[HW] Attendance Swipe

Or: Practicing with Classes and, finally, using std::vector

[Overview](#_4wbswp5ji4hs)

[Objectives](#_3znysh7)

[Starter Code](#_2et92p0)

[Requirements & Roadmap](#_tyjcwt)

[Step 1: class Student](#_3dy6vkm)

[Step 2: Class Date](#_1t3h5sf)

[Step 3: list students, check if enrolled, list courses for student](#_4d34og8)

[Step 4: Add courses (from a file); list courses](#_2s8eyo1)

[Step 5: Add attendance records (from a file); list attendance records](#_17dp8vu)

[Data Files](#_3rdcrjn)

[Useful functions & classes](#_26in1rg)

[Requirements](#_lnxbz9)

[Recommendations](#_35nkun2)

[Getting Started](#_1ksv4uv)

[Submission](#_44sinio)

[Background Information: std::vector](#_2jxsxqh)

[Have Fun!](#_z337ya)

# Overview

Back when we had CSCE 121 lectures in the ZACH classrooms, some instructors collected attendance information by having students swiping their IDs as they arrived to class. Using your formidable knowledge of C++ classes, you’ve been asked to set up a system that loads attendance data into a program and produces reports of who was present.

# Objectives

1. Additional practice with classes
   1. Constructors
   2. Getters and Setters
   3. Encapsulation
      1. Public/Private
2. Work with vectors rather than arrays.
3. More practice with throwing and catching exceptions.

# Starter Code

The starter code on Mimir contains files: School.h, School.cpp, Course.h, Course.cpp, Student.h, Student.cpp, AttendanceRecord.h, AttendanceRecord.cpp, and attendance-main.cpp The file attendance-main.cpp is useful to test your code locally.

To compile the starter code successfully before coding all the necessary functions, you can comment out invocations until you implement them.

# Requirements & Roadmap

You are given the main function that will use the following classes. You need to implement the following classes according to the header files provided for each.

Note: Just because you have a member function declaration, you don’t have to have the function definition for the program to compile unless the program needs to use that function.

* As you develop your code, you only need to define the functions you need at the moment. You don’t have to get everything done at once. It is not a good strategy to do so.
* You might need to comment out calls in the attendance.cpp file to functions you have not yet defined/written to be able to compile, execute and debug.

## Step 1: class Student

We provide:

* Student class definition (Student.h)

You implement in Student.cpp:

* Student constructor
* Student::addCourse  
  If the course being already exists, throws std::invalid\_argument
* Student::listCourses  
  The figure below shows expected format for students with and without courses.

|  |
| --- |
| Courses for 166100166  121  222  410 Courses for 660166834  No courses |

## Step 2: Class Date

We provide:

* Date class definition, constructors, and getter methods (Date.h)
* The implementation of methods that generate strings for Date data attributes (getDateTime(), getDate(), getTime())
* The implementation of operator<= , operator>=, and operator> (using the operators that you will implement)

You implement in Date.cpp:

* Comparison operators: operator<

## Step 3: list students, check if enrolled, list courses for student

We provide:

* Student class definition (Student.h)
* Implementation of School::addStudents(string filename)(School.cpp)
  + Yes, we implemented this one for you. Take a look at it. Now.
* You can use attendance-main.cpp for testing.

You implement:

* Student constructor (Student.cpp)
* School::isEnrolled(string student\_id)
* School::listStudents() (School.cpp)
  + Use to output and test
  + Should output “No students” if there are no students
  + The output must follow this format:

|  |
| --- |
| 166100166,Tanja Cypert 660166834,Bernice Ringer 930740679,Brent Slagle |

## Step 4: Add courses (from a file); list courses

We provide:

* Course class definition (Course.h)
* You can use attendance-main.cpp for testing.

You implement:

* Course constructor, four getter methods
* School::addCourses(string filename)
  + Add Course instances to the vector of courses
  + If there is an error opening the file, the function must thrown an std::invalid\_argument exception;
  + The input file will have the information formatted as the following figure. You can assume that the data in the file is correct:
    - Every line is well-formed
    - The data for start and end time, id, and title are correct
    - There are no repeated lines.
  + The file may have empty lines at the end.

|  |
| --- |
| 551,15:00,15:50,Biochemistry, Wine, and Practices in the Environmental Sciences 587,15:55,17:10,Pirates, Mineralogy, and Purpose 521,08:00,08:50,Cryptozoology |

* School::listCourses(ostream& os)
  + Use to output and test
  + Should output “No courses” if there are no courses
  + The output is expected to be formatted as in the figure above.

## Step 5: Add attendance records (from a file); list attendance records

We provide:

* AttendanceRecord class definition (AttendanceRecord.h)
* You can use attendance-main.cpp for testing.

You implement:

* The AttendanceRecord constructor and all other member functions.
* unsigned int School::addAttendanceData(string filename)  
  The input file has the attendance information (date, course id, and student id) formatted as the following figure. There may be blank lines at the end of the file.  
  This member function will process the attendance records in the file and returns the number of invalid attendance records. A record is valid when:
  + The student id appears in the list of students
  + The course id appears in the list of courses
  + The date is a valid date (Date::isValid() can be used to determine if it is valid)
  + Course::addAttendanceRecord did not throw an exception (as described in the next item)

|  |
| --- |
| 2020-08-26 15:08:56,551,166100166 2020-08-26 15:39:06,551,930740679 2020-08-26 15:41:54,551,660166834 2020-08-28 15:02:45,551,930740679 2020-08-28 15:10:15,551,166100166 2020-08-28 15:17:46,551,660166834 2020-08-30 15:05:03,551,930740679 2020-08-30 15:35:24,551,166100166 2020-09-02 15:12:05,551,930740679 2020-09-02 15:20:19,551,660166834 2020-09-04 15:06:47,551,930740679 2020-09-04 15:24:52,551,166100166 2020-09-04 15:48:34,551,660166834 2020-09-06 15:28:29,551,166100166 |

* Course::addAttendanceRecord(AttendanceRecord ar)
  + We consider an AttendanceRecord valid if the time is greater than or equal to the course start time and less than or equal to the end time for the course. The function must throw the exception std::invalid\_argument if the time is invalid
* Output course attendance:   
  School::outputCourseAttendance(string course\_id)  
  List all attendance records for the specified course
  + Throws std::invalid\_argument exception if course is not found
  + Should output “No records.” if none are found.
  + Figure above illustrates the expected formatting.
* Output course attendance for a specific student:   
  School::outputStudentAttendance(string student\_id,   
   string course\_id)  
  List all attendance records for the specified course and student
  + Throws std::invalid\_argument exception if course or student is not found
  + Should output “No records” if none are found.
  + Example output in the figure below:

|  |
| --- |
| 8-27-2019 16:36:44,587,930740679 8-29-2019 16:20:05,587,930740679 9-3-2019 16:05:59,587,930740679 9-5-2019 16:39:17,587,930740679 |

## Data Files

The input files we use in the Mimir tests are available in the Google Shared Drive ([link to folder)](https://drive.google.com/drive/folders/1ODxmXblR0AO38dEIR0qq0BywmimACstg?usp=sharing)

* student-small.csv
* course-small.csv
* attendance-small.csv
* student-medium.csv
* course-medium.csv
* attendance-medium.csv
* student-large.csv
* course-large.csv
* attendance-large.csv

## Useful functions & classes

You may use any of the functionality provided by C++ strings and C++ vectors in your solution. Most students may find the following operations useful:

* getline
  + An example of how to use this is provided in  
    School::addStudents(string filename)
* vector
  + push\_back(elem)  
    Adds a new element elem at the end of the vector
  + size()  
    Returns the number of elements in the vector
  + More information about vectors is available in the [Background Information section](#_2jxsxqh)
* istringstream
  + You already used this in a labwork. If you need to refresh your memory, an example of how to use this is provided in  
    School::addStudents(string filename)
* setw()and setfill()
  + An example of how to use this is provided in string Date::getDate()

## Requirements

* You are required to implement classes School, Student, Course, and AttendanceRecord. You are required to implement any of the class member functions not already provided for you in the starter code.
* Use of an unapproved header file will result in a score of 0.
* The program must compile without warnings or errors.

g++ -std=c++17 -Wall -Wextra -pedantic -g School.cpp Student.cpp Course.cpp Attenance.cpp

* The program must run without errors or unhandled exceptions.

# Recommendations

1. Read the header and source files.
2. Think before you code.
3. Pick small problems to solve first, write test cases first, and develop incrementally.
4. Use descriptive (long) naming conventions for variables and functions.
5. Add comments to the code to describe anything which is not obvious from the code.
6. Use whitespace (indentation, newlines) to visually organize code.
7. Use functions to reduce code duplication and increase abstraction.

# Getting Started

1. It is the end of the semester. You are busy. Start and finish early.
2. Download the starter code.
3. Download the Shared Drive input files for testing your program locally.
4. Read the header and source files.
5. Follow the suggested order of adding functionality: Step 1 (add Students), Step 2 (add Courses), and Step 3 (add Attendance Records).  
   As you implement the functions needed in each step, compile the program.
   1. It may not compile due to undefined reference errors.
   2. You can comment out the code you don’t need yet
6. Follow the instructions for Submit it to Mimir.
   1. So that you can see the test cases.
   2. Recommendations:
      1. Think before you code: plan your program on paper (digital or analog) before mashing the keyboard
      2. Write test cases before writing the implementation of the method
7. Recompile and rerun (run == test).
   1. Check for errors.
   2. If no errors, move on
   3. Else, start debugging
   4. When changing your code to fix errors, before executing or uploading to Mimir, stop and explain to yourself (or to a [rubber duck](https://en.wikipedia.org/wiki/Rubber_duck_debugging)) why your bug fix will work.
8. Submit to Mimir.
   1. So that you can checkpoint your code and see your progress.
9. When you get stuck, ask for help on Piazza and attend office hours. The TAs and the instructors will help you.
   1. Read the prior posts before posting a new question (which may be a duplicate).

# Submission

The source files to submit are named

* attendance-main.cpp
* School.h and School.cpp
* Student.h and Student.cpp
* Course.h and Course.cpp
* AttendanceRecord.h and AttendanceRecord.cpp

# Background Information: std::vector

std::vector is a C++ standard class that provides the functionality of dynamically allocated arrays. You can learn more about C++ vectors in the documentation ([link to std::vector documentation](http://www.cplusplus.com/reference/vector/vector/)) or in the Zybook Chapter 8.2.

A simple example example of using std::vector to store a list of double values is below:

|  |
| --- |
| #include <iostream> #include <vector>  #include <iostream>  #include <vector>  int main() {   size\_t n = 10;  std::vector<double> vec\_of\_doubles(n);  for (size\_t i = 0; i < n; i++) {  vec\_of\_doubles[i] = i;  }   std::vector<int> v;  // another way of adding elements to vector v  for (size\_t i = 0; i < n; i++) {  v.push\_back(i); // add i to the last position in v  }   // let's print the elements in a vector  // we can use a for loop as before, or we can use vector iterators:  for (double x: vec\_of\_doubles) { // x will iterate over all elements  std::cout << x << " " ;  }  std::cout << std::endl;  return 0; } |

# Have Fun!

It is much more fun for you and you get better support from instructors and TAs when you do not start working on the homework just before it is due.