Thực Hành 11

1. Bài tập 1

Định nghĩa kiểu dữ liệu mới **PhanSo** sử dụng cấu trúc **struct** để biểu diễn phân số theo dạng a/b trong đó a và b là các số nguyên với b≠0, ngoài ra phân số luôn ở dạng tối giản.

Viết hàm thực hiện các công việc sau:

1. Nhập dữ liệu cho tử số và mẫu số của 1 phân số kiểu **PhanSo** dưới dạng "**a/b**"
   ```
   void nhapPhanSo(PhanSo& ps);
   ```
2. In ra tử số và mẫu số theo dạng "**a/b**"
   ```
   void inPhanSo(const PhanSo& ps);
   ```
3. Tối giản phân số, phân số luôn ở dạng tối giản, ví dụ 2/4, quy đổi thành 1/2
   ```
   void toiGianPhanSo(PhanSo& ps);
   ```
4. Tính tổng 2 phân số
   ```
   void tongPhanSo(const PhanSo& ps1,
                   const PhanSo& ps2,
                         PhanSo& psTong);
   ```
5. Tính hiệu 2 phân số
   ```
   void hieuPhanSo(const PhanSo& ps1,
                   const PhanSo& ps2,
                         PhanSo& psHieu);
   ```
6. Tính tích 2 phân số
   ```
   void tichPhanSo(const PhanSo& ps1,
                   const PhanSo& ps2,
                         PhanSo& psTich);
   ```
7. Tính thương 2 phân số
   ```
   void thuongPhanSo(const PhanSo& ps1,
                     const PhanSo& ps2,
                           PhanSo& psThuong);
   ```
8. So sánh nhỏ hơn 2 phân số (trả về đúng nếu **ps1** nhỏ hơn **ps2**)
   ```
   bool soSanhNhoHonPhanSo(const PhanSo& ps1,
                           const PhanSo& ps2);
   ```

Viết chương trình kiểm tra các hàm trên đã cài đặt đúng hay chưa.

Nhập vào một mảng các phân số kiểu **PhanSo**, theo cả 2 cách sau:

- Đọc dữ liệu từ bàn phím
- Đọc dữ liệu từ file (có thể phải sửa hàm **nhapPhanSo**)

Sắp xếp mảng các phân số theo thứ tự giảm dần và in ra mảng sau khi sắp xếp.

Đếm số lượng các phân số nhỏ hơn 0, lớn hơn 0, và bằng 0.

Tính tổng các phân số có trị tuyệt đối nhỏ hơn 1.

Tính tích các phân số có trị tuyệt đối lớn hơn 1.

Nâng cao: viết các hàm trên sử dụng khái niệm toán tử nạp chồng.

2. Bài tập 2

Định nghĩa kiểu dữ liệu mới **Time** sử dụng cấu trúc **struct** để biểu diễn thời gian theo dạng giờ (**hour**), phút (**minute**), và giây (**second**) theo định dạng 24 giờ.

Viết hàm thực hiện các công việc sau:

1. Nhập dữ liệu giờ, phút, và giây cho 1 thời gian kiểu **Time**
   ```
   void nhapTime(Time& tm);
   ```
2. In ra thời gian theo dạng "**hh:mm:ss**"
   ```
   void inTime(const Time& tm);
   ```
3. Quy chuẩn giờ phút giây theo đúng định dạng 24 giờ
   ```
   void quyChuanTime(Time& tm);
   ```
4. Tính tổng 2 thời gian
   ```
   void tongTime(const Time& tm1,
                 const Time& tm2,
                       Time& tmTong);
   ```
5. Tính khoảng cách thời gian giữa 2 thời gian
   ```
   void diffTime(const Time& tm1,
                 const Time& tm2,
                       Time& tmDiff);
   ```
6. So sánh lớn hơn 2 thời gian (trả về đúng nếu **tm1** lớn hơn **tm2**)
   ```
   bool soSanhLonHonTime(const Time& tm1,
                         const Time& tm2);
   ```
7. Chuyển đổi 1 thời gian sang giờ, phút hoặc giây tương ứng với tham số thứ hai, trong đó **type** là một trong ba hằng số **HOUR**, **MINUTE**, **SECOND**,
   ```
   double convertTimeToD(const Time& tm, const int type);
   PhanSo convertTimeToPS(const Time& tm, const int type);
   ```

Viết chương trình kiểm tra các hàm trên đã cài đặt đúng hay chưa.

Nhập vào một mảng các thời gian kiểu **Time**, theo cả 2 cách sau:

- Đọc dữ liệu từ bàn phím
- Đọc dữ liệu từ file (có thể phải sửa hàm **nhapTime**)

Sắp xếp mảng các thời gian theo thứ tự tăng dần và in ra mảng sau khi sắp xếp.

Đếm số lượng các thời gian trước và sau 12:00:00.

Xác định các thời gian lặp (bằng nhau) và đếm số lượng mỗi lặp.

Xác định các cặp thời gian có khoảng cách giống nhau, đếm số lượng chúng, ví dụ: 01:00:00; 02:00:00; 03:00:00; 04:00:00, có các cặp (1,2); (2,3); (3;4) khoảng cách 1 giờ (3 cặp), và các cặp (1, 3); (2, 4) khoảng cách 2 giờ (2 cặp). Gợi ý: tính khoảng cách giữa các cặp, lưu vào mảng 2 chiều, sau đó xử lý dữ liệu mảng 2 chiều.

Sử dụng kiểu dữ liệu mới **Time** để in ra thời gian bắt đầu và kết thúc của các tiết học. Gợi ý: sử dụng 2 biến điều khiển thời gian cho thời gian bắt đầu và kết thúc, sau mỗi lần lặp, tăng 2 biến này thêm 1 giờ. Có thể sử dụng cú pháp lặp **for** như sau:

```
for (int i=0, j=1; i <= 10; i++, j++) … … …
```

3. Bài tập 3

Định nghĩa kiểu dữ liệu mới **DongXu** sử dụng cấu trúc **struct** để biểu diễn khoản tiền theo dạng đồng và xu, trong đó xu trong khoảng 0 đến 99, 1 đồng bằng 100 xu.

Viết hàm thực hiện các công việc sau:

1. Nhập dữ liệu đồng và xu cho 1 khoản tiền kiểu **DongXu**
   ```
   void nhapDongXu(DongXu& dx);
   ```
2. In ra khoản tiền theo dạng "**a dong va b xu**"
   ```
   void inDongXu(const DongXu& dx);
   ```
3. Quy chuẩn khoản tiền, nếu là 4 đồng 120 xu đưa về 5 đồng 20 xu
   ```
   void quyChuanDongXu(DongXu& dx);
   ```
4. Tính tổng 2 khoản tiền
   ```
   void tongDongXu(const DongXu& dx1,
                   const DongXu& dx2,
                         DongXu& dxTong);
   ```
5. Tính khoản tiền khác biết giữa 2 khoản tiền
   ```
   void diffDongXu(const DongXu& dx1,
                   const DongXu& dx2,
                         DongXu& dxDiff);
   ```
6. So sánh nhỏ hơn hai khoản tiền (trả về đúng nếu **dx1** nhỏ hơn **dx2**)
   ```
   bool soSanhNhoHonDongXu(const DongXu& dx1,
                           const DongXu& dx2);
   ```

Viết chương trình kiểm tra các hàm trên đã cài đặt đúng hay chưa.

Lưu ý: phải thực hiện tính toán trên các giá trị đồng và xu, không được phép đổi hết ra xu và tính toán trên giá trị xu này.

Nhập vào một mảng các khoản tiền kiểu **DongXu**, theo cả 2 cách sau:

- Đọc dữ liệu từ bàn phím
- Đọc dữ liệu từ file (có thể phải sửa hàm **nhapDongXu**)

Nếu coi mỗi khoản tiền trên là số tiền trong ví của mỗi sinh viên trong lớp, gộp các khoản tiền này lại và đem chia đều cho tất cả sinh viên trong lớp, thì mỗi sinh viên được bao nhiêu tiền. Sau đó xác định xem những sinh viên được lợi nhất và những sinh viên bị thiệt nhất với cách chia như trên. Người thực hiện việc chia đều như trên được trả bao nhiêu tiền?

Sắp xếp mảng các khoản tiền theo thứ tự giảm dần và in ra mảng sau khi sắp xếp.

Lần lượt nhập vào từng khoản tiền, sau đó loại khỏi mảng đã sắp xếp (ở trên) khoản tiền gần nhất với khoản tiền vừa nhập.

Nâng cao: Cài đặt toán tử so sánh để có thể sử dụng hàm **sort** trong thư viện **algorithm** để sắp xếp mảng các khoản tiền kiểu **DongXu**.

4. Bài tập 4

Định nghĩa kiểu dữ liệu mới **SoPhuc** sử dụng cấu trúc **struct** để biểu diễn số phức theo dạng $a + bi$ trong đó a biểu diễn phần thực, b biểu diễn phần ảo, và $i = \sqrt{-1}$.
Viết hàm thực hiện các công việc sau:

1. Nhập dữ liệu phần thực và phần ảo cho 1 số phức kiểu **SoPhuc**
   ```
   void nhapSoPhuc(SoPhuc& sp);
   ```
2. In ra số phức theo dạng "**a + bi**"
   ```
   void inSoPhuc(const SoPhuc& sp);
   ```
3. Tính tổng 2 số phức
   ```
   void tongSoPhuc(const SoPhuc& sp1,
                   const SoPhuc& sp2,
                         SoPhuc& spTong);
   ```
4. Tính hiệu 2 số phức
   ```
   void hieuSoPhuc(const SoPhuc& sp1,
                   const SoPhuc& sp2,
                         SoPhuc& spHieu);
   ```
5. Tính tích 2 số phức
   ```
   void tichSoPhuc(const SoPhuc& sp1,
                   const SoPhuc& sp2,
                         SoPhuc& spTich);
   ```
6. Tính thương 2 số phức
   ```
   void thuongSoPhuc(const SoPhuc& sp1,
                     const SoPhuc& sp2,
                           SoPhuc& spThuong);
   ```

Viết chương trình kiểm tra các hàm trên đã cài đặt đúng hay chưa.
Nhập vào một mảng các khoản tiền kiểu **SoPhuc**, theo cả 2 cách sau:

- Đọc dữ liệu từ bàn phím
- Đọc dữ liệu từ file (có thể phải sửa hàm **nhapSoPhuc**)

Nếu coi mỗi khoản tiền trên là số tiền trong ví của mỗi sinh viên trong lớp, gộp các khoản tiền này lại và đem chia đều cho tất cả sinh viên trong lớp, thì mỗi sinh viên được bao nhiêu tiền. Sau đó xác định xem những sinh viên được lợi nhất và những sinh viên bị thiệt nhất với cách chia như trên. Người thực hiện việc chia đều như trên được trả bao nhiêu tiền?
Sắp xếp mảng các số phức theo thứ tự tăng dần của môđun và in ra mảng sau sắp xếp.
Tính tổng của mảng các số phức.
Xác định các cặp số phức có tích bằng 9 hoặc -9.

5. Bài tập 5

Định nghĩa kiểu dữ liệu mới **`Matrix`** sử dụng cấu trúc **`struct`** để biểu diễn ma trận có kích thước $M \times N$

Viết hàm thực hiện các công việc sau:

1. Nhập dữ liệu cho 1 ma trận kiểu **`Matrix`**
   ```
   void nhapMatrix(Matrix& mt);
   ```
2. In ra ma trận
   ```
   void inMatrix(const Matrix& mt);
   ```
3. Tính tổng 2 ma trận
   ```
   void tongMatrix(const Matrix& mt1,
                   const Matrix& mt2,
                         Matrix& mtTong);
   ```
4. Tính tích 2 ma trận
   ```
   void tichMatrix(const Matrix& mt1,
                   const Matrix& mt2,
                         Matrix& mtTich);
   ```
5. Tính ma trận chuyển vị
   ```
   void tranformMatrix(const Matrix& mt, Matrix& mtTran);
   ```

Viết chương trình kiểm tra các hàm trên đã cài đặt đúng hay chưa.

6. Bài tập 6

Định nghĩa kiểu dữ liệu mới **`SinhVien`** sử dụng cấu trúc **`struct`** để biểu diễn thông tin của sinh viên bao gồm:

- Mã số sinh viên: 8 ký tự chữ số
- Họ và tên
- Điểm 3 môn học:
  - Tin học cơ sở 4 (4 tín chỉ)
  - Tiếng anh chuyên ngành (2 tín chỉ)
  - Cấu trúc dữ liệu (3 tín chỉ)
- Điểm trung bình cộng của 3 môn học trên (số tín chỉ là hệ số môn học)

Viết hàm thực hiện các công việc sau:

1. Nhập dữ liệu ngẫu nhiên cho 1 sinh viên (yêu cầu: điểm số các môn học trong khoảng từ 0.0 đến 10.0, và có một chữ số ở phần thập phân)
   ```
   void nhapSinhVien(SinhVien& sv);
   ```
2. Tính điểm trung bình cộng 03 môn học cho 1 sinh viên
   ```
   void tinhTBC(SinhVien& sv);
   ```
3. Xác định các sinh viên đạt yêu cầu về điểm số dựa trên các tiêu chí sau:
   - Điểm trung bình cộng không dưới 4.0
   - Không có môn nào dưới 3.0
   - Có nhiều nhất một môn dưới 4.0
4. In danh sách sinh viên (bao gồm toàn bộ các thông tin) vào file định dạng **`.csv`**, sắp xếp theo thứ tự họ và tên từ A đến Z