Lab 1: Ôn tập

1 Con trỏ

1. Viết hàm nhập vào một mảng số nguyên gồm n phần tử
   - `void inputArray(int* a, int n)`
2. Viết hàm in các giá trị của mảng ra màn hình.
   - `void printArray(int* a, int n);`
3. Viết hàm tìm giá trị nhỏ nhất trong mảng:
   - `int FindMin(int* a, int n);`
4. Viết hàm tìm phần tử có trị tuyệt đối lớn nhất trong mảng:
   - `int FindMaxModulus(int* a, int n);`
5. Viết hàm đếm số lần xuất hiện của giá trị bất kì trong mảng.
   - `int countValue(int* a, int n, int key)`
6. Viết hàm đảo ngược chuỗi và in ra màn hình.
   - `void StrRev(char* a)`
7. Viết hàm in hoa chữ cái đầu của từng từ và in ra màn hình. VD "Cau TRuC dU liEu" -> "Cau Truc Du Lieu".
   - `void PrettyStr(char* a)`

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 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)`

4 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.