. Implement the `Course` class

You will create a new `Course` class that represents a course offering. This class will contain all the data members and member functions indicated in the given UML class diagram. In addition:

3.4.1. The course's `Time` data member must be stored as a pointer, and the object must be dynamically allocated in the constructor.

3.4.2. The default constructor must take eight (8) parameters: the term, course subject, code, section, instructor, lecture days, and the hours and minutes of the course's start time. The course id will be set to its permanent value when the course is added to the school's course collection.

3.4.3. The destructor must deallocate the required memory.

3.4.4. The `computeDays()` member function computes the enumerated type values of the two week days that correspond to the lecture days data member. For example, if a course is offered on Tue/Thu, then the enumerated values returned by the function will correspond to Tuesday and Thursday.

3.4.5. The `computeSlot()` member function computes the time slot when the course is offered, based on the `Time` data member.

3.4.6. The `lessThan()` member function compares whether the `Course` object should be ordered before the given parameter, when compared first by subject, then by code, then by term, then by section.

3.4.7. The `print()` member function must print every data member, in the **same format** shown in the workshop video. The lecture days must be printed as an informative string, and not a numeric value.

3.4.8. You must provide getter member functions for the course id, term, instructor, as well as a getter member function that formats the entire course code into a single string using the following format: `"subject code-section"`, for example "COMP 2404-C".

3.4.9. You will also need a setter member function for the course id, so that a course can be assigned a unique identifier when it is added to the school's primary collection.

3.5. Implement the `CourseArray` class

You will copy the `Array` class that we implemented in the coding example of section 2.2, program #1, into a new collection class called `CourseArray`. Then, you must make the following changes:

3.5.1. Modify the collection class so that it stores `Course` object pointers as data.

3.5.2. Add a new data member to keep track of the unique id that will be assigned to the next course added to the array. This data member must be initialized in the constructor using a provided constant.

3.5.3. Modify the `add()` member function so that it adds the given course to the array, in its correct position, so that the courses remain in ascending order:
- (a) you must shift the elements towards the back of the array to make room for the new element in its correct place
- (b) do not add to the end of the array and sort, as this is both inefficient and unnecessary
- (c) you must use the `Course` class's `lessThan()` member function to perform the comparisons
- (d) you must set the new course's id to its permanent value, using the next id data member

3.5.4. Implement a new `bool find(int id, Course** c)` member function that does the following:
- (a) search the array for the course with the given id
- (b) if the course is found, a pointer to that `Course` object is "returned" in the `c` parameter, and the function returns success
- (c) if the course is not found, the pointer returned in `c` is set to null, and the function returns failure

3.5.5. Modify the `print()` member function so that it takes a string as a parameter, to represent an academic term, and then prints only the courses in the array that are offered in the given term.

3.6. Implement the `Schedule` class

You will create a new `Schedule` class that represents a schedule for a specific term. This class will contain all the data members and member functions indicated in the given UML class diagram. In addition:

3.6.1. The schedule stores a 2-dimensional array of `Course` pointers, organized by week days and time slots, as described in instruction 3.1.6.

3.6.2. The default constructor must take one string parameter for the schedule's term, and initialize that data member. It must also initialize the 2D array to null pointers.

3.6.3. The `addCourse()` member function adds the given course to the schedule, as follows:
- (a) you must check that the course parameter is not null, and that the course's term matches the schedule's term
- (b) if any error occurs, the course cannot be added to the schedule; the program must print out a detailed error message and return false
- (c) compute the two week days and the time slot when the course is offered
- (d) add the course to the schedule in *two* places: in the column for the first day and the row for that time slot, and in the column for the second day and the same row
  - (i) *only **one** instance* of each course must exist! **do not** make any copies of any courses; instead, the 2D array must contain pointers to courses already in the School's course collection
  - (ii) enumerated data type values are represented internally as sequential numeric values, by default starting at zero; because of this, the mapping from enumerated value to array index is very simple

3.6.4. The `print()` member function must print every data member, in the **same format** shown in the workshop video, with columns for each week day, rows for each time slot, and all the headers and borders shown.

NOTE: The time slot start times must be printed using the `Time::print()` member function. To do this, it may be necessary to perform an explicit type conversion (a hard typecast) from an array index value to the time slot enumerated type.

3.6.5. The `clear()` member function removes all the courses from the schedule, by setting every element in the 2D array to null pointers.

3.6.6. You must provide a getter member function for the term.

3.7. **Implement the `School` class**

You will create a new `School` class that contains all the data members and member functions indicated in the given UML class diagram. In addition:

3.7.1. The school's collection of courses must be stored using a `CourseArray` object.

3.7.2. The default constructor must take one string parameter for the school's name, and initialize that data member.

3.7.3. The destructor may be empty, but its presence is required so that the destructor of the collection object is called automatically when the school object is deallocated.

3.7.4. The `addCourse()` member function adds the given course to the school's course collection.

3.7.5. The `findCourse()` member function finds the course with the given id, and returns the found course using the output parameter. The return value indicates whether or not the course id was found.

3.7.6. The `printCourses()` member function prints out the term, and the details of each course in the course collection that matches the given term.

3.8. **Implement the `Control` class**

You will implement the `Control` class with all the data members and member functions indicated in the given UML class diagram. In addition:

3.8.1. The School data member must be stored as a `School` pointer, and the object must be dynamically allocated in the constructor.

3.8.2. The collection of schedules must be stored as a primitive array of `Schedule` pointers.

3.8.3. The destructor must clean up the necessary memory.

3.8.4. The `addSched()` member function adds a given schedule to the back of the schedules array.

3.8.5. The `findSched()` member function finds the schedule for the given term and returns it using the output parameter.

3.8.6. The `launch()` member function does the following:

(a) call the initialization function provided in the `a2-posted.tar` file; you must use this function, *without modification*, to initialize the contents of the school data member

(b) use the `View` object to repeatedly print out the main menu and read the user's selection, until the user chooses to exit

(c) if the user chooses to change the selected term:

(i) use the `View` object to prompt the user to enter a term and store this value in a local variable

(ii) all menu options must be performed for this selected term, until the user chooses another

(iii) check if a schedule already exists for the new selected term; if not, dynamically allocate a new schedule for that term and add it to the schedules array

(d) if the user chooses to view the courses for the selected term, print them out in ascending order

(e) if the user chooses to view the schedule for the selected term, check that a schedule already exists for this term; if none exists, the user must be notified, and the operation cannot be completed; otherwise, print the corresponding schedule

(f) if the user chooses to add a new course to the schedule for the selected term:

(i) check that a schedule already exists for the selected term; if none exists, the user must be notified, and the operation cannot be completed

(ii) otherwise, use the `View` object to prompt the user and read in a course id; find the corresponding course in the school's collection; then add the found course to the user's schedule for the selected term

(iii) if the course is not found, or if the found course is not offered in the selected term, the user must be notified, and the operation cannot be completed

(g) if the user chooses to clear the schedule for the selected term, check that a schedule already exists for this term; if none exists, the user must be notified, and the operation cannot be completed; otherwise, clear the corresponding schedule of all courses

NOTE: The above functionality must reuse existing functions, everywhere possible.

3.9. **Write the `main()` function**

Your `main()` function must declare a `Control` object and call its `launch()` function. The entire program control flow must be implemented in the `Control` object as described in the previous instruction, and the `main()` function must do nothing else.

3.10. **Packaging and documentation**

3.10.1. Your code must be correctly separated into header and source files, as we saw in class.

3.10.2. You must provide a `Makefile` that separately compiles each source file into a corresponding object file, then links all the object files to produce the program executable. Your `Makefile` must also include the `clean` target that removes all the object files and the program executable. **Do not** use an auto-generated Makefile; it must be specific to this program.

3.10.3. You must provide a plain text `README` file that includes:

(a) a preamble (program author, purpose, list of source, header, and data files, if applicable)

(b) compilation and launching instructions, including any command line arguments

3.10.4. Use either the `tar` or the `zip` utility in Linux to package into one `.tar` or `.zip` file **all the files** required to build and execute your program.

3.10.5. Do not include any additional files or directories in your submission.

3.10.6. All class definitions must be documented, as described in the course material, section 1.3. Please **DO NOT** place inline comments in your function implementations.

4. Constraints

Your design and implementation must comply with all the rules of correct software development and programming conventions that we have learned during the course lectures, including but not restricted to:

4.1. The code must be written using the C++11 standard that is supported by the course VM, and it must compile and execute in that environment. It must not require the installation of libraries or packages or any software not already provided in the course VM.

4.2. Your program must comply with the principle of least privilege, and it must follow correct encapsulation principles, including the separation of control, view, entity, and collection object functionality.

4.3. Your program must not use any library classes or programming techniques that are not used in the in-class coding examples. Do not use any classes, containers, or algorithms from the C++ standard template library (STL), unless explicitly permitted in the instructions.

4.4. If base code is provided, do not make any unauthorized changes to it.

4.5. Your program must follow basic OO programming conventions, including the following:

4.5.1. Do not use any global variables or any global functions other than `main()`.

4.5.2. Do not use `struct`s. You must use classes instead.

4.5.3. Objects must always be passed by reference, never by value.

4.5.4. Functions must return data using parameters, not using return values, except for getter functions and where otherwise permitted in the instructions.

4.5.5. Existing functions and predefined constants must be reused everywhere possible.

4.5.6. All basic error checking must be performed.

4.5.7. All dynamically allocated memory must be explicitly deallocated.

4.6. You may implement helper classes and helper member functions, but only if the design is consistent with the provided instructions. Design and implementation choices that undermine the learning outcomes will not earn any marks.

5. Submission Requirements

5.1. You will submit in *Brightspace*, before the due date and time, one `tar` or `zip` file that includes all your program files, as described in the **Packaging and documentation** instructions.

5.2. Late submissions will be subject to the late penalty described in the course outline. No exceptions will be made, including for technical and/or connectivity issues. Do not wait until the last day to submit.

5.3. Only files uploaded into *Brightspace* will be graded. Submissions that contain incorrect, corrupt, or missing files will be graded as is. Corrections to submissions will not be accepted after the due date and time, for any reason.

6. Grading Criteria

• 35 marks:	code quality and design
• 35 marks:	coding approach and implementation
• 25 marks:	program execution
• 5 marks:	program packaging