

A. 删除重复的节点

描述

给定一个按升序排列的链表，链表中可能包含一些重复的节点。

请编写一个函数，删除链表中所有重复的节点，使得最终链表中只包含**唯一出现过的节点**。

注意，重复的节点**不保留**。

输入

输入一个已经排序的链表。

输出

输出删除重复节点后的链表，**重复的节点不保留**。

输入样例 1

```
1->2->3->3->4->4->5->NULL
```

输出样例 1

```
1->2->5->NULL
```

输入样例 2

```
1->1->1->1->2->3->NULL
```

输出样例 2

```
2->3->NULL
```

提示

```
#include <iostream>
#include <sstream>
#include <string>
#include <cctype>
using namespace std;

struct ListNode{
    int val;
    ListNode *next;
}

ListNode (int x): val(x), next(NULL) {} // 在 struct 中也可以写构造函数，用于对数据的初始化。例如，ListNode x(5) 表示：创建结构体变量 x，其中 x.val 为 5，x.next 为空。
```

```
//create_list 函数用于从输入数据中创建链表，并返回链表头指针
ListNode* create_list(const string &s) {
    stringstream ss(s);
    string token;
    ListNode* head = NULL;
    ListNode* tail = NULL;
    while(getline(ss, token, '-')) {
        while(!token.empty() && !isdigit(token[0])) {// 删除非数字的字符
            token.erase(token.begin());
        }
    }
}
```

```
if(token.empty() || token == "NULL") continue;

int num = stoi(token);

ListNode* node = new ListNode(num);

if(!head) {

    head = node;

    tail = node;

} else {

    tail->next = node;

    tail = node;

}

return head;

}
```

//delete_duplicates 函数用于删除链表中的重复结点，并返回新链表的头指针

```
ListNode* delete_duplicates(ListNode* head) {

//本函数需要你补充完整

}
```

//print_list 函数用于将链表打印出来

```
void print_list(ListNode* head){

    ListNode* cur = head;

    while(cur) {

        cout << cur->val;

    }

}
```

```
    if(cur->next) cout << "->";
    cur = cur->next;
}

cout << "->NULL" << endl;

}

int main(){
    string list_str;
    getline(cin, list_str); // 从输入中读取一整行数据到 list_str 中
    ListNode* head = create_list(list_str);
    ListNode* new_head = delete_duplicates(head);
    print_list(new_head);
    return 0;
}
```

B. 第一个公共结点

描述

输入两个链表，编写一个函数来找出它们的第一个公共结点。

第一个公共结点的定义：从这一结点开始，两个链表的后续结点序列的值相同。

如果两个链表没有公共结点，则返回 **NULL**。

输入

输入为两个链表。保证两个链表不会完全相同。

输出

输出为第一个公共结点。

输入样例 1

```
a1->a2->c1->c2->c3->NULL
```

```
b1->b2->b3->c1->c2->c3->NULL
```

输出样例 1

```
c1
```

输入样例 2

```
b2->a2->b1->d1->d1->b1->c1->NULL
```

```
d1->d1->b1->c1->NULL
```

输出样例 2

```
d1
```

提示

```
#include <iostream>
#include <sstream>
#include <string>
#include <utility>

using namespace std;
```

// 本题使用的数据结构为双链表，每个结点都有 prev 和 next 指针，可以从链表的任意位置向前或向后遍历。使用 node->prev 访问 node 结点的前一结点，使用 node->next 访问 node 结点的后一结点。

// 头结点（链表的第一个结点）的 prev 指针和尾结点（最后一个结点）的 next 指针均为 NULL。

```
struct ListNode {
    string val;           // string 类，用于处理字符串
    ListNode *next; // 指向下一个结点的指针
    ListNode *prev; // 指向前一个结点的指针
    ListNode(string x) : val(x), prev(NULL), next(NULL) {} // struct 中的构造函数，用于对链表结点的初始化
};
```

//buildList 函数用于从输入字符串 input 创建链表，创建好的链表头指针存储在 head 中，尾指针存储在 tail 中

```
void buildList(const string& input, ListNode*& head, ListNode*& tail) {
```

```
if (input.empty())
{
    head = NULL;
    tail = NULL;
    return;
}

// 移除结尾的 "->NULL"

const string tail_str = "->NULL";

string trimmedInput = input;

if (trimmedInput.size() >= tail_str.size() && trimmedInput.substr(trimmedInput.size() - tail_str.size()) == tail_str) {

    trimmedInput = trimmedInput.substr(0, trimmedInput.size() - tail_str.size());
}

// 分割字符串并构建链表

stringstream ss(trimmedInput);

string item;

ListNode* dummyHead = new ListNode("");
ListNode* current = dummyHead;
tail = dummyHead;

while (getline(ss, item, '-')) {

    if (item.back() == '>') {

        item.pop_back();
    }

    if (!item.empty() && (item.front() == '>' && item.size() > 1)) {
```

```
        item = item.substr(1);

    }

    if (!item.empty()) {

        ListNode* newNode = new ListNode(item);

        current->next = newNode;

        newNode->prev = current;

        current = current->next;

        tail = newNode;

    }

}

head = dummyHead->next;

return;

}

//findFirstCommonNode 函数从两条链表（A 和 B）中寻找第一个公共结点，  
并返回指向公共结点的指针

ListNode* findFirstCommonNode(ListNode* headA, ListNode* tailA, ListNode* headB, ListNode* tailB) {

//本函数需要你补充完整

}

int main() {

    string lineA, lineB;

    getline(cin, lineA); //从输入中读取一整行数据到 lineA 中

    getline(cin, lineB); //从输入中读取一整行数据到 lineB 中
```

```
ListNode *headA, *tailA;  
ListNode *headB, *tailB;  
headA = tailA = headB = tailB = NULL;  
buildList(lineA, headA, tailA); //从输入数据中创建链表  
buildList(lineB, headB, tailB); //从输入数据中创建链表  
  
ListNode* commonNode = findFirstCommonNode(headA, tailA, head  
B, tailB);  
if (commonNode != NULL) {  
    cout << commonNode->val << endl;  
} else {  
    cout << "NULL" << endl;  
}  
return 0;  
}
```

C. 把字符串转换成整数

描述

请你写一个函数 `StrToInt`, 实现把字符串转换成整数这个功能。

注意以下各种情况, 都需要处理:

- 1) 空字符串: 空输入。
- 2) 普通数字: 基本转换。
- 3) 前导零: 去掉前导零的逻辑。
- 4) 负号: 需处理负号。
- 5) 前导空格: 空格忽略。
- 6) 非数字字符: 只提取数字部分。

输入

输入一个字符串。

输出

输出转换后的整数。

输入样例 1

```
"123"
```

输出样例 1

```
123
```

输入样例 2

```
""
```

输出样例 2

```
0
```

输入样例 3

```
" -000123abc"
```

输出样例 3

```
-123
```

提示

```
#include <iostream>
#include <string>
using namespace std;
```

```
struct CharNode {
```

```
    char val;
```

```
    CharNode* next;
```

```
    CharNode(char c): val(c), next(NULL) {} // struct 中的构造函数，用于  
    对结点的初始化
```

```
};
```

//StringToList 函数将字符串（string 类）转化为链表，该函数返回链表的头指针

```
CharNode* StringToList(const string& str) {  
    CharNode dummy(0);  
    CharNode* tail = &dummy;  
  
    for (int i = 0; i < str.size(); ++i) {  
  
        char c = str[i];  
        CharNode* newNode = new CharNode(c);  
        tail->next = newNode;  
        tail = newNode;  
    }  
  
    return dummy.next;  
}
```

//FreeList 函数清理链表

```
void FreeList(CharNode* head) {  
    while (head) {  
        CharNode* temp = head;  
        head = head->next;  
        delete temp;  
    }  
}
```

// StrToInt 函数将链表转换为字符串（string 类）

```
string StrToInt(CharNode* head) {  
    // 本函数需要你补充完整
```

```
}

int main() {
    string input;
    getline(cin, input); //从输入中读取一整行数据到 input 中
    if (!input.empty() && input.front() == '"') {
        input = input.substr(1); // 去掉输入字符串中开头的引号
    }
    if (!input.empty() && input.back() == '"') {
        input = input.substr(0, input.size() - 1); // 去掉输入字符串中末尾
        的引号
    }
    CharNode* head = StringToList(input);
    string result = StrToInt(head);
    cout << result << endl;
    FreeList(head);
    return 0;
}
```

D. 左旋字符串

描述

字符串的左旋转操作是把字符串前面的若干个字符转移到字符串的尾部。

请编写一个函数实现字符串左旋转操作的功能。

比如输入字符串 "abcdefg" 和数字 2，该函数将返回左旋转 2 位得到的结果 "cdefgab" 。

输入

输入一个字符串和一个整数 n 。

输出

输出左旋转后的字符串。

输入样例 1

```
"abcdefg"
```

```
2
```

输出样例 1

```
"cdefgab"
```

提示

```
#include <iostream>
#include <string>
using namespace std;
struct CharNode {
```

```
char val;  
CharNode* next;  
  
CharNode(char c): val(c), next(NULL){}  
};  
  
// 将字符串转为链表  
  
CharNode* StringToList(const string& str) {  
  
    CharNode dummy(0);  
  
    CharNode* tail = &dummy;  
  
    for (int i = 0; i < str.size(); ++i) {  
  
        CharNode* newNode = new CharNode(str[i]);  
  
        tail->next = newNode;  
  
        tail = newNode;  
    }  
  
    return dummy.next;  
}  
  
// 获取链表长度  
  
int ListLength(CharNode* head) {  
  
    int length = 0;  
  
    while (head) {  
  
        length++;  
  
        head = head->next;  
    }  
  
    return length;  
}
```

```
// 链表转为字符串

string ListToString(CharNode* head) {

    string result = "";
    while (head) {
        result += head->val;
        head = head->next;
    }
    return result;
}

// 释放链表内存

void FreeList(CharNode* head) {

    while (head) {
        CharNode* temp = head;
        head = head->next;
        delete temp;
    }
}

//左旋操作

CharNode* LeftRotate(CharNode* head, int n) {

    //本函数需要你补充完整
}

int main() {

    string s;
```

```
getline(cin, s);           // 读取带引号的字符串，比如 "abcdefg"  
s = s.substr(1, s.size() - 2); // 去掉前后的引号  
  
string n_str;  
  
getline(cin, n_str);       // 读取数字 n  
  
int n = stoi(n_str);      // 转换为整数  
  
// 字符串 → 链表  
  
CharNode* head = StringToList(s);  
  
// 左旋操作  
  
CharNode* rotated = LeftRotate(head, n);  
  
// 输出结果  
  
cout << ListToString(rotated) << endl;  
  
// 清理内存  
  
FreeList(rotated);  
  
return 0;  
}
```