Final Project – Object Oriented Programming

Due Date: June 25th 11:59pm, 2020

Team Name: King Size Yakult

Team Members: 曾柏硯 (Jacky), 余世暘 (Roy), 洪翌鈞 (Dennis), 蕭妤珊 (Kasa)

Topic & Introduction

NTUCE features an **advisor preselection system**, which annually assigns a professor as an advisor for each student. It is claimed that the assignment system operates according to each student's first three priorities, and also some specific criteria if some professors provide ones.

However, when it comes to the result, we heard that some student have not been assigned to their beloved professors for years, doubting if the system is reasonable.

Therefore, we decide to write a program to accomplish this distribution process, since as long as the rule is set up, the computer is always **FAIR**.

Similar approaches and applications

The only reference in this project is *Study of the Distribution Program for the Joint Entrance Examination of Universities and Related problems*. In this article, the author introduced how to define a "steady" assignment, and how will the result differ by the priority of criteria, that is to say, different views of assignment strategy.

Since the context in this project is quite different from that in *Joint Entrance Examination of Universities* (for example, in an exam there are scores for assignment), we did not totally mimic the logic in the article, and developed our system according to the definition of a "steady" assignment mentioned in the article.

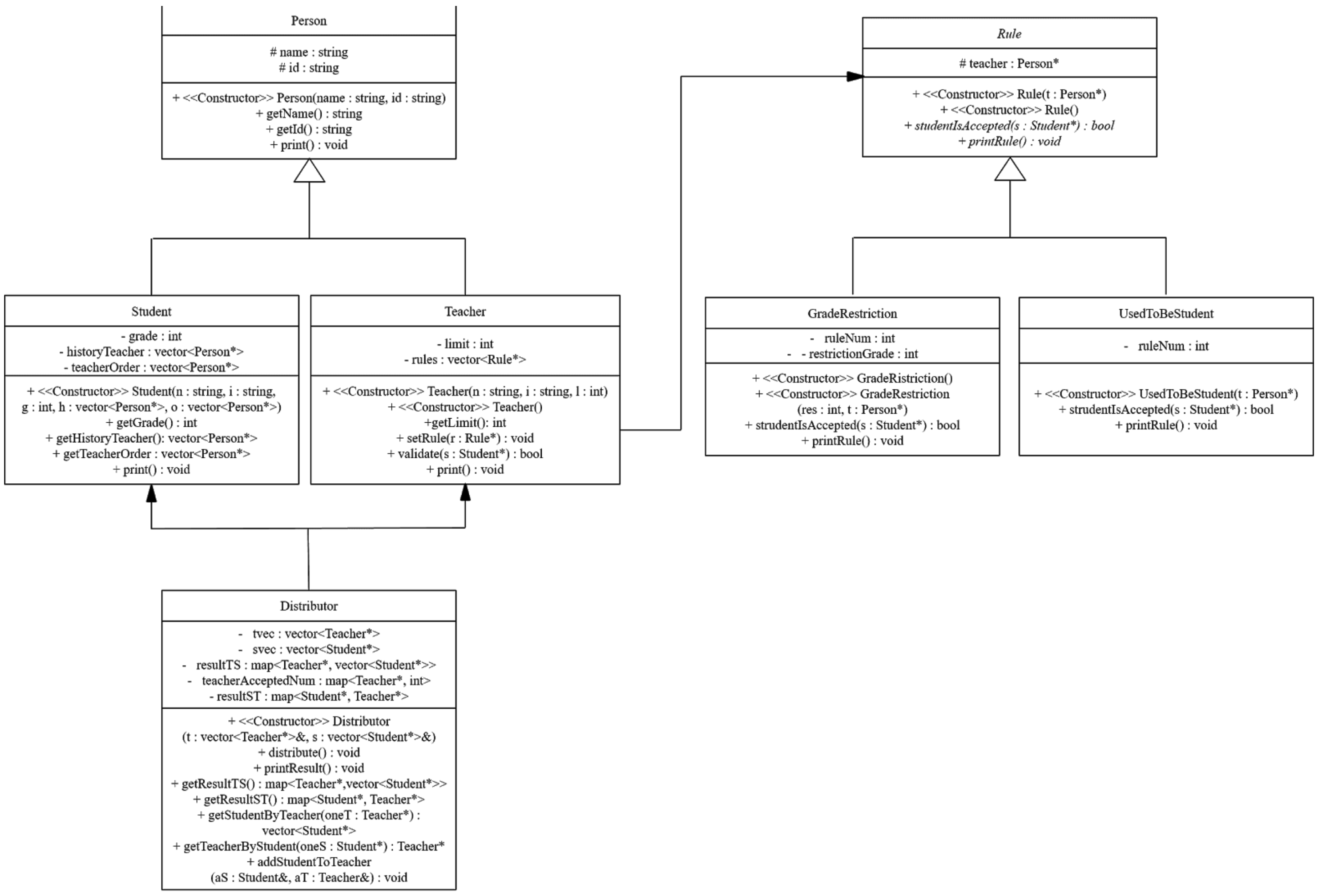
Assumption & Goals

Assumption : There are two kinds of "Person", "Teacher" and "Student". The teacher can decide how many students they want to accept. The total quota of students the teachers will accept is more than the true number of students. Besides, the teachers can set up what kind of "Rule"s they want. If the student doesn't meet the rules, they won't be accepted by the teacher. Each student can choose 3 teachers as candidates, and each student have a property of the "history teacher", that is , the teacher taught them before. This may be one of the guidelines the teacher decides whether the student is accepted. Briefly saying, the students can choose at most 3 teachers actively, while the teacher can choose the students passively by setting up the rules.

<u>Goals</u> : Using the gaming rule just mentioned, the “Distributor” will distribute the students to all the teachers **FAIRLY**. The result may be a table showing the final relationship between the teachers and the students.

Modeling processes

UML


Person
# name : string
# id : string
+ <<Constructor>> Person(name : string, id : string)
+ getName() : string
+ getId() : string
+ print() : void
Rule
# teacher : Person*
+ <<Constructor>> Rule(t : Person*)
+ <<Constructor>> Rule()
+ studentIsAccepted(s : Student*) : bool
+ printRule() : void
Student
- grade : int
- historyTeacher : vector<Person*>
- teacherOrder : vector<Person*>
+ <<Constructor>> Student(n : string, i : string, g : int, h : vector<Person*>, o : vector<Person*>)
+ getGrade() : int
+ getHistoryTeacher(): vector<Person*>
+ getTeacherOrder : vector<Person*>
+ print() : void
Teacher
- limit : int
- rules : vector<Rule*>
+ <<Constructor>> Teacher(n : string, i : string, l : int)
+ <<Constructor>> Teacher()
+getLimit(): int
+ setRule(r : Rule*) : void
+ validate(s : Student*) : bool
+ print() : void
GradeRestriction
- ruleNum : int
- - restrictionGrade : int
+ <<Constructor>> GradeRistriction()
+ <<Constructor>> GradeRestriction (res : int, t : Person*)
+ strudentIsAccepted(s : Student*) : bool
+ printRule() : void
UsedToBeStudent
- ruleNum : int
+ <<Constructor>> UsedToBeStudent(t : Person*)
+ strudentIsAccepted(s : Student*) : bool
+ printRule() : void
Distributor
- tvec : vector<Teacher*>
- svec : vector<Student*>
- resultTS : map<Teacher*, vector<Student*>>
- teacherAcceptedNum : map<Teacher*, int>
- resultST : map<Student*, Teacher*>
+ <<Constructor>> Distributor (t : vector<Teacher*>&, s : vector<Student*>&)
+ distribute() : void
+ printResult() : void
+ getResultTS() : map<Teacher*,vector<Student*>>
+ getResultST() : map<Student*, Teacher*>
+ getStudentByTeacher(oneT : Teacher*) : vector<Student*>
+ getTeacherByStudent(oneS : Student*) : Teacher*
+ addStudentToTeacher (aS : Student&, aT : Teacher&) : void

Person

```
class Person {
    protected:
        string name; //Person's name
        string id; //Person's ID
    public:
        Person(const string name, const string id); //Person class constructor
        string getName(); //get Person's name
        string getId(); //get Person's ID
        virtual void print(); //print Person's information
}
```

In “Person” class there two private members which are name, a string storing person’s name; and id, a string storing person’s id.

In public part, there is a constructor, two members functions getting each private properties, and a “print()” member function which prints out person’s information and needed to be overridden.

Students

```
class Student : public Person {
    private:
        int grade; //Student's grade
        vector<Person*> historyTeacher; //Student's history advisor
        vector<Person*> teacherOrder; //Student's advisor order

    public:
        Student(const string n, const string i, const int grade, const vector<Person*> h,
        const vector<Person*> o); //Student class constructor
        int getGrade(); // get Student's grade
        vector<Person*> getHistoryTeacher(); //get history advisor
        vector<Person*> teacherOrder(); //get advisor order
        void print()const override; //print Student's information
        ~Student(); //Student class destructor
}
```

In “Student” class, there are three private members, which are grade, an integer storing student’s grade; historyTeacher, a vector of “Person” objects’ pointer storing student’s history advisor; and teacherOrder, a vector of “Person” objects’ pointer storing student’s advisor order.

And then move on to the public part. First, there are constructor and destructor. Then there are three member functions which each can get private properties. Finally, the “print()” member function is for printing out the information of the “Student” class.

Teacher

```
class Teacher : public Person {
    private:
        int limit; // limit of the number of students
        std::vector<Rule*> rules; // rules of accepting students
    public:
        Teacher(); // default constructor
        Teacher(const std::string n, const std::string i, const int l);
        // constructor(name, id, limit)
        void print() const override; // print Teacher's information
        void setRule(Rule* r); // set rules
        bool validate(Student* s);
        // validate whether the Student is accepted by the Teacher
        int getLimit();
        // get the limit of the number of students
         ~Teacher() = default; // default destructor
};
```

In “Teacher” class, there are two private members. The first is “limit”, meaning the limit of number of students the teacher want to accept. The second is "rules", which is the vector, storing the rules of accepting students the teacher want to set up.

There are also seven member functions. “Teacher()” is the default constructor. “Teacher(n, i, l)” is the constructor needing “name”, “id”, “limit” as the inputs. “print()” is the function to print Teacher’s information. “setRule()” is the function for Teacher to set up the rules for accepting students and store these rules in the vector “rules”. “validate(s)” is the function taking Student object as the input, which validate whether the input Student is qualified to be the Teacher’s student. “getLimit()” is the function to get the “limit” member. “~Teacher()” is the default destructor.

Rule

```
class Rule {
    protected:
        Person* teacher = nullptr; // the teacher who owns the rule
    public:
        Rule() = default; // default constructor
        Rule(Person* t) {teacher = t;} // constructor(person)
        virtual bool studentIsAccepted(Student* s) = 0;
        // whether the student is accepted by this rule
        virtual void printRule() = 0; // print the information of this rule
};
```

In “Rule” class, there is a protected member “teacher”, which stores the teacher who owns this rule.

There are four public member functions. “Rule()” and “Rule(t)” are constructors, while “Rule(t)” takes a Person “t” as the input. “studentIsAccepted(s)” is the function that tests whether the student is accepted by this rule. “printRule()” print the information of the rule.

Grade Restriction Rule

```
class GradeRestriction : public Rule{
    private:
        int ruleNum = 1; // Rule derivatives numbering
        int restrictionGrade = 0; // The grade restriction
    public:
        GradeRestriction() = default; // default constructor
        GradeRestriction(int res, Person* t);
        // constructor(restrictionGrade, person)
        bool studentIsAccepted(Student* s) override;
        // whether the student is accepted by this rule
        void printRule() override; // print the information of this rule
};
```

In the “GradeRestriction” class, which inherits the class “Rule”, there are two private members. “ruleNum” is the numbering of the derivatives of the “Rule” class. “restrictionGrade” is the restriction grade for the teacher. For example, if the “restrictionGrade” is “3”, this means the teacher only accepts the students who are older than 3rd grade.

There are four public member functions. “GradeRestriction()” and “GradeRestriction(res, t)” are both constructors, while “GradeRestriction(res, t)” takes an integer “res” and a Person “t” as input arguments. “studentIsAccepted(s)” and “printRule()” are functions that override the base class, which is the “Rule” class.

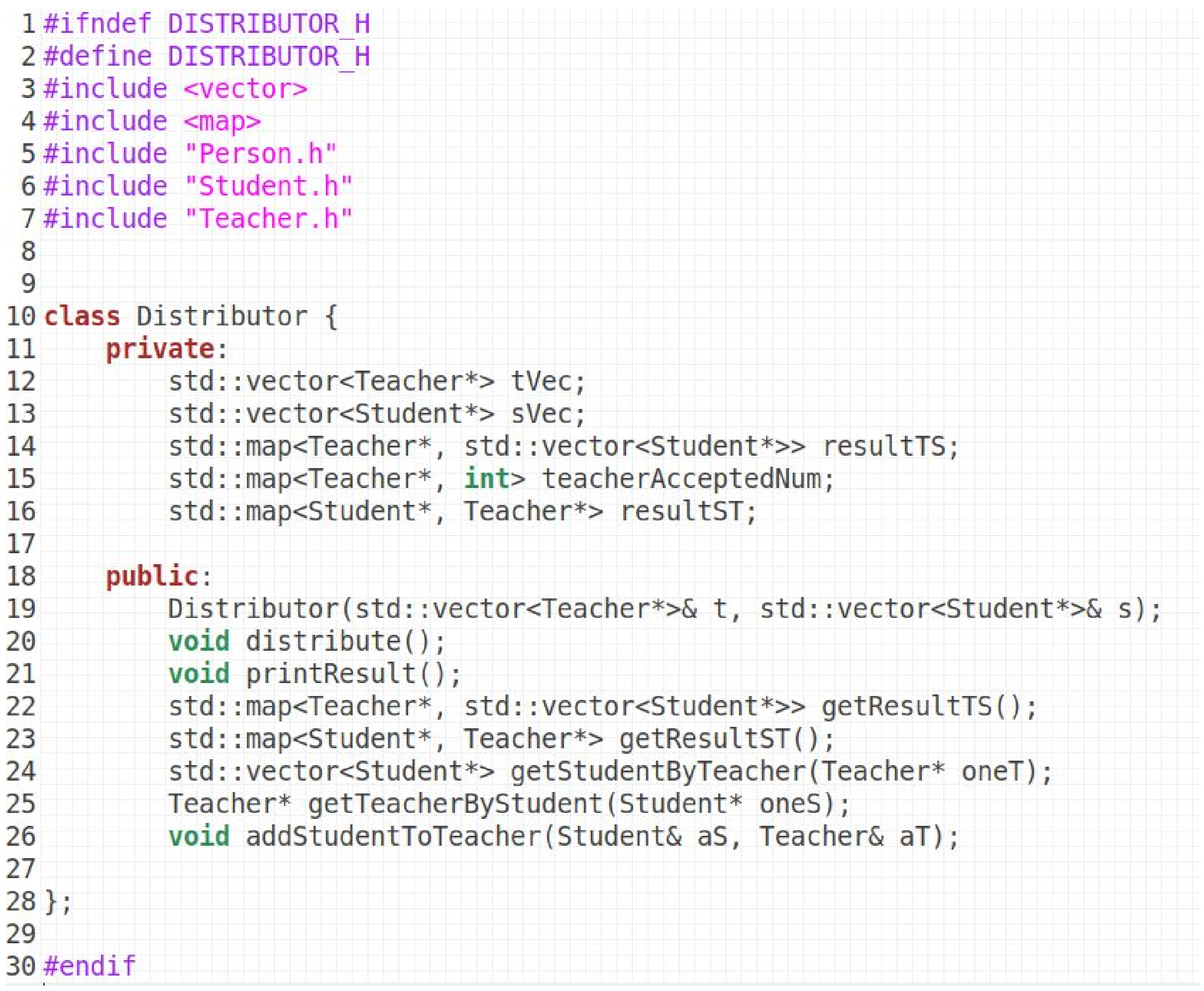
Used To Be Student Rule

```
class UsedToBeStudent : public Rule {
    private:
        int ruleNum = 2; // Rule derivatives numbering
    public:
        UsedToBeStudent(Person* t); // constructor(t)
        virtual bool studentIsAccepted(Student* s);
        // whether the student is accepted by this rule
        virtual void printRule(); // print the information of this rule
};
```

In the “UsedToBeStudent” class, there is only one private member, “ruleNum”, which is the numbering of the derivatives of the “Rule” class.

There are three public member functions. “UsedToBeStudent(t)” is the constructor take a Person “t” as an input argument. “studentIsAccepted(s)” and “printRule()” are functions that override the base class, which is the “Rule” class.

Distribution Method



```cpp
#ifndef DISTRIBUTOR_H
#define DISTRIBUTOR_H
#include <vector>
#include <map>
#include "Person.h"
#include "Student.h"
#include "Teacher.h"


class Distributor {
    private:
        std::vector<Teacher*> tVec;
        std::vector<Student*> sVec;
        std::map<Teacher*, std::vector<Student*>> resultTS;
        std::map<Teacher*, int> teacherAcceptedNum;
        std::map<Student*, Teacher*> resultST;

    public:
        Distributor(std::vector<Teacher*>& t, std::vector<Student*>& s);
        void distribute();
        void printResult();
        std::map<Teacher*, std::vector<Student*>> getResultTS();
        std::map<Student*, Teacher*> getResultST();
        std::vector<Student*> getStudentByTeacher(Teacher* oneT);
        Teacher* getTeacherByStudent(Student* oneS);
        void addStudentToTeacher(Student& aS, Teacher& aT);

};

#endif
```

The header file shows a pattern of how the class functions. It is similar to the Abstract Factory Pattern. The algorithm first assigns students by their teacherOrder in class “Student”, and then the result of validate() in the class “Teacher.” If in the first order, too many students choose a single teacher, the system randomly erases several ones. After the 3 orders are done, we randomly get a student for a teacher, and check the result of validate(). Finally, in the last step, we randomly assign the students remained. The program ends whenever all the students are assigned.

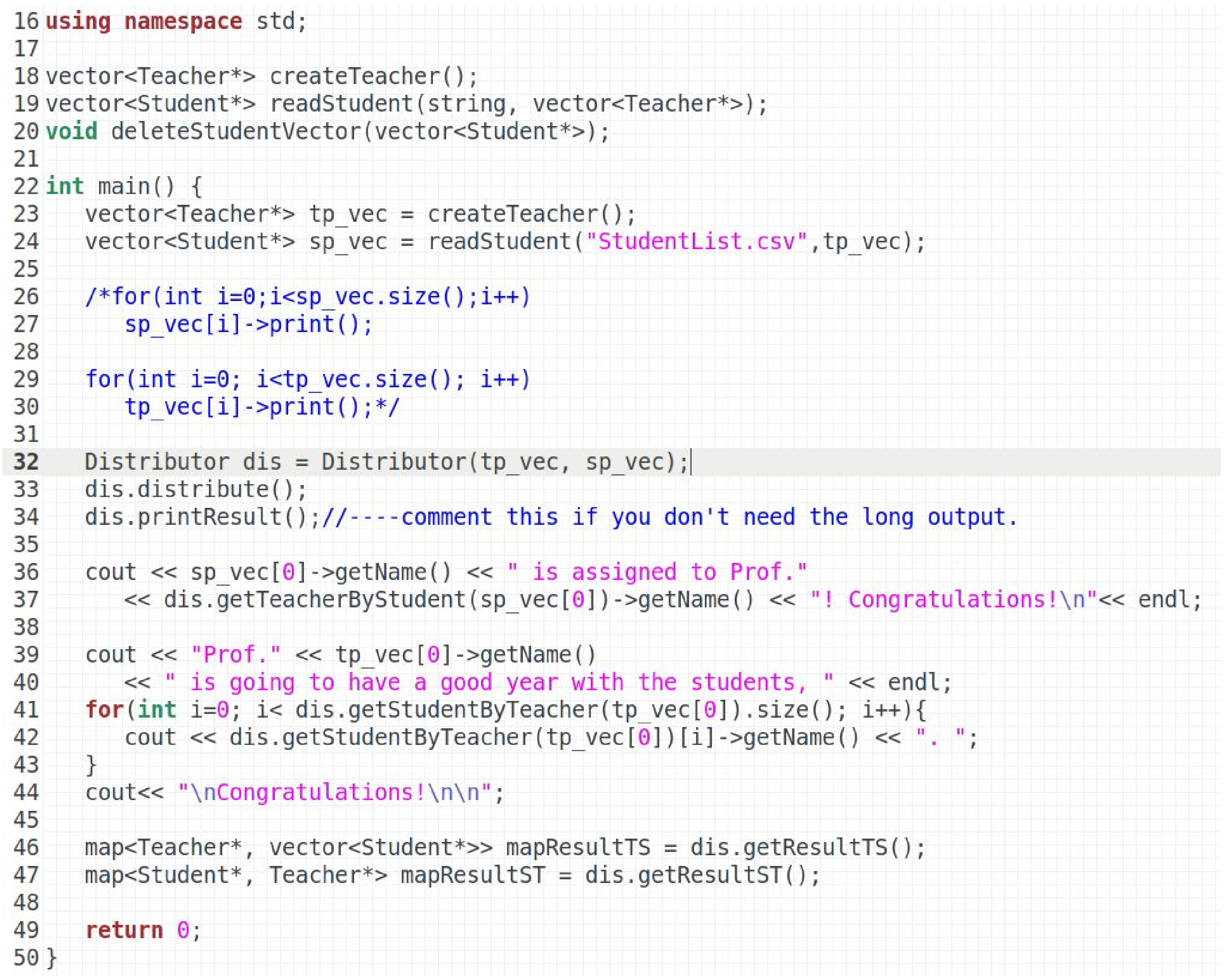
For more information, please check the comments in “Distributor.cpp.”

Results

In “ testTeacherStudent.cpp”, we test the programs with 2 csv file, “StudentList.csv” and “TeacherList.csv.”

“StudentList.csv” includes 26 students’ data, and “TeacherList.csv” includes 6 teachers’ data.

The picture below shows a part of “ testTeacherStudent.cpp.” There are 3 functions implemented in this file for reading the csv files, and deleting a vector<Student*>.



```cpp
using namespace std;

vector<Teacher*> createTeacher();
vector<Student*> readStudent(string, vector<Teacher*>);
void deleteStudentVector(vector<Student*>);

int main() {
    vector<Teacher*> tp_vec = createTeacher();
    vector<Student*> sp_vec = readStudent("StudentList.csv",tp_vec);

    /*for(int i=0;i<sp_vec.size();i++)
        sp_vec[i]->print();

    for(int i=0; i<tp_vec.size(); i++)
        tp_vec[i]->print();*/

    Distributor dis = Distributor(tp_vec, sp_vec);
    dis.distribute();
    dis.printResult();//----comment this if you don't need the long output.

    cout << sp_vec[0]->getName() << " is assigned to Prof."
        << dis.getTeacherByStudent(sp_vec[0])->getName() << "! Congratulations!\n"<< endl;

    cout << "Prof." << tp_vec[0]->getName()
        << " is going to have a good year with the students, " << endl;
    for(int i=0; i< dis.getStudentByTeacher(tp_vec[0]).size(); i++){
        cout << dis.getStudentByTeacher(tp_vec[0])[i]->getName() << ". ";
    }
    cout<< "\nCongratulations!\n\n";

    map<Teacher*, vector<Student*>> mapResultTS = dis.getResultTS();
    map<Student*, Teacher*> mapResultST = dis.getResultST();

    return 0;
}
```

Some parts of the output are showing below. For the completested information, please check “exampleOutPut.txt.” We can see that all the assignment result is reasonable according to the input data.

For example, “Amanda’ is not assigned to her first ideal teacher “Guo-Xin Yang,” but when she check the students assigned to “Guo-Xin Yang,” she will realize that they all choose “Guo-Xin Yang” as the first order, and they all meet the criterion “taught by teacher Guo-Xin Yang.”

Thus, she finds that Prof. Guo-Xin Yang is too popular with a very small limit number, 3. Then Amanda might accepted that it is a random result, and could frequently and happily meet Prof. Yin-Nan Huang, her second order, for advice during her last year in NTU.

```
==============Steps==============

( random seed: 1593092779 )

------------[ Step_1 ]-------------
```

```
Teacher/   TimesChosenByStudents/  redundancy/
 Bo-Hua Chen        0              -5
 Yin-Nan Huang      3              -3
 Yu-Ting Hsu        6              2
 Guo-Xin Yang       11             8
 Hau-Zhe He         3              -4
 Shang-Xien Xie     0              -5

------------[ Step_2 ]-------------

Teacher/   TimesChosenByStudents/  redundancy/
 Bo-Hua Chen        0              -5
 Yin-Nan Huang      2              -1
 Yu-Ting Hsu        2              2
 Guo-Xin Yang       0              0
 Hau-Zhe He         6              2
 Shang-Xien Xie     0              -5

------------[ Step_3 ]-------------

Teacher/   TimesChosenByStudents/  redundancy/
 Bo-Hua Chen        0              -5
 Yin-Nan Huang      3              2
 Yu-Ting Hsu        0              0
 Guo-Xin Yang       0              0
 Hau-Zhe He         0              0
 Shang-Xien Xie     0              -5

------------[ Step_4 ]-------------

Teacher/   TimesChosenByStudents/  redundancy/
 Bo-Hua Chen        0              -5
 Yin-Nan Huang      0              0
 Yu-Ting Hsu        0              0
 Guo-Xin Yang       0              0
 Hau-Zhe He         0              0
 Shang-Xien Xie     0              -5

------------[ Step_5 ]-------------

Teacher/   TimesChosenByStudents/  redundancy/
 Bo-Hua Chen        0              -2
 Yin-Nan Huang      0              0
 Yu-Ting Hsu        0            0
 Guo-Xin Yang       0            0
 Hau-Zhe He         0            0
 Shang-Xien Xie     0            -3

===============Student===============

[Student's]
name: Amanda
ID: b05501055
grade: 4
history teacher: Yin-Nan Huang.  Yu-Ting Hsu.  Guo-Xin Yang.
teacher order: Guo-Xin Yang.  Yin-Nan Huang.  Yu-Ting Hsu.
-----------------------
Assignment Result: Yin-Nan Huang
--------------------------------------#

[Student's]
name: Bob
ID: b05501056
grade: 4
history teacher: Hau-Zhe He.  Yin-Nan Huang.  Guo-Xin Yang.
teacher order: Guo-Xin Yang.  Yu-Ting Hsu.  Hau-Zhe He.
-----------------------
Assignment Result: Guo-Xin Yang
--------------------------------------#

[Student's]
name: Leo
ID: b06501007
grade: 3
history teacher: Hau-Zhe He.  Yin-Nan Huang.
teacher order: Guo-Xin Yang.  Yin-Nan Huang.  Yu-Ting Hsu.
-----------------------
Assignment Result: Guo-Xin Yang
--------------------------------------#

[Student's]
name: Tina
ID: b07501047
grade: 2
history teacher: Guo-Xin Yang.
teacher order: Guo-Xin Yang.  Yu-Ting Hsu.
-----------------------
```

```
Assignment Result: Guo-Xin Yang
-------------------------------------#

===============Teacher===============

[Teacher's]:
Name: Bo-Hua Chen
ID: 1
The limit of student number is: 5
Rule:
Rule No.1 is "GradeRestriction to Student."
Student should at least be at grade 1.

Student List: Yaris. Xx. Sandy. Willy.
-------------------------------------#

[Teacher's]:
Name: Yin-Nan Huang
ID: 2
The limit of student number is: 6
Rule:
Rule No.1 is "GradeRestriction to Student."
Student should at least be at grade 3.

Student List: Frank. Harry. Nick. Cindy. Amanda. Eric.
-------------------------------------#

[Teacher's]:
Name: Guo-Xin Yang
ID: 4
The limit of student number is: 3
Rule:
Rule No.1 is "GradeRestriction to Student."
Student should at least be at grade 2.
Rule No.2 is "UsedToBeStudent"
Student should have been taught by teacher Guo-Xin Yang

Student List: Bob. Leo. Tina.
-------------------------------------#

Amanda is assigned to Prof.Yin-Nan Huang! Congratulations!

Prof.Bo-Hua Chen is going to have a good year with the students,
Yaris. Xx. Sandy. Willy.
```

Congratulations!

Feedback

Jacky

At beginning, the limit of topic was so free that we couldn’t come up a good idea until Roy talked the sad story. Maybe having more limits is a good idea for students to decide the topic of term project.

After this final project, I have more experience to help me decide what kind of interface we should design for our class, and it should be straightforward and extendable. We keep modify our class interface during this week. It make our communication become more and more chaos, because we don’t know some certain class has been modified or not.

I also experienced that OOP is very friendly for cooperation. I just need to tell someone what kinds of function I provided in my class, and they don’t need to know how it works, just use my class in their classes or functions. If I update something without changing interface, they don’t need to update their code for me. It’s very convenient.

Roy

This is the first time for me to write a project using Object Oriented Programming. Although this is not a large or complex project, I have learned some coding concepts and design patterns that are very different from using programming languages like a calculator.

In the beginning, the topic is too free to come up with. Since the time and skill for us is limited, we try to find something by which we can practice not only the skills but also the communication with teammates. Communicating with teammates sometimes cost a lot of time but valuable. We can exchange ideas with each other and improve what we have learned in the class. Besides, the OOP framework favors cooperation. We can split the whole project into several smaller problems and distribute them to each member. If everyone is responsible for their parts, the project can be done efficiently.

To sum up, although the project is small, what I have mainly learned is the concept and teamwork. Besides, this course help me build up a new knowledge that we can think a question not only by functional programming but also the Object Oriented Programming.

Dennis

This final project is a big experience for me to more understand what object oriented programming is actually working. When I was learning OOP in class, it was a great difficulty for me. The concept was too abstract. Now, through this project, I can actually apply this program language. Moreover, I can realize how this program language work and what is the advantage of object oriented program.

Besides, the cooperation of this project is very wonderful. Everyone does his work well so we can finally complete it. If I will have an another programming project in the future, this will be a great help for me.

Kasa

Let’s be honest (Ahh...so impolite), I am now exhausted and could barely find a cheerful word. Being not talented enough, and not hard-working enough for this course during the whole semester, within the limited period of time, I have done my best on building this system with the technics and deep thoughts provided in class. When I look back on my former programming project right now, I can point out lots of parts for improvements, and, probably, so will I do when looking back on this project. Then, that’s the meaning and worth of the course for me.

Work Distribution

- Jacky : 25%
  - Teacher class, student class, distribution approach design
  - Technical Support (debug, point out some potential bug, provide some ideas about solution)
- Roy : 25%
  - Teacher class design and implementation
  - Rule class and it’s derivatives design and implementation
- Dennis : 25%
  - Student class design and implementation
- Kasa : 25%
  - Distribution approach design and implementation

Appendix

Study of the Distribution Program for the Joint Entrance Examination of Universities and Related problems, http://ip194097.ntcu.edu.tw/ungian/Chokphin/Hoagu/hunhoat/hunhoat.htm?fbclid=IwAR0gADyBRxdpHnTFwkq3B3F2p6CuR5Fl23P-dIMUImVOA-6hg3dZ2Ei3f_M