**第2章 线性表**

**一、基础知识题**

**2.1❶ 描述以下三个概念的区别：头指针，头结点，首元结点（第一个元素结点）。**

**首元结点是指链表中存储线性表中第一个数据元素a1的结点。**

**头结点是为了操作方便，在链表的首元结点之前附设的一个结点，该结点的数据域中不存储线性表的数据元素，其作用是为了对链表进行操作时，可以对空表、非空表的情况以及对首元结点进行统一处理。**

**头指针是指向链表中第一个结点（或为头结点或为首元结点）的指针。若链表中附设头结点，则不管线性表是否为空表，头指针均不为空，否则表示空表的链表的头指针为空。**

**这三个概念对单链表、双向链表和循环链表均适用。是否设置头结点，是不同的存储结构表示同一逻辑结构的间题。**

**2.2❶ 填空题**

**（1）在顺序表中插入或删除一个元素，需要平均移动\_\_表中一半\_\_元素，具体移动的元素个数与\_\_该元素的位置\_\_有关。**

**（2）顺序表中逻辑上相邻的元素的物理位置\_\_必定\_\_相邻。单链表中逻辑上相邻的元素在物理位置\_\_不一定\_\_相邻。**

**（3）在单链表中，除了首元结点外，任一结点的存储位置由\_\_其直接前驱结点的链域的值\_\_指示。**

**（4）在单链表中设置头结点的作用是\_\_插入和删除首元素时不必进行特殊处理\_\_。**

**2.3❷ 在什么情况下用顺序表比链表好？**

**当不需频繁在存储的元素间进行插入和删除操作时，用顺序表较好。**

**2.4❶ 对以下单链表分别执行下列各程序段，并画出结果示意图。**

****

**（1）Q=P->next;**

**（2）L=P->next;**

**（3）R->data=P->data;**

**（4）R->data=P->next->data;**

**（5）P->next->next->next->data=P->data;**

**（6）T=P;**

**while(T!=NULL)**

**{**

**T->data=T->data\*2;**

**T=T->next;**

**}**

**（7）T=P;**

**while(T->next!=NULL)**

**{**

**T->data=T->data\*2;**

**T=T->next;**

**}**

**(1)**



**(2)**



**(3)**



**(4)**



**(5)**



**(6)**



**(7)**



**2.5❶ 画出执行下列各行语句后各指针及链表的示意图。**

**L = (LinkList) malloc (sizeof(LNode));**

**P = L;**

**for(i=1; i<=4; i++)**

**{**

**P->next= (LinkList) malloc (sizeof(LNode));**

**P = P->next;**

**P->data = i\*2-1;**

**}**

**P->next = NULL;**

**for(i=4; i>=1; i--)**

**Ins\_LinkList(L, i+1, i\*2);**

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

**Del\_LinkList(L, i);**









**2.6❷ 已知L是无表头结点的单链表，且P结点既不是首元结点，也不是尾元结点，试从下列提供的答案中选择合适的语句序列。**

**a.在P结点后插入S结点的语句序列是\_\_（4）（1）\_\_。**

**b.在P结点前插入S结点的语句序列是\_\_（7）（11）（8）（4）（1）\_\_。**

**c.在表首插入S结点的语句序列是\_\_（5）（12）\_\_。**

**d.在表尾插入S结点的语句序列是\_\_（9）（1）（6）\_\_。**

**（1）P->next=S;**

**（2）P->next=P->next->next;**

**（3）P->next=S->next;**

**（4）S->next=P->next;**

**（5）S->next=L;**

**（6）S->next=NULL;**

**（7）Q=P;**

**（8）while(P->next!=Q)**

**P=P->next;**

**（9）while(P->next!=NULL)**

**P=P->next;**

**（10）P=Q;**

**（11）P=L;**

**（12）L=S;**

**（13）L=P;**

**2.7❷ 已知L是带表头结点的非空单链表，且P结点既不是首元结点，也不是尾元结点，试从下列提供的答案中选择合适的语句序列。**

**a.删除P结点的直接后继结点的语句序列是\_\_（11）（3）（14）\_\_。**

**b.删除P结点的直接前驱结点的语句序列是\_\_（10）（12）（8）（3）（14）\_\_。**

**c.删除P结点的语句序列是\_\_（10）（12）（7）（3）（14）\_\_。**

**d.删除首元结点的语句序列是\_\_（12）（11）（3）（14）\_\_。**

**e.删除尾元结点的语句序列是\_\_（9）（11）（3）（14）\_\_。**

**（1）P=P->next;**

**（2）P->next=P;**

**（3）P->next=P->next->next;**

**（4）P=P->next->next;**

**（5）while(P!=NULL)**

**P=P->next;**

**（6）while(Q->next!=NULL)**

**{**

**P=Q;**

**Q=Q->next;**

**}**

**（7）while(P->next!=Q)**

**P=P->next;**

**（8）while(P->next->next!=Q)**

**P=P->next;**

**（9）while(P->next->next!=NULL)**

**P=P->next;**

**（10）Q=P;**

**（11）Q=P->next;**

**（12）P=L;**

**（13）L=L->next;**

**（14）free(Q);**

**2.8❷ 已知P结点是某双向链表的中间结点，试从下列提供的答案中选择合适的语句序列。**

**a.在P结点后插入S结点的语句序列是\_\_（7）（12）（3）（6）\_\_。**

**b.在P结点前插入S结点的语句序列是\_\_（8）（4）（5）（13）\_\_。**

**c.删除P结点的直接后继结点的语句序列是\_\_（15）（1）（11）（18）\_\_。**

**d.删除P结点的直接前驱结点的语句序列是\_\_（16）（2）（10）（18）\_\_。**

**e.删除P结点的语句序列是\_\_（14）（9）（17）\_\_。**

**（1）P->next=P->next->next;**

**（2）P->priou=P->priou->priou;**

**（3）P->next=S;**

**（4）P->priou=S;**

**（5）S->next=P;**

**（6）S->priou=P;**

**（7）S->next=P->next;**

**（8）S->priou=P-priou;**

**（9）P->priou->next=P->next;**

**（10）P->priou->next=P;**

**（11）P->next->priou=P;**

**（12）P->next->priou=S;**

**（13）P->priou->next=S;**

**（14）P->next->priou=P->priou;**

**（15）Q=P->next;**

**（16）Q=P->priou;**

**（17）free(P);**

**（18）free(Q);**

**2.9❷ 简述下列算法的功能。**

**（1）Status A(LinkedList L) //L是无表头结点的单链表**

**{**

**if(L&&L->next)**

**{**

**Q=L;**

**L=L->next;**

**P=L;**

**while(P->next)**

**P=P->next;**

**P->next=Q;**

**Q->next=NULL;**

**}**

**return OK;**

**}//A**

**（2）void BB(LNode \*s, LNode \*q)**

**{**

**p=s;**

**while(p->next!=q)**

**p=p->next;**

**p->next=s;**

**}//BB**

**void AA(LNode \*pa, LNode \*pb)**

**{//pa和pb分别指向单循环链表中的两个结点**

**BB(pa, pb);**

**BB(pb, pa);**

**}//AA**

**（1）如果L的长度不小于2，则将首元结点删去并插入表尾。**

**（2）将单循环链表拆成两个单循环链表。**

**二、算法设计题**

**本章算法设计题设计的顺序表和线性链表的类型定义如下：**

**#define LIST\_INIT\_SIZE 100**

**#define LISTINCREMENT 10**

**typedef struct**

**{**

**ElemType \*elem; //存储空间基址**

**int length; //当前长度**

**int listsize; //当前分配的存储容量**

**}SqList; //顺序表类型**

**注：此文档中，ElemType被定义为int类型。**

**typedef struct LNode**

**{**

**ElemType data;**

**Struct Lnode \*next;**

**}LNode, \*LinkList; //线性链表类型**

**┏━━━━┓**

**┣ 顺序表 ┫**

**┗━━━━┛**

**2.10❷ 指出以下算法的错误和低效（即费时）之处，并将它改写为一个既正确又高效的算法。**

**Status DeleteK(SqList &a, int i, int k)**

**{//本过程从顺序存储结构的线性表a中删除第i个元素起的k个元素**

**if(i<1 || k<0 || i+k>a.length)**

**return INFEASIBLE; //参数不合法**

**else**

**for(count=1; count<k; count++)**

**{//删除一个元素**

**for(j=a.length; j>=i+1; j--)**

**a.elem[j-1] = a.elem[j];**

**a.length--;**

**}**

**return OK;**

**}//DeleteK**

**错误有两处：**

**（1）参数不合法的判别条件不完整。合法的入口参数条件为：（删除时包括第i个元素）**

**(0<i≤a.length)&& (0≤k≤a.length-i+1)**

**（2）第二个for语句中，元素前移的次序错误。**

**低效之处是每次删除一个元素的策略。**

**修改如下：**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/01 SequenceList/SequenceList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_10(SqList \*a, int i, int k);**

**void PrintElem(LElemType\_Sq e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**SqList L;**

**int i;**

**if(InitList\_Sq(&L))** **//链表L创建成功**

**{**

**for(i=1; i<=20; i++)** **//链表L中元素1~20**

**ListInsert\_Sq(&L, i, i);**

**}**

**printf("L = ");**

**ListTraverse\_Sq(L, PrintElem);** **//输出L**

**printf("\n\n");**

**printf("删除第 5 个元素起的 10 个元素...\n");**

**Algo\_2\_10(&L, 5, 10);**

**printf("此时L = ");**

**ListTraverse\_Sq(L, PrintElem);** **//输出L**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━━━━━━━━┓**

**┃题2.10：删除顺序表中从第i个元素起的k个元素┃**

**┗━━━━━━━━━━━━━━━━━━━━━\*/**

**Status Algo\_2\_10(SqList \*a, int i, int k)**

**{**

**int j;**

**if(i<1 || i>(\*a).length || k<0 || i+k-1>(\*a).length)**

**return ERROR;**

**for(j=i+k; j<=(\*a).length; j++)**

**(\*a).elem[j-k-1] = (\*a).elem[j-1];**

**(\*a).length -= k;**

**return OK;**

**}**

**void PrintElem(LElemType\_Sq e)**

**{**

**printf("%d ", e);**

**}**

**2.11❷ 设顺序表va中的数据元素递增有序。试写一算法，将x插入到顺序表的适当位置上，以保持该表的有序性。**

**#include <stdio.h>**

**#include <stdlib.h> //提供malloc、realloc、free、exit原型**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/01 SequenceList/SequenceList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_11(SqList \*va, LElemType\_Sq x);**

**void PrintElem(LElemType\_Sq e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**SqList L;**

**int i;**

**if(InitList\_Sq(&L))** **//链表L创建成功**

**{**

**for(i=1; i<=10; i++)** **//链表L中元素1~20**

**ListInsert\_Sq(&L, i, 2\*i);**

**}**

**printf("L = ");**

**ListTraverse\_Sq(L, PrintElem);** **//输出L**

**printf("\n\n");**

**printf("将元素 \"5\" 插入到链表L中...\n");**

**Algo\_2\_11(&L, 5);**

**printf("此时L = ");**

**ListTraverse\_Sq(L, PrintElem);** **//输出L**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━━┓**

**┃题2.11：将x插入到递增序列va中 ┃**

**┗━━━━━━━━━━━━━━━\*/**

**Status Algo\_2\_11(SqList \*va, LElemType\_Sq x)**

**{**

**int i;**

**LElemType\_Sq \*newbase;**

**if(!(\*va).length)**

**return ERROR;**

**if((\*va).length==(\*va).listsize)** **//若存储空间已满，需开辟新空间**

**{**

**newbase = (LElemType\_Sq\*)realloc((\*va).elem, ((\*va).listsize+LISTINCREMENT)\*sizeof(LElemType\_Sq));**

**if(!newbase)**

**exit(OVERFLOW);**

**(\*va).elem = newbase;**

**(\*va).listsize += LISTINCREMENT;**

**}**

**for(i=(\*va).length; i>=1; i--)**

**{**

**if((\*va).elem[i-1]>x)**

**(\*va).elem[i] = (\*va).elem[i-1];**

**else**

**break;**

**}**

**(\*va).elem[i] = x;**

**(\*va).length++;**

**return OK;**

**}**

**void PrintElem(LElemType\_Sq e)**

**{**

**printf("%d ", e);**

**}**

**2.12❸ 设A=(a1,...,an)和B=(b1,...,bn)均为顺序表，A'和B'分别为A和B中除去最大共同前缀后的子表（例如，A=(x,y,y,z,x,z)，B=(x,y,y,z,y,x,x,z)，则两者中最大的共同前缀为(x,y,y,z)，在两表中除去最大共同前缀后的子表分别为A'=(x,z)和B'=(y,x,x,z)）。若A'=B'=空表，则A=B；若A'=空表，而B'≠空表，或者两者均不为空表，且A'的首元小于B'的首元，则A<B；否则A>B。试写一个比较A，B大小的算法（请注意：在算法中，不要破坏原表A和B，并且，也不一定先求得A'和B'才进行比较）。**

**#include <stdio.h>**

**#include <stdlib.h> //提供malloc、realloc、free、exit原型**

**#include "../../../▲课本算法实现/▲02 线性表/01 SequenceList/SequenceList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**int Algo\_2\_12(SqList A, SqList B);**

**void PrintElem(LElemType\_Sq e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**int i, mark;**

**SqList A, B;**

**int a[7] = {1, 2, 3, 4, 5, 9, 12};**

**int b[7] = {1, 2, 3, 4, 5, 11, 12};**

**InitList\_Sq(&A);**

**InitList\_Sq(&B);**

**for(i=1; i<=7; i++)**

**{**

**ListInsert\_Sq(&A, i, a[i-1]);**

**ListInsert\_Sq(&B, i, b[i-1]);**

**}**

**printf("构建完成的顺序表为：\n");**

**printf("A = ");**

**ListTraverse\_Sq(A, PrintElem);**

**printf("\n");**

**printf("B = ");**

**ListTraverse\_Sq(B, PrintElem);**

**printf("\n\n");**

**mark = Algo\_2\_12(A, B);**

**printf("后缀比较结果：");**

**if(mark<0)**

**printf("A<B\n");**

**else if(mark>1)**

**printf("A>B\n");**

**else**

**printf("A==B\n");**

**printf("\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━┓**

**┃题2.12：比较顺序表后缀大小┃**

**┗━━━━━━━━━━━━━\*/**

**int Algo\_2\_12(SqList A, SqList B)**

**{**

**int i;**

**i = 0;**

**while(i<A.length && i<B.length)** **//A、B均未扫描完**

**{**

**if(A.elem[i]>B.elem[i])**

**return 1;**

**else if(A.elem[i]<B.elem[i])**

**return -1;**

**else**

**i++;** **//相等时比较下一元素**

**}**

**if(i<A.length)** **//A还有剩余，A大**

**return 1;**

**else if(i<B.length)** **//B还有剩余，B大**

**return -1;**

**else**

**return 0;** **//同时扫描完，相等**

**}**

**void PrintElem(LElemType\_Sq e)**

**{**

**printf("%d ", e);**

**}**

**┏━━━━┓**

**┣ 单链表 ┫**

**┗━━━━┛**

**2.13❷ 试写一算法在带头结点的单链表结构上实现线性表操作LOCATE(L,X)。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**int Algo\_2\_13(LinkList L, LElemType\_L x);**

**void PrintElem(LElemType\_L e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**LinkList L;**

**int i;**

**if(InitList\_L(&L))** **//链表L创建成功**

**{**

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

**ListInsert\_L(L, i, 2\*i);**

**}**

**printf("L = ");**

**ListTraverse\_L(L, PrintElem);** **//输出L**

**printf("\n\n");**

**printf("元素 \"12\" 在链表L中的位置为 %d \n", Algo\_2\_13(L, 18));**

**printf("\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━┓**

**┃题2.13：寻找元素x再L中的位置┃**

**┗━━━━━━━━━━━━━━\*/**

**int Algo\_2\_13(LinkList L, LElemType\_L x)**

**{**

**int i;**

**LinkList p;**

**if(L)**

**{**

**i = 1;**

**p = L->next;**

**while(p)**

**{**

**if(p->data==x)**

**return i;**

**i++;**

**p = p->next;**

**}**

**}**

**return 0;**

**}**

**void PrintElem(LElemType\_L e)**

**{**

**printf("%d ", e);**

**}**

**2.14❷ 试写一算法在带头结点的单链表结构上实现线性表操作LENGTH(L)。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**int Algo\_2\_14(LinkList L);**

**void PrintElem(LElemType\_L e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**LinkList L;**

**int i;**

**if(InitList\_L(&L))** **//链表L创建成功**

**{**

**for(i=1; i<=10; i++)** **//链表L中元素1~20**

**ListInsert\_L(L, i, 2\*i);**

**}**

**printf("L = ");**

**ListTraverse\_L(L, PrintElem);** **//输出L**

**printf("\n\n");**

**printf("链表L的长度为 %d \n", Algo\_2\_14(L));**

**printf("\n");**

**return 0;**

**}**

**/\*━━━━━━━━┓**

**┃题2.14：求L长度 ┃**

**┗━━━━━━━━\*/**

**int Algo\_2\_14(LinkList L)**

**{**

**return ListLength\_L(L);** **//已定义**

**}**

**void PrintElem(LElemType\_L e)**

**{**

**printf("%d ", e);**

**}**

**2.15❷ 已知指针ha和hb分别指向两个单链表的头结点，并且已知两个链表的长度分别为m和n。试写一算法将这两个链表连接在一起（即令其中一个表的首元结点连在另一个表的最后一个结点之后），假设指针hc指向连接后的链表的头结点，并要求算法以尽可能短的时间完成连接运算。请分析你的算法和时间复杂度。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_15(LinkList ha, LinkList hb, LinkList \*hc);**

**void PrintElem(LElemType\_L e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**LinkList ha, hb, hc;**

**int i;**

**if(InitList\_L(&ha) && InitList\_L(&hb) && InitList\_L(&hc))** **//链表创建成功**

**{**

**for(i=1; i<=5; i++)** **//链表赋值**

**ListInsert\_L(ha, i, i);**

**for(i=1; i<=7; i++)** **//链表赋值**

**ListInsert\_L(hb, i, 2\*i);**

**}**

**printf("ha = ");**

**ListTraverse\_L(ha, PrintElem);** **//输出**

**printf("\n");**

**printf("hb = ");**

**ListTraverse\_L(hb, PrintElem);**

**printf("\n\n");**

**Algo\_2\_15(ha, hb, &hc);**

**printf("连接ha和hb之后的链表为：\nhc = ");**

**ListTraverse\_L(hc, PrintElem);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━┓**

**┃题2.15：连接ha和hb┃**

**┗━━━━━━━━━\*/**

**Status Algo\_2\_15(LinkList ha, LinkList hb, LinkList \*hc)**

**{**

**LinkList pa, pb;** **//pa、pb分别指向ha、hb头结点**

**if(ha && hb)**

**{**

**pa = ha;**

**pb = hb;**

**while(pa->next && pb->next)**

**{**

**pa = pa->next;**

**pb = pb->next;**

**}**

**if(!pa->next)**

**{**

**\*hc = ha;**

**pa->next = hb->next;**

**}**

**if(!pb->next)**

**{**

**\*hc = hb;**

**pb->next = ha->next;**

**}**

**return OK;**

**}**

**return ERROR;**

**}**

**void PrintElem(LElemType\_L e)**

**{**

**printf("%d ", e);**

**}**

**时间复杂度分析：此算法费时的操作主要为遍历链表，故时间复杂度只与链表ha或hb的长度有关，其时间复杂度为：m≤n ? O(m) : O(n)。**

**2.16❸ 已知指针la和lb分别指向两个无头结点单链表中的首元结点。下列算法是从表la中删除自第i个元素起共len个元素后，将它们插入到表lb中的第j个元素之前。试问此算法是否正确？如有错，则请改正之。**

**Status DeleteAndInsertSub (LinkedList la, LinkedList lb, int i, int j, int len)**

**{**

**if(i<0 || j<0 || len<0)**

**return INFEASIBLE;**

**p=la; k=1;**

**while(k<i)**

**{**

**p=p->next;**

**k++;**

**}**

**q=p;**

**while(k<=len)**

**{**

**q=q->next;**

**k++;**

**}**

**s=lb;**

**k=1;**

**while(k<j)**

**{**

**s=s->next;**

**k++;**

**}**

**s->next=p;**

**q->next=s->next;**

**return OK;**

**}//DeleteAndInsertSub**

**2.17❷ 试写一算法，在无头结点的动态单链表上实现线性表操作INSERT(L, i, b)，并和在带头结点的动态单链表上实现相同操作的算法进行比较。**

**2.18❷ 同2.17题要求。试写一算法，实现线性表操作DELETE(L, i)。**

**#include <stdio.h>**

**#include <stdlib.h> //提供malloc、realloc、free、exit原型**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_16(LinkList \*la, LinkList \*lb, int i, int j, int len);**

**Status Algo\_2\_17(LinkList \*L, int i, LElemType\_L b);**

**Status Algo\_2\_18(LinkList \*L, int i);**

**void InitList\_2\_16(LinkList \*L); //初始化L**

**void Output\_2\_16(LinkList L); //输出L**

**int main(int argc, char \*argv[])**

**{**

**int i;**

**LinkList la, lb;**

**InitList\_2\_16(&la);**

**InitList\_2\_16(&lb);**

**printf("███题 2.17 验证...███\n");**

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

**{**

**Algo\_2\_17(&la, i, 2\*i-1);**

**Algo\_2\_17(&lb, i, 2\*i);**

**}**

**printf("创建好的无头结点链表为：\n");**

**printf("la = ");**

**Output\_2\_16(la);**

**printf("\n");**

**printf("lb = ");**

**Output\_2\_16(lb);**

**printf("\n\n");**

**printf("███题 2.16 验证...███\n");**

**printf("将la中从第2个结点起的5个结点插入到lb的第6个结点之前...\n");**

**Algo\_2\_16(&la, &lb, 2, 6, 5);**

**printf("la = ");**

**Output\_2\_16(la);**

**printf("\n");**

**printf("lb = ");**

**Output\_2\_16(lb);**

**printf("\n\n");**

**printf("███题 2.18 验证...███\n");**

**printf("删除lb第6个结点起的5个结点...\n");**

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

**Algo\_2\_18(&lb, 6);**

**printf("删除完成后：lb = ");**

**Output\_2\_16(lb);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓**

**┃题2.16：删除la中第i个结点起的len个结点并添加到lb第j个结点之前 ┃**

**┗━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\*/**

**/\* 本题中的链表无头结点 \*/**

**Status Algo\_2\_16(LinkList \*la, LinkList \*lb, int i, int j, int len)**

**{**

**LinkList p, q, s, prep;**

**int k;**

**if(i<0 || j<0 || len<0)**

**return INFEASIBLE;**

**p = \*la;**

**k = 1;**

**prep = NULL;**

**while(p && k<i)** **//在la中查找第i个结点，用p标记**

**{**

**prep = p;**

**p = p->next;**

**k++;**

**}**

**if(!p)** **//找不到第i个元素**

**return INFEASIBLE;**

**q = p;** **//p指向la表中第i个结点**

**while(q && k<i+len-1)** **//查找la表中第i+len-1个结点，用q标记**

**{**

**q = q->next;**

**k++;**

**}**

**if(!q)**

**return INFEASIBLE;**

**if(!prep)** **//i=1的情况**

**\*la = q->next;**

**else** **//完成删除**

**prep->next = q->next;**

**if(j==1)**

**{**

**q->next = \*lb;**

**\*lb = p;**

**}**

**else**

**{**

**s = \*lb;**

**k = 1;**

**while(s && k<j-1)** **//查找lb表中第j-1个元素**

**{**

**s = s->next;**

**k++;**

**}**

**if(!s)**

**return INFEASIBLE;**

**q->next = s->next;**

**s->next = p;** **//完成插入**

**return OK;**

**}**

**}**

**/\*━━━━━━━━━━━━━━━┓**

**┃题2.17：将b插入为L的第i个结点 ┃**

**┗━━━━━━━━━━━━━━━\*/**

**/\* 本题中的链表无头结点 \*/**

**Status Algo\_2\_17(LinkList \*L, int i, LElemType\_L b)**

**{**

**LinkList p, q;**

**int count;**

**p = (LinkList)malloc(sizeof(LNode));**

**if(!p)**

**exit(OVERFLOW);**

**p->data = b;**

**if(i>0)**

**{**

**if(i==1)**

**{**

**p->next = \*L;**

**\*L = p;**

**return OK;**

**}**

**else**

**{**

**if(\*L)**

**{**

**count = 1;**

**q = \*L;**

**while(count<i-1 && q)**

**{**

**count++;**

**q = q->next;**

**}**

**if(q)**

**{**

**p->next = q->next;**

**q->next = p;**

**return OK;**

**}**

**}**

**}**

**}**

**return ERROR;**

**}**

**/\*━━━━━━━━━━━━┓**

**┃题2.18：删除L的第i个结点┃**

**┗━━━━━━━━━━━━\*/**

**/\* 本题中的链表无头结点 \*/**

**Status Algo\_2\_18(LinkList \*L, int i)**

**{**

**LinkList p, q;**

**int count;**

**if(i>1)**

**{**

**p = \*L;**

**count = 1;**

**while(p && count<i-1)**

**{**

**count++;**

**p = p->next;**

**}**

**if(p)**

**{**

**if(count>i-1)** **//删除头结点**

**{**

**\*L = (\*L)->next;**

**free(p);**

**return OK;**

**}**

**else**

**{**

**if(p->next)** **//第i个结点存在**

**{**

**q = p->next;**

**p->next = q->next;**

**free(q);**

**return OK;**

**}**

**}**

**}**

**}**

**return ERROR;**

**}**

**void InitList\_2\_16(LinkList \*L)**

**{**

**\*L = NULL;**

**}**

**void Output\_2\_16(LinkList L)**

**{**

**LinkList p = L;**

**while(p)**

**{**

**printf("%2d ", p->data);**

**p = p->next;**

**}**

**}**

**2.19❸ 已知线性表中的元素以值递增有序排列，并以单链表作存储结构。试写一高效的算法，删除表中所有值大于mink且小于maxk的元素（若表中存在这样的元素），同时释放被删结点空间，并分析你的算法的时间复杂度（注意：mink和maxk是给定的两个参变量，它们的值可以和表中的元素相同，也可以不同）。**

**时间复杂度分析：最坏的情况是全部扫描完也没找到适合的元素，故时间复杂度与链表长度有关，为O(Length(L))。**

**2.20❷ 同2.19题条件（递增有序排列），试写一高效的算法，删除表中所有值相同的多余元素（使得操作后的线性表中所有元素的值均不相同），同时释放被删结点空间，并分析你的算法的时间复杂度。**

**#include <stdio.h>**

**#include <stdlib.h> //提供malloc、realloc、free、exit原型**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_19\_1(LinkList L, int mink, int maxk);**

**Status Algo\_2\_19\_2(LinkList L, int mink, int maxk);**

**Status Algo\_2\_20(LinkList L);**

**void PrintElem(LElemType\_L e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**LinkList L;**

**int mink, maxk\_1, maxk\_2, i;**

**int a[] = {1, 2, 2, 3, 4, 4, 5, 6, 7, 7};**

**mink = 2;**

**maxk\_1 = 4;**

**maxk\_2 = 6;**

**InitList\_L(&L);**

**for(i=0; i<10; i++)**

**ListInsert\_L(L, i+1, a[i]);**

**printf("原链表L = ");**

**ListTraverse\_L(L, PrintElem);**

**printf("\n\n");**

**printf("███题 2.19 验证...███\n");**

**printf("删除链表中大于 %d 且小于 %d 的元素之后...\n", mink, maxk\_1);**

**Algo\_2\_19\_1(L, mink, maxk\_1);**

**printf("L = ");**

**ListTraverse\_L(L, PrintElem);**

**printf("\n");**

**printf("删除链表中大于 %d 且小于 %d 的元素之后...\n", mink, maxk\_2);**

**Algo\_2\_19\_2(L, mink, maxk\_2);**

**printf("L = ");**

**ListTraverse\_L(L, PrintElem);**

**printf("\n\n");**

**printf("███题 2.20 验证...███\n");**

**printf("去掉链表中多余的重复元素...\n");**

**Algo\_2\_20(L);**

**printf("L = ");**

**ListTraverse\_L(L, PrintElem);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━━━━━━━━━━━━━━┓**

**┃题2.19：删除递增线性表中元素大于mink小于maxk之间的结点┃**

**┗━━━━━━━━━━━━━━━━━━━━━━━━━━━\*/**

**/\* 方法1 \*/**

**Status Algo\_2\_19\_1(LinkList L, int mink, int maxk)**

**{**

**LinkList p, pre, s;**

**if(!L || !L->next)** **//L不存在或为空表时，无法删除**

**return ERROR;**

**if(mink>=maxk)** **//阙值设置错误**

**return ERROR;**

**pre = L;**

**p = pre->next;** **//p指向首结点**

**while(p && p->data<maxk)** **//上限**

**{**

**if(p->data<=mink)** **//下限**

**{**

**pre = p;**

**p = p->next;**

**}**

**else** **//删掉满足条件的结点**

**{**

**s = p;**

**pre->next = p->next;**

**p = p->next;**

**free(s);**

**}**

**}**

**return OK;**

**}**

**/\*━━━━━━━━━━━━━━━━━━━━━━━━━━━┓**

**┃题2.19：删除递增线性表中元素大于mink小于maxk之间的结点┃**

**┗━━━━━━━━━━━━━━━━━━━━━━━━━━━\*/**

**/\* 方法2 \*/**

**Status Algo\_2\_19\_2(LinkList L, int mink, int maxk)**

**{**

**LinkList p, pre, s;**

**if(!L || !L->next)** **//L不存在或为空表时，无法删除**

**return ERROR;**

**if(mink>=maxk)** **//阙值设置错误**

**return ERROR;**

**pre = L;**

**p = pre->next;** **//p指向首结点**

**while(p && p->data<=mink)** **//下限**

**{**

**pre = p;**

**p = p->next;**

**}**

**if(p)**

**{**

**while(p && p->data<maxk)** **//上限**

**{**

**s = p;**

**pre->next = p->next;**

**p = p->next;**

**free(s);**

**}**

**return OK;**

**}**

**}**

**/\*━━━━━━━━━━━━━━━━┓**

**┃题2.20：删除表中值相同的多余结点┃**

**┗━━━━━━━━━━━━━━━━\*/**

**Status Algo\_2\_20(LinkList L)**

**{**

**LinkList p, pre, s;**

**if(!L || !L->next)** **//L不存在或为空表时，无法删除**

**return ERROR;**

**pre = L->next;**

**p = pre->next;** **//p指向首结点**

**while(p)**

**{**

**if(pre->data==p->data)**

**{**

**s = p;**

**pre->next = p->next;**

**p = p->next;**

**free(s);**

**}**

**else**

**{**

**pre = p;**

**p = p->next;**

**}**

**}**

**return OK;**

**}**

**void PrintElem(LElemType\_L e)**

**{**

**printf("%d ", e);**

**}**

**┏━━━━┓**

**┣ 顺序表 ┫**

**┗━━━━┛**

**2.21❸ 试写一算法，实现顺序表的就地逆置，即利用原表的存储空间将线性表(a1, a2, ..., an)逆置为(an, an-1, ..., a1)。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/01 SequenceList/SequenceList.c" //\*\*▲02 线性表\*\*//#include <stdio.h>**

**/\* 函数原型 \*/**

**Status Algo\_2\_21(SqList L);**

**void PrintElem(LElemType\_Sq e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**SqList L;**

**int i;**

**if(InitList\_Sq(&L))** **//链表L创建成功**

**{**

**for(i=1; i<=10; i++)** **//链表L中元素1~10**

**ListInsert\_Sq(&L, i, i);**

**}**

**printf("L = ");**

**ListTraverse\_Sq(L, PrintElem);** **//输出L**

**printf("\n\n");**

**printf("逆置顺序表...\n");**

**Algo\_2\_21(L);**

**printf("此时L = ");**

**ListTraverse\_Sq(L, PrintElem);** **//输出L**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━┓**

**┃题2.21：顺序表逆置┃**

**┗━━━━━━━━━\*/**

**Status Algo\_2\_21(SqList L)**

**{**

**int i;**

**int j;**

**LElemType\_Sq tmp;**

**if(L.length==0)**

**return ERROR;**

**for(j=1,i=(L.length)/2; j<=i; j++)**

**{**

**tmp = L.elem[j-1];**

**L.elem[j-1] = L.elem[L.length-j];**

**L.elem[L.length-j] = tmp;**

**}**

**return OK;**

**}**

**void PrintElem(LElemType\_Sq e)**

**{**

**printf("%d ", e);**

**}**

**┏━━━━┓**

**┣ 单链表 ┫**

**┗━━━━┛**

**2.22❸ 试写一算法，对单链表实现就地逆置。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_22(LinkList L);**

**void PrintElem(LElemType\_L e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**LinkList L;**

**int i;**

**if(InitList\_L(&L))** **//链表L创建成功**

**{**

**for(i=1; i<=10; i++)** **//链表L中元素1~10**

**ListInsert\_L(L, i, i);**

**}**

**printf("L = ");**

**ListTraverse\_L(L, PrintElem);** **//输出L**

**printf("\n\n");**

**printf("逆置单链表...\n");**

**Algo\_2\_22(L);**

**printf("此时L = ");**

**ListTraverse\_L(L, PrintElem);** **//输出L**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━┓**

**┃题2.22：单链表逆置┃**

**┗━━━━━━━━━\*/**

**Status Algo\_2\_22(LinkList L)**

**{**

**LinkList pre, p;**

**if(!L || !L->next)** **//链表不存在或链表为空**

**return ERROR;**

**p = L->next;**

**L->next = NULL;**

**while(p)** **//头插法**

**{**

**pre = p;**

**p = p->next;**

**pre->next = L->next;**

**L->next = pre;**

**}**

**return OK;**

**}**

**void PrintElem(LElemType\_L e)**

**{**

**printf("%d ", e);**

**}**

**2.23❸ 设线性表A=(a1, a2, ..., am)，B=(b1, b2, ..., bn)，试写一个按下列规则合并A，B为线性表C的算法，即使得**

**C=(a1, b1, ..., am, bm, bm+1, ..., bn) 当m<=n时；**

**或者 C=(a1, b1, ..., an, bn, an+1, ..., am) 当m>n时。**

**线性表A，B和C均以单链表作存储结构，且C表利用A表和B表中的结点空间构成。注意：单链表的长度值m和n均未显式存储。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_23\_1(LinkList La, LinkList \*Lb, LinkList \*Lc);**

**Status Algo\_2\_23\_2(LinkList La, LinkList \*Lb, LinkList \*Lc);**

**void PrintElem(LElemType\_L e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**LinkList La, Lb, Lc;**

**int i, mark;**

**if(InitList\_L(&La) && InitList\_L(&Lb) && InitList\_L(&Lc))** **//链表L创建成功**

**{**

**for(i=1; i<=5; i++)** **//创建链表La和Lb**

**{**

**ListInsert\_L(La, i, 2\*i-1);**

**ListInsert\_L(Lb, i, 2\*i);**

**}**

**}**

**printf("验证题 2.23 的方法1输入 1 ，验证题 2.23 的方法2输入 2 ：");**

**fflush(stdin);**

**scanf("%d", &mark);**

**printf("\n");**

**printf("创建好的链表为：\n");**

**printf("La = ");**

**ListTraverse\_L(La, PrintElem);**

**printf("\n");**

**printf("Lb = ");**

**ListTraverse\_L(Lb, PrintElem);** **//输出链表**

**printf("\n\n");**

**if(mark==1)**

**{**

**printf("题 2.23 方法1 验证...\n");**

**Algo\_2\_23\_1(La, &Lb, &Lc);**

**}**

**if(mark==2)**

**{**

**printf("题 2.23 方法2 验证...\n");**

**Algo\_2\_23\_2(La, &Lb, &Lc);**

**}**

**printf("归并La和Lb为Lc = ");**

**ListTraverse\_L(La, PrintElem);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━┓**

**┃题2.23：归并两个单链表┃**

**┗━━━━━━━━━━━\*/**

**/\* 方法一：顺序链接 \*/**

**Status Algo\_2\_23\_1(LinkList La, LinkList \*Lb, LinkList \*Lc)**

**{**

**LinkList prea, pa, pb;**

**if(!La || !(\*Lb) || (!La->next && !(\*Lb)->next))//La或Lb有一个不存在或两个均为空表时，合并错误**

**return ERROR;**

**\*Lc = La;** **//利用A的头结点作C的头结点**

**prea = La;**

**pa = La->next;**

**pb = (\*Lb)->next;**

**while(pa && pb)**

**{**

**(\*Lb)->next = pb->next;**

**prea = pa;**

**pa = pa->next;**

**prea->next = pb;**

**pb->next = pa;**

**prea = pb;**

**pb = (\*Lb)->next;**

**}**

**if(!pa)** **//Lb还有剩余**

**prea->next = pb;**

**free((\*Lb));**

**\*Lb = NULL;**

**return OK;**

**}**

**/\*━━━━━━━━━━━┓**

**┃题2.23：归并两个单链表┃**

**┗━━━━━━━━━━━\*/**

**/\* 方法二：交替链接 \*/**

**Status Algo\_2\_23\_2(LinkList La, LinkList \*Lb, LinkList \*Lc)**

**{**

**LinkList cur, pa, pb;**

**int i = 0;**

**if(!La || !(\*Lb) || (!La->next && !(\*Lb)->next))//La或Lb有一个不存在或两个均为空表时，合并错误**

**return ERROR;**

**\*Lc = La;** **//利用A的头结点作C的头结点**

**cur = (\*Lc);**

**pa = La->next;**

**pb = (\*Lb)->next;**

**while(pa && pb)**

**{**

**i++;**

**if(i%2)**

**{**

**cur->next = pa;**

**cur = pa;**

**pa = pa->next;**

**}**

**else**

**{**

**cur->next = pb;**

**cur = pb;**

**pb = pb->next;**

**}**

**}**

**if(!pa)** **//La先扫描完**

**cur->next = pb;**

**if(!pb)** **//Lb先扫描完，注意与方法一的区别**

**cur->next = pa;**

**free((\*Lb));**

**\*Lb = NULL;**

**return OK;**

**}**

**void PrintElem(LElemType\_L e)**

**{**

**printf("%d ", e);**

**}**

**2.24❹ 假设有两个按元素值递增有序排列的线性表A和B，均以单链表作存储结构，请编写算法将A表和B表归并成一个按元素值递减有序（即非递增有序，允许表中含有值相同的元素）排列的线性表C，并要求利用原表（即A表和B表）的结点空间构造C表。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_24(LinkList La, LinkList \*Lb, LinkList \*Lc);**

**void PrintElem(LElemType\_L e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**LinkList La, Lb, Lc;**

**int i, mark;**

**if(InitList\_L(&La) && InitList\_L(&Lb) && InitList\_L(&Lc))** **//链表L创建成功**

**{**

**for(i=1; i<=5; i++)** **//创建链表La和Lb**

**{**

**ListInsert\_L(La, i, 2\*i-1);**

**ListInsert\_L(Lb, i, 2\*i);**

**}**

**}**

**printf("创建好的链表为：\n");**

**printf("La = ");**

**ListTraverse\_L(La, PrintElem);**

**printf("\n");**

**printf("Lb = ");**

**ListTraverse\_L(Lb, PrintElem);** **//输出链表**

**printf("\n\n");**

**Algo\_2\_24(La, &Lb, &Lc);**

**printf("归并La和Lb为Lc = ");**

**ListTraverse\_L(Lc, PrintElem);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━━━━━━━━┓**

**┃题2.24：归并两个递增单链表为一个递减单链表┃**

**┗━━━━━━━━━━━━━━━━━━━━━\*/**

**Status Algo\_2\_24(LinkList La, LinkList \*Lb, LinkList \*Lc)**

**{**

**LinkList pa, pb, s;**

**if(!La || !Lb || (!La->next && !(\*Lb)->next))** **//La或Lb有一个不存在或两个均为空表时，合并错误**

**return ERROR;**

**\*Lc = La;** **//利用A的头结点作C的头结点**

**pa = La->next;**

**pb = (\*Lb)->next;**

**La->next = NULL;**

**while(pa && pb)** **//采用头插法建立递减链表**

**{**

**if(pa->data<=pb->data)**

**{**

**s = pa->next;**

**pa->next = La->next;**

**La->next = pa;**

**pa = s;**

**}**

**else**

**{**

**s = pb->next;**

**pb->next = La->next;**

**La->next = pb;**

**pb = s;**

**}**

**}**

**while(pa)**

**{**

**s = pa->next;**

**pa->next = La->next;**

**La->next = pa;**

**pa = s;**

**}**

**while(pb)**

**{**

**s = pb->next;**

**pb->next = La->next;**

**La->next = pb;**

**pb = s;**

**}**

**free(\*Lb);**

**return OK;**

**}**

**void PrintElem(LElemType\_L e)**

**{**

**printf("%d ", e);**

**}**

**┏━━━━┓**

**┣ 顺序表 ┫**

**┗━━━━┛**

**2.25❹ 假设以两个元素依值递增有序排列的线性表A和B分别表示两个集合（即同一表中的元素值各不相同），现要求另辟空间构成一个线性表C，其元素为A和B中元素的交集，且表C中的元素也依值递增有序排列。试对顺序表编写求C的算法。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲02 线性表/01 SequenceList/SequenceList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**void Algo\_2\_25(SqList La, SqList Lb, SqList \*Lc);**

**void PrintElem(LElemType\_Sq e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**SqList La, Lb, Lc;**

**int i;**

**if(InitList\_Sq(&La) && InitList\_Sq(&Lb)) //链表L创建成功**

**{**

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

**{**

**ListInsert\_Sq(&La, i, i);**

**ListInsert\_Sq(&Lb, i, 2\*i);**

**}**

**}**

**printf("La = ");**

**ListTraverse\_Sq(La, PrintElem);**

**printf("\n");**

**printf("Lb = ");**

**ListTraverse\_Sq(Lb, PrintElem);**

**printf("\n\n");**

**InitList\_Sq(&Lc);**

**Algo\_2\_25(La, Lb, &Lc);**

**printf("Lc = La∩Lb = ");**

**ListTraverse\_Sq(Lc, PrintElem);** **//输出L**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━┓**

**┃题2.25：C=A∩B┃**

**┗━━━━━━━\*/**

**void Algo\_2\_25(SqList La, SqList Lb, SqList \*Lc)**

**{**

**int i, j, k;**

**i = j = 0;**

**k = 1;**

**while(i<La.length && j<Lb.length)**

**{**

**if(La.elem[i]<Lb.elem[j])**

**i++;**

**else if(La.elem[i]>Lb.elem[j])**

**j++;**

**else**

**{**

**ListInsert\_Sq(Lc, k, La.elem[i]);**

**k++;**

**i++;**

**j++;**

**}**

**}**

**}**

**void PrintElem(LElemType\_Sq e)**

**{**

**printf("%d ", e);**

**}**

**┏━━━━┓**

**┣ 单链表 ┫**

**┗━━━━┛**

**2.26❹ 要求同2.25题。试对单链表编写求C的算法。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_26(LinkList La, LinkList Lb, LinkList Lc);**

**void PrintElem(LElemType\_L e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**LinkList La, Lb, Lc;**

**int i;**

**if(InitList\_L(&La) && InitList\_L(&Lb) && InitList\_L(&Lc))** **//链表创建成功**

**{**

**for(i=1; i<=10; i++)** **//链表赋值**

**{**

**ListInsert\_L(La, i, i);**

**ListInsert\_L(Lb, i, 2\*i);**

**}**

**}**

**printf("La = ");**

**ListTraverse\_L(La, PrintElem);** **//输出**

**printf("\n");**

**printf("Lb = ");**

**ListTraverse\_L(Lb, PrintElem);**

**printf("\n\n");**

**Algo\_2\_26(La, Lb, Lc);**

**printf("Lc=La∩Lb= ");**

**ListTraverse\_L(Lc, PrintElem);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━┓**

**┃题2.26：C=A∩B┃**

**┗━━━━━━━\*/**

**Status Algo\_2\_26(LinkList La, LinkList Lb, LinkList Lc)**

**{**

**LinkList pa, pb, pc, s;**

**if(!La || !Lb)**

**return ERROR;**

**pa = La->next;**

**pb = Lb->next;**

**pc = Lc;**

**while(pa && pb)**

**{**

**if(pa->data==pb->data)**

**{**

**s = (LinkList)malloc(sizeof(LNode));**

**if(!s)**

**exit(OVERFLOW);**

**s->data = pa->data;**

**s->next = NULL;**

**pc->next = s;**

**pc = pc->next;**

**pa = pa->next;**

**pb = pb->next;**

**}**

**else if(pa->data<pb->data)**

**pa = pa->next;**

**else**

**pb = pb->next;**

**}**

**return OK;**

**}**

**void PrintElem(LElemType\_L e)**

**{**

**printf("%d ", e);**

**}**

**┏━━━━┓**

**┣ 顺序表 ┫**

**┗━━━━┛**

**2.27❹ 对2.25题的条件作以下修改，对顺序表重新编写求得表C的算法。**

**（1）假设在同一表（A或B）中可能存在值相同的元素，但要求新生成的表C中的元素值各不相同；**

**（2）利用A表空间存放表C。**

**（1）注意当A、B中元素相等时，要判断C中是否已经出现过这个相同元素。**

**（2）注意C利用A的空间时，A长度可能要改变。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲02 线性表/01 SequenceList/SequenceList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**void Algo\_2\_27\_\_1(SqList La, SqList Lb, SqList \*Lc);**

**void Algo\_2\_27\_\_2(SqList \*La, SqList Lb);**

**void PrintElem(LElemType\_Sq e);** **//测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**SqList La, Lb, Lc\_1, \*Lc\_2;**

**int i;**

**int a[10] = {1,2,2,3,4,5,6,7,7,8};**

**int b[10] = {2,2,3,4,4,6,7,8,8,9};**

**if(InitList\_Sq(&La) && InitList\_Sq(&Lb))** **//链表L创建成功**

**{**

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

**{**

**ListInsert\_Sq(&La, i, a[i-1]);**

**ListInsert\_Sq(&Lb, i, b[i-1]);**

**}**

**}**

**printf("La = ");**

**ListTraverse\_Sq(La, PrintElem);**

**printf("\n");**

**printf("Lb = ");**

**ListTraverse\_Sq(Lb, PrintElem);**

**printf("\n\n");**

**printf("题 2.27 第(1)问验证：\n");**

**InitList\_Sq(&Lc\_1);**

**Algo\_2\_27\_\_1(La, Lb, &Lc\_1);**

**printf("Lc = La∩Lb = ");**

**ListTraverse\_Sq(Lc\_1, PrintElem);** **//输出L**

**printf("\n\n");**

**printf("题 2.27 第(2)问验证：\n");**

**Algo\_2\_27\_\_2(&La, Lb);**

**Lc\_2 = &La;**

**printf("Lc = La∩Lb = ");**

**ListTraverse\_Sq(\*Lc\_2, PrintElem);** **//输出L**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━┓**

**┃题2.27：C=A∩B┃**

**┗━━━━━━━\*/**

**/\* (1) \*/**

**void Algo\_2\_27\_\_1(SqList La, SqList Lb, SqList \*Lc)**

**{**

**int i, j, k;**

**i = j = 0;**

**k = 1;**

**while(i<La.length && j<Lb.length)**

**{**

**if(La.elem[i]<Lb.elem[j])**

**i++;**

**else if(La.elem[i]>Lb.elem[j])**

**j++;**

**else**

**{**

**if(!i || La.elem[i]!=La.elem[i-1])**

**{**

**ListInsert\_Sq(Lc, k, La.elem[i]);**

**k++;**

**}**

**i++;**

**j++;**

**}**

**}**

**}**

**/\*━━━━━━━┓**

**┃题2.27：C=A∩B┃**

**┗━━━━━━━\*/**

**/\* (2) \*/**

**void Algo\_2\_27\_\_2(SqList \*La, SqList Lb)**

**{**

**int i, j, k;**

**int len\_a;**

**i = j = k = 0;**

**len\_a = 0;**

**while(i<(\*La).length && j<Lb.length)**

**{**

**if((\*La).elem[i]<Lb.elem[j])**

**i++;**

**else if((\*La).elem[i]>Lb.elem[j])**

**j++;**

**else**

**{**

**if(!i || (\*La).elem[i]!=(\*La).elem[i-1])**

**{**

**(\*La).elem[k] = (\*La).elem[i];**

**len\_a++;**

**k++;**

**}**

**i++;**

**j++;**

**}**

**}**

**(\*La).length = len\_a;**

**}**

**void PrintElem(LElemType\_Sq e)**

**{**

**printf("%d ", e);**

**}**

**┏━━━━┓**

**┣ 单链表 ┫**

**┗━━━━┛**

**2.28❹ 对2.25题的条件作以下两点修改，对单链表重新编写求得表C的算法。**

**（1）假设在同一表（A或B）中可能存在值相同的元素，但要求新生成的表C中的元素值各不相同。**

**（2）利用原表（A表或B表）中的结点构造表C，并释放A表中的无用结点空间。**

**（1）注意事项同上题。**

**（2）注意事项同上题。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_28\_\_1(LinkList La, LinkList Lb, LinkList Lc);**

**Status Algo\_2\_28\_\_2(LinkList La, LinkList Lb);**

**void PrintElem(LElemType\_L e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**LinkList La, Lb, Lc\_1, \*Lc\_2;**

**int i;**

**int a[10] = {1,2,2,3,4,5,6,7,7,8};**

**int b[10] = {2,2,3,4,4,6,7,8,8,9};**

**if(InitList\_L(&La) && InitList\_L(&Lb))**  **//链表创建成功**

**{**

**for(i=1; i<=10; i++)** **//链表赋值**

**{**

**ListInsert\_L(La, i, a[i-1]);**

**ListInsert\_L(Lb, i, b[i-1]);**

**}**

**}**

**printf("La = ");**

**ListTraverse\_L(La, PrintElem);** **//输出**

**printf("\n");**

**printf("Lb = ");**

**ListTraverse\_L(Lb, PrintElem);**

**printf("\n\n");**

**printf("题 2.28 第(1)问验证：\n");**

**InitList\_L(&Lc\_1);**

**Algo\_2\_28\_\_1(La, Lb, Lc\_1);**

**printf("Lc = La∩Lb = ");**

**ListTraverse\_L(Lc\_1, PrintElem);** **//输出L**

**printf("\n\n");**

**printf("题 2.28 第(2)问验证：\n");**

**Algo\_2\_28\_\_2(La, Lb);**

**Lc\_2 = &La;**

**printf("Lc = La∩Lb = ");**

**ListTraverse\_L(\*Lc\_2, PrintElem);** **//输出L**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━┓**

**┃题2.28：C=A∩B┃**

**┗━━━━━━━\*/**

**/\* (1) \*/**

**Status Algo\_2\_28\_\_1(LinkList La, LinkList Lb, LinkList Lc)**

**{**

**LinkList pa, pb, pc, s;**

**if(!La || !Lb)**

**return ERROR;**

**pa = La->next;**

**pb = Lb->next;**

**pc = Lc;**

**while(pa && pb)**

**{**

**if(pa->data==pb->data)**

**{**

**if(pc==Lc || pc->data!=pa->data)**

**{**

**s = (LinkList)malloc(sizeof(LNode));**

**if(!s)**

**exit(OVERFLOW);**

**s->data = pa->data;**

**s->next = NULL;**

**pc->next = s;**

**pc = pc->next;**

**}**

**pa = pa->next;**

**pb = pb->next;**

**}**

**else if(pa->data<pb->data)**

**pa = pa->next;**

**else**

**pb = pb->next;**

**}**

**return OK;**

**}**

**/\*━━━━━━━┓**

**┃题2.28：C=A∩B┃**

**┗━━━━━━━\*/**

**/\* (2) \*/**

**Status Algo\_2\_28\_\_2(LinkList La, LinkList Lb)**

**{**

**LinkList pa, pb, p;**

**if(!La || !Lb)**

**return ERROR;**

**pa = La->next;**

**pb = Lb->next;**

**p = La;**

**while(pa && pb)**

**{**

**if(pa->data==pb->data)**

**{**

**if(p==La || p->data!=pa->data)**

**{**

**p->next = pa;**

**p = pa;**

**pa = pa->next;**

**}**

**else**

**{**

**p->next = pa->next;**

**free(pa);**

**pa = p->next;**

**}**

**pb = pb->next;**

**}**

**else if(pa->data<pb->data)**

**{**

**p->next = pa->next;**

**free(pa);**

**pa = p->next;**

**}**

**else**

**pb = pb->next;**

**}**

**return OK;**

**}**

**void PrintElem(LElemType\_L e)**

**{**

**printf("%d ", e);**

**}**

**┏━━━━┓**

**┣ 顺序表 ┫**

**┗━━━━┛**

**2.29❺ 已知A，B和C为三个递增有序的线性表，现要求对A表作如下操作：删去那些既在B表中出现，又在C表中出现的元素。试对顺序表编写实现上述操作的算法，并分析你的算法的时间复杂度（注意：同一表中各元素值可能相同）。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲02 线性表/01 SequenceList/SequenceList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**void Algo\_2\_29(SqList \*La, SqList Lb, SqList Lc);**

**void PrintElem(LElemType\_Sq e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**SqList La, Lb, Lc;**

**int i;**

**int a[] = {1,2,3,4,5,6,7,8,9,10};**

**int b[] = {1,2,2,3,5,7,8,8,9,11};**

**int c[] = {2,3,3,4,5,6,7,8,11,12};**

**if(InitList\_Sq(&La) && InitList\_Sq(&Lb) && InitList\_Sq(&Lc)) //链表L创建成功**

**{**

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

**{**

**ListInsert\_Sq(&La, i, a[i-1]);**

**ListInsert\_Sq(&Lb, i, b[i-1]);**

**ListInsert\_Sq(&Lc, i, c[i-1]);**

**}**

**}**

**printf("La = ");**

**ListTraverse\_Sq(La, PrintElem);**

**printf("\n");**

**printf("Lb = ");**

**ListTraverse\_Sq(Lb, PrintElem);**

**printf("\n");**

**printf("Lc = ");**

**ListTraverse\_Sq(Lc, PrintElem);**

**printf("\n\n");**

**Algo\_2\_29(&La, Lb, Lc);**

**printf("La = La-Lb∩Lc = ");**

**ListTraverse\_Sq(La, PrintElem);** **//输出L**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━┓**

**┃题2.29：A=A-B∩C┃**

**┗━━━━━━━━\*/**

**void Algo\_2\_29(SqList \*La, SqList Lb, SqList Lc)**

**{**

**int i, j, k, count;**

**i = j = k =0;**

**count = 0;**

**while(i<(\*La).length && j<Lb.length && k<Lc.length)**

**{**

**if(Lb.elem[j]<Lc.elem[k])**

**j++;**

**else if(Lb.elem[j]>Lc.elem[k])**

**k++;**

**else**

**{**

**while(i<(\*La).length && (\*La).elem[i]<Lb.elem[j])**

**{**

**(\*La).elem[count] = (\*La).elem[i];**

**count++;**

**i++;**

**}**

**while(i<(\*La).length && (\*La).elem[i]==Lb.elem[j])**

**i++;**

**if(i<(\*La).length)**

**{**

**j++;**

**k++;**

**}**

**}**

**}**

**while(i<(\*La).length)**

**{**

**(\*La).elem[count] = (\*La).elem[i];**

**count++;**

**i++;**

**}**

**(\*La).length = count;**

**}**

**void PrintElem(LElemType\_Sq e)**

**{**

**printf("%d ", e);**

**}**

**┏━━━━┓**

**┣ 单链表 ┫**

**┗━━━━┛**

**2.30❺ 要求同2.29题。试对单链表编写算法，请释放A表中的无用结点空间。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_30(LinkList La, LinkList Lb, LinkList Lc);**

**void PrintElem(LElemType\_L e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**LinkList La, Lb, Lc;**

**int i;**

**int a[] = {1,2,3,4,5,6,7,8,9,10};**

**int b[] = {1,2,2,3,5,7,8,8,9,11};**

**int c[] = {2,3,3,4,5,6,7,8,11,12};**

**if(InitList\_L(&La) && InitList\_L(&Lb) && InitList\_L(&Lc)) //链表L创建成功**

**{**

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

**{**

**ListInsert\_L(La, i, a[i-1]);**

**ListInsert\_L(Lb, i, b[i-1]);**

**ListInsert\_L(Lc, i, c[i-1]);**

**}**

**}**

**printf("La = ");**

**ListTraverse\_L(La, PrintElem);**

**printf("\n");**

**printf("Lb = ");**

**ListTraverse\_L(Lb, PrintElem);**

**printf("\n");**

**printf("Lc = ");**

**ListTraverse\_L(Lc, PrintElem);**

**printf("\n\n");**

**Algo\_2\_30(La, Lb, Lc);**

**printf("La = La-Lb∩Lc = ");**

**ListTraverse\_L(La, PrintElem);** **//输出L**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━┓**

**┃题2.30：A=A-B∩C┃**

**┗━━━━━━━━\*/**

**Status Algo\_2\_30(LinkList La, LinkList Lb, LinkList Lc)**

**{**

**LinkList pa, pb, pc, pre;**

**if(!La || !Lb || !Lc)**

**return ERROR;**

**pa = La->next;**

**pb = Lb->next;**

**pc = Lc->next;**

**pre = La;**

**while(pa && pb && pc)**

**{**

**if(pb->data<pc->data)**

**pb = pb->next;**

**else if(pb->data>pc->data)**

**pc = pc->next;**

**else**

**{**

**while(pa && pa->data<pb->data)**

**{**

**pre = pa;**

**pa = pa->next;**

**}**

**while(pa && pa->data==pb->data)**

**{**

**pre->next = pa->next;**

**free(pa);**

**pa = pre->next;**

**}**

**if(pa)**

**{**

**pb = pb->next;**

**pc = pc->next;**

**}**

**}**

**}**

**return OK;**

**}**

**void PrintElem(LElemType\_L e)**

**{**

**printf("%d ", e);**

**}**

**┏━━━━━━┓**

**┣ 单循环链表 ┫**

**┗━━━━━━┛**

**2.31❷ 假设某个单向循环链表的长度大于1，且表中既无头结点也无头指针。已知s为指向链表中某个结点的指针，试编写算法在链表中删除指针s所指结点的前驱结点。**

**#include <stdio.h>**

**#include <stdlib.h> //提供malloc、realloc、free、exit原型**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/04 SinglyLinkedList/SinglyLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_31(LinkList L, LinkList s);**

**void InitList\_2\_31(LinkList \*L);**

**Status CreatList\_2\_31(LinkList \*L, int n, LElemType\_L Data[]);**

**void Output\_2\_31(LinkList L);**

**int main(int argc, char \*argv[])**

**{**

**LinkList L, s;**

**int a[10] = {1,2,3,4,5,6,7,8,9,10};**

**InitList\_2\_31(&L);**

**CreatList\_2\_31(&L, 10, a);**

**printf("L = ");**

**Output\_2\_31(L);**

**printf("\n\n");**

**s = L->next->next->next->next;**

**printf("删除 %d 的前驱后...\n", s->data);**

**Algo\_2\_31(L, s);**

**printf("L = ");**

**Output\_2\_31(L);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━┓**

**┃题2.31：删除s指向结点的前驱 ┃**

**┗━━━━━━━━━━━━━━\*/**

**/\* 本题中链表为无头结点的单循环链表 \*/**

**Status Algo\_2\_31(LinkList L, LinkList s)**

**{**

**LinkList r, p;**

**if(s->next==s)**

**return ERROR;**

**r = s;**

**while(r->next->next!=s)**

**r = r->next;**

**p = r->next;**

**r->next = p->next;**

**if(p==L)**

**L = p->next;**

**free(p);**

**return OK;**

**}**

**void InitList\_2\_31(LinkList \*L)**

**{**

**\*L = NULL;**

**}**

**Status CreatList\_2\_31(LinkList \*L, int n, LElemType\_L Data[])**

**{**

**LinkList s, r;**

**int i;**

**if(\*L)** **//创建之前保证链表为空**

**return ERROR;**

**for(i=1; i<=n; i++)**

**{**

**s = (LinkList)malloc(sizeof(LNode));**

**s->data = Data[i-1];**

**if(!s)**

**exit(OVERFLOW);**

**if(i==1)**

**{**

**\*L = s;**

**s->next = \*L;**

**r = \*L;**

**}**

**else**

**{**

**s->next = r->next;**

**r->next = s;**

**r = r->next;**

**}**

**}**

**return OK;**

**}**

**void Output\_2\_31(LinkList L)**

**{**

**LinkList p;**

**if(L)**

**{**

**printf("%2d ", L->data);**

**p = L->next;**

**while(p!=L)**

**{**

**printf("%2d ", p->data);**

**p = p->next;**

**}**

**}**

**}**

**2.32❷ 已知有一个单向循环链表，其每个结点中含三个域：pre，data和next，其中data为数据域，next为指向后继结点的指针域，pre也为指针域，但它的值为空(NULL)，试编写算法将此单向循环链表改为双向循环链表，即使pre称为指向前驱结点的指针域。**

**#include <stdio.h>**

**#include <stdlib.h> //提供malloc、realloc、free、exit原型**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**/\* 类型定义 \*/**

**typedef int LElemType\_L;**

**typedef struct LNode** **//双循环链表结构体**

**{**

**LElemType\_L data;**

**struct LNode\* next;**

**struct LNode\* pre;**

**}LNode;**

**typedef LNode\* LinkList;** **//指向结构的指针**

**/\* 函数原型 \*/**

**Status Algo\_2\_32(LinkList L);**

**Status InitList\_2\_32(LinkList \*L);**

**Status CreatList\_2\_32(LinkList L, int n, LElemType\_L Data[]);**

**void Output\_2\_32(LinkList L);**

**void OutputReverse\_2\_32(LinkList L);**

**int main(int argc, char \*argv[])**

**{**

**LinkList L;**

**LElemType\_L a[10] = {1,2,3,4,5,6,7,8,9,10};**

**InitList\_2\_32(&L);**

**CreatList\_2\_32(L, 10, a);**

**printf("L = ");**

**Output\_2\_32(L);**

**printf("\n\n");**

**printf("为链表设置前驱后逆序输出链表...\n");**

**Algo\_2\_32(L);**

**printf("L = ");**

**OutputReverse\_2\_32(L);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━━━━┓**

**┃题2.32：为单循环链表各结点设置前驱┃**

**┗━━━━━━━━━━━━━━━━━\*/**

**Status Algo\_2\_32(LinkList L)**

**{**

**LinkList r;**

**if(!L)**

**return ERROR;**

**for(r=L; r->next!=L; r=r->next)**

**r->next->pre = r;**

**L->pre = r;**

**return OK;**

**}**

**Status InitList\_2\_32(LinkList \*L)**

**{**

**(\*L) = (LinkList)malloc(sizeof(LNode));**

**if(!(\*L))**

**exit(OVERFLOW);**

**(\*L)->pre = NULL;**

**(\*L)->next = \*L;**

**return OK;**

**}**

**Status CreatList\_2\_32(LinkList L, int n, LElemType\_L Data[])**

**{**

**LinkList s, r;**

**int i;**

**if(!L || L->next!=L)**

**return ERROR;**

**for(i=1,r=L; i<=n; i++)**

**{**

**s = (LinkList)malloc(sizeof(LNode));**

**if(!s)**

**exit(OVERFLOW);**

**s->data = Data[i-1];**

**s->pre = NULL;**

**s->next = r->next;**

**r->next = s;**

**r = r->next;**

**}**

**return OK;**

**}**

**void Output\_2\_32(LinkList L)**

**{**

**LinkList p;**

**if(L)**

**{**

**p = L->next;**

**while(p!=L)**

**{**

**printf("%2d ", p->data);**

**p = p->next;**

**}**

**}**

**}**

**void OutputReverse\_2\_32(LinkList L)**

**{**

**LinkList p;**

**if(L)**

**{**

**p = L->pre;**

**while(p!=L)**

**{**

**printf("%2d ", p->data);**

**p = p->pre;**

**}**

**}**

**}**

**2.33❸ 已知由一个线性链表表示的线性表中含有三类字符的数据元素（如：字母字符、数字字符和其他字符），试编写算法将该线性链表分割为三个循环链表，其中每个循环链表表示的线性表中均只含一类字符。**

**#include <stdio.h>**

**#include <stdlib.h> //提供malloc、realloc、free、exit原型**

**#include <ctype.h> //提供isalpha、isdigit原型**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**/\* 类型定义 \*/**

**typedef char LElemType\_L;**

**typedef struct LNode** **//双循环链表结构体**

**{**

**LElemType\_L data;**

**struct LNode\* next;**

**}LNode;**

**typedef LNode\* LinkList;** **//指向结构的指针**

**/\* 函数原型 \*/**

**Status Algo\_2\_33(LinkList L, LinkList L\_char, LinkList L\_num, LinkList L\_other);**

**Status InitList\_2\_33(LinkList \*L); //单链表初始化**

**Status CreatList\_2\_33(LinkList L, int n, LElemType\_L Data[]);**

**Status InitList\_CL\_2\_33(LinkList \*L); //单循环链表初始化**

**void Output\_2\_33(LinkList L); //输出单链表**

**void Output\_CL\_2\_33(LinkList L); //输出单循环链表**

**int main(int argc, char \*argv[])**

**{**

**LinkList L, L\_char, L\_num, L\_other;**

**LElemType\_L \*a = "2a3@A5e$T\*";**

**InitList\_2\_33(&L);**

**CreatList\_2\_33(L, 10, a);**

**printf("L = ");**

**Output\_2\_33(L);**

**printf("\n\n");**

**printf("拆分单链表之后...\n");**

**InitList\_CL\_2\_33(&L\_char);**

**InitList\_CL\_2\_33(&L\_num);**

**InitList\_CL\_2\_33(&L\_other);**

**Algo\_2\_33(L, L\_char, L\_num, L\_other);**

**printf("L\_char = ");**

**Output\_CL\_2\_33(L\_char);**

**printf("\n");**

**printf("L\_num = ");**

**Output\_CL\_2\_33(L\_num);**

**printf("\n");**

**printf("L\_other = ");**

**Output\_CL\_2\_33(L\_other);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓**

**┃题2.33：按字符类型（字母、数字、其他）拆分线性表为循环链表┃**

**┗━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\*/**

**Status Algo\_2\_33(LinkList L, LinkList L\_char, LinkList L\_num, LinkList L\_other)**

**{**

**LinkList r, rc, rn, ro, s;**

**if(!L)**

**return ERROR;**

**r = L->next;**

**rc = L\_char;**

**rn = L\_num;**

**ro = L\_other;**

**while(r)**

**{**

**s = (LinkList)malloc(sizeof(LNode));**

**if(!s)**

**exit(OVERFLOW);**

**s->data = r->data;**

**if(isalpha(r->data))** **//结点值为字符**

**{**

**s->next = rc->next;**

**rc->next = s;**

**rc = rc->next;**

**}**

**else if(isdigit(r->data))** **//结点值为数字**

**{**

**s->next = rn->next;**

**rn->next = s;**

**rn = rn->next;**

**}**

**else** **//其它**

**{**

**s->next = ro->next;**

**ro->next = s;**

**ro = ro->next;**

**}**

**r = r->next;**

**}**

**return OK;**

**}**

**Status InitList\_2\_33(LinkList \*L)** **//单链表初始化**

**{**

**\*L = (LinkList)malloc(sizeof(LNode));**

**if(!(\*L))**

**exit(OVERFLOW);**

**(\*L)->next = NULL;**

**return OK;**

**}**

**Status CreatList\_2\_33(LinkList L, int n, LElemType\_L Data[])**

**{**

**LinkList s, r;**

**int i;**

**if(!L || L->next)**

**return ERROR;**

**for(i=1,r=L; i<=n; i++)**

**{**

**s = (LinkList)malloc(sizeof(LNode));**

**if(!s)**

**exit(OVERFLOW);**

**s->data = Data[i-1];**

**s->next = r->next;**

**r->next = s;**

**r = r->next;**

**}**

**return OK;**

**}**

**Status InitList\_CL\_2\_33(LinkList \*L)** **//单循环链表初始化**

**{**

**\*L = (LinkList)malloc(sizeof(LNode));**

**if(!(\*L))**

**exit(OVERFLOW);**

**(\*L)->next = \*L;**

**return OK;**

**}**

**void Output\_2\_33(LinkList L)** **//输出单链表**

**{**

**LinkList p;**

**if(L)**

**{**

**p = L->next;**

**while(p)**

**{**

**printf("%2c ", p->data);**

**p = p->next;**

**}**

**}**

**}**

**void Output\_CL\_2\_33(LinkList L)** **//输出单循环链表**

**{**

**LinkList p;**

**if(L)**

**{**

**p = L->next;**

**while(p!=L)**

**{**

**printf("%2c ", p->data);**

**p = p->next;**

**}**

**}**

**}**

**在2.34至2.36题中，“异或指针双向链表”类型XorLinkedList和指针异或函数XorP定义为：**

**typedef struct XorNode**

**{**

**char data;**

**struct XorNode LRPtr;**

**}XorNode, \*XorPointer;**

**typedef struct**

**{ //无头结点的异或指针双向链表**

**XorPointer Left, Right; //分别指向链表的左端和右端**

**}XorLinkedList;**

**XorPointer XorP(XorPointer p, XorPointer q);**

**//指针异或函数XorP返回指针p和q的异或(XOR)值**

**异 或 表**



**异或指针链表的动态创建过程**













**┏━━━━━━━━━━━━━━━┓**

**┣ 扩展的双链表（异或指针链表） ┫**

**┗━━━━━━━━━━━━━━━┛**

**2.34❹ 假设在算法描述语言中引入指针的二元运算“异或”（用“⊕”表示），若a和b为指针，则a⊕b的运算结果仍为原指针类型，且**

**a****⊕(a⊕b)=(a⊕a)****⊕b=b**

**(a⊕b)⊕b=a⊕(b⊕b)=a**

**则可利用一个指针域来实现双向链表L。链表L中的每个结点只含两个域：data域和LRPtr域，其中LRPtr域存放该结点的左邻与右邻结点指针（不存在时为NULL）的异或。若设指针L.Left指向链表中的最左结点，L.Right指向链表中的最右结点，则可实现从左向右或从右向左遍历此双向链表的操作。试写一算法按任一方向依次输出链表中各元素的值。**

**2.35❹ 采用2.34题所述的存储结构，写出在第i个结点之前插入一个结点的算法。**

**2.36❹ 采用2.34题所述的存储结构，写出删除第i个结点的算法。**

**#include <stdio.h>**

**#include <stdlib.h> //提供malloc、realloc、free、exit原型**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**/\* 类型定义 \*/**

**typedef char XElemType;**

**typedef struct XorNode** **//异或指针链表结点结构**

**{**

**XElemType data;**

**struct XorNode \*LRPtr;**

**}XorNode;**

**typedef XorNode\* XorPointer;** **//指向结点结构的指针**

**typedef struct** **//无头结点的异或指针双向链表的头尾指针**

**{**

**XorPointer Left;** **//分别指向链表的最左端（头指针）和最右端（尾指针）**

**XorPointer Right;**

**}XorLinkedList;**

**typedef XorLinkedList\* Xor;**

**/\* 异或指针链表函数原型 \*/**

**Status Algo\_2\_34\_1(Xor L, int mark);**

**Status Algo\_2\_34\_2(Xor L, int mark);**

**Status Algo\_2\_35(Xor L, int i, XElemType e);**

**Status Algo\_2\_36(Xor L, int i, char \*e);**

**XorPointer XorP\_XorL(XorPointer p, XorPointer q);**

**Status Init\_XorL(Xor \*L);**

**Status Create\_XorL(Xor L, int n, XElemType Data[]);**

**XorPointer Prior\_Next\_Ptr\_XorL(Xor L, XorPointer cur, int mark);**

**int Length\_XorL(Xor L);**

**XorPointer GetElemPtr\_XorL(Xor L, int i);**

**int main(int argc, char \*\*argv)**

**{**

**Xor L;**

**XElemType e;**

**int n = 9;**

**XElemType \*data = "123456789";**

**Init\_XorL(&L);** **//先初始化**

**Create\_XorL(L, n, data);**

**printf("███题 2.34 验证...███\n");**

**printf("顺序输出L = ");**

**Algo\_2\_34\_1(L, 0);**

**printf("\n");**

**printf("逆序输出L = ");**

**Algo\_2\_34\_1(L, 1);**

**printf("\n\n");**

**printf("███题 2.35 验证...███\n");**

**Algo\_2\_35(L, 5, '\*');**

**printf("插入 '\*' 为L的第5个结点后顺序输出...\n");**

**Algo\_2\_34\_2(L, 0);**

**printf("\n\n");**

**printf("███题 2.36 验证...███\n");**

**Algo\_2\_36(L, 5, &e);**

**printf("删除L的第5个结点 '%c' 后逆序输出...\n", e);**

**Algo\_2\_34\_2(L, 1);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━━━━━━┓**

**┃(01)指针异或函数，返回指针p和q的异或值┃**

**┗━━━━━━━━━━━━━━━━━━━\*/**

**XorPointer XorP\_XorL(XorPointer p, XorPointer q)**

**{**

**unsigned long x, y, z;**

**x = (unsigned long)p;**

**y = (unsigned long)q;**

**z = x^y;** **//求异或**

**return (XorPointer)z;**

**}**

**/\*━━━━━━━━━━━━┓**

**┃(02)初始化异或指针链表L ┃**

**┗━━━━━━━━━━━━\*/**

**Status Init\_XorL(Xor \*L)**

**{**

**\*L = (Xor)malloc(sizeof(XorLinkedList));**

**if(!(\*L))**

**exit(OVERFLOW);**

**(\*L)->Left = NULL;**

**(\*L)->Right = NULL;**

**return OK;**

**}**

**/\*━━━━━━━━━━━┓**

**┃(03)创建异或指针链表L ┃**

**┗━━━━━━━━━━━\*/**

**Status Create\_XorL(Xor L, int n, XElemType Data[])**

**{**

**int i;**

**XorPointer pre\_l, pre\_m, pre\_r;**

**pre\_l = pre\_m = NULL;**

**for(i=1; i<=n; i++)**

**{**

**pre\_r = (XorPointer) malloc (sizeof(XorNode));**

**if(!pre\_r)**

**exit(OVERFLOW);**

**pre\_r->data = Data[i-1];**

**if(i==1)** **//左指针（创建首结点时设立）**

**L->Left = pre\_r;**

**if(pre\_m)** **//pre\_m不为空时执行,从创立第二个结点时执行**

**pre\_m->LRPtr = XorP\_XorL(pre\_l, pre\_r);**

**L->Right = pre\_r;** **//右指针**

**pre\_r->LRPtr = XorP\_XorL(pre\_m, NULL);**

**pre\_l = pre\_m;**

**pre\_m = pre\_r;**

**}**

**return OK;**

**}**

**/\*━━━━━━━━━━━━━━━━━━━━━━━━━┓**

**┃(04)获取当前结点(cur)的前驱(mark=0)或后继(mark=1) ┃**

**┗━━━━━━━━━━━━━━━━━━━━━━━━━\*/**

**XorPointer Prior\_Next\_Ptr\_XorL(Xor L, XorPointer cur, int mark)**

**{**

**XorPointer p\_l, p\_m, p\_r;**

**if(mark!=1 && mark!=0)**

**return NULL;**

**if(!L || !L->Left || !cur)** **//链表不存在或链表为空或cur为空则无法求地址**

**return NULL;**

**if(mark==1)** **//求后继，从左向右**

**{**

**p\_l = NULL;**

**p\_m = L->Left;**

**while(p\_m!=cur)**

**{**

**p\_r = XorP\_XorL(p\_l, p\_m->LRPtr);** **//计算右侧第一个结点地址**

**p\_l = p\_m;** **//指针不断向右推进**

**p\_m = p\_r;**

**}**

**p\_r = XorP\_XorL(p\_l, p\_m->LRPtr);**

**return p\_r;**

**}**

**if(mark==0)** **//求前驱，从右向左**

**{**

**p\_m = L->Right;**

**p\_r = NULL;**

**while(p\_m!=cur)**

**{**

**p\_l = XorP\_XorL(p\_m->LRPtr, p\_r);** **//计算左侧第一个结点地址**

**p\_r = p\_m;** **//指针不断向左推进**

**p\_m = p\_l;**

**}**

**p\_l = XorP\_XorL(p\_m->LRPtr, p\_r);**

**return p\_l;**

**}**

**}**

**/\*━━━━━━━┓**

**┃(05)获取L长度 ┃**

**┗━━━━━━━\*/**

**int Length\_XorL(Xor L)**

**{**

**int count;**

**XorPointer p, pre, psuc;**

**if(!L)**

**exit(OVERFLOW);** **//链表不存在则退出**

**count = 0;**

**pre = NULL;**

**p = L->Left;** **//从左向右遍历**

**while(p)**

**{**

**count++;**

**psuc = XorP\_XorL(pre, p->LRPtr);**

**pre = p;**

**p = psuc;**

**}**

**return count;**

**}**

**/\*━━━━━━━━━━━━━━┓**

**┃(06)返回指向第i个结点的指针 ┃**

**┗━━━━━━━━━━━━━━\*/**

**XorPointer GetElemPtr\_XorL(Xor L, int i)** **//i为元素位序**

**{**

**int count;**

**XorPointer p, pre, psuc;**

**if(!L || !L->Left)** **//链表不存在或链表为空无法求地址**

**return ERROR;**

**if(i<1)**

**return ERROR;**

**count = 1;**

**pre = NULL;**

**p = L->Left;**

**while(count<i && p)**

**{**

**psuc = XorP\_XorL(pre, p->LRPtr);**

**pre = p;**

**p = psuc;**

**count++;**

**}**

**return p;**

**}**

**/\*━━━━━━━━━━━━━━━┓**

**┃(07-1)输出L，mark表示输出方向 ┃**

**┗━━━━━━━━━━━━━━━\*/**

**/\* 方法1：借助Prior\_Next\_Ptr\_XorL函数遍历链表 \*/**

**Status Algo\_2\_34\_1(Xor L, int mark)**

**{**

**XorPointer cur;**

**if(mark!=1 && mark!=0)**

**return ERROR;**

**if(!L || !L->Left)** **//链表不存在或链表为空则无法输出**

**return ERROR;**

**if(mark==0)**

**{**

**cur = L->Left;**

**do**

**{**

**printf("%c ",cur->data);**

**cur = Prior\_Next\_Ptr\_XorL(L, cur, !mark);**

**}while(cur);**

**}**

**if(mark==1)**

**{**

**cur = L->Right;**

**do**

**{**

**printf("%c ",cur->data);**

**cur = Prior\_Next\_Ptr\_XorL(L, cur, !mark);**

**}while(cur);**

**}**

**return OK;**

**}**

**/\*━━━━━━━━━━━━━━━┓**

**┃(07-2)输出L，mark表示输出方向 ┃**

**┗━━━━━━━━━━━━━━━\*/**

**/\* 方法2：不借助前驱\_后继函数遍历链表 \*/**

**Status Algo\_2\_34\_2(Xor L, int mark)** **//i等于0代表从左向右输出，1代表从右向左输出**

**{**

**XorPointer p\_l, p\_m, p\_r;**

**if(mark!=1 && mark!=0)**

**return ERROR;**

**if(!L && !L->Left)** **//链表不存在或链表为空则无法输出**

**return ERROR;**

**if(mark==0)** **//从左向右**

**{**

**p\_l = NULL;**

**p\_m = L->Left;**

**while(p\_m)** **//p\_m为空时说明已输出完毕**

**{**

**printf("%c ",p\_m->data);**

**p\_r = XorP\_XorL(p\_l, p\_m->LRPtr); //计算右侧第一个结点地址**

**p\_l = p\_m;** **//指针不断推进**

**p\_m = p\_r;**

**}**

**}**

**if(mark==1)**

**{**

**p\_m = L->Right;**

**p\_r = NULL;**

**while(p\_m)**

**{**

**printf("%c ",p\_m->data);**

**p\_l = XorP\_XorL(p\_m->LRPtr, p\_r);**

**p\_r = p\_m;**

**p\_m = p\_l;**

**}**

**}**

**return OK;**

**}**

**/\*━━━━━━━━━━━━━━━┓**

**┃(08)将元素e插入到第i个结点之前┃**

**┗━━━━━━━━━━━━━━━\*/**

**Status Algo\_2\_35(Xor L, int i, XElemType e)**

**{**

**XorPointer s, i\_pre\_pre, i\_pre, i\_cur, i\_suc;**

**if(!L)** **//L必须存在**

**return ERROR;**

**if(i<1 || i>Length\_XorL(L)+1)** **//i限制在正确的范围内**

**return ERROR;**

**s = (XorPointer)malloc(sizeof(XorNode));**

**if(!s)**

**return OVERFLOW;**

**s->data = e;**

**i\_cur = GetElemPtr\_XorL(L, i);** **//指向第i个结点**

**i\_pre = GetElemPtr\_XorL(L, i-1);** **//指向第i-1个结点**

**i\_pre\_pre = GetElemPtr\_XorL(L, i-2);** **//指向第i-2个结点**

**if(!i\_cur)** **//第i个结点不存在**

**{**

**if(!i\_pre)** **//链表为空，插入为首结点**

**{**

**s->LRPtr = NULL;**

**L->Left =s;**

**}**

**else** **//链表不空，插入为尾结点**

**{**

**s->LRPtr = i\_pre;**

**i\_pre->LRPtr = XorP\_XorL(i\_pre\_pre, s);**

**}**

**L->Right = s;**

**}**

**else** **//第i个结点存在**

**{**

**i\_suc = XorP\_XorL(i\_pre, i\_cur->LRPtr);//指向第i+1个结点**

**s->LRPtr = XorP\_XorL(i\_pre, i\_cur);** **//设定s的后继指针**

**i\_cur->LRPtr = XorP\_XorL(s, i\_suc);** **//当前结点的后继指针也要改变**

**if(!i\_pre)** **//链表不空，插入为首结点**

**L->Left = s;**

**else**

**i\_pre->LRPtr = XorP\_XorL(i\_pre\_pre, s);**

**}**

**return OK;**

**}**

**/\*━━━━━━━━━━━━━━━━━━━━┓**

**┃(09)删除第i个结点，并用e接受结点中元素值┃**

**┗━━━━━━━━━━━━━━━━━━━━\*/**

**Status Algo\_2\_36(Xor L, int i, char \*e)**

**{**

**XorPointer i\_pre\_pre, i\_pre, i\_cur, i\_suc, i\_suc\_suc;**

**if(i<1 || i>Length\_XorL(L))**

**return ERROR;**

**if(!L || !L->Left)**

**return ERROR;**

**i\_cur = GetElemPtr\_XorL(L, i);** **//指向第i个结点**

**\*e = i\_cur->data;**

**i\_pre = GetElemPtr\_XorL(L, i-1);** **//指向第i-1个结点**

**i\_suc = XorP\_XorL(i\_pre, i\_cur->LRPtr);** **//指向第i+1个结点**

**if(!i\_suc)** **//删除尾结点**

**{**

**L->Right = i\_pre;**

**if(!i\_pre)** **//只有一个结点的情况**

**L->Left = NULL;**

**else**

**i\_pre->LRPtr = XorP\_XorL(i\_pre->LRPtr, i\_cur);**

**}**

**else**

**{**

**if(!i\_pre)** **//删除首结点**

**{**

**L->Left = i\_suc;**

**i\_suc->LRPtr =** **XorP\_XorL(i\_cur, i\_suc->LRPtr);**

**}**

**else**

**{**

**i\_pre\_pre = XorP\_XorL(i\_pre->LRPtr, i\_cur);** **//指向第i-2个结点**

**i\_suc\_suc = XorP\_XorL(i\_cur, i\_suc->LRPtr);** **//指向第i+2个结点**

**i\_pre->LRPtr = XorP\_XorL(i\_pre\_pre, i\_suc);**

**i\_suc->LRPtr = XorP\_XorL(i\_pre, i\_suc\_suc);**

**}**

**}**

**free(i\_cur);**

**i\_cur = NULL;**

**return OK;**

**}**

**┏━━━━━━┓**

**┣ 双循环链表 ┫**

**┗━━━━━━┛**

**2.37❹ 设以带头结点的双向循环链表表示的线性表L=(a1, a2, ..., an)，试写一时间复杂度为O(n)的算法，将L改造为L=(a1, a3, ..., an, ..., a4, a2)。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**#include "../../../▲课本算法实现/▲02 线性表/08 DualCycleLinkedList/DualCycleLinkedList.c" //\*\*▲02 线性表\*\*//**

**/\* 函数原型 \*/**

**Status Algo\_2\_37(DuLinkList L);**

**void PrintElem(LElemType\_DC e); //测试函数，打印整型**

**int main(int argc, char \*argv[])**

**{**

**DuLinkList L;**

**int i;**

**InitList\_DuL(&L);**

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

**ListInsert\_DuL(L, i, i);**

**ListTraverse\_DuL(L, PrintElem);**

**printf("\n\n");**

**Algo\_2\_37(L);**

**printf("重新排序后 L = ");**

**ListTraverse\_DuL(L, PrintElem);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━━━━━┓**

**┃题2.37：按奇偶次序重排链表各元素位序┃**

**┗━━━━━━━━━━━━━━━━━━\*/**

**Status Algo\_2\_37(DuLinkList L)**

**{**

**DuLinkList head, tail, p;**

**if(!L)**

**return ERROR;**

**head = L->next;** **//向右推进**

**tail = L->prior;** **//固定不动**

**while(head!=tail)**

**{**

**if(head->next!=tail)**

**{**

**p = head->next;** **//摘下此结点**

**p->next->prior= head;**

**head->next = p->next;**

**p->next = tail->next;**

**p->prior = tail;**

**tail->next->prior= p;**

**tail->next = p;**

**head = head->next;**

**}**

**else**

**break;**

**}**

**return OK;**

**}**

**void PrintElem(LElemType\_DC e)**

**{**

**printf("%d ", e);**

**}**

**2.38❹ 设有一个双向循环链表，每个结点中除有pre，data和next三个域外，还增设了一个访问频度域freq。在链表被起用之前，频度域freq的值均初始化为零，而每当对链表进行一次****LOCATE(L, x)的操作后，被访问的结点（即元素值等于x的结点）中的频度域freq的值便增1，同时调整链表中结点之间的次序，使其按访问频度非递增的次序顺序排列，以便始终保持被频繁访问的结点总是靠近表头结点，试编写符合上述要求的LOCATE操作的算法。**

**#include <stdio.h>**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**/\* 类型定义 \*/**

**typedef int LElemType\_DC;**

**typedef struct DuLNode** **//双向链表结构体**

**{**

**LElemType\_DC data;**

**struct DuLNode \*prior;**

**struct DuLNode \***

**next;**

**int freq;** **//访问频度域**

**}DuLNode;**

**typedef DuLNode\* DuLinkList;** **//指向双循环链表结构的指针**

**/\* 函数原型 \*/**

**int Algo\_2\_38(DuLinkList L, LElemType\_DC e);**

**Status CreatList\_DuL\_2\_38(DuLinkList \*L, int n, LElemType\_DC Data[]);**

**Status Output\_DuL\_2\_38(DuLinkList L);**

**int main(int argc, char \*argv[])**

**{**

**DuLinkList L, p;**

**int i;**

**LElemType\_DC data[10] = {1,2,3,4,5,6,7,8,9,10};**

**LElemType\_DC tmp[10] = {1,2,2,2,3,4,4,5,5,9};**

**CreatList\_DuL\_2\_38(&L, 10, data);**

**printf("初始化的链表为L = ");**

**Output\_DuL\_2\_38(L);**

**printf("\n\n");**

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

**Algo\_2\_38(L, tmp[i-1]);**

**printf("访问之后：L = ");**

**Output\_DuL\_2\_38(L);**

**printf("\n");**

**printf("对应访问次数：");**

**for(p=L->next; p!=L; p=p->next)**

**printf("%2d ", p->freq);**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓**

**┃题2.38：定位L中元素e的位序，并按访问频度重排链表假设链表中元素不重复┃**

**┗━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━\*/**

**int Algo\_2\_38(DuLinkList L, LElemType\_DC e)**

**{**

**DuLinkList p, pre;**

**int i;**

**if(!L || L->next==L)** **//链表不存在或链表为空**

**return 0;**

**i = 1;**

**pre = L;**

**p = L->next;**

**while(p!=L && p->data!=e)**

**{**

**i++;**

**pre = p;**

**p = p->next;**

**}**

**if(p!=L)** **//找到了与e相等的结点**

**{**

**p->freq++;**

**while(pre!=L && pre->freq<p->freq)**

**{**

**i--;**

**pre = pre->prior;**

**}**

**if(pre->next!=p)** **//若pre位置发生改变，p位置也需要改变**

**{**

**p->next->prior= p->prior;** **//摘下结点p**

**p->prior->next = p->next;**

**p->next = pre->next;** **//将p插入到合适位置**

**p->prior = pre;**

**pre->next->prior= p;**

**pre->next = p;**

**}**

**return i;**

**}**

**return 0;**

**}**

**Status CreatList\_DuL\_2\_38(DuLinkList \*L, int n, LElemType\_DC Data[])** **//从无到有创建**

**{**

**DuLinkList s, h;**

**int i;**

**\*L = (DuLinkList)malloc(sizeof(DuLNode));**

**if(!(\*L))**

**exit(OVERFLOW);**

**(\*L)->next = (\*L)->prior = \*L;**

**for(i=1,h=\*L; i<=n; i++)**

**{**

**s = (DuLinkList)malloc(sizeof(DuLNode));**

**if(!s)**

**exit(OVERFLOW);**

**s->data = Data[i-1];**

**s->freq = 0;**

**s->next = \*L;**

**s->prior = h;**

**h->next = s;**

**if(h==\*L)**

**h->prior = s;**

**h = h->next;**

**}**

**}**

**Status Output\_DuL\_2\_38(DuLinkList L)**

**{**

**DuLinkList p;**

**if(!L)**

**return ERROR;**

**else**

**{**

**p = L->next;**

**while(p!=L)**

**{**

**printf("%2d ", p->data);**

**p = p->next;**

**}**

**}**

**return OK;**

**}**

**在2.39至2.40题中，稀疏多项式采用的顺序存储结构SqPoly定义为**

**typedef struct**

**{**

**int coef;**

**int exp;**

**}PolyTerm;**

**typedef struct**

**{ //多项式的顺序存储结构**

**PolyTerm \*data;**

**int last;**

**}SqPoly;**

**┏━━━━┓**

**┣ 多项式 ┫**

**┗━━━━┛**

**2.39❸ 已知稀疏多项式，其中n=em>em-1>…>e1≥0，ci≠0(i=1,2,...,m),m≥1。试采用存储量同多项式项数m成正比的顺序存储结构，编写求Pn(x0)的算法(x0为给定值)，并分析你的算法的时间复杂度。**

**时间复杂度为O(n)，只与顺序表长度有关。**

**2.40❸ 采用2.39题给定的条件和存储结构，编写求的算法，将结果多项式存放在新辟的空间中，并分析你的算法的时间复杂度。**

**由于要遍历p1表和p2表，故时间复杂度为O(n1+n2)。**

**#include <stdio.h>**

**#include <math.h> //提供pow原型**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**/\* 类型定义 \*/**

**typedef struct** **//稀疏多项式顺序结构**

**{**

**int coef;** **//系数**

**int exp;** **//指数**

**}PolyTerm;**

**typedef struct**

**{** **//多项式的顺序存储结构**

**PolyTerm \*data;**

**int last;** **//多项式项数**

**}SqPoly;**

**/\* 函数原型 \*/**

**int Algo\_2\_39(SqPoly P, int x);**

**Status Algo\_2\_40(SqPoly Pa, SqPoly Pb, SqPoly \*Pc);**

**Status CreatPoly\_Sq(SqPoly \*P, int n, PolyTerm Data[]);**

**void OutputPoly\_Sq(SqPoly P);**

**int main(int argc, char \*argv[])**

**{**

**SqPoly Pa, Pb, Pc;**

**PolyTerm a[5] = {{-2,1},{4,3},{11,5},{-7,7},{8,9}};**

**PolyTerm b[5] = {{3,2},{-5,3},{12,4},{-7,7},{6,11}};**

**CreatPoly\_Sq(&Pa, 5, a);**

**CreatPoly\_Sq(&Pb, 5, b);**

**printf("Pan(x) = ");**

**OutputPoly\_Sq(Pa);**

**printf("\n");**

**printf("Pbn(x) = ");**

**OutputPoly\_Sq(Pb);**

**printf("\n\n");**

**printf("███题 2.39 验证...███\n");**

**printf("计算多项式Pa的值 Pa5(2) = %d", Algo\_2\_39(Pa, 2));**

**printf("\n");**

**printf("计算多项式Pb的值 Pb5(2) = %d", Algo\_2\_39(Pb, 2));**

**printf("\n\n");**

**printf("███题 2.40 验证...███\n");**

**Algo\_2\_40(Pa, Pb, &Pc);**

**printf("Pcn(x) = ");**

**OutputPoly\_Sq(Pc);**

**printf("\n");**

**printf("计算多项式Pc的值 Pc5(2) = %d", Algo\_2\_39(Pc, 2));**

**printf("\n\n");**

**return 0;**

**}**

**/\*━━━━━━━━━━┓**

**┃题2.39：求多项式的值┃**

**┗━━━━━━━━━━\*/**

**int Algo\_2\_39(SqPoly P, int x)**

**{**

**int i, sum;**

**for(i=0,sum=0; i<P.last; i++)**

**sum += P.data[i].coef \* (int)(pow(x, P.data[i].exp));**

**return sum;**

**}**

**/\*━━━━━━━━┓**

**┃题2.40：Pc=Pa-Pb┃**

**┗━━━━━━━━\*/**

**Status Algo\_2\_40(SqPoly Pa, SqPoly Pb, SqPoly \*Pc)**

**{**

**int i, j, k, sum;**

**(\*Pc).data = (PolyTerm \*)malloc((Pa.last+Pb.last)\*sizeof(PolyTerm));**

**if(!(\*Pc).data)**

**exit(OVERFLOW);**

**i = j = k =0;**

**while(i<Pa.last && j<Pb.last)**

**{**

**if(Pa.data[i].exp<Pb.data[j].exp)**

**{**

**(\*Pc).data[k++] = Pa.data[i];**

**i++;**

**}**

**else if(Pa.data[i].exp>Pb.data[j].exp)**

**{**

**(\*Pc).data[k].coef = -Pb.data[j].coef;** **//符号改变**

**(\*Pc).data[k].exp = Pb.data[j].exp;**

**k++;**

**j++;**

**}**

**else**

**{**

**sum = Pa.data[i].coef - Pb.data[j].coef;**

**if(sum)**

**{**

**(\*Pc).data[k].coef = sum;**

**(\*Pc).data[k].exp = Pa.data[i].exp;**

**k++;**

**}**

**i++;**

**j++;**

**}**

**}**

**while(i<Pa.last)** **//Pa未扫描完**

**{**

**(\*Pc).data[k++] = Pa.data[i];**

**i++;**

**}**

**while(j<Pb.last)** **//Pb未扫描完**

**{**

**(\*Pc).data[k].coef = -Pb.data[j].coef;** **//符号改变**

**(\*Pc).data[k].exp = Pb.data[j].exp;**

**k++;**

**j++;**

**}**

**(\*Pc).last = k;**

**return OK;**

**}**

**Status CreatPoly\_Sq(SqPoly \*P, int n, PolyTerm Data[])**

**{**

**int i;**

**(\*P).last = n;**

**(\*P).data = (PolyTerm \*)malloc(n\*sizeof(PolyTerm));**

**if(!(\*P).data)**

**exit(OVERFLOW);**

**for(i=0; i<n; i++)**

**(\*P).data[i] = Data[i];**

**return OK;**

**}**

**void OutputPoly\_Sq(SqPoly P)**

**{**

**int i;**

**for(i=0; i<P.last; i++)**

**{**

**if(i==0)**

**printf("%d" , P.data[i].coef);**

**else**

**{**

**if(P.data[i].coef<0)**

**{**

**printf(" - ");**

**printf("%d", -P.data[i].coef);**

**}**

**else**

**{**

**printf(" + ");**

**printf("%d", P.data[i].coef);**

**}**

**}**

**if(P.data[i].exp)**

**{**

**printf("x");**

**if(P.data[i].exp!=1)**

**printf("^%d", P.data[i].exp);**

**}**

**}**

**}**

**在2.41至2.42题中，稀疏多项式采用的循环链表存储结构LinkedPoly定义为**

**typedef struct PolyNode**

**{**

**PolyTerm data;**

**struct PolyNode \*next;**

**}PolyNode, \*PolyLink;**

**typedef PolyLink LinkedPoly;**

**2.41❷ 试以循环链表作稀疏多项式的存储结构，编写求其导函数的算法，要求利用原多项式中的结点空间存放其导函数（多项式），同时释放所有无用（被删）结点。**

**2.42❷ 试编写算法，将一个用循环链表表示的稀疏多项式分解成两个多项式，使这两个多项式中各自仅含奇次项或偶次项，并要求利用原链表中的结点空间构成这两个链表。**

**#include <stdio.h>**

**#include <stdlib.h> //提供malloc、realloc、free、exit原型**

**#include "../../../▲课本算法实现/▲01 绪论/Status.h" //\*\*▲01 绪论\*\*//**

**/\* 类型定义 \*/**

**typedef struct** **//稀疏多项式链式结构**

**{**

**int coef;** **//系数**

**int exp;** **//指数**

**}PolyTerm;**

**typedef struct PolyNode**

**{**

**PolyTerm data;**

**struct PolyNode \*next;**

**}PolyNode;**

**typedef PolyNode \*PolyLink;**

**typedef PolyLink LinkedPoly;**

**/\* 函数原型 \*/**

**Status Algo\_2\_41(LinkedPoly P);**

**Status Algo\_2\_42(LinkedPoly \*P, LinkedPoly \*P\_Odd, LinkedPoly \*P\_Even);**

**Status CreatPoly\_L(LinkedPoly \*P, int n, PolyTerm Data[]);**

**Status OutputPoly\_L(LinkedPoly P);**

**int main(int argc, char \*argv[])**

**{**

**LinkedPoly P1, P2;**

**PolyTerm a[7] = {-2,1,4,2,2,3,11,5,-7,6,8,9,-3,10};**

**CreatPoly\_L(&P1, 7, a);**

**CreatPoly\_L(&P2, 7, a);**

**printf("Pn(x) = ");**

**OutputPoly\_L(P1);**

**printf("\n\n");**

**printf("███题 2.41 验证...███\n");**

**{**

**Algo\_2\_41(P1);**

**printf("求多项式的导函数之后...\n");**

**printf("Pn(x) = ");**

**OutputPoly\_L(P1);**

**printf("\n\n");**

**}**

**printf("███题 2.42 验证...███\n");**

**{**

**LinkedPoly P\_odd, P\_even;**

**Algo\_2\_42(&P2, &P\_odd, &P\_even);**

**printf("按指数奇偶性拆分多项式之后...\n");**

**printf("奇次幂 P\_odd = ");**

**OutputPoly\_L(P\_odd);**

**printf("\n");**

**printf("偶次幂 P\_even = ");**

**OutputPoly\_L(P\_even);**

**printf("\n\n");**

**}**

**}**

**/\*━━━━━━━━┓**

**┃题2.41：求导函数┃**

**┗━━━━━━━━\*/**

**Status Algo\_2\_41(LinkedPoly P)**

**{**

**LinkedPoly r, pre;**

**if(!P)**

**return ERROR;**

**pre = P;**

**r = P->next;**

**while(r!=P)**

**{**

**if(r->data.exp)**

**{**

**r->data.coef \*= r->data.exp;** **//系数与指数想乘**

**r->data.exp--;** **//指数减一**

**pre = r;**

**}**

**else** **//删除当前结点**

**{**

**pre->next = r->next;**

**free(r);**

**}**

**r = pre->next;**

**}**

**return OK;**

**}**

**/\*━━━━━━━━━━━━━━━┓**

**┃题2.42：按指数奇偶性拆分多项式┃**

**┗━━━━━━━━━━━━━━━\*/**

**Status Algo\_2\_42(LinkedPoly \*P, LinkedPoly \*P\_Odd, LinkedPoly \*P\_Even)**

**{**

**LinkedPoly r, s, p, q;**

**\*P\_Odd = (LinkedPoly)malloc(sizeof(PolyNode));**

**\*P\_Even = (LinkedPoly)malloc(sizeof(PolyNode));**

**if(!(\*P\_Odd) || !(\*P\_Even))**

**exit(OVERFLOW);**

**(\*P\_Odd)->next = \*P\_Odd;**

**(\*P\_Even)->next = \*P\_Even;**

**if(!(\*P))**

**return ERROR;**

**p = \*P\_Odd;**

**q = \*P\_Even;**

**r=(\*P)->next;**

**while(r!=(\*P))**

**{**

**s = r;**

**r = r->next;**

**if(s->data.exp%2)** **//指数为奇次**

**{**

**s->next = p->next;**

**p->next = s;**

**p = p->next;**

**}**

**else**

**{**

**s->next = q->next;**

**q->next = s;**

**q = q->next;**

**}**

**}**

**free(\*P);**

**\*P = NULL;**

**}**

**Status CreatPoly\_L(LinkedPoly \*P, int n, PolyTerm Data[])** **//从无到有创建**

**{**

**int i;**

**LinkedPoly s, r;**

**\*P = (LinkedPoly)malloc(sizeof(PolyNode));** **//创建头结点**

**if(!(\*P))**

**exit(OVERFLOW);**

**(\*P)->next = \*P;**

**for(i=0,r=\*P; i<n; i++)**

**{**

**s = (LinkedPoly)malloc(sizeof(PolyNode));**

**if(!s)**

**exit(OVERFLOW);**

**s->data = Data[i];**

**s->next = r->next;**

**r->next = s;**

**r = r->next;**

**}**

**return OK;**

**}**

**Status OutputPoly\_L(LinkedPoly P)**

**{**

**int i;**

**LinkedPoly r;**

**if(!P)**

**return ERROR;**

**for(r=P->next; r!=P; r=r->next)**

**{**

**if(r==P->next)**

**printf("%d" , r->data.coef);**

**else**

**{**

**if(r->data.coef<0)**

**{**

**printf(" - ");**

**printf("%d", -r->data.coef);**

**}**

**else**

**{**

**printf(" + ");**

**printf("%d", r->data.coef);**

**}**

**}**

**if(r->data.exp)**

**{**

**printf("x");**

**if(r->data.exp!=1)**

**printf("^%d", r->data.exp);**

**}**

**}**

**return OK;**

**}**

**习题中涉及到的数据结构的完整实现以及更多《数据结构》课本源码和《数据结构题集》习题解析见博客更新...**

[**http://www.cnblogs.com/kangjianwei101/**](http://www.cnblogs.com/kangjianwei101/)