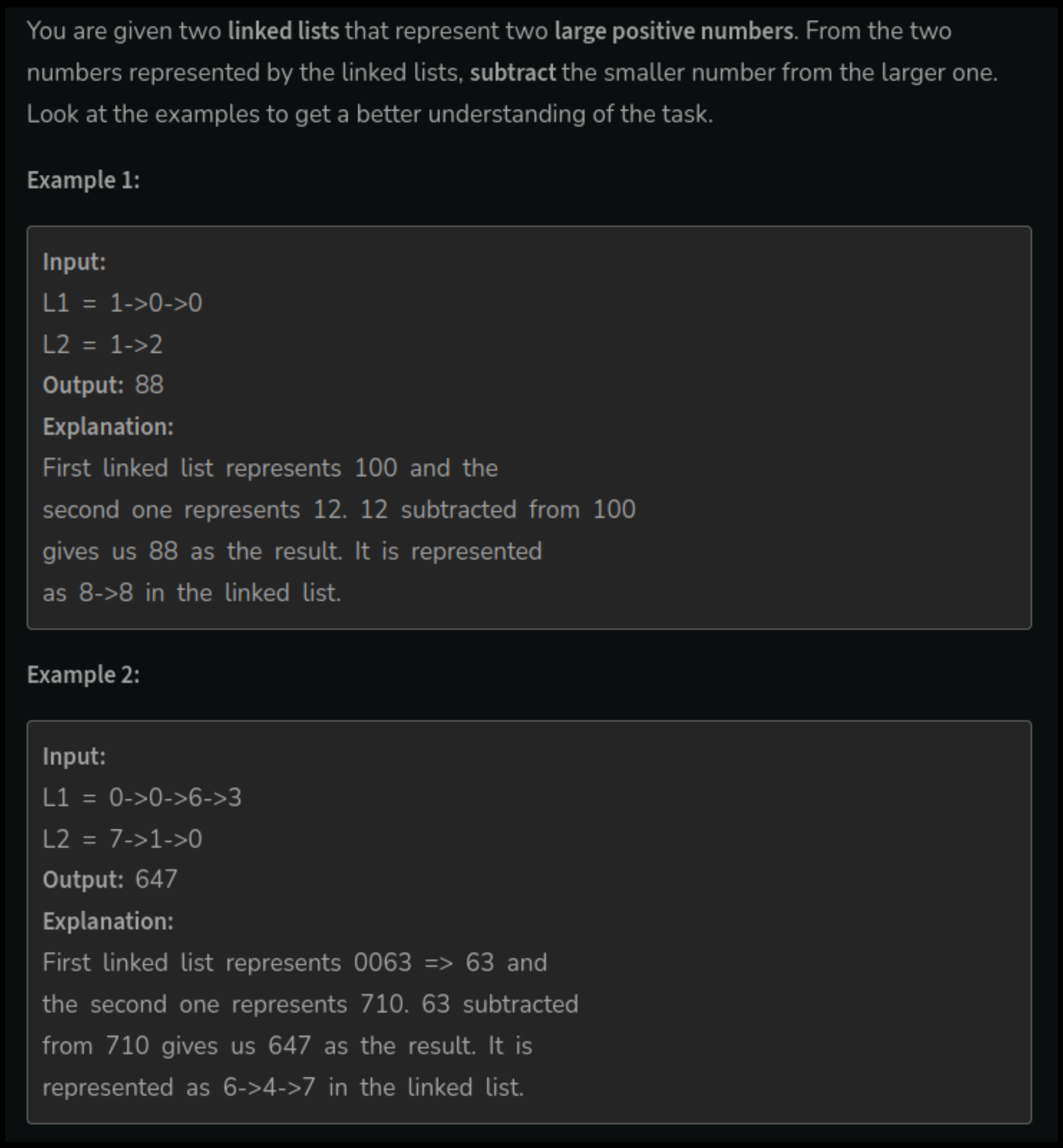
④ Subtraction in Linked List



You are given two **linked lists** that represent two **large positive numbers**. From the two numbers represented by the linked lists, **subtract** the smaller number from the larger one. Look at the examples to get a better understanding of the task.

Example 1:

```
Input:
L1 = 1->0->0
L2 = 1->2
Output: 88
Explanation:
First linked list represents 100 and the
second one represents 12. 12 subtracted from 100
gives us 88 as the result. It is represented
as 8->8 in the linked list.
```

Example 2:

```
Input:
L1 = 0->0->6->3
L2 = 7->1->0
Output: 647
Explanation:
First linked list represents 0063 => 63 and
the second one represents 710. 63 subtracted
from 710 gives us 647 as the result. It is
represented as 6->4->7 in the linked list.
```

L1 = $1 \rightarrow 0 \rightarrow 0$ L2 = $1 \rightarrow 2$.

Output : $8 \rightarrow 8$

Brute force :-

→ Convert linked list to vector then subtract it.

→ Again convert vector to linked list and return the ans.

```
class Solution {
public:
    vector<int> subtract(vector<int>&v1,vector<int>&v2){
        int borrow=0,i=v1.size()-1,j=v2.size()-1;
        vector<int>p;
        while(i>=0&& j>=0){
            if(v1[i]>=v2[j]){
                 borrow=0;
                 p.push_back(v1[i]-v2[j]);
                 i--,j--;
            }
            else{
                int k=i-1;
                while(k>=0&&v1[k]==0){
                    v1[k]=9,k--;
                }
                if(k>=0&&v1[k]>0){
                    v1[k]-=1;
                }
                p.push_back(v1[i]+10-v2[j]);
                i--,j--;
            }
        }
        while(i>=0){
            if(borrow+v1[i]>=0){
                borrow=0;
                 p.push_back(v1[i]+borrow);
                 i--;
            }
        }
        return p;
    }
```

```
Node* subLinkedList(Node* head1, Node* head2) {
        vector<int>v1,v2;
        while(head1!=NULL&& head1->data==0){
            head1=head1->next;
        }
         while(head2!=NULL&& head2->data==0){
            head2=head2->next;
        }
        while(head1!=NULL) {
            v1.push_back(head1->data);
            head1=head1->next;
        }
        while(head2!=NULL) {
            v2.push_back(head2->data);
            head2=head2->next;
        }
        vector<int>ans;
        if(v1.size()>v2.size()){
              ans=subtract(v1,v2);
        }
        else if(v1.size()==v2.size()){
            int i=0,j=0;
            while(i<v1.size() && v1[i]==v2[i]) i++;
            if(i<v1.size()){
                if(v1[i]>v2[i]) ans=subtract(v1,v2);
                else ans=subtract(v2,v1);
            }
        }
        else{
            ans=subtract(v2,v1);
        }
        int i=0;
        reverse(ans.begin(),ans.end());
        while(i<ans.size()&& ans[i]==0) i++;
        for(;i<ans.size();i++){
            cout<<ans[i];
        }
        if(ans.size()==0) cout<<"0";
        return NULL;
    }
};
```

Better Approach :-

→ Reverse linked list

→ Subtract

→ Reverse & return the result.

```
class Solution {
public:
    int len(Node* temp){
        int n = 0;
        while(temp){
            n++;
            temp = temp->next;
        }
        return n;
    }

    Node *Reverse(Node* temp){
        Node* prev = NULL;
        while(temp){
            Node* save = temp->next;
            temp->next = prev;
            prev = temp;
            temp = save;
        }
        return prev;
    }

    pair<Node*, Node*> GreaterSmaller(Node* head1, Node* head2){

        int n1 = len(head1);
        int n2 = len(head2);

        Node *Greater = head1;
        Node *Smaller = head2;
        if(n1<n2)
        {
            Smaller = head1;
            Greater = head2;
        }
        else if (n1==n2)
        {
            Node* check1 = head1;
            Node* check2 = head2;
            while(check1->data==check2->data && check1->next!=NULL){
                check1=check1->next;
                check2=check2->next;
            }
            if(check1->data<check2->data && check1!=NULL){
                Greater = head2;
                Smaller = head1;
            }
        }
        return {Smaller, Greater};
    }
    Node* subLinkedList(Node* head1, Node* head2) {

        // Clear the leading Zeroes
        while(head1->data==0 && head1->next){
            head1=head1->next;
        }
        while(head2->data==0 && head2->next){
            head2=head2->next;
        }

        Node *Smaller = GreaterSmaller(head1, head2).first;
        Node *Greater = GreaterSmaller(head1, head2).second;

        Smaller = Reverse(Smaller);
        Greater = Reverse(Greater);

        string answer = "";
        int carry = 0;

        while(Smaller!=NULL){
            int sub = Greater->data - Smaller->data - carry;
            if(sub<0){
                carry = 1;
                answer.push_back((10+sub)+'0');
            }else{
                answer.push_back((sub+'0'));
                carry = 0;
            }
            Greater = Greater->next;
            Smaller = Smaller->next;
        }

        while(Greater!=NULL){
            int sub = Greater->data-carry;
            if(sub<0){
                carry = 1;
                answer.push_back((10+sub)+'0');
            }else{
                answer.push_back((sub+'0'));
                carry = 0;
            }
            Greater = Greater->next;
        }

        // Make ans LL using current LL
        Node* ans = NULL;
        Node* current = NULL;
        while (answer.size() > 0) {
            long long add = (answer[answer.size() - 1] - '0');
            if (ans == NULL) {
                ans = new Node(add);
                current = ans;
            } else {
                current->next = new Node(add);
                current = current->next;
            }
            answer.pop_back();
        }

        // Remove leading zeroes from ans;
        while(ans->data==0 && ans->next){
            ans=ans->next;
        }
        return ans;
    }
};
```