Danh Sách Liên Kết

MSSV	20280083
Tên	Lại Toàn Thắng

1. Biên dịch đoạn chương trình nêu trên.
2. Cho biết kết quả in ra màn hình khi người dùng nhập vào các dữ liệu sau:

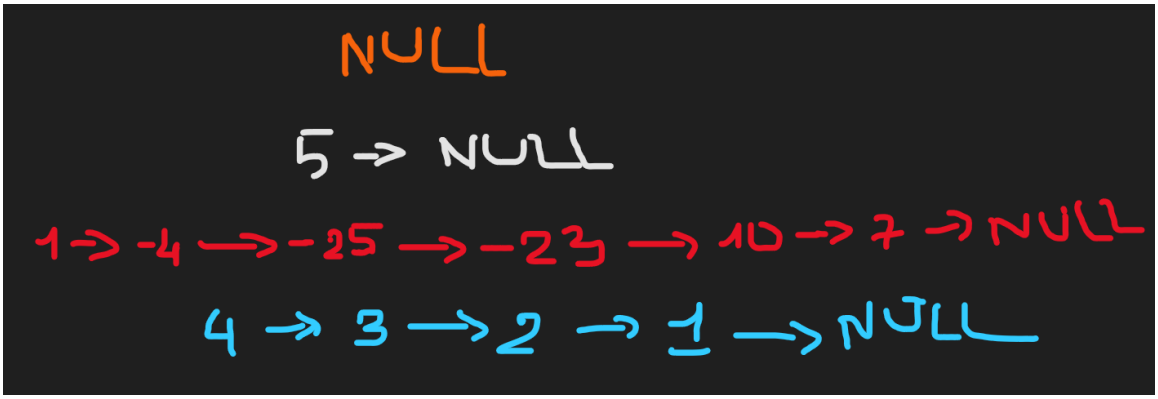
```
-1
5 -1
7 10 -23 -25 -4 1 -1
1 2 3 4 -1
```

- Danh sách liên kết rỗng
- List có một phần tử: "5"
- List: "1 -4 -25 -23 10 7"
- List: "4 3 2 1"

3. Nêu nhận xét ngắn gọn mối liên hệ giữa thứ tự nhập dữ liệu vào với thứ tự in dữ liệu ra màn hình.

Thứ tự nhập dữ liệu vào với thứ tự in dữ liệu ra màn hình ngược nhau.

4. Vẽ hình danh sách liên kết theo dữ liệu được nhập ở câu 2.



5. Nếu trong hàm main (_tmain) thứ tự hai dòng lệnh sau đây bị hoán đổi cho nhau thì kết quả kết xuất ra màn hình sẽ như thế nào đối với dữ liệu câu 2? Giải thích lý do?

```
//Ghi chu
PrintList(pRoot);
//Ghi chu
RemoveAll(pRoot);
```

Nếu thứ tự hai dòng lệnh trên bị hoán đổi thì kết quả xuất ra màn hình sẽ luôn là rỗng. Vì lệnh RemoveAll(pRoot) đã xoá bỏ các phần tử trong danh sách liên kết ra khỏi bộ nhớ.

6. Nếu trong hàm main (_tmain) vòng lặp do…while được thay đổi như dưới đây thì kết quả kết xuất ra màn hình sẽ như thế nào đối với dữ liệu câu 2? Giải thích lý do?

```
do
{
	printf("Nhap vao du lieu, -1 de ket thuc: ");
	scanf("%d", &Data);
	AddHead(pRoot, Data);
	if (Data == -1)
		break;
}while (Data != -1);
```

Nếu thay đổi như vậy thì List (trước khi thay đổi do...while) sẽ AddHead thêm phần tử -1. Tức List mới = "-1" + List cũ

7. Với các hàm CreateNode, AddHead được cung cấp sẵn, hãy cho biết ý nghĩa của các giá trị trả về của hàm.

- Đối với hàm CreateNode
  - Giá trị trả về của hàm là một giá trị con trỏ Node với kiểu dữ liệu là Node*
  - Ý nghĩa: Hình thành một node mới với liên kết kế tiếp là NULL
- Đối với hàm AddHead
  - Giá trị trả về của hàm là kiểu dữ liệu bool, tức TRUE or FALSE
  - Ý nghĩa:
    - trả về FALSE khi bộ nhớ không có đủ dung lượng để cấp phát một con trỏ mới
    - Trả về TRUE khi đã cấp phát thành công và liên kết Node đó với List và con trỏ vừa tạo là pHead của List.

8. Hãy ghi chú các thông tin bằng cách trả lời các câu hỏi ứng với các dòng lệnh có yêu cầu ghi chú (//Ghi chú) trong các hàm RemoveAll, PrintList, _tmain.

```
pHead = NULL; // Khởi tạo lại cho pHead giá trị NULL, nếu không thì ảnh hưởng đến pHead thật trong caller.
// Hiểu cách khác thì gán pHead lại bằng NULL thì pHead mới thật sự là NULL, nếu không gán thì pHead
// chưa phải là NULL mà pHead chỉ mang địa chỉ của NULL (trỏ vào NULL) - pHead vẫn còn giữ giá trị
// địa chỉ (vẫn chi bộ nhớ cho pHead)
```

9. Kết quả sẽ như thế nào nếu hàm RemoveAll được thay đổi như dưới đây? Giải thích lý do

```
void RemoveAll(NODE* &pHead)
{
	while (pHead != NULL)
	{
		pHead = pHead->pNext;
		delete pHead;
	}
	pHead = NULL; //Ghi chu: Tại sao phải thực hiện phép gán này?
}
```

Kết quả trên màn hình console sẽ vẫn giống như cũ nhưng chương trình sẽ không kết thúc do lệnh RemoveAll ở trên.

Giải thích: lệnh delete pHead; xoá bỏ phần tử pHead ra khỏi bộ nhớ. Sau đó thì ... không có sau đó nữa. Nói chung là cần một con trỏ pCur để trỏ đến vị trị pHead để con trỏ pHead di chuyển về sau. Sau đó mình muốn làm gì con trỏ pCur thì làm thuii 😂

10. Giá trị cuối cùng của biến pRoot trong đoạn chương trình mẫu là gì? Giải thích lý do.

Giá trị cuối cùng của biến pRoot là NULL vì như đã biết, lệnh RemoveAll (Node* &pHead) kết thúc bằng con trỏ pHead = NULL. Vậy nên RemoveAll(pRoot) sẽ tương tự như vậy.

Áp dụng – Nâng cao

1. Bổ sung chương trình mẫu cho phép tính **tổng giá trị** các phần tử trên danh sách liên kết đơn gồm các giá trị nguyên.

Gợi ý: tham khảo hàm PrintList để viết hàm **SumList**.

```cpp
int SumList(NODE* pHead)
{
    int s = 0;
    NODE* pCur = pHead;
    while (pCur != nullptr)
    {
        s = s + pCur->Key;
        pCur = pCur->pNext;
    }
    return s;
}
```

2. Bổ sung chương trình mẫu cho phép tìm **giá trị nguyên lớn nhất** trong số các phần tử nguyên trên danh sách liên kết đơn gồm các giá trị nguyên.

Gợi ý: tham khảo hàm `PrintList` để viết hàm **`MaxList`**.

```cpp
int MaxList(NODE* pHead)
{
    int max = pHead->Key;
    NODE* pCur = pHead->pNext;
    while (pCur != nullptr)
    {
        if (pCur->Key > max)
            max = pCur->Key;
        pCur = pCur->pNext;
    }
    return max;
}
```

3. Bổ sung chương trình mẫu cho phép tính **số lượng các phần tử** của danh sách liên kết đơn gồm các giá trị nguyên.

Gợi ý: tham khảo hàm `PrintList` để viết hàm **`CountList`**.

```cpp
int CountList(NODE* pHead)
{
    int count = 0;
    NODE* pCur = pHead;
    while (pCur != nullptr){
        count++;
        pCur = pCur->pNext;
    }
    return count;
}
```

4. Bổ sung chương trình mẫu cho phép **thêm vào cuối** danh sách liên kết đơn một giá trị nguyên. Gợi ý: tham khảo hàm `AddHead` để viết hàm **`AddTail`**.

```cpp
bool AddTail(NODE* &pHead, int data)
{
    NODE* p = nullptr;
    p = CreateNode(data);
    if (p == nullptr)
        return false;
    if (pHead == nullptr)
        pHead = p;
    NODE* pCur = pHead;
    while (pCur->pNext != nullptr)
    {
        pCur = pCur->pNext;
    }
    pCur->pNext = p;
    return true;
}
```

5. Bổ sung chương trình mẫu cho phép **xóa phần tử đầu** danh sách liên kết đơn.

```
void DeleteFirst(NODE* &pHead)
{
    NODE* pCur = pHead;
    pHead = pHead->pNext;
    delete pCur;
}
```

6. Bổ sung chương trình mẫu cho phép **xóa phần tử cuối** danh sách liên kết đơn.

```
void DeleteTail(NODE* &pHead)
{
    NODE* pCur = pHead;
    while (pCur->pNext->pNext != nullptr)
    {
        pCur = pCur->pNext;
    }
    NODE* pDel = pCur->pNext;
    delete pDel;
    pCur->pNext = nullptr;
}
```

7. Bổ sung chương trình mẫu cho biết **số lượng các phần tử** trên danh sách liên kết đơn có giá trị trùng với giá trị ***x*** được cho trước.

Gợi ý: tham khảo thao tác duyệt danh sách liên kết trong hàm **`PrintList`**.

```
int Count_Duplicate(NODE* pHead, int x)
{
    int count = 0;
    for (NODE* i = pHead; i != nullptr; i = i->pNext)
        if (i->Key == x)
            count++;
    return count;
}
```

8. Bổ sung chương trình mẫu cho phép tạo một danh sách liên kết đơn gồm các phần tử mang giá trị nguyên trong đó không có cặp phần tử nào mang giá trị giống nhau.

Gợi ý: sử dụng hàm `AddHead` hoặc `AddTail` có bổ sung thao tác kiểm tra phần tử giống nhau.

```
void Not_Duplicate(NODE*& pRoot)
{
    int Data;
    do
    {
        printf("Nhap vao du lieu, -1 de ket thuc: ");
        scanf("%d", &Data);

        if (Data == -1)
            break;
        if (pRoot == nullptr)
            AddHead(pRoot, Data);
        else {
            int count = 0;
            for (NODE* i = pRoot; i != nullptr; i = i->pNext) {
                if (i->Key == Data)
                    count++;
            }
            if (count == 0) AddHead(pRoot, Data);
        }
    } while (Data != -1);
    printf("\nDu lieu da duoc nhap: \n");
    PrintList(pRoot);
}
```

9. Cho sẵn một danh sách liên kết đơn gồm các phần tử mang giá trị nguyên và một giá trị nguyên x. Hãy tách danh sách liên kết đã cho thành 2 danh sách liên kết: một danh sách gồm các phần tử có giá trị nhỏ hơn giá trị x và một danh sách gồm các phần tử có giá trị lớn hơn giá trị x. Giải quyết trong 2 trường hợp:

a. Danh sách liên kết ban đầu không cần tồn tại.
b. Danh sách liên kết ban đầu bắt buộc phải tồn tại.

```
void SeparateList(NODE* pHead, int x)
{
    NODE* pHead1 = nullptr; // List contains elements smaller than x
    NODE* pHead2 = nullptr; // List contains elements more than x
    for (NODE* i = pHead; i != nullptr; i = i->pNext)
    {
        if (i->Key < x)
            AddTail(pHead1, i->Key);
        if (i->Key > x)
            AddTail(pHead2, i->Key);
    }
    cout << "\nLess than 5: ";
    PrintList(pHead1);
    cout << endl;
    cout << "\nGreater than 5: ";
    cout << endl;
    PrintList(pHead2);
}
```