

## 单向链表习题

- 1: 分别采用头插法和尾插法创建一个非空单向链表（上课讲过）
- 2: 遍历一个非空单向链表（正向、逆向）（上课讲过）
- 3: 判断一个非空单向链表是否带环？

```
#include <stdio.h>

#include <stdlib.h>

#include "e:\\fan\\type.h"    //加载 ElemSN 类型
#include "e:\\fan\\link.h"    //创建以及输出链表的函数
//一个单向链表如果带环，只可能从尾部带环，所以一定无 NULL

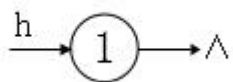
int IsHasRing(ElemSN * h)
{
    ElemSN * pl, * ps; //分别表示长短指针
    pl=ps=h;
    while(pl&&pl->next){
        //若不带环则链表有 NULL，但要区分奇数和偶数节点个数
        pl=pl->next->next; //长指针一次跑两个节点
        ps=ps->next;      //短指针一次只跑一个节点
        if(pl==ps){       //若连个指针指向同一个节点则有环
            return 1;
        }
    }

    return 0;             //若 p 为 NULL 或 p->next 为 NULL 则无环
}
```

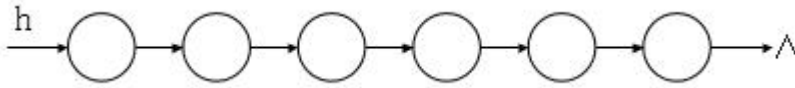
```
}  
  
int main(void)  
{  
    int a[]={3,2,5,8,4,7,6,9};  
    int n=sizeof(a)/sizeof(a[0]);  
    ElemSN * head;  
    head=CreateLink(a,n); //创建单向链表  
    //若不加下面两条语句则无环，加上则带环  
    //for(ElemSN * t=head;t->next;t=t->next);  
    //t->next=head->next->next->next;  
    if(IsHasRing(head)){  
        printf("YES!\n");  
    }  
    else{  
        printf("NO!\n");  
    }  
    return 0;  
}
```

4: 返回一个非空单向链表的中间结点（上课讲过）

若只有一个结点，返回该节点



若有偶数个结点，返回后半段起始节点



5: 返回两个非空单向链表是否有公共交点

若无, 返回 NULL

若有, 返回第一个公共交点

答案: //本段代码是演示两个链表有公共交点

```
#include <stdio.h>
#include <stdlib.h>
#include "e:\\fan\\type.h" //加载 ElemSN 类型
#include "e:\\fan\\link.h" //创建以及输出链表的函数
//模拟生成两个有公共交点的链表, 第一个公共交点是 6
void Intersect(ElemSN * h1, ElemSN * h2)
{
    ElemSN * p, * q;
    int i;
    for(p=h1, i=0; i<5; p=p->next, ++i);
    for(q=h2; q->next; q=q->next);
    q->next=p->next;
}

//判断两个链表是否有公共交点
ElemSN * IsIntersect(ElemSN * h1, ElemSN * h2)
{
    int cnt=0, flag=0;
```

```
ElemSN * p=h1,*q=h2,*t=h1;
//计算两个链表的节点个数之差
for(t=h1;t;cnt++, t=t->next);
for(t=h2;t;cnt--, t=t->next);
//先把长的链表长的部分跑完
t=h1;
if(cnt<0){
flag=1;
t=h2;
cnt=-cnt;
}
while(cnt>0){
    t=t->next;
    cnt--;
}
flag?(q=t):(p=t);
//两个链表再同步向前跑
while(p!=q){
    p=p->next;
    q=q->next;
}
return p;//这里也可以是 q
```

```
}

int main(void)
{
    int a1[]={3, 2, 5, 8, 4, 7, 6, 9};
    int n1=sizeof(a1)/sizeof(a1[0]);
    int a2[]={5, 2, 7, 8};
    int n2=sizeof(a2)/sizeof(a2[0]);
    ElemSN * head1,* head2,* p;
    //生成两个链表
    head1=CreateLink(a1,n1);
    head2=CreateLink(a2,n2);
    //模拟生成两个相交链表，第一个公共交点是 6
    Intersect(head1, head2);
    //输出两个有公共交点的链表
    PrintLink(head1);
    PrintLink(head2);
    //判断两个链表是否相交
    p=IsIntersect(head1, head2);
    if(p) {
        printf("有公共交点，第一个公共交点是: %d!\n", p->data);
    }
    else{
```

```
    printf("没有公共交点!\n");  
}  
  
return 0;  
}
```

\*\*\*\*\*以下是无公共交点的代码演示\*\*\*\*\*

```
//没有公共交点代码，更详细的注释请参考有公共交点  
#include <stdio.h>  
#include <stdlib.h>  
#include "e:\\fan\\type.h" //加载 ElemSN 类型  
#include "e:\\fan\\link.h" //创建以及输出链表的函数  
//判断两个链表是否有公共交点  
ElemSN * IsIntersect(ElemSN * h1, ElemSN * h2)  
{  
    int cnt=0, flag=0;  
    ElemSN * p=h1, *q=h2, *t=h1;  
    for(t=h1; t; cnt++, t=t->next);  
    for(t=h2; t; cnt--, t=t->next);  
    t=h1;  
    if(cnt<0) {  
        flag=1;  
    }
```

```
t=h2;

cnt=-cnt;

}

while(cnt>0) {

    t=t->next;

    cnt--;

}

flag?(q=t):(p=t);
while(p!=q) {

    p=p->next;

    q=q->next;

}

return p;

}

int main(void)

{

    int a1[]={3, 2, 5, 8, 4, 7, 6, 9};

    int n1=sizeof(a1)/sizeof(a1[0]);

    int a2[]={5, 2, 7, 8};

    int n2=sizeof(a2)/sizeof(a2[0]);

    ElemSN * head1,* head2,* p;
```

```
head1=CreateLink(a1,n1);
head2=CreateLink(a2,n2);
PrintLink(head1);
PrintLink(head2);
//这里这创建了两个链表，所以没有公共交点，是两个独立链表
p=IsIntersect(head1,head2);
if(p){
    printf("有公共交点，第一个公共交点是%d!\n",p->data);
}
else{
    printf("没有公共交点!\n");
}
return 0;
}
```

注：本代码只是演示过程，注意力不要集中在如何生成有公共交点的链表上，只要搞明白 IsIntersect(ElemSN \* h1, ElemSN \* h2) 函数的算法过程即可

这是方法 A    时间复杂度  $O(n)$     空间复杂度  $O(1)$

6: 设 head 指向一个非空单向链表，数据域值均为正整数，将该链表拆分成一个奇数链，一个偶数链

答案: #include <stdio.h>  
#include <stdlib.h>



```
#include "e:\\fan\\type.h"    //加载 ElemSN 类型
#include "e:\\fan\\link.h"    //创建以及输出链表的函数
//将一个非空链表拆分成一个奇数链，一个偶数链
ElemSN * SplitLink(ElemSN ** hh)
{
    ElemSN * hodd, * heven, * p, * q, * ins;
    hodd=*hh;
    heven=NULL;
    p=*hh;
    while(p) { //将偶数节点从原链表中取出，头插法建一个偶数链
        if (p->data%2==0) {
            ins=p;
            if (p==*hh)    *hh=p=(*hh)->next;
            else    p=q->next=p->next;
            ins->next=heven;
            heven=ins;
        }
        else {
            q=p;
            p=p->next;
        }
    }
}
```

```
    }  
    return heven;//返回偶数链  
}  
  
int main(void)  
{  
    int a1[]={3, 2, 5, 8, 4, 7, 6, 9};  
    int n1=sizeof(a1)/sizeof(a1[0]);  
    ElemSN * head,*head_odd,*head_even;  
    head=CreateLink(a1,n1);  
    PrintLink(head);  
    head_even=SplitLink(&head);  
    printf("奇数链: ");  
    PrintLink(head);  
    printf("偶数链: ");  
    PrintLink(head_even);  
    return 0;  
}
```

7: 将两个升序连合并成一个升序链

答案: #include <stdio.h>

#include <stdlib.h>

#include "e:\\fan\\type.h" //加载 ElemSN 类型

#include "e:\\fan\\link.h" //创建以及输出链表的函数

//请参考合并两个升序顺序表，算法和思路都相同

ElemSN \* MergeLink(ElemSN \* h1, ElemSN \* h2)

{

ElemSN \* hn=NULL, \*t, \*p;

while(h1 || h2) { //直至两个链表全部处理完

if(!h2 || h1 && h1->data < h2->data) {

//若第二个链表为空或两个链表都不空第一个链表的首元结点小于  
第二个链表的首元结点

p=h1;

h1=h1->next;

}

else { //否则取第二个链表的首元结点

p=h2;

h2=h2->next;

}

if(!hn) { //尾插法插入新链表

hn=t=p;

}

else {

t=t->next=p;

```
    }  
}  
  
return hn;//返回新链表的头指针  
}  
  
int main(void)  
{  
    int a1[]={2,4,6,8};  
    int n1=sizeof(a1)/sizeof(a1[0]);  
    int a2[]={3,9,11};  
    int n2=sizeof(a2)/sizeof(a2[0]);  
    ElemSN * head1,* head2,*head;  
    //生成两个链表  
    head1=CreateLink(a1,n1);  
    head2=CreateLink(a2,n2);  
    PrintLink(head1);  
    PrintLink(head2);  
    //合并两个升序连  
    head=MergeLink(head1,head2);  
    PrintLink(head);  
    return 0;  
}
```

8: 设顺序表 a, 其数据元素值无序且重复

按照顺序表 a 生成一个升序有序且不含重复值的单向链表

答案: #include <stdio.h>

#include <stdlib.h>

#include "e:\\fan\\type.h" //加载 ElemSN 类型

#include "e:\\fan\\link.h" //输出链表的函数

ElemSN \* OrderedLink(int a[], int n)

{

ElemSN \* h=NULL, \* p, \* q, \* ins;

for(int i=0; i<n; ++i) {

for(p=h; p && p->data < a[i]; q=p, p=p->next);

//在已生成的链表中查找插入位置

if(!p || p->data != a[i]) {

//p 为空要插入、p 不为空且 p 指向的节点的 data 不等于当前  
元素要插入

ins=(ElemSN \*)malloc(sizeof(ElemSN)); //生成新节点

ins->data=a[i];

if(p==h) { //如果 p 等于 h---头插

ins->next=h;

h=ins;

}

else { //中间或尾插

ins->next=q->next;

```
        q->next=ins;
    }
}

}

return h;
}

int main(void)
{
    int a[]={2, 2, 3, 2, 5, 2, 3, 3, 5, 7, 8, 9, 6, 7, 2, 5, 9, 2};
    int n=sizeof(a)/sizeof(a[0]);
    ElemSN * head;
    head=OrderedLink(a,n);
    PrintLink(head);
    return 0;
}
```

9: 有一个数据域值均为正整数的非空单向链表

i>: 将所有奇数结点移至偶数结点之前

ii>: 将比首元结点数据域值小的结点移至首元结点之前

**【递归】**课后布置的作业题

1、任意输入一个正整数  $X$  ( $0 < X \leq 2147483647$ )

(1) 返回  $X$  多个位上数字之和;

例:  $X=1234$     返回: 10

(2) 返回  $X$  的逆置数

例:  $X=1234$     返回: 4321 (不是输出)

(3) 返回  $1+2+3+4+5+\cdots+n$  之和

答案:

```
#include <stdio.h>

int SumNum(int x)
{
    if(x==0){
        return 0;
    }
    else{
        return SumNum(x/10)+x%10;
    }
}

int PreNum(int x,int y)
{
    if(x==0){
        return y;
    }
    else{
        y=y*10+x%10;
        return PreNum(x/10,y);
    }
}

int Sum(int n)
{
    if(n==0){
        return 0;
    }
    else{
        return Sum(n-1)+n;
    }
}

int main(void)
{
    printf("%d\n",Sum(100));
    return 0;
}
```

## 2、设有顺序表 a ( $0 < \text{长度} \leq 9$ )

数据域的值均为正整数且介于 1~9 之间，将该顺序表数据元素的值生成一个逆置数。

要求：递归方程、递归代码

答案：



```
int PreNumArray(int a[],int n)
{
    if(n==0){
        return 0;
    }
    else{
        return PreNumArray(a+1,n-1)*10+a[0];
    }
}
```

3、返回顺序表的最大值。

要求：递归方程、递归代码

```
int MaxValueArray(int a[],int n)
{
    if(n==1){
        return a[0];
    }
    else{
        int max=MaxValueArray(a+1,n-1);
        return a[0]>max?a[0]:max;
    }
}

int main(void)
{
    int a[]={1,2,3,4,5};
    int n=sizeof(a)/sizeof(a[0]);
    printf("%d\n",MaxValueArray(a,n));
    return 0;
}
```

答案：

4、设 head 指向一个非空单向链表（无重复值，无返回值）

(1) 返回节点个数

(2) 返回奇数节点个数

(3) 返回最大值

(4) 任意输入一个关键字 key, 返回关键字为 key 的结点

答案：

```

#include <stdio.h>
#include <stdlib.h>

typedef struct node{
    int data;
    struct node * next;
}ElemSN;

ElemSN * CreateLink(int a[],int n)
{
    ElemSN * h,* ins,* t;
    h=NULL;
    for(int i=0;i<n;++i){
        ins=(ElemSN *)malloc(sizeof(ElemSN));
        ins->data=a[i];
        ins->next=NULL;
        if(!h){
            h=t=ins;
        }
        else{
            t=t->next=ins;
        }
    }
    return h;
}

int CountNode(ElemSN * h)
{
    if(!h){
        return 0;
    }
    else{
        return CountNode(h->next)+1;
    }
}

int CountOddNode(ElemSN * h)
{
    if(!h){
        return 0;
    }
    else{
        return CountOddNode(h->next)+h->data%2;
    }
}

int MaxValue(ElemSN * h)
{
    if(h->next==NULL){
        return h->data;
    }
    else{
        int max=MaxValue(h->next);
        return h->data>max ? h->data :max;
    }
}

ElemSN * FindKeyNodeA(ElemSN * h,int key)
{
    if(h==NULL){
        return NULL;
    }
    else{
        if(h->data==key){
            return h;
        }
        else{
            return FindKeyNodeA(h->next,key);
        }
    }
}

ElemSN * FindKeyNodeB(ElemSN * h,int key)
{
    if(h==NULL||h->data==key){
        return h;
    }
    else{
        return FindKeyNodeB(h->next,key);
    }
}

int main(void)
{
    int a[]={3,2,5,8,4};
    int n=sizeof(a)/sizeof(a[0]);
    ElemSN * head,* pkey;
    head=CreateLink(a,n);
    if(pkey=FindKeyNodeB(head,5)){
        printf("%d\n",pkey->data);
    }
    else{
        printf("not found!\n");
    }
    return 0;
}

```