ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

CƠ SỞ LẬP TRÌNH

Bài thực hành 06

DỮ LIỆU CẤU TRÚC VÀ XỬ LÝ TỆP TIN

Giảng viên thực hành: Lê Đức Khoan

Thành phố Hồ Chí Minh, 11/2025

Mục lục

1 Khái niệm

1.1 Dữ liệu cấu trúc

Trong C++, ngoài những kiểu dữ liệu phổ biến thường được sử dụng như: char, string, int, float, double, ... Trong nhiều trường hợp chúng ta cần sử dụng kết hợp nhiều kiểu dữ liệu hơn để biểu diễn được thông tin mong muốn. Ví dụ để biểu diễn một phân số ta cần lưu: tử số và mẫu số. Tương tự khi lưu thông tin của sinh viên ta cần lưu: tên, ngày tháng năm sinh, MSSV, ... Với các ví dụ trên ta cần sử dụng dữ liệu có cấu trúc để biểu diễn thay vì những kiểu dữ liệu có sẵn trong ngôn ngữ lập trình.

1.2 Xử lí tệp tin

Với các chương trình lớn, dữ liệu lớn. Đầu vào của chương trình chúng ta không thể nhập từ bàn phím. Do đó việc đọc dữ liệu từ tệp tin đóng vai trò quan trọng trong trường hợp này. Với việc đọc dữ liệu tệp tin chúng ta có các thao tác cơ bản bao gồm:

1. Mở tập tin, ta cần cung cấp đường dẫn và tên tập tin chính xác.
2. Sử dụng tập tin (sau khi mở thành công).
   (a) Đọc dữ liệu tập tin đưa vào biến bộ nhớ
   (b) Ghi dữ liệu từ biến bộ nhớ trong chương trình lên tập tin sau khi đã xử lý
3. Đóng tập tin.

2 Thao tác với dữ liệu có cấu trúc

2.1 Khai báo

Ta đã có các kiểu cơ bản như int, float, double, char... Giờ giả sử ta cần tạo ra một kiểu dữ liệu mới ví dụ như Student với các thông tin cơ bản như: tên, MSSV. Ta sẽ khởi tạo cấu trúc Student với các biến: name đại diện cho tên và student_code đại diện cho MSSV. Bên dưới là cách khởi tạo:

```
struct Student {
    string name;
    int student_code;
};
```

Câu lệnh sẽ bắt đầu bằng từ khoá struct theo sau là tên của cấu trúc và các thuộc tính sẽ được khai báo trong cặp dấu {} và kết thúc bằng dấu ";".

Bây giờ ta có thể sử dụng cấu trúc **Student** như một kiểu dữ liệu thông thường. Tuy nhiên việc nhập xuất các giá trị của cấu trúc thì chúng ta cần định nghĩa.

```
#include <iostream>
#include <string>

using namespace std;

struct Student {
    string name;
    int student_code;
};

void input_student(Student& student){
    cout << "Input name: ";
    getline(cin, student.name);
    cout << "Input student code: ";
    cin >> student.student_code;
```

```
16          cin.ignore();
17      }
18
19      void output_student(Student& student){
20          cout << "Student name: " << student.name << endl;
21          cout << "Student code: " << student.student_code << endl;
22      }
23      int main() {
24          Student student; // Khởi tạo một sinh viên
25          const int MAX_STUDENT = 10;
26          Student students[MAX_STUDENT]; // Khởi tạo một mảng chưa 10 sinh viên
27          // Xuất nhập một sinh viên
28          input_student(student);
29          output_student(student);
30          // Xuất nhập nhiều sinh viên
31          for(int i = 0; i < MAX_STUDENT; i++) {
32              input_student(students[i]);
33          }
34          // Xuất nhiều sinh viên
35          for(int i = 0; i < MAX_STUDENT; i++) {
36              output_student(students[i]);
37          }
38          return 0;
39      }
```

Sinh viên tự tìm hiểu thêm về **cin.ignore()** khi xuất nhập vừa nhập số và vừa nhập chuỗi sử dụng **getline()**.

2.2 Cấu trúc lồng nhau

Với ví dụ bên trên nếu ta cần lưu thêm trường thông tin liên quan đến ngày sinh của sinh viên ta cần khai báo một cấu trúc Date chưa thông tin về ngày, tháng, năm. Cấu trúc này sẽ được sử dụng lại trong cấu trúc dữ liệu Student.

```
struct Date {
    int day;
    int month;
    int year;
}
struct Student {
    string name;
    Date birth_day;
    int student_code;
};
```

2.3 Khởi tạo

Chúng ta có thể khởi tạo giá trị cho một biến có kiểu dữ liệu cấu trúc thông qua sử dụng các hàm tự hiện thực như ví dụ trên ngoài ra có thể sử dụng khởi tạo lúc tạo biến như sau:

```
struct Date {
    int day;
    int month;
    int year;
}
struct Student {
    string name;
    Date birth_day;
    int student_code;
};
int main() {
    Student student = {"Nguyen Van A", {1, 2, 2003}, 12345}
    // Khởi tạo giá trị cho biến student
    return 0;
}
```

2.4 Truy cập dữ liệu trong cấu trúc

Sau đây là một số cách truy cập dữ liệu trong cấu trúc:

- Sử dụng toán tử "**.**".

```
int main() {
    Student student = {"Nguyen Van A", 12345};
    cout << "Student name: " << student.name << endl;
    cout << "Student code: " << student.student_code << endl;
    return 0;
}
```

- Sử dụng "**->**" khi làm việc với con trỏ.

```
int main() {
    Student student = {"Nguyen Van A", 12345};
    Student *sp = &student;
    cout << "Student name: " << sp->name << endl;
    cout << "Student code: " << sp->student_code << endl;
    return 0;
}
```

- Sử dụng "**.**" và "**.**" hoặc "**->**" và "**.**" trong trường hợp cấu trúc lồng nhau.

```
int main() {
    Student student = {"Nguyen Van A", {1, 2, 2000} 12345};

    // Truy cập sử dụng "."
    cout << "Birth day: " << student.birth_day.day << "/"
        << student.birth_day.month << "/"
        << student.birth_day.year << endl;

    // Truy cập sử dụng "->" với con trỏ
```

```
10          Student *sp = &student;
11          cout << "Birth day: " << student->birth_day.day << "/"
12              << student->birth_day.month << "/"
13              << student->birth_day.year << endl;
14          return 0;
15      }
```

3 Thao tác với file dữ liệu

Để thực hiện các thao tác đọc/ghi dữ liệu ta sử dụng thư viện **fstream** trong C++. Sau đây là trình bày các bước cơ bản cho quá trình đọc và ghi file. Ở đây chúng ta sẽ thực hiện đọc ghi với file (.txt).

3.1 Đọc file

Thao tác này giúp chúng ta đọc dữ liệu từ file và lưu vào các biến hoặc in ra màn hình.

```
1   #include <iostream>
2   #include <fstream>
3   using namspace std:
4   int main() {
5       ifstream fin("data.txt");
6       return 0;
7   }
```

Ở đây chúng ta sử dụng lệnh **ifstream** để đọc file và đưa vào trong **fin**. Chúng ta có thể đổi bên **fin** thành bất cứ tên gì chúng ta muốn.

```
1   #include <iostream>
2   #include <fstream>
3   using namspace std:
4   int main() {
5       ifstream file_input("data.txt");
```

```
6         return 0;
7     }
```

Ở ví dụ trên ta sử dụng tên **file_input**. Để đọc từng dòng trong file ta sử dụng hàm **getline**.

```
1     // main.cpp
2     #include <iostream>
3     #include <fstream>
4     #include <string>
5     using namespace std;
6     int main() {
7         ifstream file_input("data.txt");
8         string line;
9         while (getline(file_input, line)) {
10            cout << line << endl;
11        }
12        return 0;
13    }
```

Giả sử file data.txt có:

```
// data.txt
Nguyen Van A
Nguyen Van B
Le Van C
```

Sau khi thực hiện đoạn mã trên chúng ta sẽ có kết quả:

```
// g++ main.cpp -o main && ./main
Nguyen Van A
Nguyen Van B
Le Van C
```

Các dùng dữ liệu sẽ được đọc và ghi từng dòng ra terminal.

Bên cạnh việc đọc các dòng văn bản chúng ta có thể đọc được các số từ file. Thông qua việc sử dụng toán tử "»".

```cpp
// main.cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
    ifstream file_input("data.txt");
    int number;
    while (file_input >> number) {
        cout << number << endl;
    }
    return 0;
}
```

Ví dụ file data.txt có:

```
// data.txt
3
2 3 4
1
```

Ở đây khi thực thi đoạn chương trình trên chúng ta sẽ không có kết quả là từng dòng dữ liệu được in ra như ví dụ ban đầu mà mỗi chữ số trong file sẽ được in trên mỗi dòng. Lí do cho việc này là do toán tử "»" chỉ đọc đến khi gặp các ký từ đặc biệt như khoảng trắng, xuống dòng thì sẽ dừng lại. Do đó ta có kết quả là:

```
// g++ main.cpp -o main && ./main
3
2
3
4
```

```
1
```

Ngoài ra trong một số trường hợp dòng dữ liệu có thể cách nhau bằng các dấu như: "/", "-", "_", ",". Ta có thể đọc từng dòng theo cách thứ nhất và sử dụng hàm **getline()** với tham số thứ 3 chỉ định Delimeter. Trường hợp này chúng ta sẽ gặp trong thực tế khi muốn phân tách dữ liệu trên cùng một dòng và đưa vào các biến. Tuy nhiên đối số thứ nhất phải thuộc lớp **istream**. Chúng ta cần thêm việc sử dụng thư viện **sstream** để chuyển dữ liệu từ string về dạng istream. Dưới đây là ví dụ.

```cpp
// main.cpp
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
int main() {
    string fruits = "apple,orange,cherry,mango";
    string fruit;

    // Chuyển string thành istream
    stringstream fruits_ss(fruits);

    // Đọc từng thành phần và in ra màn hình
    // fruits_ss: luồng nhập từ string
    // fruit: Biến chứa giá trị được đọc
    // ',': Delimeter (Dấu phân tách dữ liệu)
    while (getline(fruits_ss, fruit, ',')) {
        cout << fruit << endl;
    }
    return 0;
}
```

Khi chạy ta sẽ có kết quả:

```
// g++ main.cpp -o main && ./main
```

```
apple
orange
cherry
mango
```

3.2 Ghi file

Với việc ghi file ta có thể có nhiều lựa chọn: ghi tiếp hoặc xoá bỏ dữ liệu cũ và ghi lại dữ liệu mới. Để ghi file ta sử dụng **ofstream**.

3.2.1 Ghi đè và xoá dữ liệu cũ

Với lựa chọn này toàn bộ dữ liệu trong file ban đầu sẽ bị xoá bỏ và dữ liệu mới sẽ được ghi vào file.

```cpp
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream fout("output.txt");
    fout << "Hello World!" << endl;
    return 0;
}
```

Giả sử ban đầu file output.txt chưa:

```
// output.txt
Nguyen Van A
Nguyen Van B
```

Sau khi thực thi chương trình ta sẽ có kết quả:

```
// output.txt
Hello World!
```

Chúng ta có thể xem fout tương tự như cout. Lúc này thay vì ghi ra terminal ta sẽ ghi vào file.

3.2.2 Ghi tiếp vào dữ liệu cũ

Việc này cho phép ta tiếp tục ghi vào cuối file mà không xoá bỏ nội dung cũ.

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    ofstream fout("output.txt", ios::app);
    fout << "Hello World!" << endl;
    return 0;
}
```

Giả sử ban đầu file output.txt chưa:

```
// output.txt
Nguyen Van A
Nguyen Van B
```

Sau khi thực thi chương trình ta sẽ có kết quả:

```
// output.txt
Nguyen Van A
Nguyen Van B
Hello World!
```

Chúng ta làm được việc này nhờ tham số **ios::app** chỉ định việc append (thêm) dữ liệu vào file. Còn với trường hợp đầu tiên tham số mặc định là **ios::out**.

3.3 Đọc và ghi file

Chúng ta có thể mở một file vừa ở dạng đọc và dạng ghi. Ta sử dụng **fstream**.

```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main() {
    // Mở file ở chế độ đọc, ghi và thêm vào
    fstream file("data.txt", ios::in | ios::out | ios::app);

    string line;
    while (getline(file, line)) {
        cout << line << endl;
    }
    // Xoá bỏ trạng thái lỗi EOF
    file.clear();

    // Ghi thêm vào cuối file
    file << "Hello World\n";

    file.close();
    return 0;
}
```

Giả sử ban đầu file data.txt chưa:

```
// data.txt
Nguyen Van A
```

Sau khi thực thi chương trình ta sẽ có kết quả:

```
// g++ main.cpp -o main && ./main
Nguyen Van A
```

Kết quả trong file data.txt.

```
// data.txt
```

```
Nguyen Van A
Hello World!
```

Nếu không sử dụng dòng lệnh số 14 thì chương trình sẽ không thể ghi được dữ liệu vào file vì trạng thái của pointer đọc đang mang giá trị EOF điều này dẫn đến việc không thể ghi được. Do đó ta dùng dòng lệnh 14 để xoá bỏ trạng thái EOF và tiếp tục ghi vào file. **Sinh viên có thể nghiên cứu thêm về vấn đề này.**

4 Yêu cầu bài nộp

Sinh viên được cung cấp một thư mục mã nguồn **lab_06** chứa thư viện, các định nghĩa hàm. Mã nguồn đã bao gồm đầy đủ các định nghĩa hàm cho tất cả các bài tập trong Lab 06. Sinh viên thực hiện thực các hàm theo prototype đã được định nghĩa ở file **.cpp** và có thể viết thêm hàm nếu cần thiết.

Chú ý: Sinh viên không được tự ý sửa đổi bất kỳ thành phần nào của mã nguồn mẫu. Nếu mã nguồn biên dịch không thành công khi chấm thì sẽ nhận điểm 0 cho bài tập đó.

Khi nộp bài sinh viên **đổi tên thư mục lab_06 thành MSSV và zip toàn bộ mã nguồn** thành file **MSSV.zip** với MSSV là mã số sinh viên được nhà trường cung cấp. Các thành phần ở trong thư mục phải giống như thư mục được cung cấp ban đầu. Không thêm bất kỳ file nay thư mục nào khác.

Tên file zip mẫu:

```
25123456.zip
```

Tất cả các trường hợp làm sai yêu cầu sẽ nhận điểm 0 cho bài thực hành. Vì thế sinh viên cần đọc kỹ và thực hiện đúng yêu cầu.

5 Hướng dẫn chạy file thực hành lab_08

Tổ chức thư mục của lab_06 như sau:

```
lab_08
   |-- fraction_helper
   |           |-- fraction_utils.h
   |           |-- fraction_utils.cpp
   |           |-- test_result
   |                   |-- input_fraction.txt
   |                   |-- output_fraction.txt
   |
   |-- student_helper
   |           |-- student_utils.h
   |           |-- student_utils.cpp
   |           |-- data
   |           |     |-- input_student.txt
   |           |-- result
   |                 |-- output_student.txt
   |                 |-- search_sc.txt
   |                 |-- search_name.txt
   |                 |-- remove_sc.txt
   |                 |-- sorted_student.txt
   |                 |-- average_score.txt
   |                 |-- good_student.txt
   |-- main.cpp
   |-- Makefile
```

Vì có nhiều file được include vào trong file main để chạy chương trình do đó để nhanh chóng compile và chạy chương trình. Trong bài lab này chúng ta sẽ tiếp cận với giải pháp dùng Makefile để compile và chạy chương trình. Gõ vào terminal để kiểm tra version của Make:

```
make -v
```

Nếu không xuất hiện version thì máy chưa có Make. Khi đó, các bạn có thể tham khảo hướng

dẫn cài đặt Make. Sau khi đã cài xong, các bạn đi đến thư mục đang chưa file **Makefile** và có thể thực hiện các câu lệnh sau:

- **make all**: Compile code ở tất cả các file để tạo ra file thực thi (**.exe**). Đây là default command nên nếu bạn gõ **make** thì nó sẽ tự động hiểu là **make all**.
- **make clean**: Để xoá bỏ file thực thi vừa tạo.
- **make run**: Thực hiện compile code và chạy chương trình.

Các câu lệnh trong Makefile bao gồm:

```
1  # Compiler and flags
2  CXX = g++
3  CXXFLAGS = -std=c++17 -Wall -g -I fraction_helper -I student_helper
4
5  # Name of execution file
6  TARGET = main
7
8  # List of source files
9  SRC = main.cpp fraction_helper/fraction_utils.cpp
10                 student_helper/student_utils.cpp
11
12 # Default command
13 all:
14         $(CXX) $(CXXFLAGS) $(SRC) -o $(TARGET)
15
16 # Clean command
17 clean:
18         rm -f $(TARGET)
19
20 # Compile and run command
21 run: all
22         ./$(TARGET)
```

- **Câu lệnh số 2**: Khai báo compiler là **g++**.

- **Câu lệnh số 3**: Khai báo thêm các flags khi compile code. Với các tham số sau **-I** chỉ các thư mục chưa file **.h**. Nếu có thêm nhiều thư viện tự định nghĩa ta sẽ thêm tiếp các cặp **-I library_directory**. Mỗi thư mục sẽ đi với một ký tự **-I**.
- **Câu lệnh số 6**: Khai báo tên file **Thực thi** được tạo ra khi compile mã nguồn.
- **Câu lệnh số 13**: Compile mã nguồn và tạo ra file thực thi. Với câu lệnh tương tự trong những tuần trước: **g++ main.cpp -o main**. Câu lệnh này sẽ được chạy khi ta gọi: **make all** hoặc **make**.
- **Câu lệnh số 17**: Xoá file thực thi vừa được tạo ra. Lệnh được thực thi khi gọi: **make clean**.
- **Câu lệnh số 21**: Thực hiện compile và chạy file thực thi. Lệnh được thực thi khi gọi: **make run**. Khi gọi lệnh này sẽ thực hiện hai việc liên tục là **make all** và **./main**. Với **make all** đã được trình bày ở trên và **./main** là để chạy file thực thi.

6 Bài tập

6.1 Tính toán phân số

Viết chương trình xây dựng cấu trúc để chứa phân số (Các phân số luôn hợp lệ) và hiện thực các chức năng:

1. Rút gọn phân số

```
1       Fraction reduce_fraction(const Fraction& fraction);
```

2. Cộng hai phân số

```
1       Fraction sum_fraction(const Fraction& first_fraction,
2                             const Fraction& second_fraction);
```

3. Trừ hai phân số

```
1       Fraction minus_fraction(const Fraction& first_fraction,
2                               const Fraction& second_fraction);
```

4. Nhân hai phân số

```
1       Fraction multiply_fraction(const Fraction& first_fraction,
2                                  const Fraction& second_fraction);
```

5. Chia hai phân số

```
1       Fraction divide_fraction(const Fraction& first_fraction,
2                                const Fraction& second_fraction);
```

6. So sánh hai phân số: Hiện thực ba hàm để so sánh lớn hơn, bé hơn và bằng nhau.

```
1       bool is_equal(const Fraction& first_fraction,
```

```
                                const Fraction& second_fraction);
        bool is_greater(const Fraction& first_fraction,
                                const Fraction& second_fraction);
        bool is_less(const Fraction& first_fraction,
                                const Fraction& second_fraction);
```

Yêu cầu bài tập: Sinh viên hiện thực các hàm như yêu cầu phía trên. Bên cạnh đó thực hiện hàm **auto_test_fraction** để nhận vào file (input_fraction.txt) và trả kết quả ra file (output_fraction.txt).

```
    void auto_test_fraction(input_file_name = "input_fraction.txt",
                            output_file_name = "output_fraction.txt");
```

Sinh viên sử dụng ghi đọc file để đọc các phân số từ trong file (input_fraction.txt) và hiện thực các phép toán theo yêu cầu trong file sau đó lưu kết quả đã được tối giản vào file (output_fraction.txt). Dưới đây là format của file (input_fraction.txt):

```
4
+
1/2, 3/4, 5/6
*
1/2, 1/3
-
4/3, 1/3
>
6/5, 20/3
```

Giải thích file **input_fraction.txt**:

- Dòng đầu tiên: Số lượng testcase có trong file
- Mỗi testcase sẽ bao gồm 2 dòng:
  - Dòng 1: Phép toán được thực hiện.

– Dòng 2: Các phân số cần tính toán cách nhau bằng dấu ",". Phân số được viết theo dạng "a/b". Không có testcase nào quá 5 phân số và với các phép so sánh luôn là 2 phân số.

Sinh viên ghi kết quả tính toán được ra file **output_fraction.txt**. Theo yêu cầu sau:

- Kết quả của mỗi testcase ghi ở một dòng theo đúng format. Nếu rút gọn về số thì sẽ để ở dạng số (xem testcase số 3).
- Với các phép toán so sánh. Ghi "1" cho đúng và "0" cho sai(xem testcase số 4).

Với testcase trên ta có kết quả:

```
25/12
1/6
1
0
```

6.2 Quản lí sinh viên

Viết chương trình xây dựng cấu trúc Student (tên, MSSV, điểm số) và thực hiện các chức năng sau. Kết quả của mỗi chức năng sẽ được ghi ra một file riêng biệt với tên được cho sẵn như bên dưới.

1. Ghi danh sách sinh viên ra file (**output_student.txt**)
2. Đọc danh sách sinh viên từ file (**input_student.txt**)
3. Tìm kiếm sinh viên theo MSSV (**search_sc.txt**)
4. Tìm kiếm sinh viên theo Tên (**search_name.txt**)
5. Xoá sinh viên theo MSSV (**remove_sc.txt**)
6. Sắp xếp sinh viên theo điểm số (**sorted_student.txt**)
7. Tính điểm trung bình lớp (**average_score.txt**)
8. Lọc các sinh viên đạt mức giỏi >= 8.0 (**good_student.txt**)

Sinh viên đọc file **input_student.txt** và thực hiện các chức năng và ghi ra file tương ứng. Sau đây sẽ trình bày chi tiết cho từng chức năng.

6.2.1 Đọc danh sách sinh viên từ file

Hiện thực hàm đọc danh sách sinh viên từ file và lưu vào trong mảng để phục vụ cho các chức năng tiếp theo. Mỗi sinh viên sẽ có 3 trường thông tin tương ứng: tên, MSSV, điểm số. Các trường cách nhau bằng dấu ",".

```
1    void read_students(Student students[], int& num_students,
2                       std::string file_name = "input_student.txt");
```

6.2.2 Ghi danh sách sinh viên ra file

Hiện thực hàm ghi danh sách các sinh viên ra file.

```
1    void write_students(Student students[], int& num_students,
2                        std::string out_file_name = "output_student.txt");
```

6.2.3 Tìm kiếm sinh viên theo MSSV

Hiện thực hàm tìm kiếm sinh viên theo MSSV và ghi ra file thông tin của sinh viên đó theo đúng format trong file **input_student.txt**.

```
1    void search_by_student_code(Student students[], int& num_students,
2                                int student_code,
3                                std::string out_file_name = "search_sc.txt");
```

6.2.4 Tìm kiếm sinh viên theo tên

Hiện thực hàm tìm kiếm sinh viên theo tên và ghi ra file thông tin của sinh viên theo đúng format trong file **input_student.txt**.

```
1    void search_by_name(Student students[], int& num_students,
2                        std::string student_name,
3                        std::string out_file_name = "search_name.txt");
```

Chú ý: student_name có thể là substring của tên trong file. Do đó sinh viên cần tìm kiếm và lưu tất cả các tên chưa substring.

6.2.5 Xoá sinh viên theo MSSV

Hiện thực hàm xoá sinh viên theo MSSV và ghi ra file thông tin các sinh viên còn lại theo format trong file **input_student.txt**.

```
1    void remove_student(Student students[], int& num_students,
2                        int student_code, std::string out_file_name = "remove_sc.txt");
```

6.2.6 Sắp xếp sinh viên theo điểm số

Hiện thực hàm sắp xếp sinh viên theo điểm số (tăng dần) và ghi ra file thứ tự đã sắp xếp theo format trong file **input_student.txt**.

```
1    void sort_by_score(Student students[], int& num_students,
2                       std::string out_file_name = "sorted_student.txt");
```

6.2.7 Tính điểm trung bình lớp

Hiện thực hàm tính điểm trung bình toàn lớp và ghi ra file giá trị điểm trung bình. Làm tròn đến 2 chữ số thập phân sử dụng hàm **round_to_decimal**.

```
1    void average_score(Student students[], int& num_students,
2                       std::string out_file_name = "average_score.txt");
```

6.2.8 Lọc sinh viên đạt mức giỏi

Hiện thực hàm lọc các sinh viên đạt mức giỏi (≥ 8) và ghi các sinh viên vào file theo format trong file **input_student.txt**.

```
1    void filter_good_students(Student students[], int& num_students,
2                              std::string out_file_name = "good_student.txt");
```