Lab 2: Ôn tập

1 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)`

13. Chèn một số nguyên vào vị trí bất kì một `List` cho trước:
    - `bool addPos(List* L, int data, int pos)`
14. Xóa `NODE` ở vị trí bất kì của một `List` cho trước:
    - `void removePos(List* L, int pos)`
15. Cho 2 `List` đã được sắp xếp tăng dần. Ghép 2 `List` lại thành một `List` mới được sắp xếp tăng dần.
    - `List* ConcatList(List* L1, List* L2)`
16. Cho 2 `List` đã được sắp xếp tăng dần. Ghép `List` 2 vào `List` 1 mà vẫn giữ sắp xếp tăng dần.
    - `void MergeList(List* L1, List* L2)`

2 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. Viết hàm đọc theo một chuỗi ký tự và in chúng theo thứ tự ngược lại. Sử dụng một ngăn xếp.
    - `void ReverseString(char* s)`
2. Viết chương trình đọc trong một chuỗi ký tự và xác định xem dấu ngoặc đơn (), dấu ngoặc nhọn `{}` và dấu ngoặc vuông [ ] của nó có "cân bằng" hay không.

   VD:

   Input: "(ab))"

   Output: false
    - `void CheckBalance(char* s)`
3. Viết chương trình đọc một số nguyên dương và in ra biểu diễn nhị phân của số nguyên đó VD:

   Input: 2

   Output: "0000000000000010"
    - `void DecToBin(int dec)`