链表

7-1 两个有序链表序列的合并

已知两个非降序链表序列S1与S2，设计函数构造出S1与S2合并后的新的非降序链表S3。

输入格式:

输入分两行，分别在每行给出由若干个正整数构成的非降序序列，用-1表示序列的结尾（-1不属于这个序列）。数字用空格间隔。

输出格式:

在一行中输出合并后新的非降序链表，数字间用空格分开，结尾不能有多余空格；若新链表为空，输出`NULL`。

输入样例:

```
1 3 5 -1
2 4 6 8 10 -1
```

输出样例:

```
1 2 3 4 5 6 8 10
```

代码:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node *Node;
struct Node
{
    int data;
    struct Node *Next;
};
Node CList()
{
    int d;
    Node head = (Node)malloc(sizeof(struct Node)),q;
    head -> Next  = NULL;
    q = head;
    while(~scanf("%d",&d)&&d!=-1)
    {
        Node p = (Node)malloc(sizeof(struct Node));
        p -> data = d;
        p -> Next = NULL;
        q -> Next = p;
        q = p;
    }
    return head;
```

```
}
Node Merge(Node a,Node b)
{
    a = a -> Next;
    b = b -> Next;
    Node head = (Node)malloc(sizeof(struct Node));
    head -> Next = NULL;
    Node q = head;
    while(a || b)
    {
        Node p = (Node)malloc(sizeof(struct Node));
        p -> Next = NULL;
        if(a == NULL || a -> data > b -> data)
        {
            p -> data = b -> data;
            b = b -> Next;
            q -> Next = p;
            q = p;
        }
        else
        {
            p -> data = a -> data;
            a = a -> Next;
            q -> Next = p;
            q = p;
        }
    }
    return head;
}
void printL(Node a)
{
    a = a -> Next;
    if(a == NULL)printf("NULL");
    // int flag = 0;
    while(a)
    {
        if(flag)printf(" %d",a -> data);
        else printf("%d",a -> data);
        a = a -> Next;
        // flag = 1;
    }
}
int main()
{
    Node a = CList();
    Node b = CList();
    Node c = Merge(a,b);
    printL(c);
}
```

7-2 两个有序链表序列的交集

已知两个非降序链表序列S1与S2，设计函数构造出S1与S2的交集新链表S3。

输入格式:

输入分两行，分别在每行给出由若干个正整数构成的非降序序列，用-1表示序列的结尾（-1不属于这个序列）。数字用空格间隔。

输出格式:

在一行中输出两个输入序列的交集序列，数字间用空格分开，结尾不能有多余空格；若新链表为空，输出`NULL`。

输入样例:

```
1 2 5 -1
2 4 5 8 10 -1
```

输出样例:

```
2 5
```

代码:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct list{
    int data;
    struct list *next;
}list;
void set(list *head)
{
    if(head==NULL)
    {
        printf("NULL");
    }else
    while(head)
    {
        printf("%d",head->data);
        head=head->next;
        if(head)
        printf(" ");

    }
}
list *Greatlist()
{
    list *t,*p,*head=NULL;
    int n;
    while(1)
    {
        scanf("%d",&n);
        if(n==-1)
        break;
        t=(list*)malloc(sizeof(struct list));
        t->data=n;
```

```
        t->next=NULL;
        if(head==NULL)
        head=t;
        else
        {
            p->next=t;
        }
        p=t;
     }return head;
}
void hed(list *s1,list *s2)
{
    list *s3=NULL,*t,*p;
    while(s1&&s2){
    if(s1->data==s2->data)
    {
        t=(list *)malloc(sizeof(struct list));
        t->data=s1->data;
        t->next=NULL;
        if(s3==NULL)
        s3=t;
        else
        p->next=t;
        p=t;
        s1=s1->next;
        s2=s2->next;
    }else if(s1->data<s2->data)
    {
        s1=s1->next;
    }
    else{
    s2=s2->next;}
}
    set(s3);
}
int main()
{
    list *s1=Greatlist();
    list *s2=Greatlist();
    hed(s1,s2);
}
```

7-3 重排链表

给定一个单链表 $L1→L2→⋯→L**n-1→L**n$，请编写程序将链表重新排列为 $L**n→L1→L**n-1→L2→⋯$。例如：给定L为1→2→3→4→5→6，则输出应该为6→1→5→2→4→3。

输入格式：

每个输入包含1个测试用例。每个测试用例第1行给出第1个结点的地址和结点总个数，即正整数N (≤105)。结点的地址是5位非负整数，NULL地址用−1表示。

接下来有N行，每行格式为：

```
Address Data Next
```

其中`Address`是结点地址；`Data`是该结点保存的数据，为不超过105的正整数；`Next`是下一结点的地址。题目保证给出的链表上至少有两个结点。

输出格式：

对每个测试用例，顺序输出重排后的结果链表，其上每个结点占一行，格式与输入相同。

输入样例：

```
00100 6
00000 4 99999
00100 1 12309
68237 6 -1
33218 3 00000
99999 5 68237
12309 2 33218
```

输出样例：

```
68237 6 00100
00100 1 99999
99999 5 12309
12309 2 00000
00000 4 33218
33218 3 -1
```

代码：

```
#include<stdio.h>

struct Node  // 链表
{
    int next, data;
} node[100001];  // 用下标表示地址

int main()
{
    int start_address, num, temp;
    scanf("%d%d", &start_address, &num);
    for (int i = 0; i < num; i++)  // 输入每个点
    {
        scanf("%d", &temp);
        scanf("%d%d", &node[temp].data, &node[temp].next);
    }

    int order[num]; order[0] = start_address;  // 新建一个数组 order 用来
    //按顺序 存储链表地址
    for (int i = 1;; i++)
    {
        // 如果该节点的指向-1就break,同时更新有效节点的数量
```

```
        // if (node[order[i - 1]].next == -1) {num = i; break;}
        // order[i] = node[order[i - 1]].next;
        if(node[order[i-1]].next=-1){num=i;break;}
        order=node[order[i-1]].next;
    }

    int flag = -1, a = 0, b = num - 1;          // 根据 order 的顺序来重新链接链表,a是头 b是尾;
    for (int i = 1; i < num; i++, flag *= -1)  // flag在-1 1反复横跳来表示从左/从右来连接
    {
        if (flag == -1) {node[order[b--]].next = order[a];}
        else {node[order[a++]].next = order[b];}
    }  // 循环完之后 a,b 记录的就是排序结束后最后一个节点在order里的索引

    // 把最后一个节点的指向设置为-1; temp用来记录当前读取到的节点地址
    // (在这里temp应该为排序后第一个节点,应该为原先链表的最后一位)
    node[order[a]].next = -1; temp = order[num - 1];

    // 就可以依据链表一个一个输出了
    for (int i = 1; i < num; i++)
    {
        printf("%05d %d %05d\n", temp, node[temp].data, node[temp].next);
        temp = node[temp].next;
    }
    printf("%05d %d %d\n", temp, node[temp].data, node[temp].next);
    return 0;
}
```

7-4 约瑟夫环

N个人围成一圈顺序编号，从1号开始按1、2、3……顺序报数，报p者退出圈外，其余的人再从1、2、3开始报数，报p的人再退出圈外，以此类推。
请按退出顺序输出每个退出人的原序号。

输入格式:

输入只有一行，包括一个整数N(1<=N<=3000)及一个整数p(1<=p<=5000)。

输出格式:

按退出顺序输出每个退出人的原序号，数据间以一个空格分隔，但行尾无空格。

输入样例:

在这里给出一组输入。例如：

```
7 3
```

输出样例:

```
3 6 2 7 5 1 4
```

代码:

```
#include<stdio.h>
struct circle{
    int num;
    struct circle *next;
};
int main()
{
    int n,number,i;
    struct circle a[3000],*p,*q;//注意数组长度要足够大
    scanf("%d %d",&n,&number);

    for(i=0;i<n;i++){
        a[i].num=i+1;
    }
    for(i=0;i<n-1;i++){
        a[i].next=&a[i+1];
    }
    a[n-1].next=a;

    /*核心算法*/
    q=p=a;
    while(p!=p->next){
        for(i=0;i<number-1;i++){
            q=p;
            p=p->next;
        }
        q->next =p->next ;
        printf("%d ",p->num);
        p=q->next ;
    }
    printf("%d\n",p->num);
    return 0;
}
```

7-5 单链表的创建及遍历

读入n值及n个整数，建立单链表并遍历输出。

输入格式:

读入n及n个整数。

输出格式:

输出n个整数，以空格分隔（最后一个数的后面没有空格）。

输入样例:

在这里给出一组输入。例如:

```
2
10 5
```

输出样例:

在这里给出相应的输出。例如:

```
10 5
```

代码:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct math
{
    int a;
    struct math*next;
};
int main()
{
    struct math *head=NULL;
    struct math *p,*q;
    int n,i;
    scanf("%d",&n);
    q=(struct math*)malloc(sizeof(struct math));
    head = (struct math*)malloc(sizeof(struct math));
    for(i=0;i<n;i++)
    {
        p=(struct math*)malloc(sizeof(struct math));
        // if(head==NULL)
        // {
        //     head=p;
        // }
        // else
        // {
            q->next=p;
        // }
        p->next=NULL;
        scanf("%d",&p->a);
        q=p;
    }
    int r=0;
    p=head;
    while(p!=NULL)
```

```
    {
        if(r==0)
        {
            printf("%d",p->a);
            r++;
        }
        else
        {
            printf(" %d",p->a);
        }
        p=p->next;
    }
    return 0;
}
```

7-6 链表去重

给定一个带整数键值的链表 L，你需要把其中绝对值重复的键值结点删掉。即对每个键值 K，只有第一个绝对值等于 K 的结点被保留。同时，所有被删除的结点须被保存在另一个链表上。例如给定 L 为 21→-15→-15→-7→15，你需要输出去重后的链表 21→-15→-7，还有被删除的链表 -15→15。

输入格式：

输入在第一行给出 L 的第一个结点的地址和一个正整数 N（≤105，为结点总数）。一个结点的地址是非负的 5 位整数，空地址 NULL 用 -1 来表示。

随后 N 行，每行按以下格式描述一个结点：

```
地址 键值 下一个结点
```

其中`地址`是该结点的地址，`键值`是绝对值不超过104的整数，`下一个结点`是下个结点的地址。

输出格式：

首先输出去重后的链表，然后输出被删除的链表。每个结点占一行，按输入的格式输出。

输入样例：

```
00100 5
99999 -7 87654
23854 -15 00000
87654 15 -1
00000 -15 99999
00100 21 23854
```

输出样例：

```
00100 21 23854
23854 -15 99999
99999 -7 -1
00000 -15 87654
87654 15 -1
```

代码：

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
typedef struct Node
{
    int key;
    int nextid;
}node;
int main()
{
    node no[100000];//存放原始链表
    int n,r1=0,r2=0,id,h1;
    int key[100000]={0},nextid[100000],nextid1[100000];//key数组表示是否遇到过此key绝对值，nextid存放去重后ID，nextid1存放去掉的ID
    cin>>h1>>n;
    for(int i=0;i<n;i++)
    {
        cin>>id;
        cin>>no[id].key>>no[id].nextid;
    }
    while(h1!=-1)//遍历链表
    {
        if(key[abs(no[h1].key)])//如果key绝对值遇到过，将当前ID放入数组nextid1中
        {
            nextid1[r2++]=h1;
            nextid1[r2]=-1;//使存放当前ID的下一个数组元素等于-1，为输出准备
            h1=no[h1].nextid;//改变h1使得链表向后遍历
        }
        else//否则将ID放入数组nextid中
        {
            key[abs(no[h1].key)]=1;
            nextid[r1++]=h1;
            nextid[r1]=-1;
            h1=no[h1].nextid;
        }
    }
    for(int i=0;i<r1;i++)//遍历输出key值去重后的链表
    {
        if(nextid[i+1]!=-1)
            cout<<setw(5)<<setfill('0')<<nextid[i]<<" "<<no[nextid[i]].key<<" "
<<setw(5)<<setfill('0')<<nextid[i+1]<<endl ;
        else//如果nextid[i+1]==-1，则在输出它的时候不需要强制输出五位
            cout<<setw(5)<<setfill('0')<<nextid[i]<<" "<<no[nextid[i]].key<<" "
<<nextid[i+1]<<endl ;
    }
    for(int i=0;i<r2;i++)//遍历输出去掉的链表
    {
        if(nextid1[i+1]!=-1)
            cout<<setw(5)<<setfill('0')<<nextid1[i]<<" "<<no[nextid1[i]].key<<" "
<<setw(5)<<setfill('0')<<nextid1[i+1]<<endl ;
        else
```

```
            cout<<setw(5)<<setfill('0')<<nextid1[i]<<" "<<no[nextid1[i]].key<<" "
<<nextid1[i+1]<<endl ;
    }
    return 0;
}
```

7-7 单链表就地逆置

输入多个整数，以-1作为结束标志，顺序建立一个带头结点的单链表，之后对该单链表进行就地逆置（不增加新结点），并输出逆置后的单链表数据。

输入格式:

首先输入一个正整数T，表示测试数据的组数，然后是T组测试数据。每组测试输入多个整数，以-1作为该组测试的结束（-1不处理）。

输出格式:

对于每组测试，输出逆置后的单链表数据（数据之间留一个空格）。

输入样例:

```
1
1 2 3 4 5 -1
```

输出样例:

```
5 4 3 2 1
```

代码:

```
#include<stdio.h>
#include<string.h>
typedef struct node
{
        int data;
        struct node *next;
}*list;
list creat()//直接逆序建立链表（头插法）
{
        list head,l;
        int elem;
        head=(list)malloc(sizeof(struct node));
        head->next=NULL;

        while(scanf("%d",&elem)&&elem!=-1)
        {
                l=(list)malloc(sizeof(struct node));
                l->data=elem;
                l->next=head->next;
                head->next=l;
```

```
        }
        return head;
}
void print(list head)
{
        list temp,p;
        temp=head->next;
        while(temp->next)
        {
                printf("%d ",temp->data);
                temp=temp->next;
        }
        printf("%d\n",temp->data);
        return ;
}
int main()
{
        int T,i;
        scanf("%d",&T);
        for(i=1;i<=T;i++)
        {
                list head;
                head=creat();
                print(head);
        }
        return 0;
}
```

7-8 带头节点的双向循环链表操作

本题目要求读入一系列整数，依次插入到双向循环链表的头部和尾部，然后顺序和逆序输出链表。

链表节点类型可以定义为

```
typedef int DataType;
typedef struct LinkedNode{
    DataType data;
    struct LinkedNode *prev;
    struct LinkedNode *next;
}LinkedNode;
```

链表类型可以定义为

```
typedef struct LinkedList{
  int length; /* 链表的长度 */
  LinkedNode head; /* 双向循环链表的头节点 */
}LinkedList;
```

初始化链表的函数可声明为

```
void init_list(LinkedList *list);
```

分配节点的函数可声明为

```
LinkedNode *alloc_node(DataType data);
```

头部插入的函数可声明为

```
void push_front(LinkedList *list, DataType data);
```

尾部插入的函数可声明为

```
void push_back(LinkedList *list, DataType data);
```

顺序遍历的函数可声明为

```
void traverse(LinkedList *list);
```

逆序遍历的函数可声明为

```
void traverse_back(LinkedList *list);
```

输入格式:

输入一行整数（空格分隔），以-1结束。

输出格式:

第一行输出链表顺序遍历的结果，第二行输出逆序遍历的结果。

输入样例:

在这里给出一组输入。例如：

```
1 2 3 4 5 6 -1
```

输出样例:

```
5 3 1 2 4 6
6 4 2 1 3 5
```

代码：

```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <cstring>
#include <vector>
#include <queue>
#include <map>
#include <set>
//#include <bits/stdc++.h>
using namespace std;
//#define int long long
```

```
typedef long long ll;
#define mem(a, b) memset(a, b, sizeof(a))
#define PI acos(-1)
#define LLu unsigned long long
#define PLL pair<ll, ll>
#define PII pair<int, int>
#define xx first
#define yy second
#define endl '\n'
#define O_O ios::sync_with_stdio(0),cin.tie(0),cout.tie(0);
int gcd(int a, int b) {return b ? gcd(b, a%b) : a; }
int lcm(int a, int b) {return a/gcd(a, b)*b;}
const int N = 1e6 + 10, INF = 0x3f3f3f3f, mod = 1e9 + 7;
const double eps = 1e-6;
struct node
{
    int data;
    node *next;
    node *pre;
};
int main()
{
    node *head, *tail, *tt, *p;
    head = new node;
    tail = new node;
    tt = new node;
    head -> next = NULL;
    head -> pre = NULL;
    tail = head;
    tt = head;
    int x, cnt = 0, n = 0;
    while(cin >> x && x != -1)
    {
        n ++;
        p = new node;
        p -> next = NULL;
        p -> pre = NULL;
        p -> data = x;
        cnt ++;
        if(cnt & 1)
        {
            tail -> pre = p;
            p -> next = tail;
            tail = p;
        }
        else
        {
            tt -> next = p;
            p -> pre = tt;
            tt = p;
        }
    }
    int m = n;
    while(tail && m)
    {
        if(tail == head)
```

```
        {
            tail = tail -> next;
            continue;
        }
        m --;
        if(m)
        cout << tail -> data << " ";
        else cout << tail -> data << endl;
        tail = tail -> next;
    }
    m = n;
    while(tt && m)
    {
        if(tt == head)
        {
            tt = tt -> pre;
            continue;
        }
        m --;
        if(m)
        cout << tt -> data << " ";
        else cout << tt -> data << endl;
        tt = tt -> pre;
    }
    return 0;
}
```

7-9 头插法创建单链表、遍历链表、删除链表

输入一系列自然数（0和正整数），输入-1时表示输入结束。按照输入的顺序，用头插法建立单链表，并遍历所建立的单链表，输出这些数据。注意 -1 不加入链表。

输入格式:

第一行是一个正整数k，表示以下会有k组测试数据。

每组测试数据是一系列以空格隔开的自然数（0和正整数）。数列末尾的 -1 表示本组测试数据结束。按照输入的顺序，用头插法建立单链表，并遍历所建立的单链表，输出这些数据。注意 -1 不加入链表。

输出格式:

对于每组测试数据，输出链表中各节点的数据域。每个数据后有一个空格。每组测试数据的输出占1行。

输入样例:

```
3
1 2 3 4 5 -1
30 20 10 -1
4 2 2 1 1 2 0 2 -1
```

输出样例:

在这里给出相应的输出。例如:

```
5 4 3 2 1
10 20 30
2 0 2 1 1 2 2 4
```

注意:对每组测试数据,创建链表,遍历链表输出之后,一定要删除链表,否则会出现“内存超限”。

代码:

```
#include <stdio.h>
#include <stdlib.h>
// 定义单链表节点结构
struct Node {
    int data; // 数据域
    struct Node* next; // 指针域
};
// 创建链表的函数
struct Node* createLinkedList() {
    struct Node* head = NULL;
    int value;

    // 输入链表元素
    while (1) {
        scanf("%d", &value);

        // 判断是否插入节点
        if (value == -1) {
            break;
        }
        // 头插法
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = value;
        newNode->next = head;
        head = newNode;
    }
    return head;
}
// 遍历链表
void traverseAndPrint(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

// 删除链表释放内存
void deleteLinkedList(struct Node* head) {
    struct Node* current = head;
```

```c
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }
}
int main() {
    // 输入链表节点个数
    int k;
    scanf("%d", &k);
    // 输入元素
    for (int i = 0; i < k; i++) {
        struct Node* head = createLinkedList();
        traverseAndPrint(head);
        deleteLinkedList(head);
    }
    return 0;
}
```