Team Members:
Kimberly Price (Team Leader)
Kaji Rashad
Andrew Chabab
Jamal West
Alexander Gonzalez
Alberto Spadaro

Professor: Dr. Liu
Class: COP3503 (Programming Fundamentals II)
Date: 21st March 2015

Team Project Milestone #2 (Analysis and Design)

Definition:

1) What is the system definition?
   - The Schedule Optimizer program will make an optimal academic schedule based on user preferences and class needs.

2) Why is the system important?
   - The Schedule Optimizer will save users time when deciding on their semester schedules.

Analysis:

3) What should be the inputs (as it relates to all modules)?
   i. What commands will be available?
      - Refer to UML (Unified Modeling Language) diagram
   ii. What will be the format for each data type?
      - Course object: Course name (string), course number (int), section (string array), times available (string array), priority (bool) and credits (int)
      - Preferences object: Min and max credits (int), days off (bool), max period in a row (int), max period in a day (int) and unwanted periods (string)
      - Schedule object: Course object(s)
      - Comparison function: Schedule object(s) and preference object(s)
      - Student object: Stores student's information and initial inputs

4) What should be the outputs (as it relates to all modules)?
   i. How will data be output?
      - The data will be outputted to the console as matrices (mxn), displaying possible academic schedules for the user. The "m" will be the periods (time) and the "n" will be the days i.e. Monday, Tuesday, Wednesday, etc.
   ii. Are there different ways it may potentially need to be output?
      - It can be outputted on the console or exported to a file. The schedule will have different views for the user. (Ex: Weekly Planner View and Course List View)

iii. Might the user only output a subset of the data?
- Yes, the user will be able to save data from the program depending on their needs.
- Instead of outputting all possible class schedules, the program will use an algorithm to output the best schedule based on the user preferences. The user may also request to see additional schedules that meet the specified preferences.

5) What is the flow/logic required for the proposed system?
   - The user will input the required parameters about the course information and their schedule preferences. Then a method will use these parameters to come up with all the possible combinations of courses. Once we have all the combinations, another method will compare these schedules with the preferences and delete the ones that do not match with the user's preferences. Then another method will print out the combinations that match the user's preferences. (Note: If all preferences are not able to be met, the program will display the closet match and list out the preferences that were not met.)

6) What constraints should be enforced to model the real world more accurately?
   - The user must follow a very strict input process.
   - The outputs will be constructed as clearly as possible in order to provide as little confusion as possible.
   - The input process will model a well known input processes many students are familiar with. (i.e. "What's the desired course number:" Input: "MAC2311")

7) Are we operating under any sort of assumptions?
   - The user will follow the guidelines for inputting.
     - Validation will also be implemented.
   - The user will be able to understand the schedules outputted by the program.

8) Whenever an object within the program is modified, will any corresponding changes be expected elsewhere within the system?
   - If any schedule computing algorithms are modified, then the outputs as well as the inputs will be modified to follow the changes.
   - If the data files used in the program are altered in anyway then the algorithms used to build the schedules will be altered as well to accompany the changes made.

9) How are system-level modification related to system constraints?
   - System-level modifications can be related to system constraints in a sense that modifications can terminate some initial constraints and actually benefit the algorithm.

Design:

- Ideal program division: Control/UI, Data, Algorithms

- Core functionality of a program should not be linked directly to any single UI within the system.
- Refer to UML diagram

1) How many modules are required?
    - A module will be required for each class we create because it will allow better compile-time scalability.

2) What are the classes and methods for each module/component?
    - The module "Preferences" will include the following components: min credit, max credit, max period in a row, max period in a day, unwanted period and find days off from M-F
    - The module "Course" will include the following components: course name, course number, section number, credits, priority, and times/days the user want to meet

3) What are the shared classes/methods across all modules?
    - The user input in terms of his or her preferences will be shared across all modules.

Execution Plan:

1) What is the best way to divide the coding tasks?
    - The coding tasks will be divided six ways (for the six people present). Two people will work towards completing the Control/UI, two people will work towards completing data, and two people will work towards the main algorithm. Responsibilities for each person will align with their skill set. For example, the person who is best at prompting the user for input and validating user input will be responsible for the Control/UI of the program. In addition, the tasks will be divided by time so that we are consistently progressing with the project. See the planned deadlines below.

2) What will the makefile look like?
    - If our algorithm is written into one file named "main.cpp" then a makefile can be as simple as the following:

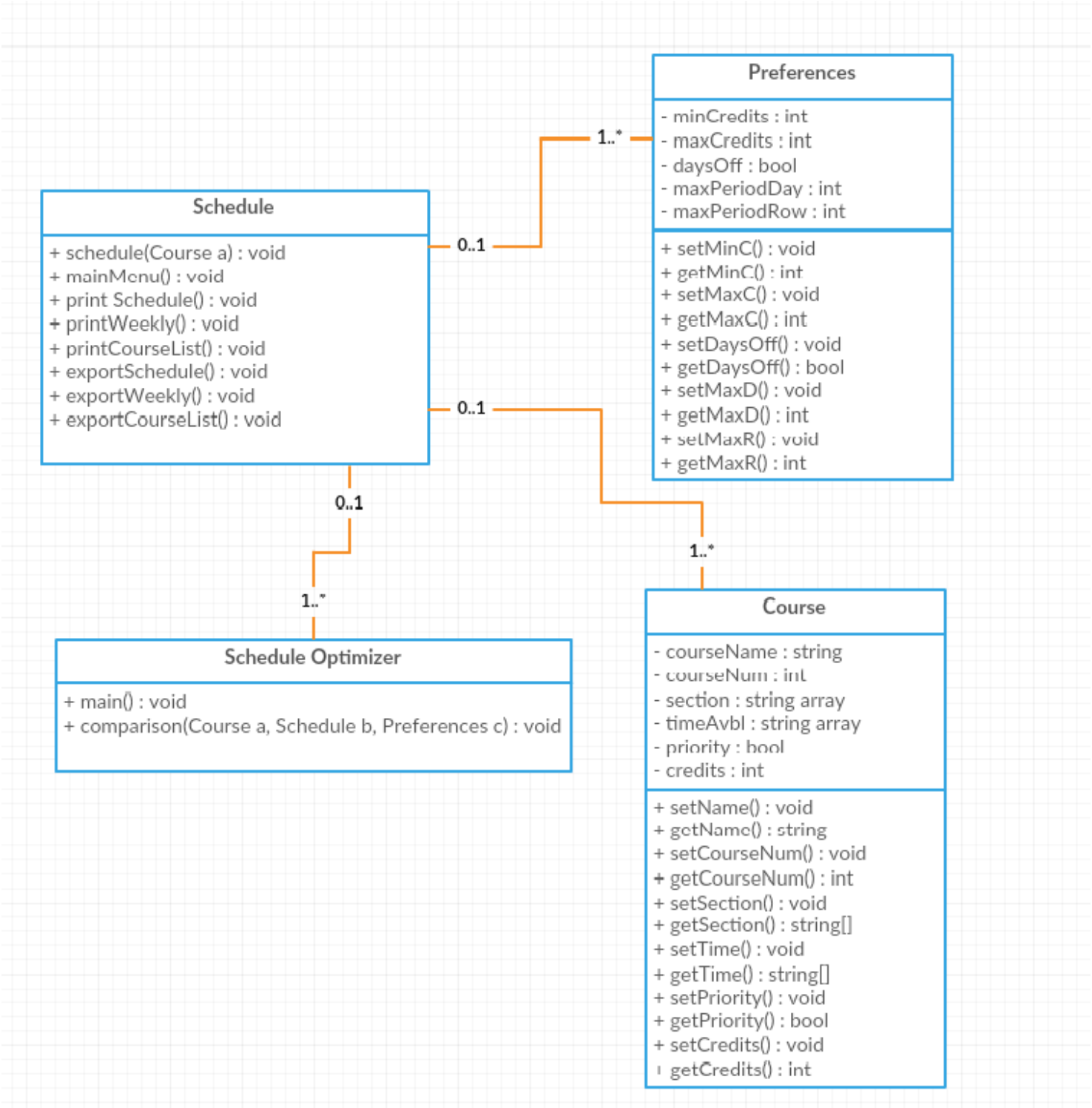
        All:

        g++ -o main.cpp classScheduleOptimizer

3) What are the planned deadlines to carry out the implementation/testing on each module?
    - On a weekly basis, our team will hold a meeting for at least two hours. During this time, we will be using the code we have been composing for that week to improve our program, and analyzing it to make sure it works as expected. In addition, we will be coming up with new ideas to enhance the algorithm or debug the algorithm. This process will continue every week until we either

have a working program that we are satisfied with or until the deadline is reached.

Deadline	Accomplishments	Comments
Feb. 19	Milestone #1: Set up team members	Create the team, in our case, it was six people
Feb. 20	Come up with topic	Class schedule optimizer
Feb. 29	Create the skeleton of the program	Skeleton of algorithm and division of work
Mar. 23	Milestone #2: Analysis and Design	Detailed plan of action
Mar. 25	Rough Draft 1 of Program	Partially-working program
Week of Mar. 28th	Rough Draft 2 of Program	Complete Algorithm
Week of April 4th	Final Draft	Complete working program
Week of April 11th	Debugging and Final Testing	Try to break code and fix any errors, make program robust
April 18th	Milestone #3: Implementation and Testing	Final submission


Preferences
- minCredits : int
- maxCredits : int
- daysOff : bool
- maxPeriodDay : int
- maxPeriodRow : int
+ setMinC() : void
+ getMinC() : int
+ setMaxC() : void
+ getMaxC() : int
+ setDaysOff() : void
+ getDaysOff() : bool
+ setMaxD() : void
+ getMaxD() : int
+ setMaxR() : void
+ getMaxR() : int
1..*
Schedule
+ schedule(Course a) : void
+ mainMenu() : void
+ print Schedule() : void
+ printWeekly() : void
+ printCourseList() : void
+ exportSchedule() : void
+ exportWeekly() : void
+ exportCourseList() : void
0..1
0..1
0..1
1..*
1..*
Schedule Optimizer
+ main() : void
+ comparison(Course a, Schedule b, Preferences c) : void
Course
- courseName : string
- courseNum : int
- section : string array
- timeAvbl : string array
- priority : bool
- credits : int
+ setName() : void
+ getName() : string
+ setCourseNum() : void
+ getCourseNum() : int
+ setSection() : void
+ getSection() : string[]
+ setTime() : void
+ getTime() : string[]
+ setPriority() : void
+ getPriority() : bool
+ setCredits() : void
+ getCredits() : int