Programming II

Practice 1

Basic aspects of the OOP

Report

Grzegorz Skowroński

Introduction

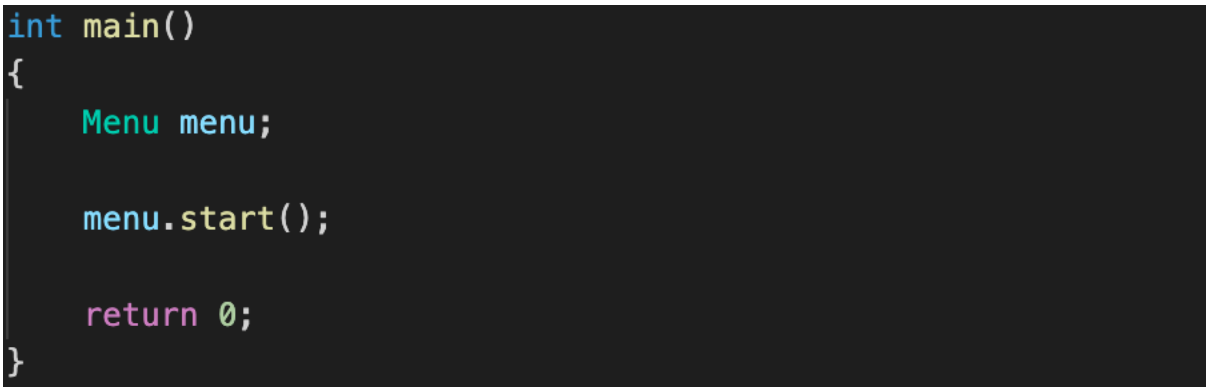
We are going to make a C++ program to manage students of different degrees at a university. Four classes are going to be created: Student, Degree, University and Menu. First three are going to have the data about the object and in case of Degree and Menu a vector of the specific objects. (e.g. Degree class will have vector of Student objects etc.) The last class – Menu – will contain all the functionality of the program: add student, delete graduates, list of all students, exit.

Operation overload, file input/output support, one dimensional vectors, classes and stream string were used to accomplish all of this.

Source code

Files: check.h ; degree.h ; menu.h ; student.h ; university.h ; main.cpp ; data.txt

Main() function is pretty simple.

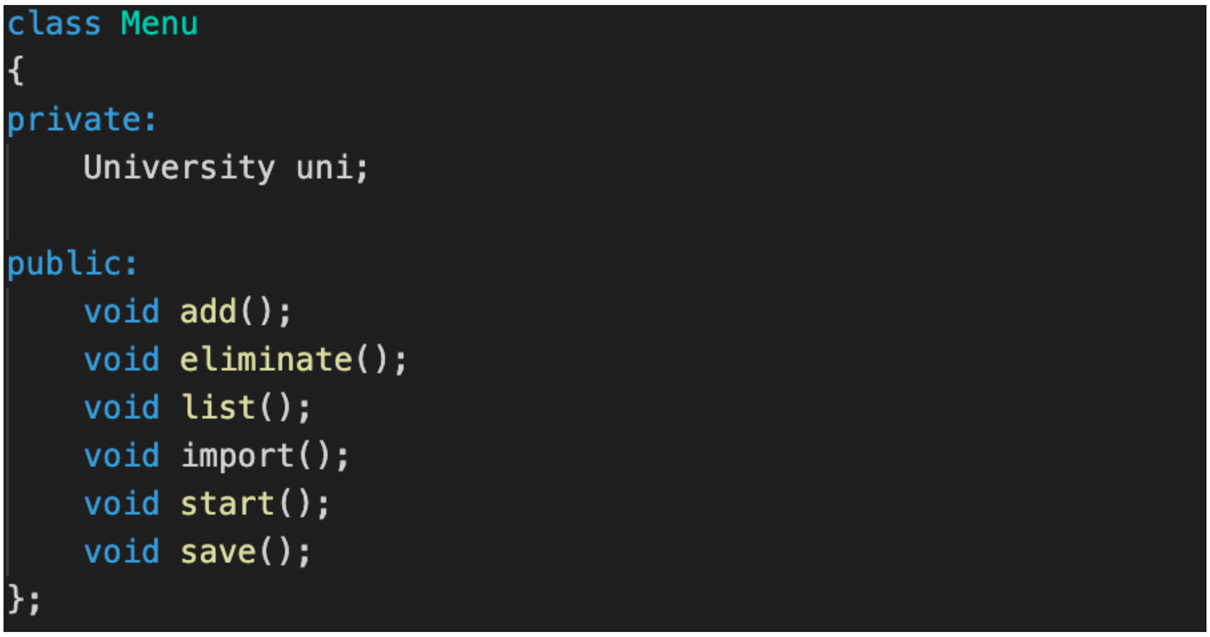


```cpp
int main()
{
    Menu menu;

    menu.start();

    return 0;
}
```

It just calls start function from Menu class. The start() function displays all text and the menu in the terminal.



```cpp
class Menu
{
private:
    University uni;

public:
    void add();
    void eliminate();
    void list();
    void import();
    void start();
    void save();
};
```

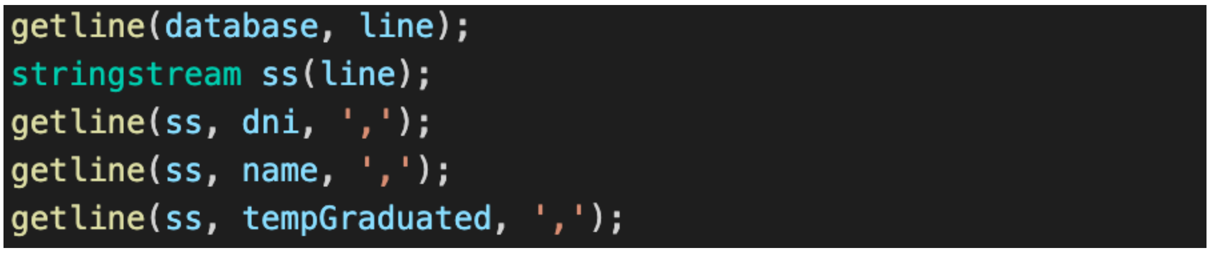
As we can see above, Menu class has 6 functions:

- Add() to add the student
- Eliminate() to delete graduates
- List() to display all the students
- Import() to import data from data.txt
- Start() to run the program and display menu
- Save() to export data to data.txt

In the Start() we can see that the Import() function is called and after this the user needs to enter a digit between 1 and 4. The correctness of the entered string is checked by its size and compared with values of 1, 2, 3 and 4 from ASCII table.

In the Import() the external file data.txt is being opened. The first line in the file corresponds to number of degrees. Next two lines to the name of the degree and the number of the students in this degree. Thanks to that numbers it was much easier to create safe loop function that import all the data from this file.

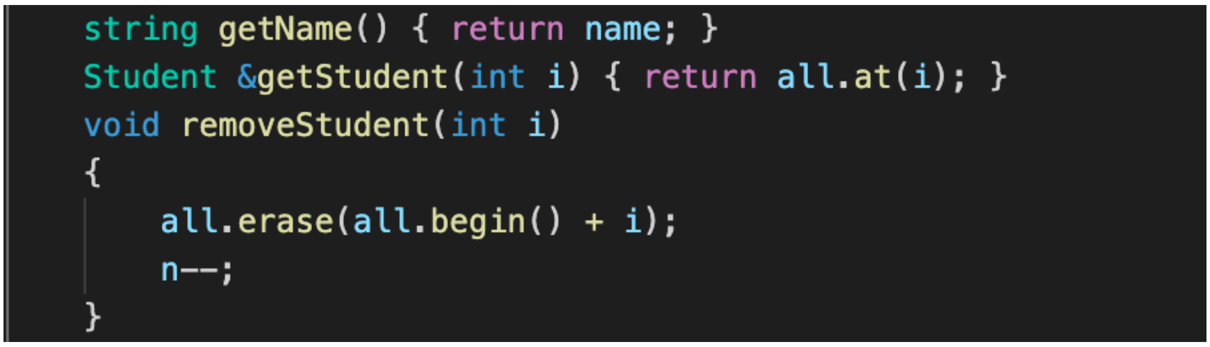
The data of each student is read by a stringstream class as we can see below.



```cpp
getline(database, line);
stringstream ss(line);
getline(ss, dni, ',');
getline(ss, name, ',');
getline(ss, tempGraduated, ',');
```

Stringstream works alike an input file. We pass there an information from each line that can be accessed by us later to divide this data by separating them with comma. Thanks to that we achieve separate strings.

Remaining three header files – student.h, degree.h and university.h consists of the class and its functions. We can find there mostly setters, getters, constructors and operation overloading.



```
    string getName() { return name; }
    Student &getStudent(int i) { return all.at(i); }
    void removeStudent(int i)
    {
        all.erase(all.begin() + i);
        n--;
    }
```

A tricky thing in the Degree and University class was about returning a Student and a Degree respectively by its reference. Thanks to that we were able to edit not the copy but the vector contained within the class.

The last header file – check.h – contains all function that were needed to validate input data or to convert Boolean variable.

Function dni_check() checks the length of the entered string. If the length is equal to 9 then it checks whether first eight chars are digits and if the last one is a letter.

Both name_check() and degree_check() works alike. The check whether the entered string consists only of letters. We have assumed that both student name and degree name consist only of capital or small letters and at least some Rome numeral e.g.: Firstname Lastname III.

Function graduated_check() just check whether the input string is equal to "yes" or to "no".

Experimentation and Validation

To test the functionality of the program a short database data.txt was created that consist of almost all difficult cases of the program.

- One student in signed up in two degrees.
- One degree has only one student that has already graduated.
- Some students have graduated, some of them have not.

Thanks to that we could add new students, try to update their data, remove the graduates and see how the degree that has only one member that has already graduated is being deleted.

Conclusions

Thanks to the class objects the program was much easier to maintain. Moreover, the code seems to be more clear and easier to understand by a 3rd person. We have an access to all the classes separately and we have possibility to expand program functionality with almost no effort. In comparison to a program that would be written without class and objects we have saved a lot of time and efforts that could be put to good use of improving its functionality or fixing some bugs. Summing up, object-oriented programming has many advantages.

- Modularity for easier troubleshooting
- Reuse of code through inheritance
- Flexibility trough polymorphism
- Effective problem solving