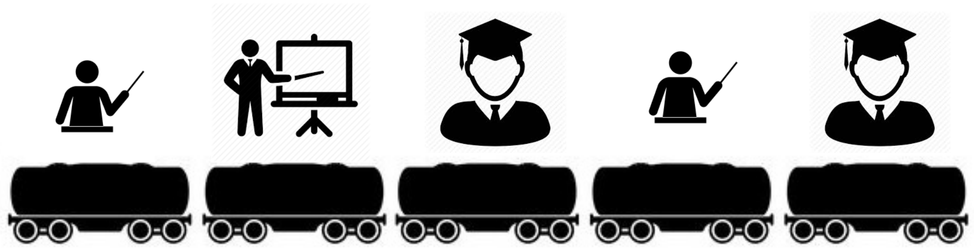
Project 2C:
Another List of CourseMembers -> Polymorphism



1. Modify the `CourseMember`, `Student`, `TeachingAssistant` and `Instructor` classes to support **Polymorphism**. (This part is almost trivial if you understand the basics of Polymorphism)

2. Store pointers to `CourseMember` in a List, where the pointers will actually point to either `Student`, `TeachingAssistant` or `Instructor` objects.

Implementation - 2 parts:

1. Modify CourseMember and derived classes

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

- `Student::displayMember()`
  Sample output:
  `first_name_` ***`last_name_`*** `majors in` ***`major_`*** `with gpa:` ***`gpa_`***

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

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

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

Space, not new line

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.

Also modify the `ta_role` in `TeachingAssistant` as follows:
```
enum ta_role {LAB_ASSISTANT, LECTURE_ASSISTANT, FULL_ASSISTANT};
```

2. Read data from file and add object pointers to List

Write the following helper functions in `polytest` files: **`polytest.hpp`** will contain the function definition and **`polytest.cpp`** will contain the function implementation. **Note** that this is not a class but just auxiliary functions for the project.

Together these functions will do the following:
- Read data from an input file with format like roster.csv: `id,first_name,last_name,title`
- For each line in the input file, instantiate an object of type `Student`, `TeachingAssistant` or `Instructor`, as indicated by the title (the last field on each line).
- Randomly generate data specific to an object that is not provided int 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).
- **Append** each of these objects via pointer to a `List<CourseMember*>`

The breakdown of these functions is as follows:

```
/**
 @param cm_list the list to be populated with pointers to CourseMember
 @param input_file the file containing data used to generate CourseMember-
derived objects to add to cm_list
 @post reads parameters from input_file to call addMemberToList()
 */
void populateCmList(List<CourseMember*>& cm_list, std::string input_file);
```

Adapt this function from Projects1C. If you successfully did this in Project1C you can skip to the next page. If you are still struggling with reading input from a file here is some pseudocode:

```
Include fstream
Instantiate an ifstream object (say in_stream) and open it with
input_file
If in_stream fail
     output an error message that says something like "cannot
     read from input_file" // it is helpful to output the name
     of the actual file but not necessary
Else{
     while it is not in_stream end of file{
          getline of in_stream up to a ',' character into an int
          id variable by doing some string-to-int conversion

          getline of in_stream up to a ',' character into a
          string first_name variable

          getline of in_stream up to a ',' character into  a
          string last_name variable

          getline of in_stream up to a '\n' character into a
          string title variable

          If it is not in_stream end of file
               Trim the \n character off the end of title
               (string::pop_back())

          call addMemberToList() with the parameters just read
          from the file
     }//end while
}//end else
close in_stream
```

To find out about the `ifstream` methods `open()`, `fail()`, `eof()` and `close()` you can take a look at the `ifstream` class documentation:
http://www.cplusplus.com/reference/fstream/ifstream/

```
/**
 @post instantiates a new object - Student, TeachingAssistant or Instructor -
as indicated by the title parameter, randomly generate relevant data not
provided by parameters (e.g. major_, gpa_ etc.) and append a pointer to
cm_list that points to the newly instantiated object
 */
void addMemberToList(List<CourseMember*>& cm_list, int id, const std::string&
first_name, const std::string& last_name, const std::string& title);
```

Note: First to access derived-class specific members you must point to the object via a pointer of type derived-class. Then point a `CourseMember*` to add it to `cm_list`
For example:

```
     Student* s_ptr = new Student(id, first_name, last_name);
     //select random gpa and major
     s_ptr->setGpa(randGpa());
     s_ptr->setMajor(randMajor());
     CourseMember* c_ptr = s_ptr;
     cm_list.insert(cm_list.getLength(), c_ptr);
     s_ptr = nullptr;
```

```
/**
 @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.

Testing:

In `main()`, test that you are reading the data into the List correctly by printing the entire list.
Then make calls to `displayMember()` to test that you are doing late binding and properly calling the correct method for each derived class.

For example:

```
populateCmList(cm_list, "roster.csv");

cm_list.getItem(x)->displayMember();
```

with different values of x to test at least:
one call to `Student::displayMember()`,
one call to `TeachingAssistant::displayMember()`
and one call to `Instructor::displayMember()`

Project Notes: The List class will issue some **warnings** for `getItem()`, because an uninitialized item is returned if the function is called with an invalid `position`. This is so because we have not discussed error handling yet. We will be able to fix this later in the semester.

Submission:

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

Your project must be submitted on Gradescope. The due date is Friday March 15 by 6pm. No late submissions will be accepted.

Have Fun!!!!!