班级：

学号：

姓名：

**实验3 链表的基本操作**

【实验目的】

1、掌握单链表的尾部操作方法；

2、掌握单链表的头部操作方法；

3、掌握单链表的指定位置操作方法；

4、熟悉[单链表(快慢指针)](http://10.200.11.160/admin/courseAdmin/assignmentAdmin/javascript:displayProContent('problemDesc1122', '1122');) 的应用；

5、熟悉[单链表的综合运用](http://10.200.11.160/admin/courseAdmin/assignmentAdmin/javascript:displayProContent('problemDesc1123', '1123');)。

【实验要求】

1. 实验共5题，每题20分，满分100分；
2. 实验需在截至日期前提交实验平台，由实验平台自动判分，补交时间段内提交酌情扣分；
3. 学生对实验分数有异议，提交申请，带教老师修正得分；
4. 原则上学生需独立完成实验题，个别难题可互相讨论或查阅资料；
5. 若对试验题有疑问，且仔细阅读测试用例后仍然不理解，可咨询带教老师。

【实验内容】

1. 单链表的尾部操作
2. 单链表的头部操作
3. 单链表的指定位置操作
4. 单链表快慢指针的应用
5. 单链表的综合应用

【试题列表】

**2.1单链表(尾部操作)**

本题主要考查内容：

1>单链表初始化

2>在单链表尾部插入节点

3>在单链表尾部删除节点

4>求单链表的有效长度

5>打印单链表

6>撤销单链表

【程序片段】#include <stdio.h>

#include <stdlib.h>

#define MaxSize 100

typedef int ElemType;

typedef struct SingleNode{

ElemType data;

struct SingleNode \*next;

}SingleNodeList,\*Linklist;

void LinkedListInit(SingleNodeList \*\*head){//1初始化有头节点的单链表

Linklist p;

if((\*head=(SingleNodeList \*)malloc(sizeof(SingleNodeList)))==NULL){

exit(1);

}

(\*head)->next=NULL;

}

int LinkedList\_PushBack(SingleNodeList \*head,ElemType x){//2单链表尾插入

SingleNodeList \*p=head,\*q;

while(p->next!=NULL)

p=p->next;

if((q=(SingleNodeList \*)malloc(sizeof(SingleNodeList)))==NULL){

exit(1);

}

q->data=x;

q->next=NULL; //尾节点的数据域与指针域赋值

【 1 】;//尾节点加入链表

return 1;

}

int LinkedList\_PopBack(SingleNodeList \*head,ElemType \*x){//3单链表尾删除

SingleNodeList \*p=head,\*q;

if(p->next==NULL){

printf("There is no data in the Linkedlist to delete.\n");

\*x = -12345;//未成功删除则在x指向单元赋特定值

return 0;

}

while(p->next!=NULL){

q=p;

p=p->next;

}//p是尾节点，q是p的直接前驱

【 2 】 ;//赋值q的指针域

\*x=p->data;

free(p);//释放p

return 1;

}

int LinkedListLength(SingleNodeList \*head){//4求单链表长度

SingleNodeList \*p=head;

int size=0;

while(p->next!=NULL){

【 3 】;

【 4 】;

}

return size;

}

int LinkedListShow(SingleNodeList \*head){//5打印单链表

SingleNodeList \*p=head;

if(p->next==NULL){

printf("There is no data in the Linkedlist to print.\n");

return 0;

}

while(【 5 】){

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

p=p->next;

}

printf("\n");

return 1;

}

void LinkedListDestroy(SingleNodeList \*\*head){//6释放链表

SingleNodeList \*p=\*head,\*q;

while(p!=NULL){

【 6 】;

【 7 】;

free(q);

}

\*head=NULL;

}

int main(){

SingleNodeList \*head;

ElemType i,x;

int switch\_num;

scanf("%d",&switch\_num);

switch(switch\_num){

case 1:

LinkedListInit(&head);

LinkedList\_PushBack(head,1);

LinkedList\_PushBack(head,3);

LinkedList\_PushBack(head,2);

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

break;

case 2:

LinkedListInit(&head);

LinkedList\_PushBack(head,11);

LinkedList\_PushBack(head,31);

LinkedList\_PushBack(head,22);

for(i=1;i<=5;i++)

LinkedList\_PushBack(head,i);

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

break;

case 3:

LinkedListInit(&head);

for(i=1;i<=5;i++)

LinkedList\_PushBack(head,i\*i);

for(i=1;i<=5;i++){

LinkedList\_PopBack(head,&x);

printf("The %d PopBack delete num = %d\n",i,x);

}

break;

case 4:

LinkedListInit(&head);

LinkedList\_PushBack(head,11);

LinkedList\_PushBack(head,31);

LinkedList\_PushBack(head,22);

for(i=1;i<=5;i++){

LinkedList\_PopBack(head,&x);

printf("The %d PopBack delete num = %d\n",i,x);

}

break;

case 5:

LinkedListInit(&head);

LinkedList\_PopBack(head,&x);

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

break;

}

LinkedListDestroy(&head);

return 0;

}

【测试用例】

1

3

1 3 2

2

8

11 31 22 1 2 3 4 5

3

The 1 PopBack delete num = 25

The 2 PopBack delete num = 16

The 3 PopBack delete num = 9

The 4 PopBack delete num = 4

The 5 PopBack delete num = 1

4

The 1 PopBack delete num = 22

The 2 PopBack delete num = 31

The 3 PopBack delete num = 11

There is no data in the Linkedlist to delete.

The 4 PopBack delete num = -12345

There is no data in the Linkedlist to delete.

The 5 PopBack delete num = -12345

5

There is no data in the Linkedlist to delete.

0

There is no data in the Linkedlist to print.

**2.2单链表(头部操作)**

本题主要考查内容：

1>在单链表头部插入节点

2>在单链表头部删除节点

3>修改当前节点数据域

4>获取当前节点数据域

【程序片段】

#include <stdio.h>

#include <stdlib.h>

#define MaxSize 100

typedef int ElemType;

typedef struct SingleNode{

ElemType data;

struct SingleNode \*next;

}SingleNodeList,\*Linklist;

void LinkedListInit(SingleNodeList \*\*head){//1初始化有头节点的单链表

Linklist p;

if((\*head=(SingleNodeList \*)malloc(sizeof(SingleNodeList)))==NULL){

exit(1);

}

(\*head)->next=NULL;

}

int LinkedList\_PushFront(SingleNodeList \*head,ElemType x){//2单链表头插入

SingleNodeList \*q;

if((q=【 1 】)==NULL){

exit(1);

}

【 2 】

【 3 】//头节点的数据域与指针域赋值

【 4 】//头节点加入链表

return 1;

}

int LinkedList\_PopFront(SingleNodeList \*head,ElemType \*x){//3单链表头删除

SingleNodeList \*p=head,\*q;

if(p->next==NULL){

printf("There is no data in the Linkedlist to delete.\n");

\*x = -12345;//未成功删除则在x指向单元赋特定值

return 0;

}

【 5 】//请填写多行代码

}

int LinkedListGet\_current(SingleNodeList \*p,ElemType \*x){//4取当前指针指数据

【 6 】=p->data;

return 1;

}

int LinkedListUpdata\_current(SingleNodeList \*p,ElemType x){//5修改当前指针数据

p->data=【 7 】;

return 1;

}

int LinkedListShow(SingleNodeList \*head){//6打印单链表

SingleNodeList \*p=head;

if(p->next==NULL){

printf("There is no data in the Linkedlist to print.\n");

return 0;

}

while(p->next!=NULL){

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

p=p->next;

}

printf("\n");

return 1;

}

void LinkedListDestroy(SingleNodeList \*\*head){//7释放链表

SingleNodeList \*p=\*head,\*q;

while(p!=NULL){

q=p;

p=p->next;

free(q);

}

\*head=NULL;

}

int LinkedListLength(SingleNodeList \*head){//8求单链表长度

SingleNodeList \*p=head;

int size=0;

while(p->next!=NULL){

size++;

p=p->next;

}

return size;

}

int main(){

SingleNodeList \*head,\*p;

ElemType i,x;

int switch\_num;

scanf("%d",&switch\_num);

switch(switch\_num){

case 1:

LinkedListInit(&head);

LinkedList\_PushFront(head,1);

LinkedList\_PushFront(head,3);

LinkedList\_PushFront(head,2);

LinkedListShow(head);

break;

case 2:

LinkedListInit(&head);

LinkedList\_PushFront(head,11);

LinkedList\_PushFront(head,31);

LinkedList\_PushFront(head,22);

for(i=1;i<=5;i++)

LinkedList\_PushFront(head,i);

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

break;

case 3:

LinkedListInit(&head);

for(i=1;i<=5;i++)

LinkedList\_PushFront(head,i\*i);

for(i=1;i<=5;i++){

LinkedList\_PopFront(head,&x);

printf("The %d PopFront delete num = %d\n",i,x);

}

break;

case 4:

LinkedListInit(&head);

LinkedList\_PushFront(head,11);

LinkedList\_PushFront(head,31);

LinkedList\_PushFront(head,22);

for(i=1;i<=5;i++){

LinkedList\_PopFront(head,&x);

printf("The %d PopFront delete num = %d\n",i,x);

}

break;

case 5:

LinkedListInit(&head);

for(i=1;i<=10;i++){

LinkedList\_PushFront(head,i);

}

LinkedListShow(head);

p=head;

for(i=1;i<=10;i=i++){

p=p->next;

if(i%2==0) LinkedListUpdata\_current(p,i\*i);

else{

LinkedListGet\_current(p,&x);

printf("The %d get Node is %d\n",i,x);

}

}

LinkedListShow(head);

break;

}

LinkedListDestroy(&head);

return 0;

}

【测试用例】

1

2 3 1

2

8

5 4 3 2 1 22 31 11

3

The 1 PopFront delete num = 25

The 2 PopFront delete num = 16

The 3 PopFront delete num = 9

The 4 PopFront delete num = 4

The 5 PopFront delete num = 1

4

The 1 PopFront delete num = 22

The 2 PopFront delete num = 31

The 3 PopFront delete num = 11

There is no data in the Linkedlist to delete.

The 4 PopFront delete num = -12345

There is no data in the Linkedlist to delete.

The 5 PopFront delete num = -12345

5

10 9 8 7 6 5 4 3 2 1

The 1 get Node is 10

The 3 get Node is 8

The 5 get Node is 6

The 7 get Node is 4

The 9 get Node is 2

10 4 8 16 6 36 4 64 2 100

**2.3单链表(指定位置操作)**

本题主要考查内容：

1>在单链表指定位置插入节点

2>在单链表指定位置删除节点

3>在有序单链表插入一个元素，使之仍然有序

4>删除单链表中第一个等于指定值的节点

5>删除单链表中所有等于指定值的节点

6>获取单链表指定位置节点的数据域

【程序片段】

#include <stdio.h>

#include <stdlib.h>

#define MaxSize 100

typedef int ElemType;

typedef struct SingleNode{

ElemType data;

struct SingleNode \*next;

}SingleNodeList,\*Linklist;

void LinkedListInit(SingleNodeList \*\*head){//1初始化有头节点的单链表

Linklist p;

if((\*head=(SingleNodeList \*)malloc(sizeof(SingleNodeList)))==NULL){

exit(1);

}

(\*head)->next=NULL;

}

int LinkedListLength(SingleNodeList \*head){//2求单链表长度

SingleNodeList \*p=head;

int size=0;

while(p->next!=NULL){

size++;

p=p->next;

}

return size;

}

int LinkedListShow(SingleNodeList \*head){//3打印单链表

SingleNodeList \*p=head;

//ElemType x;

if(p->next==NULL){

printf("There is no data in the Linkedlist to print.\n");

return 0;

}

while(p->next!=NULL){

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

p=p->next;

}

printf("\n");

return 1;

}

int LinkedListInsert(SingleNodeList \*head,int i,ElemType x){//4在ai（0<=i<=size）前插入节点

【 1 】//请填写多行代码

return 1;

}

int LinkedListDelete(SingleNodeList \*head,int i,ElemType \*x) {//5删除i位置（0<=i<=size）前节点

SingleNodeList \*p=head,\*q;

int j=-1;

while(p->next!=NULL && p->next->next!=NULL && j<i-1){

p=p->next;j++;

}

if(j!=i-1){

printf("The location parameter of the delete node is wrong.");

return 0;

}

【 2 】//请填写多行代码

free(q);

return 1;

}

int LinkedListInsert\_Order(SingleNodeList \*head,ElemType x){//6有序插入x

SingleNodeList \*pre,\*curr,\*q;

int j=-1;

pre=head;

curr=head->next;

while(curr!=NULL && curr->data<=x){

【 3 】//请填写多行代码

}

if((q=(SingleNodeList \*)malloc(sizeof(SingleNodeList)))==NULL){

exit(1);

}

q->data=x;

【 4 】;//赋值q->next

【 5 】; //重链接p->next

return 1;

}

int LinkedListDelete\_Order(SingleNodeList \*head,ElemType x) {//7删除第一个值为x的节点

SingleNodeList \*p=head,\*q;

while(p->next!=NULL && p->next->data!=x){

p=p->next;

}

if(p->next==NULL){

printf("There is no node %d to delete.\n",x);return -1;

}

【 6 】//请填写多行代码

free(q);

return 1;

}

int LinkedListDelete\_All(SingleNodeList \*head,ElemType x) {//8删除所有值为x的节点

//用p从头至尾扫描单链表，pre指向\*p节点的前驱。若p所指节点值为x，则删除，并让p

//移向下一个节点，否则让pre、p指针同步后移一个节点。

SingleNodeList \*p=head->next,\*pre=head,\*q;

int j=0;

while(p!=NULL){

if(p->data==x) {

【 7 】

}

else{//同步后移

pre=p;

p=p->next;

}

}

if(j==0)

printf("There is no node %d to delete.\n",x);

else

printf("There is %d node deleted.\n",j);

return 1;

}

int LinkedListGet(SingleNodeList \*head,int i,ElemType \*x){//9取单链表第i个节点的数据，存入x指针

【 9 】

}

void LinkedListDestroy(SingleNodeList \*\*head){//10释放链表

SingleNodeList \*p=\*head,\*q;

while(p!=NULL){

q=p;

p=p->next;

free(q);

}

\*head=NULL;

}

int main(){

SingleNodeList \*head;

ElemType i,x;

int switch\_num;

scanf("%d",&switch\_num);

switch(switch\_num){

case 1:

LinkedListInit(&head);

for(i=1;i<=10;i++)

LinkedListInsert(head,0,i);

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

break;

case 2:

LinkedListInit(&head);

for(i=1;i<=10;i++)

LinkedListInsert(head,0,i);

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

for(i=1;i<=3;i++)

LinkedListDelete(head,i,&x);

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

break;

case 3:

LinkedListInit(&head);

for(i=1;i<=8;i++){

scanf("%d",&x);

LinkedListInsert\_Order(head,x);

}

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

break;

case 4:

LinkedListInit(&head);

for(i=1;i<=5;i++){

scanf("%d",&x);

LinkedListInsert(head,0,x);

}

LinkedListShow(head);

scanf("%d",&x);

LinkedListDelete\_Order(head,x);

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

break;

case 5:

LinkedListInit(&head);

for(i=1;i<=5;i++){

scanf("%d",&x);

LinkedListInsert(head,0,x);

}

LinkedListShow(head);

scanf("%d",&x);

LinkedListDelete\_All(head,x);

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

break;

}

return 0;

}

【测试用例】

1

10

10 9 8 7 6 5 4 3 2 1

2

10

10 9 8 7 6 5 4 3 2 1

7

10 8 6 4 3 2 1

3

1 3 5 7 9 2 4 6

8

1 2 3 4 5 6 7 9

4

11 22 33 22 44

22

44 22 33 22 11

4

44 33 22 11

5

11 22 33 22 22

22

22 22 33 22 11

There is 3 node deleted.

2

33 11

**2.4单链表(快慢指针)**

本题主要考查内容：

1>获取单链表中间位置节点数据域

2>获取单链表倒数第位置节点数据域

3>翻转单链表

4>单链表循环前移

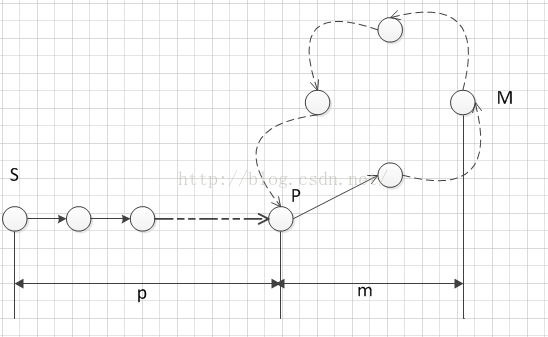
5>判断一个链表是否为循环链表

6>打印一个单链表（循环链表的前n个节点）

先简单说一下什么循环链表，循环链表其实就是单链表的尾部指针指向头指针，构建成一个环形的链表，叫做循环链表。 如 1 -> 2 - > 3 -> 1 -> 2 .....。为什么快慢指针再循环链表中总能相遇呢？你可以想象两个人在赛跑，A的速度快，B的速度慢，经过一定时间后，A总是会和B相遇，且相遇时A跑过的总距离减去B跑过的总距离一定是圈长的n倍。这也就是 Floyd判环(圈)算法。

算法描述：如果链表存在环，那么一定存在一个起点可以到达某个环的某处(这个起点也可以在某个环上)。初始状态下，假设已知某个起点[节点](https://zh.wikipedia.org/wiki/%E8%8A%82%E7%82%B9" \t "_blank" \o "节点)为节点S。现设两个指针t和h，将它们均指向S。接着，同时让t和h往前推进，但是二者的速度不同：t每前进1步，h前进2步。只要二者都可以前进而且没有相遇，就如此保持二者的推进。当h无法前进，即到达某个没有[后继](https://zh.wikipedia.org/w/index.php?title=%E5%90%8E%E7%BB%A7&action=edit&redlink=1" \t "_blank" \o "后继（页面不存在）)的节点时，就可以确定从S出发不会遇到环。反之当t与h再次相遇时，就可以确定从S出发一定会进入某个环，设其为环C。

如果确定了存在某个环，就可以求此环的起点与长度。



计算环长度

上述算法刚判断出存在环C时，显然t和h位于同一节点，设其为节点M。显然，仅需令h不动，而t不断推进，最终又会返回节点M，统计这一次t推进的步数，显然这就是环C的长度。

计算环起点

为了求出环C的起点，只要令h仍位于节点M，而令t返回起点节点S。随后，同时让t和h往前推进，且保持二者的速度相同：t每前进1步，h前进1步。持续该过程直至t与h再一次相遇，设此次相遇时位于同一节点P，则节点P即为从节点S出发所到达的环C的第一个节点，即环C的一个起点。

链表起点为节点S，环起点为节点P，t和h相遇时位于同一节点M，S和P之间的距离为p，P和M之间的距离为m，环长为C，这里两点之间的距离是指从一点走多少步可以到点另外一点。

当t和h相遇时，

t走的步数，step = p + m + a \* C，a表示相遇时t走的圈数

h走的步数，2 \* step = p + m + b \* C，b表示相遇时h走的圈数

两者相减：step = (b - a) \* C = p + m + a \* C，由此可知t走的步数是环C的倍数，即 p + m 刚好是环长度C的倍数。

t和h在M处相遇，为了计算环C的起点，令h仍位于节点M，而令t返回起点S，随后，同时让t和h往前推进，且保持两者的速度相同：t每前进1步，h前进1步。持续该过程直至t与h再一次相遇，则它们此次相遇时一定位于环的起始节点P。为什么它们此次相遇时一定在环起始节点呢？

t走了p步到达P，h在环C上p步在哪呢？h从M处出发走了p步，相对于环起始位置，h走过的距离是 m + p，而m + p刚好是环长度C的倍数，即h此时也位于环起始节点处，即t和h在P处相遇。据此就可以计算出环起始节点的位置。

算法复杂度

时间复杂度

注意到当指针t到达环C的一个起点节点P时(此时指针h显然在环C上)，之后指针t最多仅可能走1圈。若设节点S到P距离为m，环C的长度为n，则时间复杂度为O（m+n），是[线性](https://zh.wikipedia.org/wiki/%E7%B7%9A%E6%80%A7" \t "_blank" \o "线性)时间的算法。

空间复杂度

仅需要创立指针t、指针h，保存环长n、环的一个起点P。空间复杂度为O（1），是[常数](https://zh.wikipedia.org/wiki/%E5%B8%B8%E6%95%B0" \t "_blank" \o "常数)空间的算法。

【程序片段】

#include <stdio.h>

#include <stdlib.h>

#define MaxSize 100

typedef int ElemType;

typedef struct SingleNode{

ElemType data;

struct SingleNode \*next;

}SingleNodeList,\*Linklist;

void LinkedListInit(SingleNodeList \*\*head){//1初始化有头节点的单链表

Linklist p;

if((\*head=(SingleNodeList \*)malloc(sizeof(SingleNodeList)))==NULL){

exit(1);

}

(\*head)->next=NULL;

}

int LinkedListLength(SingleNodeList \*head){//2求单链表长度

SingleNodeList \*p=head;

int size=0;

while(p->next!=NULL){

size++;

p=p->next;

}

return size;

}

int LinkedListShow(SingleNodeList \*head){//3打印单链表

SingleNodeList \*p=head;

if(p->next==NULL){

printf("There is no data in the Linkedlist to print.\n");

return 0;

}

while(p->next!=NULL){

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

p=p->next;

}

printf("\n");

return 1;

}

int LinkedList\_PushBack(SingleNodeList \*head,ElemType x){//4单链表尾插入

SingleNodeList \*p=head,\*q;

while(p->next!=NULL)

p=p->next;

if((q=(SingleNodeList \*)malloc(sizeof(SingleNodeList)))==NULL){

exit(1);

}

q->data=x;q->next=NULL; //尾节点的数据域与指针域赋值

p->next=q;//尾节点加入链表

return 1;

}

int LinkedListGetMid(SingleNodeList \*head,ElemType \*x){//5取单链表中间的数据，存入x指针

【 1 】//请填入多行代码

}

int LinkedListGetLastofN(SingleNodeList \*head,int n,ElemType \*x){//6取单链表倒数第n个节点数据，存入x指针

SingleNodeList \*fast,\*slow;

int j=0;

fast=slow=head;

while(fast->next!=NULL){

【 2 】//请填入多行代码

}

if(j<n) \*x=-12345;

else \*x=slow->data;

return 1;

}

int LinkedListReversal(SingleNodeList \*head,int n){//7单链表翻转n（循环前移n位）

SingleNodeList \*fast,\*slow;

int j=0;

fast=slow=head;

while(fast->next!=NULL){

【 3 】//请填入多行代码

}

fast->next=head->next;

head->next=slow->next;

slow->next=NULL;

return 0;

}

// 循环链表

//1》判断某有头节点单链表是否为循环链表

//2》如果是循环链表，求出环的长度

//3》求出环的起点

//4》将循环链表断链成一个单链表

int Is\_circular\_linked\_list(SingleNodeList \*head,ElemType \*x){//8判断是否为循环链表 ，是则返回1，否则返回0

【 4 】//请填入多行代码}

int LinkedListShow\_N(SingleNodeList \*head,int n){//9打印单链表前n个节点

SingleNodeList \*p=head;

int i=0;

if(p->next==NULL){

printf("There is no data in the Linkedlist to print.\n");

return 0;

}

【 5 】//请填入多行代码

printf("\n");

return 1;

}

void LinkedListDestroy(SingleNodeList \*\*head){//10释放链表

SingleNodeList \*p=\*head,\*q;

while(p!=NULL){

q=p;

p=p->next;

free(q);

}

\*head=NULL;

}

int main(){

SingleNodeList \*head,\*p,\*q;

ElemType i,x;

int switch\_num;

LinkedListInit(&head);

scanf("%d",&switch\_num);

for(i=1;i<=20;i++)

LinkedList\_PushBack(head,i);

printf("%d\n",LinkedListLength(head));

LinkedListShow(head);

switch(switch\_num){

case 1:

LinkedListGetMid(head,&x);

printf("The mid of list is= %d\n",x);

break;

case 2: LinkedListGetLastofN(head,5,&x);

printf("The last of 5 in list is= %d\n",x);

break;

case 3:

LinkedListReversal(head,5);

LinkedListShow(head);

//printf("The last of 5 in list is= %d\n",x);

break;

case 4:

i=0;p=head;

while(p->next!=NULL ){

p=p->next;

i++;

if(i<=8) q=p;

}

p->next=q;

LinkedListShow\_N(head,50);

printf("\n");

i=Is\_circular\_linked\_list(head,&x);

printf("The circular\_linked\_list'length is= %d\n",i);

printf("The circular\_linked\_list start at node = %d\n",x);

LinkedListShow(head);

break;

case 5:

i=Is\_circular\_linked\_list(head,&x);

printf("The circular\_linked\_list'length is= %d\n",i);

printf("The circular\_linked\_list start at node = %d\n",x);

LinkedListShow(head);

break;

}

LinkedListDestroy(&head);

return 0;

}

【测试用例】

1

20

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

The mid of list is= 10

2

20

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

The last of 5 in list is= 16

3

20

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

17 18 19 20 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

4

20

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 8 9 10 11 12 13 14 15 16 17 1

8 19 20 8 9 10 11 12 13 14 15 16 17 18 19 20 8 9 10 11 12

The circular\_linked\_list'length is= 13

The circular\_linked\_list start at node = 8

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

5

20

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

The circular\_linked\_list'length is= 0

The circular\_linked\_list start at node = -12345

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

**2.5单链表(综合运用)**

本题主要考查内容：

1>单链表划分

2>大数加法

3>单链表原地逆置

4>两个有序单链表归并排序

5>删除有序链表中所有重复元素

6>断两个有头节点单链表是否 相交，如果相交，将交点的数据域带回，否则带回-12345

【程序片段】

#include <stdio.h>

#include <stdlib.h>

#define MaxSize 100

typedef int ElemType;

typedef struct SingleNode{

ElemType data;

struct SingleNode \*next;

}SingleNodeList,\*Linklist;

void LinkedListInit(SingleNodeList \*\*head){//1初始化有头节点的单链表

Linklist p;

if((\*head=(SingleNodeList \*)malloc(sizeof(SingleNodeList)))==NULL){

exit(1);

}

(\*head)->next=NULL;

}

int LinkedList\_PushBack(SingleNodeList \*head,ElemType x){//2单链表尾插入

SingleNodeList \*p=head,\*q;

while(p->next!=NULL)

p=p->next;

if((q=(SingleNodeList \*)malloc(sizeof(SingleNodeList)))==NULL){

exit(1);

}

q->data=x;q->next=NULL; //尾节点的数据域与指针域赋值

p->next=q;//尾节点加入链表

return 1;

}

int LinkedListShow(SingleNodeList \*head){//3打印单链表

SingleNodeList \*p=head;

if(p->next==NULL){

printf("There is no data in the Linkedlist to print.\n");

return 0;

}

while(p->next!=NULL){

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

p=p->next;

}

printf("\n");

return 1;

}

void LinkedListDestroy(SingleNodeList \*\*head){//4释放链表

SingleNodeList \*p=\*head,\*q;

while(p!=NULL){

q=p;

p=p->next;

free(q);

}

\*head=NULL;

}

//此函数完成如下功能

//1》输入一行数据，例如 1 3 5 7 9 8 6 4 2 将这些数据组织成一个有头节点的单链表A，首元素

//为1. A链表： head(A)->1->3->5->7->9->8->6->4->2

//2》输入一个数X，例如X=5.5，将A链表分成B和C两个部分，B中节点值比X小，C中节点都比X大

//如：A将分解成B与C链表：head(B)->1->3->5->4->2 head(C)->6->8->9->7

//请注意B、C链表中元素在原链表A中的次序。

//3》将B与C合并，并用A表头指向之。如 head(A)->1->3->5->4->2->6->8->9->7

int LinkedList\_Divide(SingleNodeList \*head,ElemType x) {//5单链表的划分

【 1 】//请填入多行代码

}

int LinkedList\_Add(SingleNodeList \*head,ElemType x1,ElemType x2){

//6两数转换成链表结构后相加

【 2 】//请填入多行代码

}

int LinkedList\_Rever(SingleNodeList \*head){//7原地逆置

Linklist pre=NULL,next=NULL,p;

p=head->next;

//p是当前节点，pre是p的直接前驱，next是p的后继

while(p!=NULL){

【 3 】//请填入多行代码

}

head->next=pre;

return 0;

}

//P1单链表11->23->35->47->59

//P1单链表8->36->44->52

//合并后P1单链表8->11->23->35->35->44->47->52->59

int LinkedList\_MergingSort(Linklist p1,Linklist p2){//8p1与p2归并排序，并链接至p1的头节点

【 4 】//请填入多行代码

}

//如一个链表为 36 -> 37 -> 65 -> 76 -> 97 -> 98 -> 98 -> 98 -> 98 -> 98

//删除重复元素后为: 36 -> 37 -> 65 -> 76 -> 97 -> 98

int LinkedList\_Sorted\_Delete(Linklist head) {//9删除有序链表中所有重复元素

Linklist curr=head->next,temp;

while(curr->next!=NULL){

【 5 】//请填入多行代码

}

}

//10判断两个有头节点单链表是否相交，如果相交，将交点的数据域带回，否则带回-12345

int LinkedList\_Is\_Intersect(Linklist p1,Linklist p2,ElemType \*x){

【 6 】//请填入多行代码

}

int main(){

SingleNodeList \*head,\*p,\*q;

Linklist p1,p2;

ElemType i,x,x1,x2;

int switch\_num;

scanf("%d",&switch\_num);

switch(switch\_num){

case 1:

LinkedListInit(&head);

LinkedList\_PushBack(head,1);

LinkedList\_PushBack(head,3);

LinkedList\_PushBack(head,5);

LinkedList\_PushBack(head,7);

LinkedList\_PushBack(head,9);

LinkedList\_PushBack(head,8);

LinkedList\_PushBack(head,6);

LinkedList\_PushBack(head,4);

LinkedList\_PushBack(head,2);

LinkedListShow(head);

LinkedList\_Divide(head,5);

LinkedListShow(head);

break;

case 2:

LinkedListInit(&head);

LinkedList\_Add(head,12345,5678999);

LinkedListShow(head);

LinkedList\_Rever(head);

LinkedListShow(head);

break;

case 3:

LinkedListInit(&head);

LinkedList\_PushBack(head,1);

LinkedList\_PushBack(head,3);

LinkedList\_PushBack(head,5);

LinkedList\_PushBack(head,7);

LinkedList\_PushBack(head,9);

LinkedList\_PushBack(head,8);

LinkedList\_PushBack(head,6);

LinkedList\_PushBack(head,4);

LinkedList\_PushBack(head,2);

LinkedListShow(head);

LinkedList\_Rever(head);

LinkedListShow(head);

break;

case 4:

LinkedListInit(&p1);

LinkedListInit(&p2);

LinkedList\_PushBack(p1,11);

LinkedList\_PushBack(p1,23);

LinkedList\_PushBack(p1,35);

LinkedList\_PushBack(p1,47);

LinkedList\_PushBack(p1,59);

LinkedListShow(p1);

LinkedList\_PushBack(p2,8);

LinkedList\_PushBack(p2,36);

LinkedList\_PushBack(p2,44);

LinkedList\_PushBack(p2,52);

LinkedListShow(p2);

LinkedList\_MergingSort(p1,p2);

LinkedListShow(p1);

break;

case 5:

LinkedListInit(&head);

LinkedList\_PushBack(head,1);

LinkedList\_PushBack(head,1);

LinkedList\_PushBack(head,1);

LinkedList\_PushBack(head,1);

LinkedList\_PushBack(head,1);

LinkedList\_PushBack(head,3);

LinkedList\_PushBack(head,3);

LinkedList\_PushBack(head,3);

LinkedList\_PushBack(head,19);

LinkedList\_PushBack(head,19);

LinkedList\_PushBack(head,26);

LinkedList\_PushBack(head,34);

LinkedList\_PushBack(head,34);

LinkedList\_PushBack(head,34);

LinkedList\_PushBack(head,54);

LinkedListShow(head);

LinkedList\_Sorted\_Delete(head);

LinkedListShow(head);

break;

case 6:

LinkedListInit(&head);

LinkedList\_PushBack(head,1);

LinkedList\_PushBack(head,3);

LinkedList\_PushBack(head,5);

LinkedList\_PushBack(head,7);

LinkedList\_PushBack(head,9);

LinkedList\_PushBack(head,8);

LinkedList\_PushBack(head,6);

LinkedList\_PushBack(head,4);

LinkedList\_PushBack(head,2);

LinkedListShow(head);

LinkedListInit(&p1);

LinkedListInit(&p2);

LinkedList\_PushBack(p1,11);

LinkedList\_PushBack(p1,23);

LinkedList\_PushBack(p1,35);

LinkedList\_PushBack(p1,47);

LinkedList\_PushBack(p1,59);

p=p1;

while(p->next!=NULL)

p=p->next;

p->next=head->next;

LinkedListShow(p1);

LinkedList\_PushBack(p2,8);

LinkedList\_PushBack(p2,36);

LinkedList\_PushBack(p2,44);

LinkedList\_PushBack(p2,52);

p=p2;

while(p->next!=NULL)

p=p->next;

p->next=head->next;

LinkedListShow(p2);

LinkedList\_Is\_Intersect(p1,p2,&x);

printf("The intersect node num = %d\n",x);

}

//LinkedListDestroy(&head);

return 0;

}

【测试用例】

1

1 3 5 7 9 8 6 4 2

1 3 4 2 6 8 9 7

2

5 4 3 2 1

9 9 9 8 7 6 5

4 4 3 1 9 6 5

5 6 9 1 3 4 4

3

1 3 5 7 9 8 6 4 2

2 4 6 8 9 7 5 3 1

4

11 23 35 47 59

8 36 44 52

8 11 23 35 36 44 47 52 59

5

1 1 1 1 1 3 3 3 19 19 26 34 34 34 54

1 3 19 26 34 54

6

1 3 5 7 9 8 6 4 2

11 23 35 47 59 1 3 5 7 9 8 6 4 2

8 36 44 52 1 3 5 7 9 8 6 4 2

The intersect node num = 1