Enrollment System Manager

Project Report

Object Oriented Programming

3460:421

Andrew Armstrong

Date Submitted: 05/4/2017

Contents

[Need 2](#_Toc481654856)

[Proposal 2](#_Toc481654857)

[Solution 3](#_Toc481654858)

[Requirements 3](#_Toc481654859)

[Class Hierarchy 3](#_Toc481654860)

[Value 3](#_Toc481654861)

[Position 4](#_Toc481654862)

[Person 4](#_Toc481654863)

[Applicant 5](#_Toc481654864)

[Member 5](#_Toc481654865)

[Employee 6](#_Toc481654866)

[Student 6](#_Toc481654867)

[Building 7](#_Toc481654868)

[Classroom 7](#_Toc481654869)

[Class 8](#_Toc481654870)

[Course 8](#_Toc481654871)

[Department 8](#_Toc481654872)

[College 9](#_Toc481654873)

[School 9](#_Toc481654874)

[Introspection 10](#_Toc481654875)

# Need

As mentioned by Dr. Andrew Sutton, there is currently a need for an improvement or replacement for many Universities’ enrollment software. The architecture of the proposed software will utilize databases containing but not limited to names of people, titles of courses, ID numbers, and other information relative to the University. Furthermore, the said databases will have hierarchal structures implemented within as a result of inheritance between structures. At a high-level overview, the software must manage the status all people officially affiliated with the University.

# Proposal

Examples of status management include creating entities for people who apply to the University and then updating full-time status, number of credits taken, number of credits in progress, credits needed for graduation, and other information considered to be relative to the person’s involvement with the University. The purpose of the proposed software is to provide a person a clear understanding of his or her current position at the University, and the steps that have to be taken to advance toward a goal.

Several different entities of people must be implemented, including students, faculty, staff, applicants, and more. An executable program will be constructed to combine data from several databases and data sources to create or update each entity. In the high level illustration below, the data source object shown as a red box remains abstract to represent the possibility of several different data sources, such as application information and surveys. Illustrated in Figure 1 below is the proposed software architecture

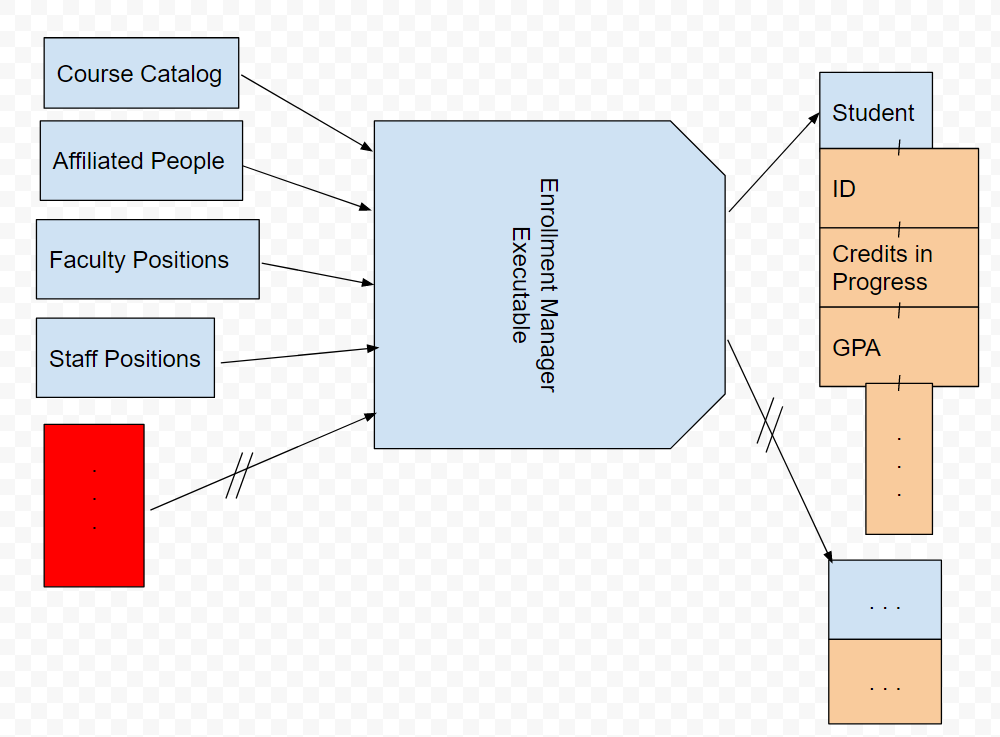


Figure 1: Software Architecture

# Solution

Problems that can be solved with the proposed enrollment system include fixing the rigidness of enrollment between different school entities, such as transient students taking courses at multiple schools. The scope of the enrollment system manager could potentially standardize enrollment by creating a universal enrollment system flexible enough to be used by all schools. Depending on the particular school, it currently may be difficult for a person to fully understand where he or she stands in terms of status at the school. That is, it may be difficult for a student to understand all of the requirements and current progress towards a degree he or she intends to earn. The proposed enrollment system will take such information into account per person and provide a clear and concise display of current standing and what needs to be done to progress toward each particular person’s intended objective at the school.

# Requirements

Dependencies for the enrollment system manager include databases containing data representing at least one member variable of the class type the database pertains to. Examples of databases with sufficient data include a Person database containing a name and social security number, a building database including the name of all buildings on a campus, a classroom database containing all rooms that classes can be taught in inside a given building, a College database containing the required courses, a Department database containing at least employees, and so on. Such databases can be simple text files with entries separated by line breaks or other formatting that can be interpreted and utilized in the parsers to be designed for each class. An example of a student database is shown in figure [@@]. The format of the student database in figure 2 is: name, id, number.

C:\Users\Armstrong\AppData\Local\Microsoft\Windows\INetCache\Content.Word\students_db.png

Figure 2: Student Database Snippet

# Class Hierarchy

The enrollment manager software contains several classes that contain member variables from various databases. Each class represents information that can be obtained through parsing the database relative to the class. Classes and databases that have been considered for implementation so far include Value, Person, Applicant, Member, Employee, Student, Building, Classroom, Class, Course, Department, College, and School. All information contained in these classes and data retrieved from databases must be necessary information for a datatype at a given University. Any extra information in the databases should be considered unnecessary and ignored.

## Value

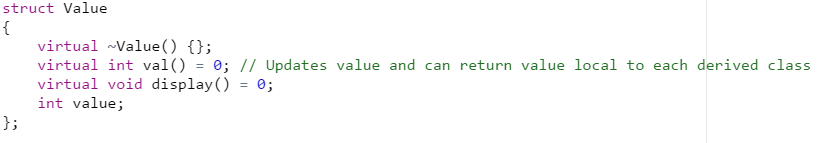
All of the classes defined in the enrollment manager are derived from the Value class. The purpose of the Value class is to provide a dynamic structure for derived classes, which must contain at least the fundamental object value for comparison between objects of the enrollment manager software. Due to classes within the enrollment manager being user-defined, no default weight value exists for comparison between objects. Providing a purely virtual function for assigning an object’s weight value requires the derived classes to dynamically override and define the behavior of the weight assignment within their definitions. For visual convenience, each derived class also overrides a purely virtual display function for printing out member variables. Each derived class also has a destructor override to ensure memory is freed properly after dynamic allocation. Shown in Figure 3 below is a code snippet illustrating the Value class. 

Figure 3: Value Class

## Position

Positions to the enrollment manager are objects containing identities including a Boolean variable determining whether or not the position is a faculty position, a string variable containing the position title, and a department object pertaining to which department the position is a part of. Positions can exist and never be referred to, such as open positions that not a single person applies to, hence the class isolation from the Member and Employee classes. Shown in Figure 4 is a code snippet of the Position class.

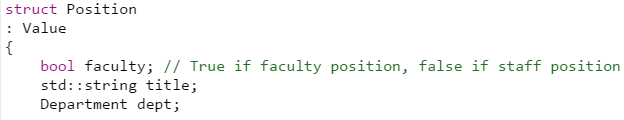


Figure 4: Position Class

## Person

The Person class essentially serves as a base class to other classes and contains member variables for holding the basic identity of a person, consisting of a name and social security number in the enrollment manager software. Creation of this class results with less member variables explicit to classes containing people identity data and thus reducing member variable assignments of such classes, provided a person object exists. A person database can be used to parse already person objects from already existing data. Shown in Figure 5 is an example Person database and Shown in Figure 6 is a code snippet illustrating the Person class’s inheritance and member variables.

C:\Users\Armstrong\AppData\Local\Microsoft\Windows\INetCache\Content.Word\People_DB.PNG

Figure 5: Person Database

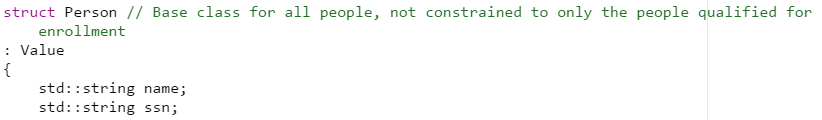


Figure 6: Person Class

## Applicant

The Applicant class is a derived class of the Person class because an applicant to the enrollment manager software is a person who has submitted an application for enrollment at the college or university. Information about applicants considered to be important for the enrollment manager is whether or not the applicant’s application has been accepted or not yet accepted. Representing the status of an application is a Boolean variable. Applications that have been denied are equivalent to applications that are pending approval to the enrollment manager, thus removing the need for a variable that is not binary. Shown in Figure 7 is a code snippet illustrating the Applicant class’s inheritance and member variable.

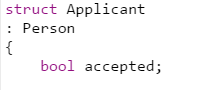


Figure 7: Applicant Class

## Member

The Member class inherits the Person class because a member to the enrollment manager software is a person who is registered at the college or university. A member is not necessarily an employee or a student, hence the implementation of the Member class. One example of a person existing in the enrollment manager explicitly as a Member object is a registered guest at the college or university. Guests at the University of Akron are given a UANET ID and ID number and may or may not be applicants or hold a position at the university. The main behavior of the Member class is to serve as a subclass for students and employees. Shown in Figure 8 is a code snippet illustrating the Member class’s inheritance and member variables.

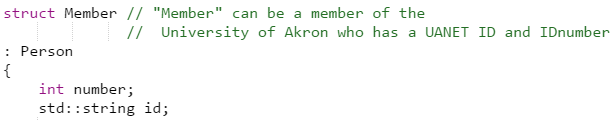


Figure 8: Member Class

## Employee

Employees to the enrollment manager software are members who are employed at the college or university and hold a position or multiple positions at a specified pay rate. To represent employment, the enrollment generates a pair consisting of a position object and a double variable representing the payment the member receives for working the specified position. Payment is specified in the Employee class rather than the Position class because payment is not fixed in terms of position and can vary between members. For example: an IT technician who has just been hired makes less than his or her coworker who holds the same exact position because the said coworker received a raise after a year of working. Due to the possibility of a member being employed at multiple positions, the enrollment manager utilizes a vector of the previously mentioned pair to represent the various paid positions. Shown in Figure 9 is a code snippet of the Employee class’s inheritance and member variables. Refer to Figure 11 for an example of a student who is also an employee with a paid position to the enrollment manager.

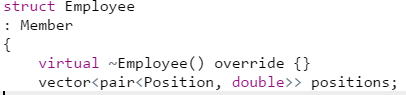


Figure 9: Employee Class

## Student

A student to the enrollment manager is a person who is both an applicant and a member of the college or university. To become a student, a person must apply to specific college or university in question and have his or her application accepted. Once an applicant is accepted, he or she becomes a member of the college or university. An applicant with an accepted application is not required to become a student, the applicant has the option to deny enrollment at the college or university. The Student class therefore inherits both the Member class and the Applicant class, which introduces a fork in the assignment of member variables pertaining to the different subclasses. Students can be enrolled in courses and classes and also have a grade point average. In the enrollment manager, classes and courses are represented through the use of vectors. Existing as a student does not imply that the student is taking classes and/or courses. Such a case is able to be efficiently represented through vector functions such as std::vector::size(). Shown in Figure 10 is a code snippet of the Student class and shown in Figure 11 is an example of assigning member variables of the Student class and a representation of how to create a Student object who is also an Employee object.

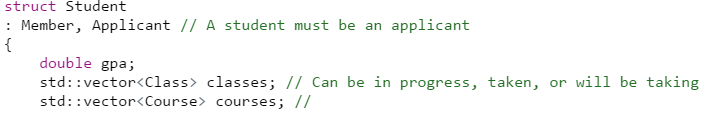


Figure 10: Student Class

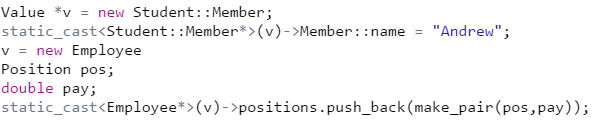


Figure 11: Student Who is Also an Employee

## Building

Most colleges and universities have a physical location rather than a purely internet or software-based location and thus contain building entities for operations. A Building object to the enrollment manager is an existing physical building which has an identity containing at least a name and a build name or address residing on the college or university’s campus. Internet or software-based locations in the enrollment manager can still be created using the Building class and should be done so in a manner such that the member variables’ contents clearly reflect the absence of a physical location. Other than representing what is on a campus, the Object class’s main purpose is to serve as a subclass for the Classroom class. Shown in Figure 12 is a code snippet of the Building class.

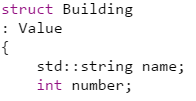


Figure 12: Building Class

## Classroom

Classrooms to the enrollment manager are physical or internet or software-based locations contained within a Building object. Classrooms cannot exist without buildings; therefore, the Classroom class inherits the Building class. The Classroom class is implemented rather than representing a container of classrooms within the Building class because not all rooms within a building are classrooms, and not all classrooms remain as classrooms. Implementing the Classroom class allows for efficient allocation and deletion of Classroom objects compared to modifying and updating member variables of the Building Class. A classroom is identified through a room number. Shown in Figure 13 is a code snippet of the Classroom class.

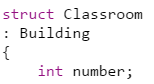


Figure 13: Classroom Class

## Class

Outside of the enrollment manager, classes and courses may seem similar and it may be difficult to distinguish characteristics between the two. However, a class can be broken down to an entity that contains a number representing the identity of the class and a size representing the number of students that can be allowed to attend. In the enrollment manager, Class objects contain integers representing the mentioned identity and available size. Classes in the enrollment manager do not have any dependencies and only inherit the base Value class. Rather than using inheritance, Class objects and Course objects are contained within other classes as member variables. Shown in Figure 14 is a code snippet of the Class class.

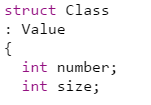


Figure 14: Class Class

## Course

As mentioned under the Class section, a sufficient amount of distinct features exist between courses and classes such that the implementation of a Course class separate from a Class class proves to be convenient. Courses contain a number of classes worth a specified number of credits. Identities of courses consist of the course title and course number. Course objects are heavily used within the enrollment manager, thus resulting with the heavy usage of Class objects due to the inclusion of Class objects within the Course class. Shown in Figure 15 is a code snippet of the Course class.

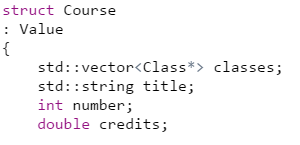


Figure 15: Course Class

## Department

Departments are abstract organization entities within a college or university, composed of several academic and industrial components. To the enrollment manager, Departments consist of an identity including both a string for the department name and an integer for the department number. One example of an actual department is the department of Distributed Technician Services (DTS) at the University of Akron. Within the DTS department are employees and available positions. Departments exist as separate entities and there are no classes that are departments other than the Department class itself, therefore departments are treated purely as objects rather than inheriting other subclasses or being inherited. Shown in Figure 16 is a code snippet of the Department class.

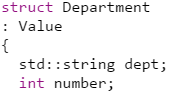


Figure 16: Department Class

## College

Colleges are entities within schools that pertain to a specific area of study, such as the college of engineering at the University of Akron. Students are admitted into a college based on their major and have at least two sets of required courses. One set of required courses comes from the college and typically includes general education courses, whereas the second set of required courses comes from the major and/or minor which typically includes core classes directly related to the area of intended study. To the enrollment manager, colleges consist of a vector of course objects to represent the required courses based on college and a vector of departments. Departments are objects contained within colleges and therefore should not be inherited. The College class is a high level class such that it contains most of the data crucial to the scope of the enrollment manager and includes composite objects. Most colleges are also assigned a building, which can be seen in Figure 17.

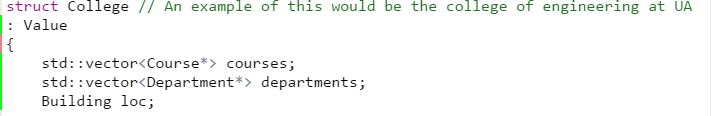


Figure 17: College Class

## School

A school is the highest level object in the enrollment system, such that all of the implemented classes are utilized within the School class. One example of a School object is the University of Akron. To the enrollment manager, schools contain an identity consisting of a string for the name of the school and a vector of colleges. Essentially, the School class is implemented to localize enrollment and provide separation between different school entities, allowing special cases such as transient students to be easily maintainable. Shown in Figure 18 is the School class.

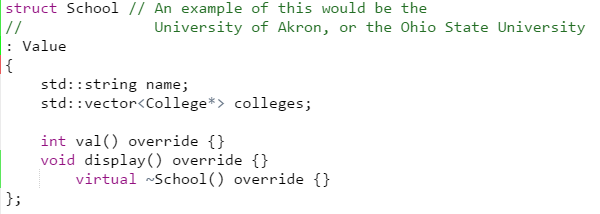


Figure 18: School Class

## Conclusion

The current version of the enrollment manager is far from complete and requires much more additional work on each class. Department objects should be further implemented to include available position, Class objects should be further implemented to include specific timeframes of which classes are attended, and Student objects should be further implemented to include features such as class schedules, recommended courses relative to the student’s major and the available electives. More classes such as a Major class and a Minor class also need to be implemented in order to achieve the goal of this system. The method of retrieving data from locally acquired CSV text documents is also impractical for the total scope of the system and should be replaced with connections to secure databases, application forms, and other internet-based software.

Overall, the enrollment manager utilizes object oriented programming such that the system specializes behavior and data of objects. Resulting from this object oriented approach is code that can be considered aesthetically pleasing and the ability to read and understand the code facilitated, provided the reader has a strong understanding of C++. Source code for the enrollment manager system can be compiled using g++ 6.3.0. Source code for the enrollment manager system can be found on GitHub via the following link: <https://github.com/ArmstrongGAndrew/enrollment_system>