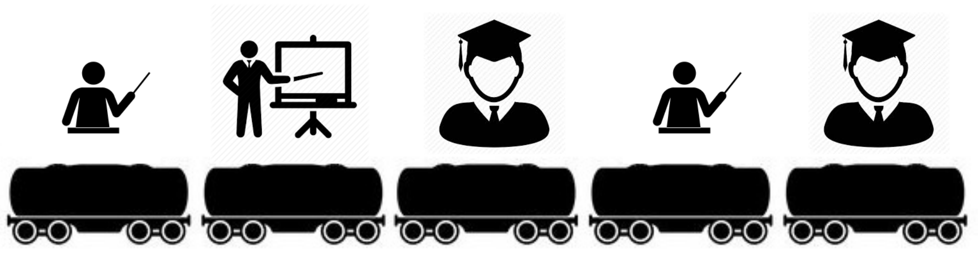
Project3:
Roster “is-a” List of Person pointers -> Polymorphism



1. Modify the `Person`, `Student`, `TeachingAssistant` and `Instructor` classes to support **Polymorphism**.

2. Modify Roster to inherit from List and store pointers to `Person` in a List, where the pointers will actually point to either `Student`, `TeachingAssistant` or `Instructor` objects.

Implementation - 2 parts:

Part1: Modify Person and derived classes

Modify the `Person` class by adding a **pure** virtual function `display()` with void return type. Each derived class can then override `display()` to display data specific to the derived object as follows:

- `Student::display()`
  Sample output:
  “***`first_name_ last_name_`*** `majors in` ***`major_`*** `with gpa:` ***`gpa_\n`***”

Space, not new line

- `TeachingAssistant::display()`
  Sample output:
  "***first_name_*** ***last_name_*** majors in ***major_*** with gpa: ***gpa_*** working [part-time/full-time] as a ***ta_role_*** *\n*"

  Outputs `part-time` if ***hours_per_week_*** < 8, otherwise outputs `full-time`

Space, not new line

- `Instructor::display()`
  Sample output:
  "***first_name_*** ***last_name_*** - office: 1000C, email: 235Instructors@hunter.cuny.edu\n"

  All instructors will have same office and contact.

Note: The formatting must match **EXACTLY** (other than for random data described next), please pay special attention to punctuation and capitalization.

Part2: Modify the Roster class

Modify the Roster class to **inherit from List** and store **pointers to Person** (You will find all necessary files on Blackboard under Course Materials/Project3 Files).Roster must have at least (but is not limited to) the following 3 methods:

1.
```
Roster(); //default constructor for empty roster
```

2.
```
/**parameterized constructor
   @pre the input file is expected to be in csv
      (comma separated value) where each line has format:
      "id,first_name_last_name,title\n"
   @param input_file_name the name of  the input csv file
   @post Pointers to Person are stored in the Roster, each pointer
         pointing to either a Student, Instructor or TeachingAssistant
         object as per the data in the input file and specified by the
         title field
   **/
  Roster(std::string input_file_name);
```

Note that there is a lot going on here. For each line in the input file it will:

- Instantiate an object of type `Student`, `TeachingAssistant` or `Instructor`, as indicated by the *title* (the last field on each line), with possible values in {student, teaching_assistant, instructor}

- **Randomly generate data** specific to the object that is not provided in the input file (`gpa_, major_, hours_per_week_, ta_role),` while all Instructor objects will have same email (*235Instructors@hunter.cuny.edu*) and same office (*1000C*).
- Add a pointer of type `Person` that points to the newly instantiate object to the `Roster`

3.
```
/**@post displays all data in roster, one per line
         as per each object's specific display method\n"
  **/
 void display();
```

You may have as many additional helper methods (public or private) as you see fit.

Hint: You may have helper functions that generate the random values needed. For example:

```
/**
 @return a number randomly sampled from
 {4.0, 3.75, 3.5, 3.25, 3.0, 2.75, 2.5, 2.25, 2.0}
 */
double randGpa();

/**
 @return a string randomly sampled from
 {"Computer Science", "Literature", "Music", "Philosophy", "Physics",
"Theatre", "Computational Biology", "Mathematics", "Geography",
"Linguistics"}
 */
std::string randMajor();

/**
 @return a ta_role randomly sampled from
 {LAB_ASSISTANT, LECTURE_ASSISTANT,FULL_ASSISTANT}
 */
ta_role randRole();
```

To implement the above functions that do random sampling, you can simply create an array or vector that contains the desired values as indicated in the comments, then generate a random number between 0 and the size of the array/vector - 1 and return the item at that random location.

For example:
```
     return gpa_list[rand() % SIZE];
```
Where `SIZE` is the size of the array/vector holding the data to you wish to sample.

You may want to add a time-stamped seed before calling `rand()` to generate non-repeating pseudo random numbers:

```
#include <ctime>
srand(static_cast<unsigned>(time(0)));
```

Testing:

In `main()` (not for submission) first test your modifications from Part 1. Instantiate objects of type Student, TeachingAssistant and Instructor and make calls to each object's display() to check that data is displayed correctly.

When Part 1 is implemented and tested, then move on to Part 2, instantiate in main() a Roster object, and call dsplay() on it. Make sure that random data is being correctly generated, and that Polymorphism is correctly implemented (each different object type is correctly calling its own version of display()).

Grading Rubric:

- **Correctness 80%** (distributed across unit testing of your submission)
- **Documentation 10%**
- **Style and Design 10%** (proper naming, modularity and organization)

Notes:

- I reserve the right to detract points given by Gradescope if your submission does not comply in some way with this specification.
- A submission that implements all required classes and functions but does not compile will receive 40 points total (including documentation and design).

Submission:

Submit the following files (**10 files**): **Person.hpp, Person.cpp, Student.hpp, Student.cpp, TeachingAssistant.hpp, TeachingAssistant.cpp, Instructor.hpp, Instructor.cpp, Roster.hpp, Roster.cpp**

Your project must be submitted on Gradescope.

Although Gradescope allows multiple submissions, it is not a platform for testing and/or debugging and it should not be used for that. You MUST test and debug your program locally.
Before submitting to Gradescope you MUST ensure that your program compiles (with g++) and runs correctly on the Linux machines in the labs at Hunter (see detailed instructions on how to upload, compile and run your files in the "Programming Rules" document). That is your baseline, if it runs correctly there it will run correctly on Gradescope and if it does not, you will have the necessary feedback (compiler error messages, debugger or program output) to guide you in debugging, which you don't have through Gradescope.
"But it ran on my machine" is not a valid excuse to get credit.
Once you have done all the above you submit it to Gradescope.

The due date is Thursday June 20 by 6pm. No late submissions will be accepted.

Have Fun!!!!!