实验2：线性表及其应

**计算机166班李彪6130116194**



**、实验 的**

帮助学 掌握线性表的基本操作在顺序和链表这两种存储结构上的实现，尤以链表的操作和应 作为重点。

**、问题描述**

1. 构造 个空的线性表L。

2. 查找第i个元素。

3. 按值查找。

4. 在线性表L的第i个元素之前插 新的元素e；

5. 在线性表L中删除第i个元素，并 e返回其值。

6. 求表 。

7. 输出线性表中的所有元素。

**三、实验要求**

1. 分别利 顺序和链表存储结构实现线性表的存储，并设计出在不同的存储

结构中线性表的基本操作算法。

2. 在实验过程中，对相同的操作在不同的存储结构下的时间复杂度和空间复杂度进 分析。

**四、实验环境**

 PC微机

 debianlinux

 编译器：gcc(GNUCCompiler)

 实现语 ：C

1/13

**五、实验步骤**



**顺序结构**

1. 程序的主要思路：因为顺序结构具有随机存取的特点，所以在直接取值的时候，是 常的 便，可以直接按照⻆标拿到值，但由于存储空间连续，所以在插 值和删除值时需要对之后的元素进 遍历并依次移位。具体实现可 代码以及注释。

2. 主要函数及基本思路：

 init\_array()初始化O(1)

 delete()删除O(n)

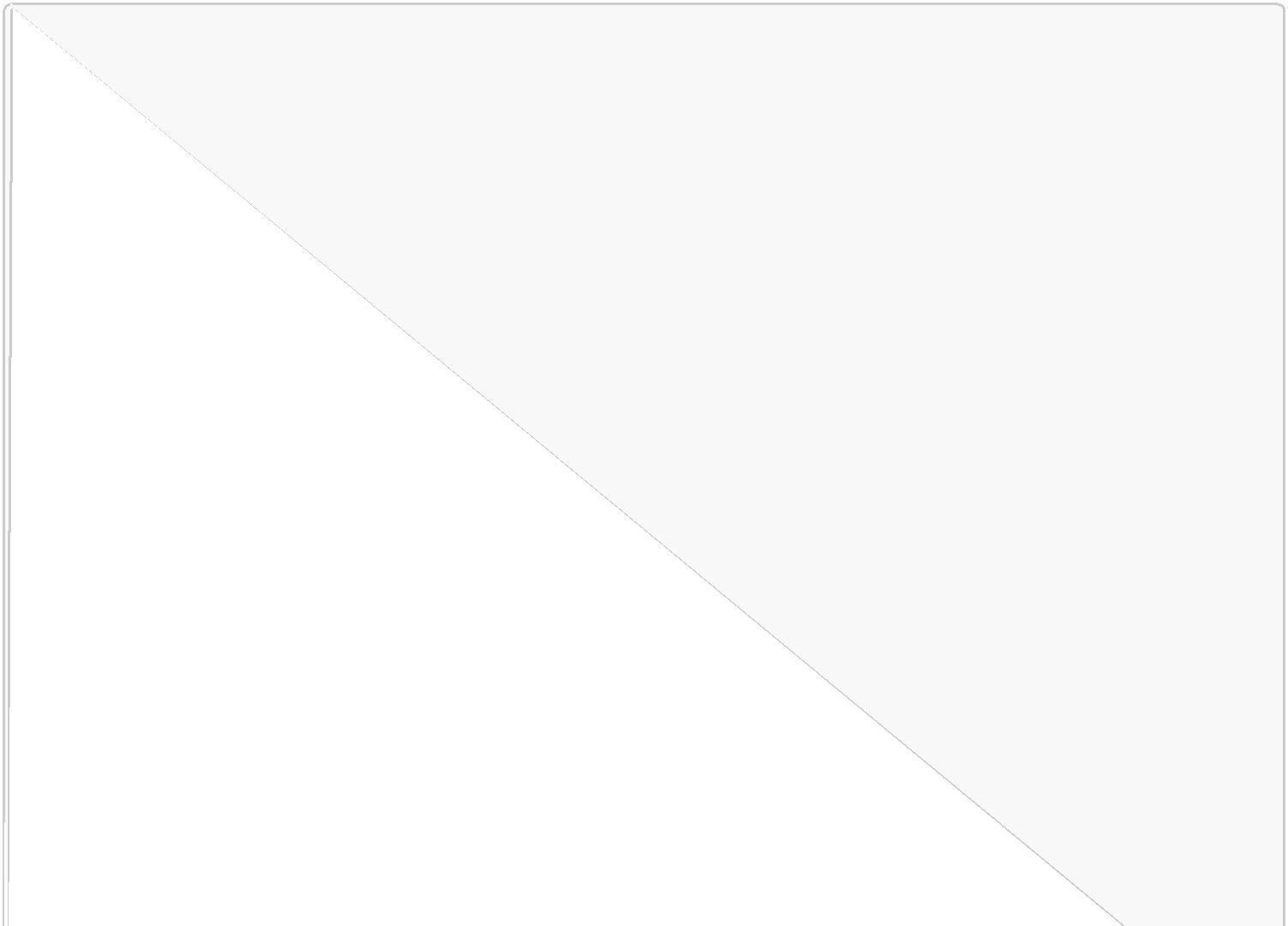
 push()插 O(n)

 query\_pos()按址查值O(1)

 query\_val()按值查址O(n)

3. 代码：

* 顺序结构



#include

#include

#include

#include

#define TRUE 1

#define FALSE 0

#define OK 1

#define ERROR 0

#define OVERFLOW 0

typedef int Status;

struct Arr //定义基本的数据结构

{

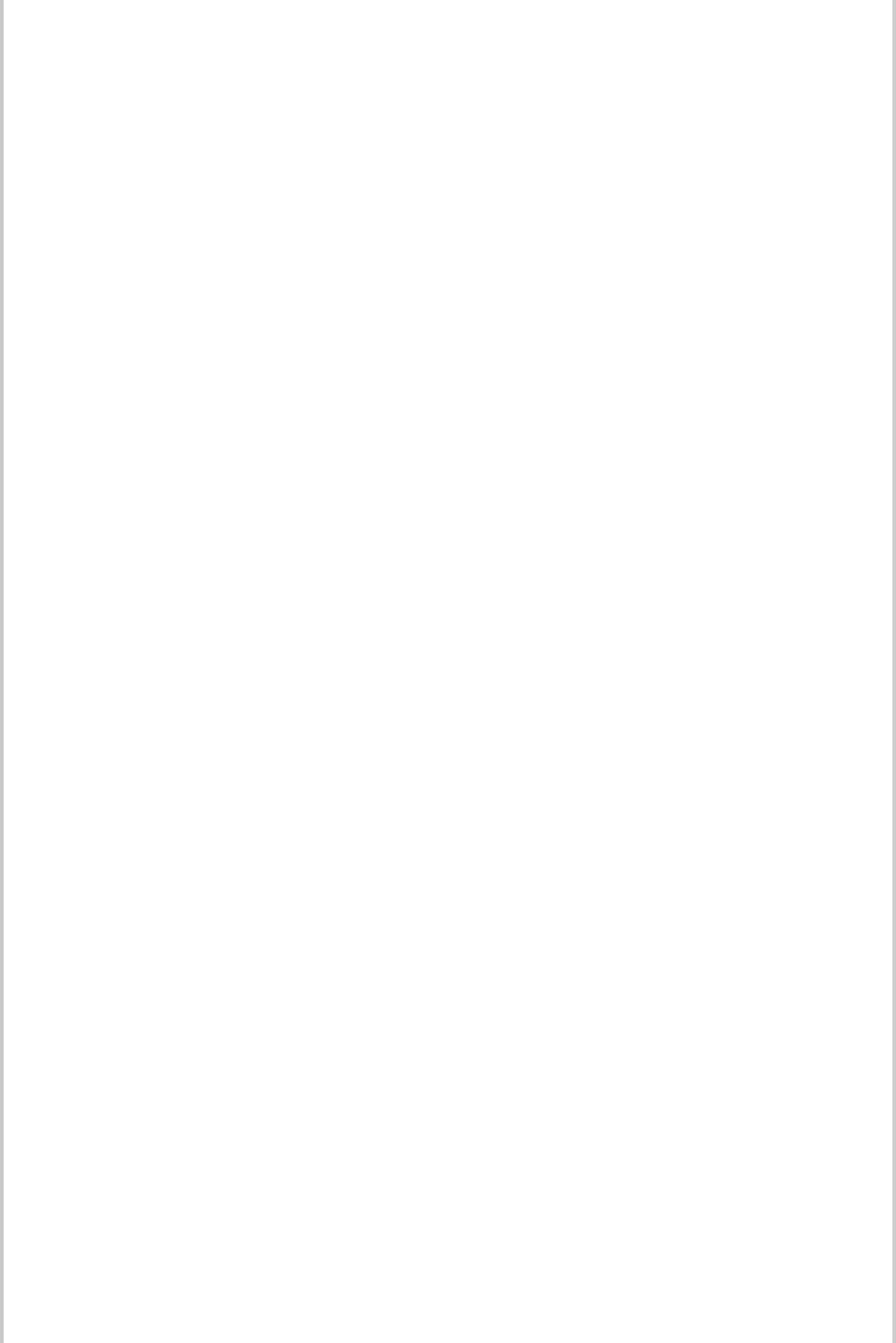
int len; //len存储顺序表的初始 度

int count; //count 存储已存的有 个数据， 便之后检查 int \*pBase;

};

2/13

* struct Arr \*pArray 指向结构的指针
* ‘ -> ’ 类似于JS的‘ . ’



void init\_array(struct Arr \*pArray, int len)

{

pArray->pBase = (int \*)malloc(sizeof(int) \* len); if (NULL == pArray->pBase) {

printf("动态分配内存失败\n");

exit(ERROR);

}

else

{

pArray->len = len;

pArray->count = 0;

printf("%d动态内存分配成功\n", len);

}

return;

}

//判断顺序结构是否为空，因为在遍历之前需要判断，所以抽离出 个单独的函数 bool isempty(struct Arr \*pArray) {

if (0 == pArray->count)

{

printf("表为空\n");

return true;

}

else

{

printf("表不为空\n");

return false;

}

}

//判断顺序结构是否已存满，由于之后插 数据需要单独判断，所以抽离出 个单独的函数

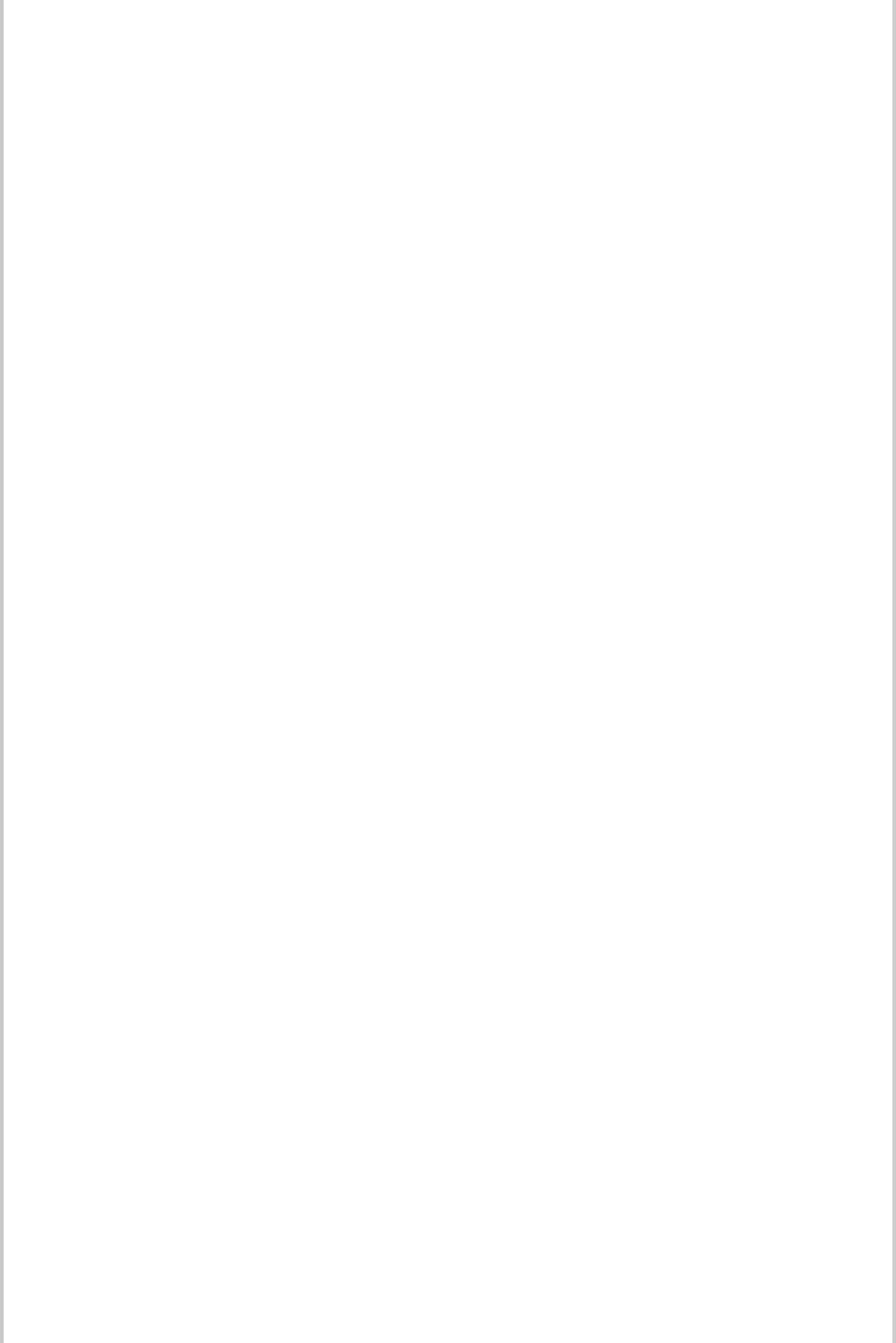
bool isfull(struct Arr \*pArray)

{

if (pArray->len == pArray->count)

{

3/13

return true;

}

else

{

return false;

}

}

bool push(struct Arr \*pArray, int val)

{

if (isfull(pArray))

{

printf("表已经满了！\n");

return false;

}

else

{

pArray->pBase[pArray->count] = val;

pArray->count++;

}

}

get\_length(struct Arr \*pArray){

printf("顺序表的 度为：%d",pArray->count);

}

//根据所给的脚标删除该数据，并遍历使之后的数据依次前移 bool delete (struct Arr \*pArray) {

int index;

printf("删除脚标:\t");

scanf("%d", &index);

if (index < 0 || index > pArray->count)

{

printf("该数字不存在");

return false;

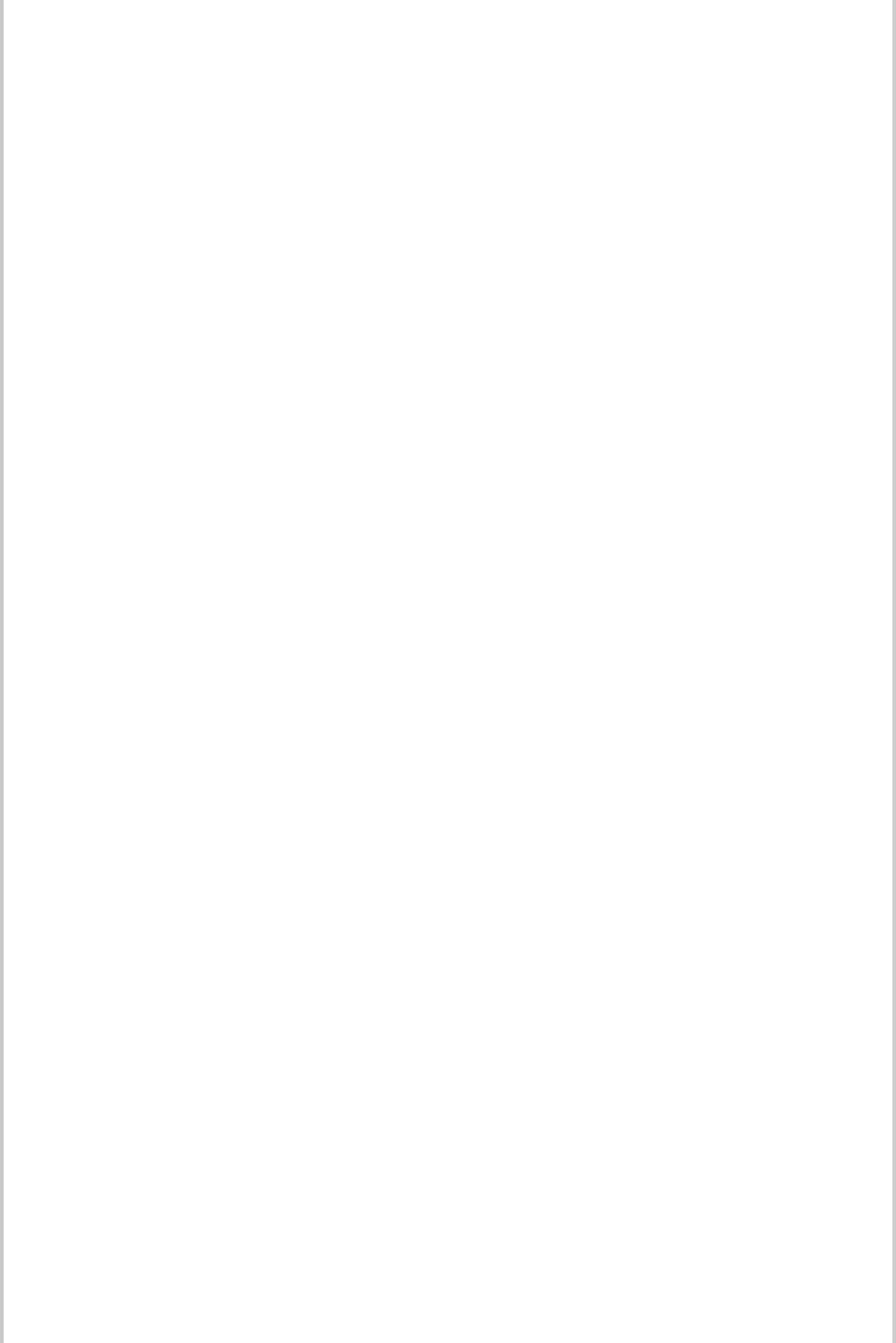
}

else

{

int i;

4/13

for (i = index; i <= parray-="">count; i++)

{

pArray->pBase[i - 1] = pArray->pBase[i];

}

pArray->count--;

return true;

}

}

//根据所给的脚标插 数据，并遍历使之后的数据依次后移

bool insert(struct Arr \*pArray, int index, int val)

{

if (index < 0 || index > pArray->count)

{

printf("插 位置不合理");

return false;

}

else

{

if (!isfull(pArray))

{

int j = index;

pArray->pBase[j + 1] = val;

for (j = index; j <= parray-="">count; j++)

{

pArray->pBase[j - 1] = pArray->pBase[j];

}

}

pArray->count++;

}

}

//因为是按值查找，所以使 遍历的 法直接依次 较 void query\_value(struct Arr \*pArray) {

int query\_val;

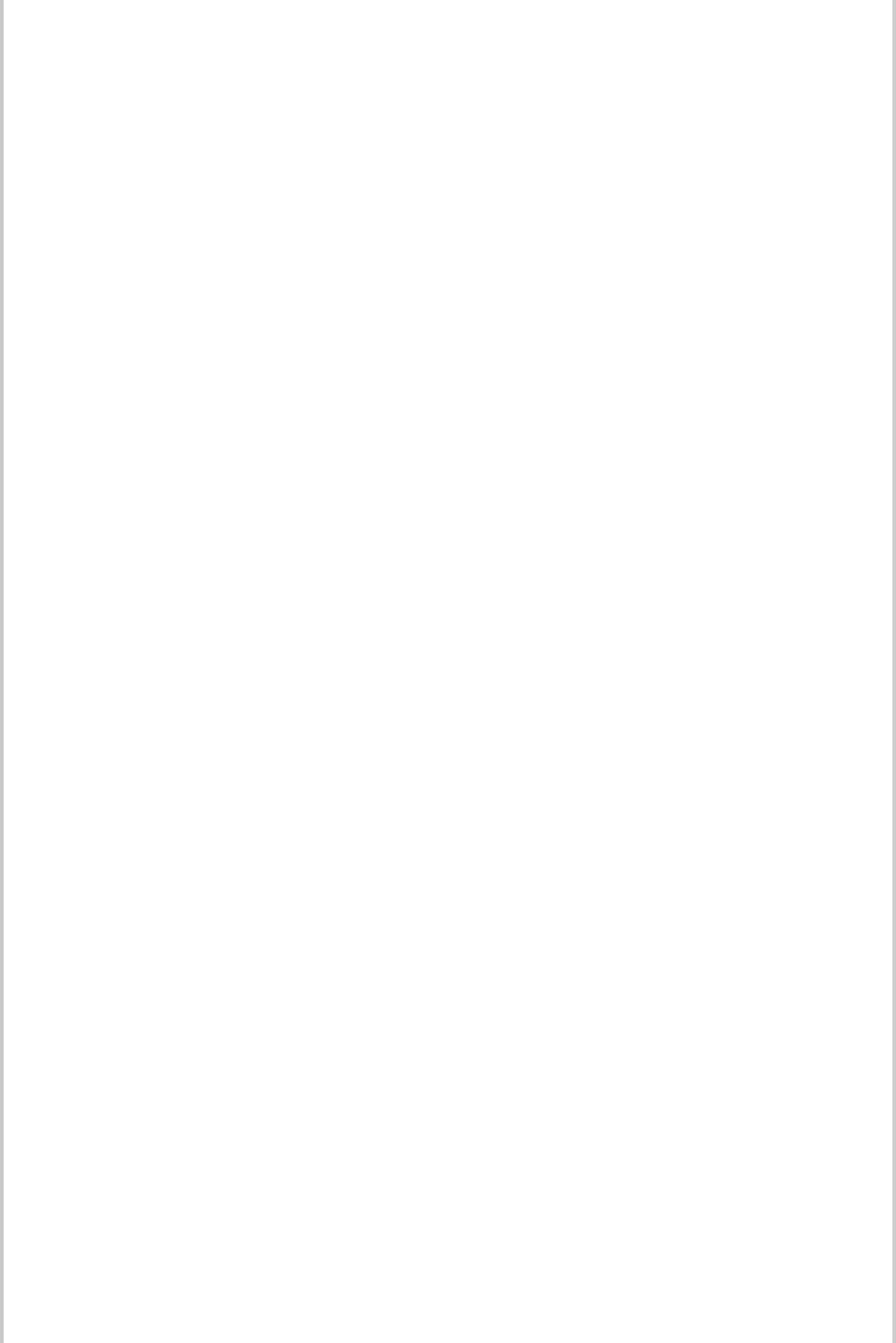
printf("请输 要查找的值");

scanf("%d", &query\_val);

for (int i = 0; i < pArray->count; i++)

{

5/13

if (pArray->pBase[i] == query\_val)

{

printf("该值的位置为：%d\n", i + 1);

}

}

}

//顺序表有随机存取的特点，所以直接 所给的⻆标拿到值 void query\_pos(struct Arr \*pArray) {

int query\_pos;

printf("请输 要查找的序列值\t");

scanf("%d", &query\_pos);

printf("垓值为：%d\n", pArray->pBase[query\_pos - 1]);

}

void show\_array(struct Arr \*pArray)

{

if (isempty(pArray))

printf("表为空！\n");

else

{

int i;

printf("------------------------------------\n"); printf("表为\n");

for (i = 0; i < pArray->count; i++)

{

printf("%d \n", pArray->pBase[i]);

}

printf("------------------------------------\n");

}

}

int main()

{

int max\_count, length, index;

struct Arr arr;

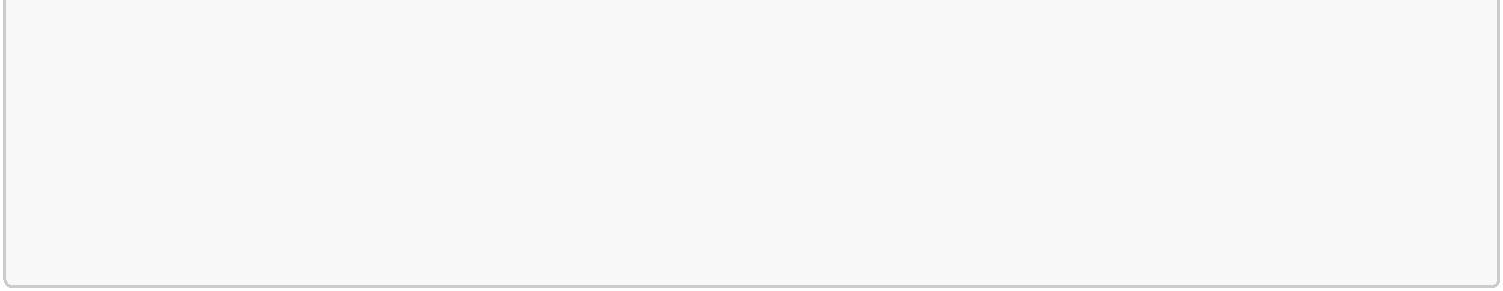
printf("创建 度：\t");

scanf("%d", &max\_count);

init\_array(&arr, max\_count);

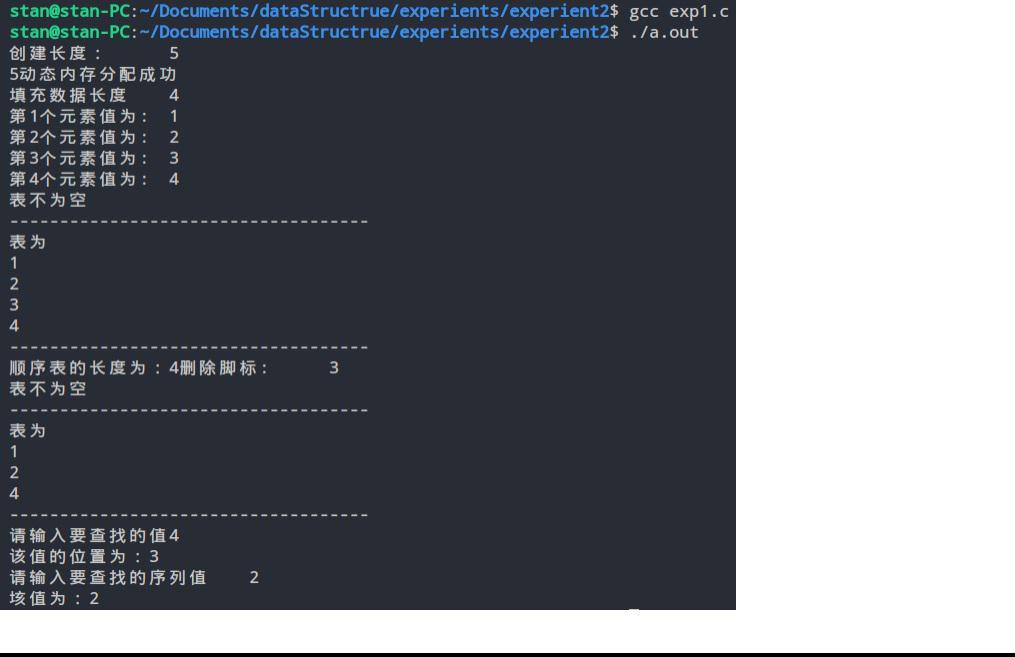
printf("填充数据 度\t");

6/13

scanf("%d", &length);

for (int a = 1; a <= length;="" a++)="" {="" int="" val;="" printf("第%d个元素值为:\t",="" a);="" scanf("%d",="" &val);="" push(&arr,="" val);="" }="" show\_array(&arr);="" get\_length(&arr);="" delete="" (&arr);="" query\_value(&arr);="" query\_pos(&arr);="" };="" <="" pre="">

3. 执 结果：



**链式结构实现**

1. 链式结构由于其物理空间并不 定是连续的，所以插 和删除节点很 便，但是查找相关的操作就 较复杂，由于位置不确定所以 般需要进 遍历。具体算法及解释 代码及注释

2.

 create\_list()初始化O(n)

 traverse()遍历O(n)

 delete\_node()删除O(1)

 insert\_node()插 O(1)

 query\_pos()按址查值O(n)

7/13

 query\_val()按值查址O(n)

3. 代码：

* C实现连式结构



#include

#include

#include

#include

//定义基本结构

typedef struct Node

{

int data;

struct Node \*next;

} Node, \*PNode;

//创建链表，顺便初始化 下。。直接往 填了值

PNode create\_list()

{

int len, i;

printf("请输 链表的 度：len=\t");

scanf("%d", &len);

PNode PHead = malloc(sizeof(Node));

PHead->next = NULL;

PNode PTail = PHead; //PTail是永远指向尾节点的指针,使 尾插法

for (i = 0; i < len; i++)

{

int val;

printf("请输 第 %d 个元素的值：", i + 1);

scanf("%d", &val);

PNode PNew = malloc(sizeof(Node));

PNew->data = val;

PNew->next = NULL;

PTail->next = PNew;

PTail = PNew;

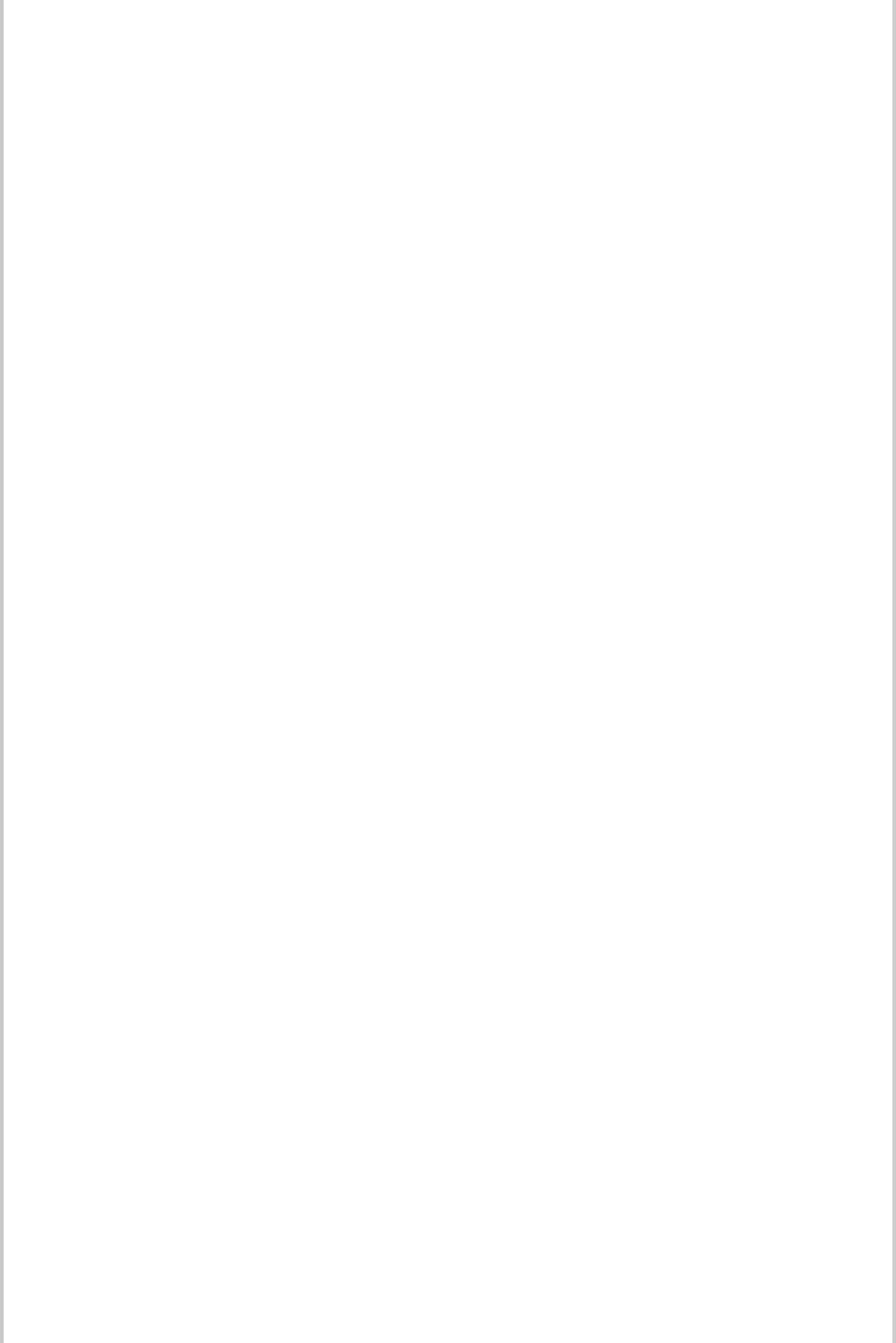
}

return PHead;

}

//从头遍历并输出该值，和下 的根据查询值类似

8/13

void traverse(PNode pHead)

{

printf("----------------\n");

PNode p = pHead->next;

while (p != NULL)

{

printf("%d ", p->data);

p = p->next;

}

printf("\n------------------\n");

}

//判断链表是否为空，因为在较多的地 需要使 所以抽离出单独的函数 bool judge\_if\_empty(PNode pHead) {

if (NULL == pHead->next)

{

return true;

}

else

{

return false;

}

}

//获取链表的 度，因为链表的特点所以使 遍历的 式 int get\_length(PNode pHead) {

int num = 0;

PNode p = pHead->next;

while (p != NULL)

{

num++;

p = p->next;

}

return num;

}

//与顺序表不同，链表不能随机存取，所以需要遍历

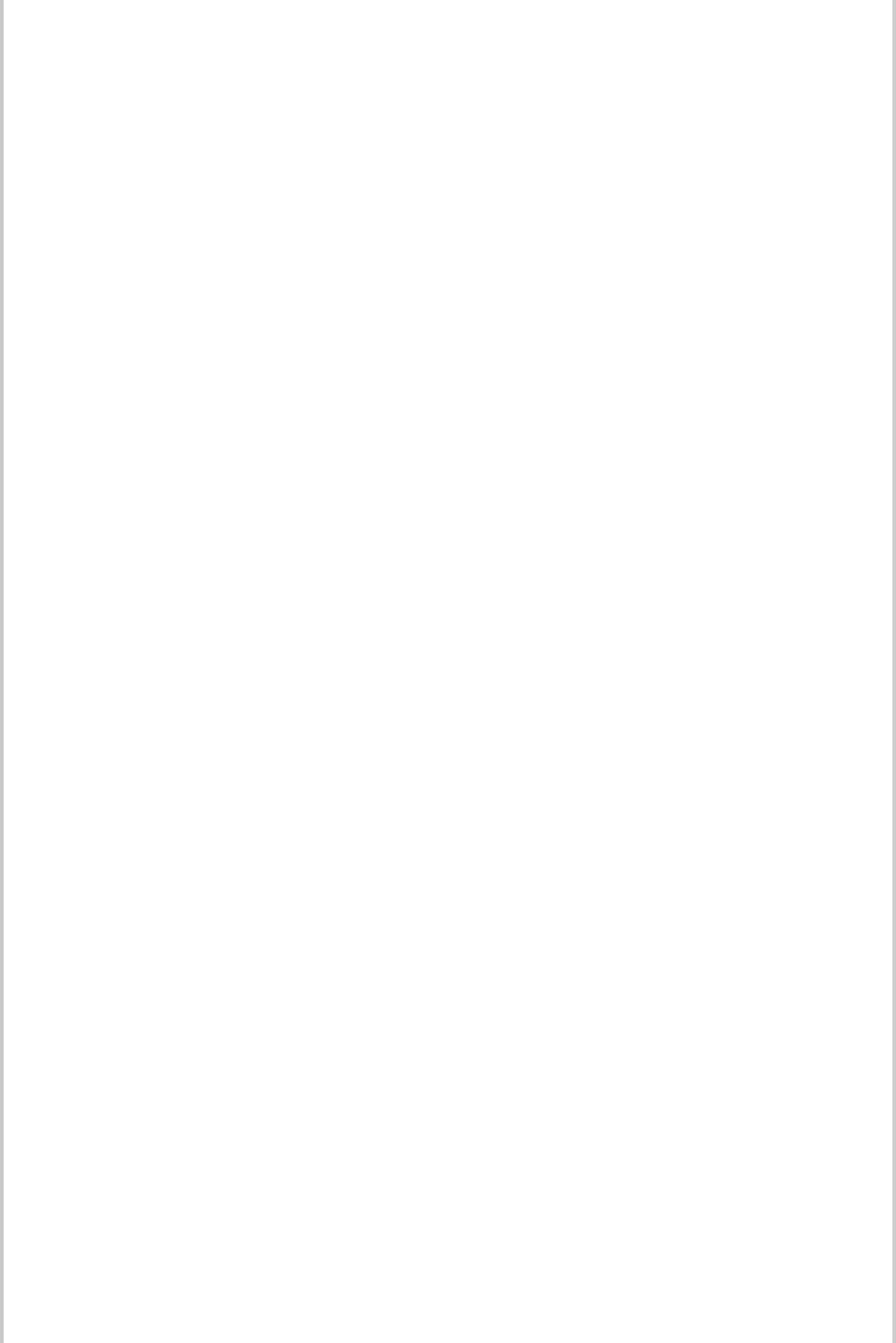
int query\_pos(PNode pHead)

{

int index;

9/13

printf("请输 要查询的位置：\t");



scanf("%d", &index);

PNode p = pHead->next;

if (get\_length(pHead) < index)

{

printf("要查询的位置不存在");

}

else

{

for (int i = 1; i < index; i++)

{

p = p->next;

}

printf("该位置的值是：%d\n", p->data);

}

return p->data;

}

int query\_val(PNode pHead)

{

int val;

printf("请输 要查询的值:\t");

scanf("%d", &val);

PNode p = pHead->next;

int j = 1;

while (p != NULL)

{

if (p->data == val)

{

printf("查询的值位置为：%d\n", j);

}

* = p->next; j++;

}

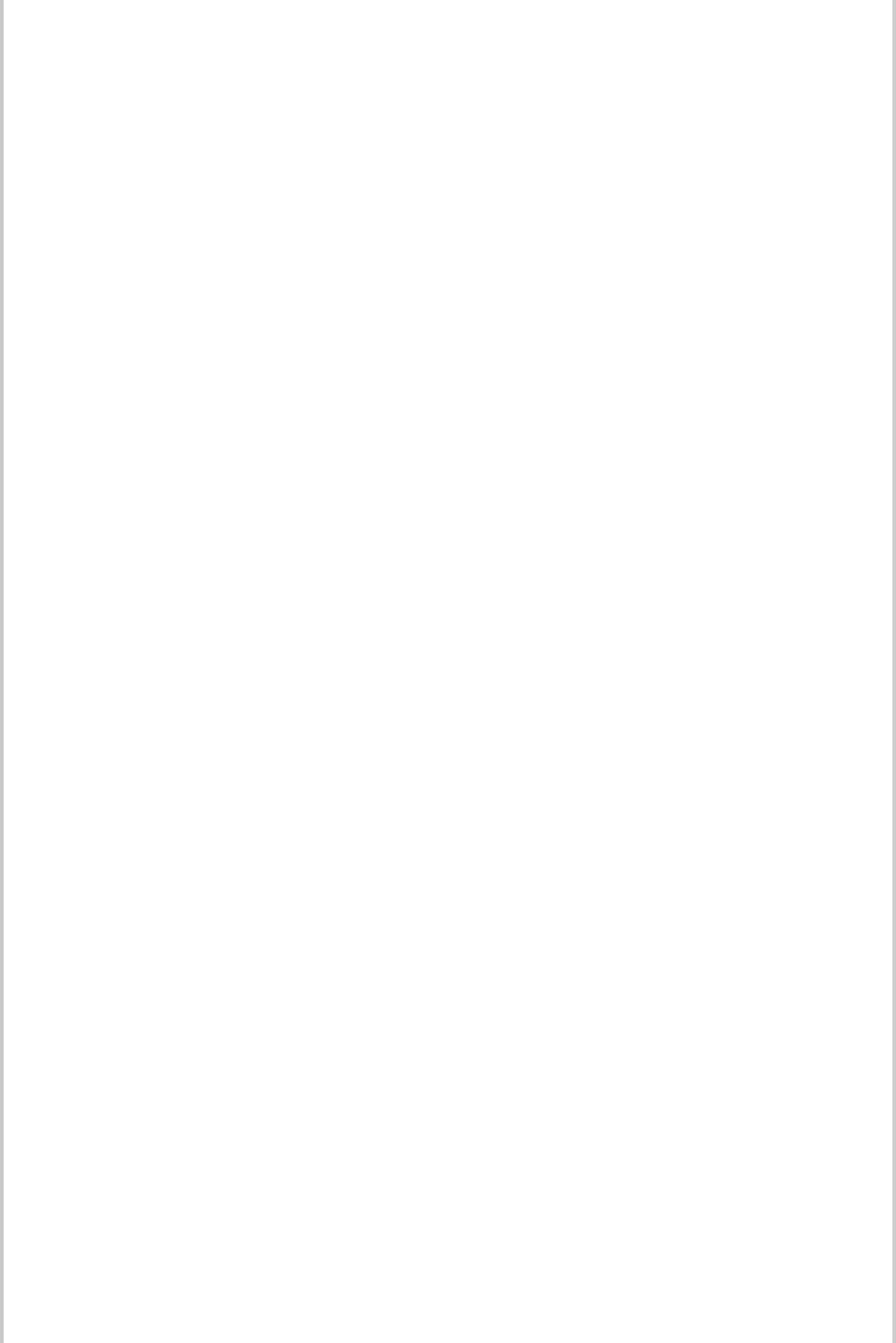
return j;

}

//插 节点， 成节点，插 ，主要是各种next的关系搞清楚 bool insert\_node(PNode pHead, int val, int pos) {

int i = 0;

10/13

PNode p = pHead;

while (NULL != p && i < pos - 1)

{

i++;

p = p->next;

}

if (i > pos - 1 || NULL == p)

{

printf("插 位置不合法\n");

return false;

}

PNode PNew = malloc(sizeof(PNode));

PNew->data = val;

PNode temp = p->next;

p->next = PNew;

PNew->next = temp;

return true;

}

//删除节点

bool delete\_node(PNode PHead, int pos)

{

|  |  |  |
| --- | --- | --- |
| int i | = | 0; |
| PNode | p | = PHead; |
| while | (p->next != NULL && i < pos - 1) | |
| { |  |  |

* = p->next; i++;

}

if (i > pos - 1 || p->next == NULL)

{

printf("删除位置不合法\n");

return false;

}

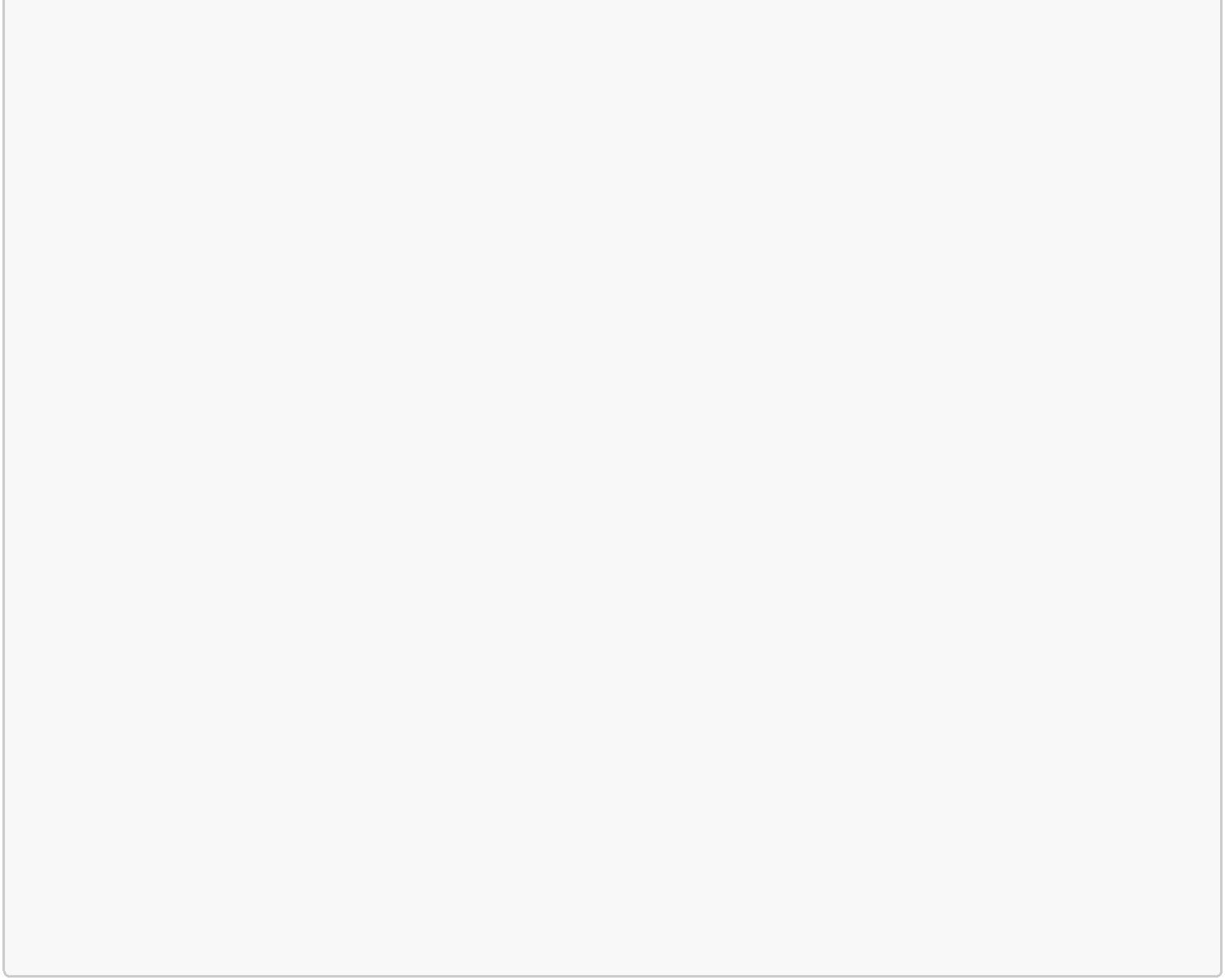
PNode temp = p->next;//注意需要临时保存节点 p->next = temp->next; free(temp);

}

int main()

{

11/13

PNode PHead = create\_list();

if (judge\_if\_empty(PHead))

printf("链表为空\n");

printf("链表的 度为：%d\n", get\_length(PHead));

traverse(PHead);

int val, pos;

printf("插 的位置：\n");

scanf("%d", &pos);

printf("插 的值：\n");

scanf("%d", &val);

insert\_node(PHead, val, pos);

traverse(PHead);

int delpos;

printf("删除的位置：\n");

scanf("%d", &delpos);

delete\_node(PHead, delpos);

traverse(PHead);

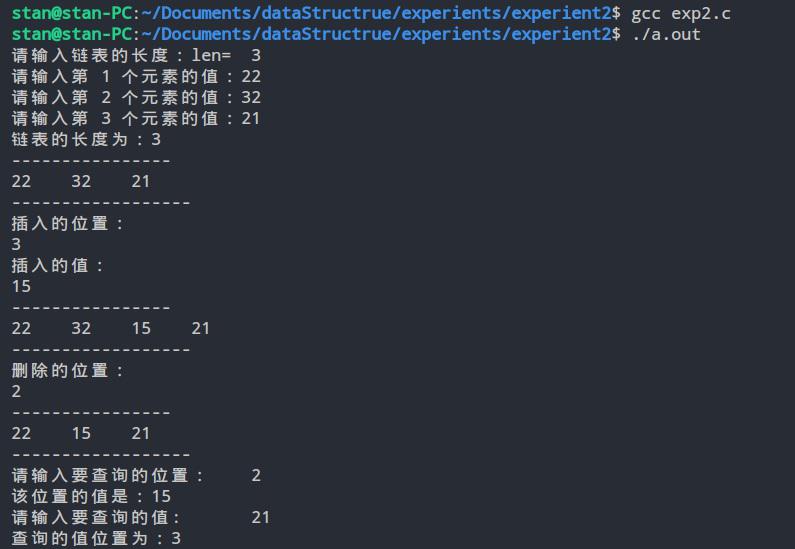
query\_pos(PHead);

query\_val(PHead);

return 0;

}

4. 运 结果：



12/13

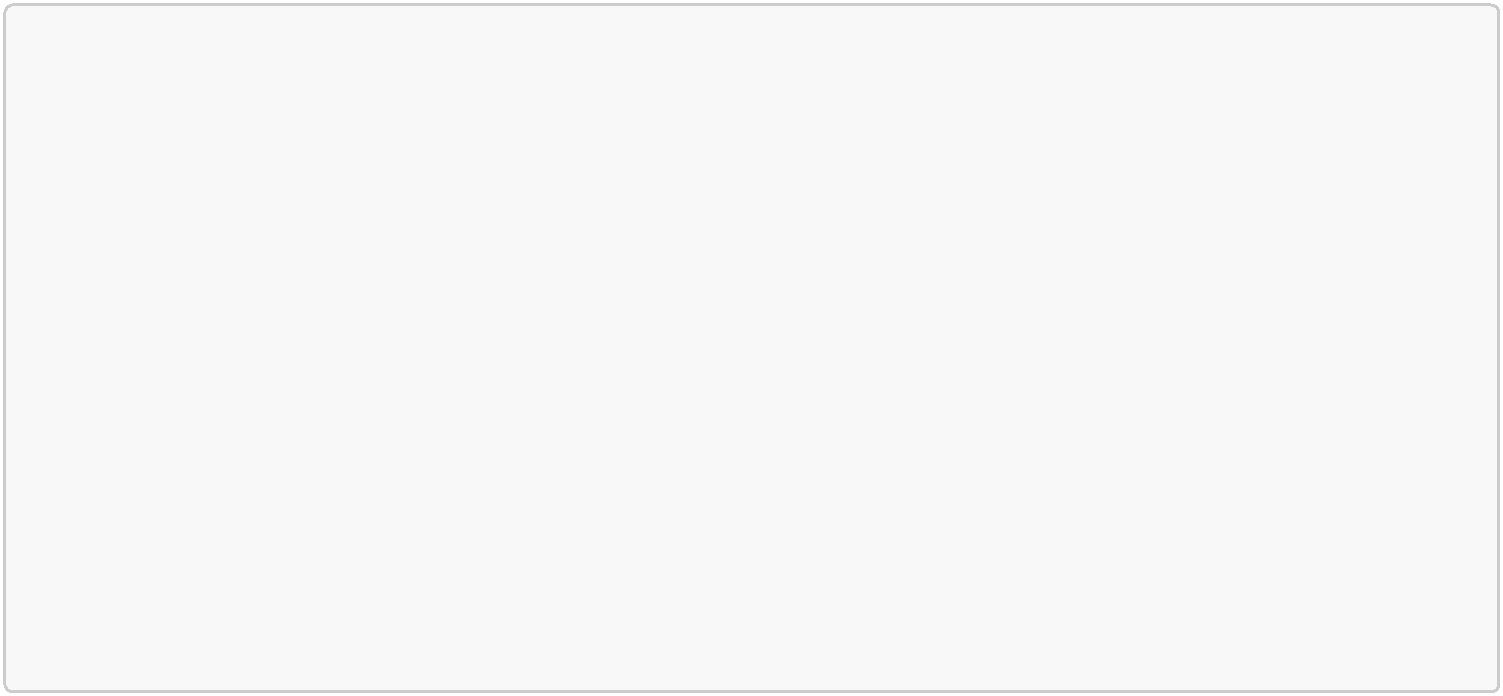
while (p != NULL)

{

**七、思考题**

1、如何设计和实现基本线性表在两种存储结构下就地逆置算法？

 顺序表：



void ReverseList(linklist \*L)

{

int i, temp;

int n = L->length;

for(i = 0; i < n/2; i ++)

{

temp = L->elem[i];

L->elem[i] = L->elem[n-1-i];

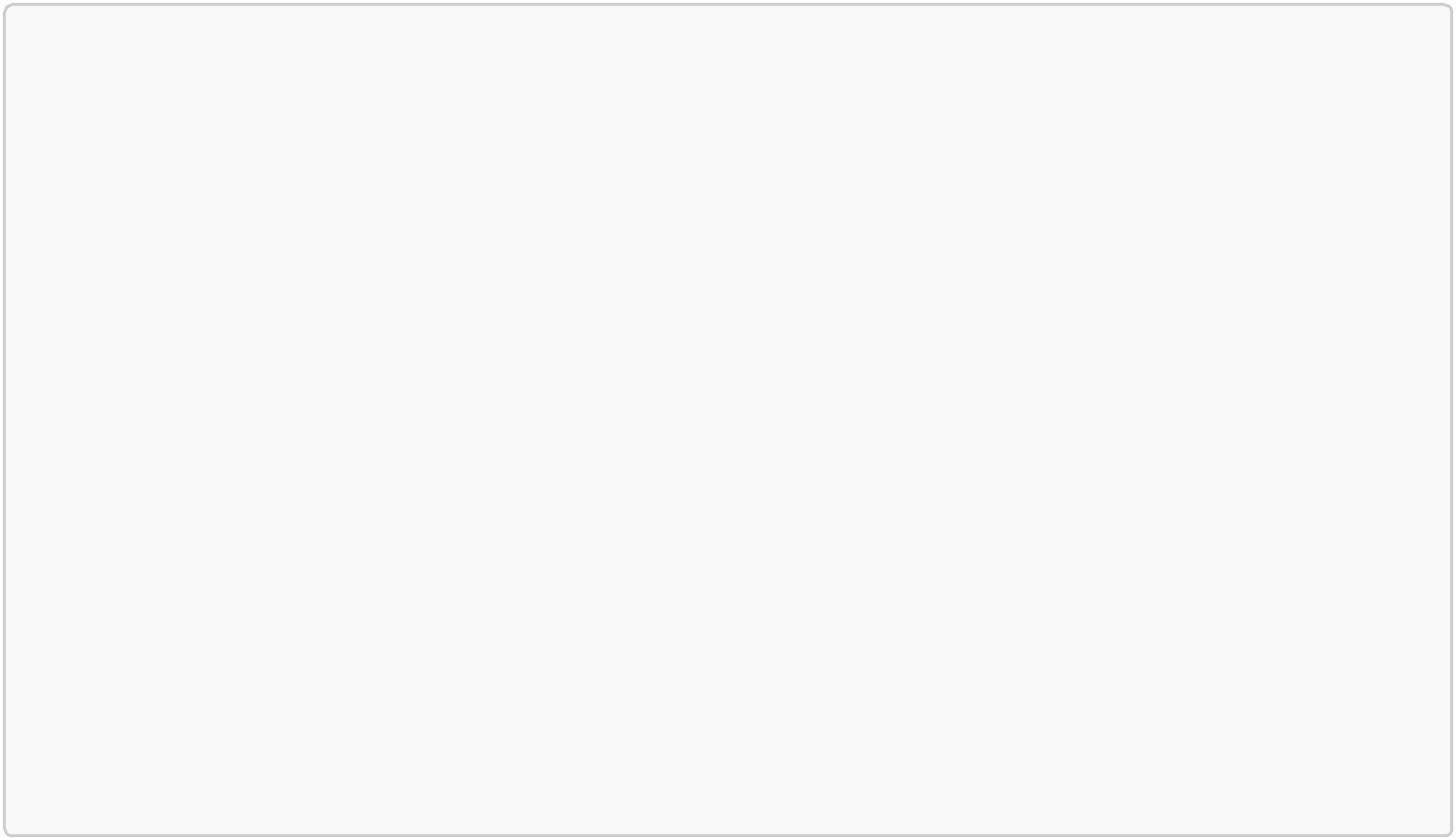
L->elem[n-1-i] = temp;

}

}

 链表：

* 主要是使 头插法把按原顺序依次拿到的数据插到头成为头结点，实现逆置



void ReverseList(LinkList L)

{

|  |  |  |
| --- | --- | --- |
| Node \*p, \*q; | /\*p为原链表的当前处理节点\*/ |  |
| p = L->next; |  |

L->next = NULL; /\*逆置单链表初始为空\*/

/\*当原链表未处理完\*/

q = p->next; /\*q指针保留原链表当前处理节点的下 个节点\*/ p->next = L->next; /\*将当前处理节点p插 到逆置L的表头\*/ L->next = p;

* = q; /\*p指向下 个待插 的节点\*/ }//头还是头@\_@

}

13/13