顺序表

**0**

Status delete\_duplicate(SqList &S){ //有序去重复

for (int i = 0; i < S.length; ){

if (S.elem[i] == S.elem[i + 1] && i != S.length-1){

cout << "delete " << S.elem[i + 1]<<" for iduplicated"<<endl;

listDelete(S, i+1);

}

else i++;

}

return 0;

}

**1**

Status delete\_0(SqList &S){ //删除0元素

for (int i = 0; i < S.length; i++){

if (S.elem[i] == 0){

listDelete(S, i);

i--;

}

}

return 0;

}

**2**

Status buble\_sort(SqList &S){ //冒泡排序

int t;

for (int i = S.length -1; i >= 0; i--){

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

if (S.elem[j] > S.elem[j + 1]){

t = S.elem[j];

S.elem[j] = S.elem[j + 1];

S.elem[j + 1] = t;

}

}

}return 0;

}

Status insertion\_sort(SqList &S){ //插入排序

int t;

for (int i = 0; i < S.length; i++){

printList(S);

for (int j = i; j >= 0; j--){

if (S.elem[i]>S.elem[j]){

t = S.elem[i];

for (int k = i; k > j + 1; k--){

S.elem[k] = S.elem[k - 1];

S.elem[j + 1] = t;

}break;

}

if (j == 0){

t = S.elem[i];

for (int k = i; k > 0; k--){

S.elem[k] = S.elem[k - 1];

S.elem[0] = t;

}

}

}

}return 0;

}

Status selection\_sort(SqList &S){ //选择排序

int t1, t2;

for (int i = 0; i < S.length; i++){

t1 = i;

for (int j = i; j < S.length - 1; j++){

if (S.elem[j + 1] < S.elem[t1])

t1 = j + 1;

}

t2 = S.elem[i]; S.elem[i] = S.elem[t1];S.elem[t1] = t2;

}

return 0;

}

**3**

Status pattern\_matching(SqList S1, SqList S2){ *//模式识别  
 if* (S2.length > S1.length)  
 *return* ERROR;  
 *for* (*int* i = 0; i < S1.length; i++){  
 *if* (S1.elem[i] == S2.elem[0]){  
 *for* (*int* j = i; j - i < S2.length; j++){  
 *if* (S2.elem[j - i] != S1.elem[j])  
 *return* ERROR;  
 cout << "match successfully, position " <<i<<endl;  
 *break*;  
 }  
 }  
 }  
}

**链表**

**0**

Status reverse(LinkList &L){ *//单链表逆置* LNode \*p = L->next, \*q = L;  
 *while*(q->next != **NULL**)  
 q = q->next;  
 q->next = p;  
 L->next = p->next;  
 q->next = p;  
 p->next = **NULL**;  
 *while*(L->next != q)  
 {  
 p = L->next;  
 L->next =p->next;  
 p->next = q->next;  
 q->next = p;  
 }  
}

**1**

Status found\_mid(LinkList &L){ *//找链表中点* LNode \*p = L, \*q = L;  
 *while*(q->next != **NULL**){  
 *if*(!q->next->next){  
 printf("mid is ");  
 printf("%d", p->next->data);  
 *return* **OK**;  
 }  
 p = p->next;  
 q = q->next->next;  
 }  
  
 printf("mid is ");  
 printf("%d", p->data);  
 printf(" and ");  
 printf("%d", p->next->data);  
  
}

**2**

Status find\_k(LinkList L, *int* k){  
 LinkList L\_tmp = L;  
 reverse(L\_tmp);  
 printLink(L\_tmp);  
 LNode \*p = L\_tmp;  
 *for*(*int* i = 0; i<k; i++)  
 p = p->next;  
 printf("%d",k);  
 printf(" from the bottom is ");  
 printf("%d",p->data);  
}

**3**

Status delete\_k (LinkList &L, *int* k){  
 reverse(L);  
 LNode \*p = L;  
 *for*(*int* i = 0; i<k-1; i++)  
 p = p->next;  
 LNode \*q = p->next;  
 p->next = p->next->next;  
 free(q);  
}

**4**

Status create\_circle(LinkList &L, *int* k){ *//构造环* LNode \*p = L, \*q = L;  
 *while*(p->next){  
 p = p->next;  
 }  
 *for*(*int* i = 0; i < k; i++){  
 q = q->next;  
 }  
 p->next = q;  
}  
  
Status find\_circle\_node(LinkList L){ *//找环节点* LNode \*p = L, \*q = L->next->next;  
 *while*(p != q ){  
 *if*(q->next == **NULL** || q->next ==**NULL**){  
 printf("not circe");  
 *return* **ERROR**;  
 }  
 p = p->next;  
 q = q->next->next;  
 }  
 p = L->next;  
 *int* count = 0;  
 *while*(p != q){  
 p = p->next;  
 q = q->next;  
 count++;  
 }  
 printf("the number of node is ");  
 printf("%d", count);  
}

**5**

Status create\_cross(LinkList &L1, LinkList &L2, *int* k){ *//链表相交* LNode \*p = L1, \*q = L2;  
 *for*(*int* i = 0; i < k; i++)  
 p = p->next;  
 *while*(q->next)  
 q = q->next;  
 q->next = p;  
}  
  
Status find\_cross\_node(LinkList L1, LinkList L2){  
 LNode \*p = L1;  
 *while*(p->next){  
 p = p->next;  
 }  
 p->next = L2->next;  
 find\_circle\_node(L1);  
}

**6**

Status delete\_LinkDuplicate(LinkList &L){ *//链表有序去重复* LNode \*p = L->next, \*q;  
 *while*(p->next){  
 *if*(p->data == p->next->data){  
 q = p->next;  
 p->next = q->next;  
 free(q);  
 }  
 *else* p = p->next;  
 }  
}

**7**

Status delete\_LinkDuplicate\_all(LinkList &L)*//链表有序去重复,不保留* LNode \*p = L, \*q;  
 *while*(p->next){  
 *int* tmp;  
 *if*(p->next->data == p->next->next->data){  
 tmp = p->next->data;  
 *while*(p->next && p->next->data == tmp){  
 q = p->next;  
 p->next = q->next;  
 free(q);  
 }  
 }  
 *else* p = p->next;  
 }  
}

**8**

Status Josephus\_kill(*int* n, *int* sep){ *//有损约瑟夫* LinkList L = linkInit();  
 *for*(*int* i = n; i >= 1; i--){  
 linkInsert(L, i);  
 }  
 printLink(L);  
  
 LNode \*p = L, \*q;  
 *while*(p->next)  
 p = p->next;  
 p->next = L->next;  
 p = L->next;  
 *while*(p->next != p){  
 *for*(*int* i=0; i < sep; i++)  
 p = p->next;  
 q = p->next;  
 p->next = q->next;  
 free(q);  
 }  
 printf("alive position is ");  
 printf("%d", p->data);  
}

*int* Josephus\_judge(LinkList L){ *//无损约瑟夫判断* LNode \*p = L->next;  
 *while*(p->data == 0)  
 p = p->next;  
 *int* tmp = p->data;  
 *while*(p->next->data == 0)  
 p = p->next;  
 *if*(p->next->data != tmp)  
 *return* 0;  
 *else return* tmp;  
}  
  
Status Josephus\_no\_kill(*int* n, *int* sep) { *//无损约瑟夫* LinkList L = linkInit();  
 *for* (*int* i = n; i >= 1; i--) {  
 linkInsert(L, i);  
 }  
 printLink(L);  
 LNode \*p = L, \*q;  
 *while* (p->next)  
 p = p->next;  
 p->next = L->next;  
 p = L->next;  
 *while*(!Josephus\_judge(L)){  
 *for*(*int* i=0; i <=sep;){  
 *if*(p->next->data == 0){  
 p = p->next;  
 }  
 *else* {  
 p = p->next;  
 i++;  
 }  
 }  
 p->data = 0;  
 }  
 printf("alive position is ");  
 printf("%d", Josephus\_judge(L));  
}

**9**

LinkList combine(LinkList &L1, LinkList&L2){ *//合并降序* reverse(L1);  
 *while*(L2->next != **NULL**)  
 {  
 LNode \*q, \*p;  
 *while*(L2->next != **NULL**)  
 {  
 p = L1;  
 q = L2->next;  
 *while* (p->next != **NULL** && q->data <= p->next->data)  
 p = p->next;  
 L2->next = q->next;  
 q->next = p->next;  
 p->next = q;  
 }  
 free(L2);  
 }  
}

LinkList combine\_upper(LinkList &L1, LinkList&L2){ *//合并升序  
 while*(L2->next != **NULL**)  
 {  
 LNode \*q, \*p;  
 *while*(L2->next != **NULL**)  
 {  
 p = L1;  
 q = L2->next;  
 *while* (p->next != **NULL** && q->data >= p->next->data)  
 p = p->next;  
 L2->next = q->next;  
 q->next = p->next;  
 p->next = q;  
 }  
 free(L2);  
 }  
}

**10**

Status is\_symmetry(LinkList L){ *//判断对称* LinkList L\_tmp = linkInit();;  
 LNode \*q = L->next, \*p = L\_tmp;  
 *while*(q){  
 linkInsert(p, q->data);  
 p = p->next;  
 q = q->next;  
 }  
 reverse(L\_tmp);  
 p = L->next;  
 q = L\_tmp->next;  
 *while*(p->next){  
 *if*(q->data != p->data){  
 printf("not symmetry");  
 *return* **OK**;  
 }  
 cout<<p->data<<"\t"<<q->data<<endl;  
 p = p->next;  
 q = q->next;  
 }  
 printf("is symmetry");  
}