Lab 1: Ôn tập

1 Con trỏ

Sử dụng kĩ thuật con trỏ để cài đặt các hàm sau:

1. Viết hàm nhập vào một mảng số nguyên gồm n phần tử với a là con trỏ trỏ tới vùng nhớ của mảng vừa nhập:
   - `void inputArray(int* &a, int &n)`
2. Viết hàm hủy cấp phát động cho mảng:
   - `void dellocateArray(int* &a)`
3. Viết hàm in ra màn hình các giá trị trong mảng:
   - `void printArray(int* a, int n)`
4. Viết hàm tìm giá trị nhỏ nhất trong mảng:
   - `int findMin(int* a, int n)`
5. Viết hàm tìm phần tử có trị tuyệt đối lớn nhất trong mảng:
   - `int findMaxModulus(int* a, int n)`
6. Viết hàm kiểm tra xem mảng có tăng dần hay không:
   - `bool isAscending(int* a, int n)`
7. Viết hàm tính tổng các phần tử trong mảng:
   - `int sumOfArray(int* a, int n)`
8. Viết hàm đếm số lượng số nguyên tố trong mảng:
   - `int countPrime(int* a, int n)`
9. Viết hàm đảo ngược mảng mà không dùng mảng phụ:
   - `void ReverseArray(int* &a, int n)`

   Từ câu 10. đến 13. yêu cầu tìm kiếm vị trí của giá trị `key` cho trước. Trả về vị trí đầu tiên tìm được. Nếu không tìm được trả về -1.
10. Tìm kiếm tuần tự:
    - `int linearSearch(int* a, int n, int key)`
11. Tìm kiếm tuần tự (sử dụng phương pháp lính canh):
    - `int sentinelLinearSearch(int* a, int n, int key)`
12. Tìm kiếm nhị phân:
    - `int binarySearch(int* a, int n, int key)`
13. Tìm kiếm nhị phân (sự dụng đệ quy):
    - `int recursiveBinarySearch(int* a, int left, int right, int key)`

2 Đệ quy

Sử dụng kỹ thuật Đệ quy để giải quyết các yêu cầu sau (sinh viên có thể khai báo thêm các hàm bổ trợ):

1. Tính tổng bình phương các số tự nhiên nhỏ hơn hoặc bằng n: $S = 1^2 + 2^2 + ... + n^2$.
   - `int sumOfSquares(int n)`
2. Tìm ước chung lớn nhất của 2 số nguyên a, b:
   - `int gcd(int a, int b)`
3. Xác định một mảng có phải là đối xứng:
   - `bool isPalindrome(int a[], int n)`
4. Tính giai thừa cho một số:
   - `int Factorial(int n)`
5. Đếm số chữ số của một số nguyên:
   - `int countDigit(int a)`
6. Số Fibonacci thứ n được tính như sau: $F(n) = F(n-1) + F(n-2)$. Tìm số Fibonacci thứ n:
   - `int FIB(int n)`

3 Xử lý tập tin

3.1 Mô tả dữ liệu

Dữ liệu được dùng trong bài tập thực hành là dữ liệu về điểm thi THPT của một tỉnh (thông tin thật của thí sinh đã được thay đổi).

Tập tin được cung cấp *"data.txt"* có một phần nội dung như sau:

```
Số Báo Danh, Họ và Tên, Toán, Ngữ Văn, Vật Lý, Hóa Học, Sinh Học, Lịch Sử, Địa Lý, GDCD, KHTN, KHXH, Ngoại Ngữ, Ghi Chú, Tỉnh
BD1200000,,8.6,6.5,4.0,7.25,5.5,,,,,,8.4,N1,BinhDinh
BD1200001,,4.0,5.0,,,,4.25,7.0,7.75,,,2.0,N1,BinhDinh
BD1200002,,7.0,6.25,6.0,6.25,6.5,,,,,,5.2,N1,BinhDinh
BD1200003,,5.2,5.75,,,,5.75,7.25,9.25,,,4.6,N1,BinhDinh
BD1200004,,7.6,6.25,7.0,6.5,4.5,,,,,,6.2,N1,BinhDinh
BD1200005,,8.6,6.5,4.0,7.25,5.5,,,,,,8.4,N1,BinhDinh
```

Trong đó:

- Dòng đầu tiên thể hiện tên các trường thông tin có trong tập tin.
- Những dòng tiếp theo thể hiện thông tin thí sinh, mỗi trường thông tin cách nhau bởi 1 dấu phẩy ",".
- Trường thông tin về Họ và tên thí sinh đã được bỏ.
- Những trường thông tin về điểm được bỏ trống nghĩa là thí sinh không tham gia thi môn đó. Để đơn giản hoá phần bài làm, khi đọc thông tin thí sinh, những trường thông tin về điểm được bỏ trống sẽ được lưu trữ trong struct mặc định là 0.
- Điểm ở trường thông tin KHTN và KHXH, sinh viên đọc phần **Một số lưu ý** để biết thêm thông tin.

3.2 Lập trình

Cho struct `Examinee` được định nghĩa như sau:

```
// Examinee.h
struct Examinee
{
    string id;
    float math, literature, physic, chemistry, biology, history, geography, civic_education, natural_science,
        social_science, foreign_language;
};
```

Thực hiện các yêu cầu sau:

1. Đọc thông tin một thí sinh:
   - `Examinee readExaminee(string line_info);`
   - Input: `line_info` - một dòng dữ liệu được đọc từ tập tin *"data.txt"* chứa thông tin của một thí sinh.
   - Output: Biến dữ liệu kiểu `Examinee` lưu trữ thông tin của thí sinh.
2. Đọc thông tin danh sách thí sinh:
   - `vector<Examinee> readExamineeList(string file_name);`
   - Input: `file_name` - tên tập tin đầu vào, trong trường hợp này là *"data.txt"*.
   - Output: Biến dữ liệu kiểu `vector<Examinee>` lưu trữ danh sách thí sinh đọc được từ tập tin.
3. Ghi xuống tập tin thông tin về tổng điểm thi của các thí sinh:
   - `void writeTotal(vector<Examinee> examinee_list, string out_file_name);`
   - Input: `examinee_list` - danh sách các thí sinh
     `out_file_name` - tên tập tin ghi xuống
   - Trong hàm thực hiện tính tổng điểm thi của thí sinh và ghi xuống tập tin *out_file_name* theo format sau:
     – Mỗi thông tin của thí sinh được ghi trên 1 dòng.
     – Thông tin thí sinh bao gồm ID và điểm tổng được cách nhau bởi 1 khoảng trắng.
     – Ví dụ:

       XX001 42.0
       XX002 38.5
       ...
       XX999 23.25

Điểm tổng sẽ được tính như sau:

- Điểm tổ hợp KHTN và KHXH trong tập tin *data.txt* mặc định là không có, do đó sinh viên cần tính điểm KHTN và KHXH để lưu vào struct `Examinee`.
- Điểm KHTN = Lý + Hóa + Sinh
- Điểm KHXH = Sử + Địa + GDCD
- Điểm tổng = Toán + Văn + Ngoại ngữ + KHTN + KHXH

4 Danh sách liên kết

Cho một danh sách liên kết đơn được định nghĩa như sau:

```
struct NODE{
    int key;
    NODE* p_next;
};
```

```
struct List{
    NODE* p_head;
    NODE* p_tail;
};
```

Hãy viết hàm thực hiện các yêu cầu sau:

1. Khởi tạo một `NODE` từ một số nguyên cho trước:
   - `NODE* createNode(int data)`
2. Khởi tạo `List` từ một `NODE` cho trước:
   - `List* createList(NODE* p_node)`
3. Chèn một số nguyên vào đầu một `List` cho trước:
   - `bool addHead(List* &L, int data)`
4. Chèn một số nguyên vào cuối một `List` cho trước:
   - `bool addTail(List* &L, int data)`
5. Xóa `NODE` đầu tiên của một `List` cho trước:
   - `void removeHead(List* &L)`
6. Xóa `NODE` cuối cùng của một `List` cho trước:
   - `void removeTail(List* &L)`
7. Xóa tất cả các `NODE` của một `List` cho trước:
   - `void removeAll(List* &L)`
8. In tất cả phần tử của một `List` cho trước:
   - `void printList(List* L)`
9. Đếm số lượng phần tử của một `List` cho trước:
   - `int countElements(List* L);`
10. Đảo một `List` cho trước *(tạo ra một* `List` *mới)*:
    - `List* reverseList(List* L)`
11. Xóa tất cả các phần từ trùng của một `List` cho trước:
    - `void removeDuplicate(List* &L)`
12. Xóa giá trị `key` khỏi một `List` cho trước:
    - `bool removeElement(List* &L, int key)`

5 Stack - Queue

Cho định nghĩa struct của một node trong danh sách liên kết đơn như sau:

```
struct NODE{
    int key;
    NODE* pNext;
};
```

Sử dụng Danh sách liên kết phía trên, định nghĩa cấu trúc dữ liệu cho Ngăn xếp và Hàng đợi, sau đó cài đặt hàm thực hiện các chức năng sau đây:

1. Ngăn xếp
   - **Khởi tạo** một ngăn xếp từ một giá trị cho trước.
   - **Push** một giá trị vào ngăn xếp.
   - **Pop** một phần tử ra khỏi ngăn xếp, trả về giá trị của phần tử.
   - **Đếm** số lượng phần tử có trong ngăn xếp.
   - Xác định một ngăn xếp có **rỗng** hay không.
2. Hàng đợi
   - **Khởi tạo** một hàng đợi từ một giá trị cho trước.
   - **Enqueue** một giá trị vào hàng đợi.
   - **Dequeue** một phần tử ra khỏi hàng đợi, trả về giá trị của phần tử đó.
   - **Đếm** số lượng phần tử có trong hàng đợi.
   - Xác định một hàng đợi có **rỗng** hay không.