2、线性表

2.1 线性表的类型定义

1、 **线性表：一个线性表是 n 个数据元素的有限序列**。是最常用且最简单的一种数据结构。
例如：26 个英文字母表：（A，B…）
在稍微复杂的线性表中，一个数据元素可以由若干个**数据项**组成。在这种情况下，常把数据元素称为**记录**，含有大量记录的线性表又称为**文件**（File）。
2、 线性表是一个非常灵活的数据结构，它的长度可根据需要增长或缩短，即插入或删除。
3、 抽象数据类型线性表的定义如下：

```
ADT List {
    数据对象:D=十 a;! ay∈ElemSet, 1=1,2,..n, n≥0 数据关系:R1={ <a-1,a> lai-
    1,a1∈D, i=2.....n |基本操作:
        InitList( &L )
        操作结果:构造一个空的线性表 L。
         DestroyList( &L )
        初始条件:线性表 L 已存在。操作结果:销毁线性表 L。
         ClearList( &L )
        初始条件:线性表 L 已存在。操作结果:将 L 重置为空表。
         ListEmpty( L )
        初始条件:线性表工已存在。
        操作结果:若 L 为空表，则返回 TRUE,否则返回 FALSE。
         ……（剩余的不写，见 P19）
```

2.2 线性表的顺序表示和实现

1、 分配存储空间
2、 插入：
在第 i 个元素之前插入一个元素时，需将第 n 至 i+1 个元素向后移动一个位置。
3、 删除：
向前移动

2.3 线性表的链式表示和实现

1、线性链表

```
Typedef struct LNode
{
    ElemType data;    //存储的数据
    Struct LNode *next; //尾指针
}LNode,*LinkList;
```

2、线性链表代码展示：

```
#include <stdio.h>
#include <stdlib.h>
#define MAXSIZE 100
#define OK 1
typedef int STATUS;
typedef struct LNode
{
	int num;
	int data;
	struct LNode *next;
}LINK;

typedef struct linkl
{
	int number;
	LINK *head;
}LINKLIST;


STATUS init_list(LINKLIST *plist)
{
	LINK *p;
	p=malloc(sizeof(LINK));//申请内存空间
	if(p==NULL)
	{
		printf("wrong!");
		exit(1);
	}
	plist->head=p;//头指针指向第一个节点
	p->data=-1;//无效数据
	p->next=NULL;
	plist->number=0;
	return OK;
}
STATUS insert(LINKLIST *plist)
{
	/*把一个数据插入到末尾*/
	int indata;
	int sum=1;
	LINK *pfirst,*p;
	pfirst=plist->head;//将头指针赋予给临时变量
	while(pfirst->next)
	{
```

```
            pfirst=pfirst->next;//向下移动指针直到最后一个元素
            sum++;
        }
        //执行插入操作
        printf("请输入要插入的数据：");
        scanf("%d",&indata);
        p=malloc(sizeof(LINK));
        if(p==NULL)
        {
            printf("wrong\n");
            exit(1);
        }
        p->data=indata;
        p->next=NULL;
        p->num=sum;
        pfirst->next=p;
        plist->number++;
        return OK;
}
STATUS disp_link(LINKLIST *plist)
{
        /*输出链表数据*/
        LINK *p;
        p=plist->head->next;
        while(p)
        {
            printf("%d\n",p->data);
            p=p->next;
        }
        return OK;
}
STATUS find_list(LINKLIST *plist)
{
        LINK *pa;
        pa=plist->head;
        int finddata;
        int index[MAXSIZE]={0};
        printf("请输入您要查找的数据：");
        scanf("%d",&finddata);
        while(pa->next)
        {
            if(pa->next->data==finddata)
            {
                index[pa->next->num-1]=1;
```

```
            }
            pa=pa->next;
        }
        disp_link(plist);
        for(int i=0;i<plist->number;i++)
        {
            if(index[i]==1)
            {
                printf("您要查找的数据位于第%d 个位置，其值为%d\n",i+1,finddata);
            }
        }
        return OK;
}
STATUS delete_list(LINKLIST *plist)
{
        disp_link(plist);
        int index;
        LINK *pa,*pb;
        pa=plist->head;
        pb=pa->next;
        printf("请输入您要删除的元素的位置号：");
        scanf("%d",&index);
        while(pb)
        {
            if(pb->num==index)
            {
                //执行删除操作
                /**由于 pa 在 pb 前面，删除 pb**/
                pa->next=pb->next;
                free(pb);
            }
            pb=pb->next;
            pa=pa->next;
        }
        return OK;
}
int main()
{
        LINKLIST Linklist;
        int i=0;
        if(!init_list(&Linklist))
            exit(1);//若初始化失败则退出程序
        while(i<5)
        {
```

```
            insert(&Linklist);
            i++;
            disp_link(&Linklist);
        }
        find_list(&Linklist);
        delete_list(&Linklist);
        disp_link(&Linklist);
        return 0;
}
```

2.4 练习实例：一元多项式的表示及相加

方法：两个链表表示相加的多项式，结果多项式新建一个链表。按幂次顺序分别寻找两个链表中的项，之后依次放入第三个链表中。

如(x^5+x+1)　　+　　(x^4+x^2+1)

①先判断第一个元素 1，两个多项式都有 1，则第一项为 2

②之后判断多项式 A 的下一项，为一次幂，小于多项式 B 中的下一项，则多项式 C 中第二项为 x

③判断多项式 A 的第三项与多项式 B 的第二项，多项式 B 的较小，则多项式 C 中的第三项为 x^2

④……直到多项式 A，B 都走到最后。

附：C 语言代码（多项式相加和多项式相乘）

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

typedef struct
{
    int coef;//系数
    unsigned int expn;//指数
}term,ElemType;

typedef struct LNode
{
    ElemType data;
    struct LNode *next;
}LINK;//节点类型

typedef struct ListL
{
    LINK *head,*tail;
    int len;
```

```
}LinkList;//链表属性

void InitialLink(LinkList *L);//初始化一个链表
int compare(term a,term b);//比较指数
void InsertLink(ElemType e,LinkList *L,int (*cmp)(term,term));//向链表中顺序插入
void DeleteLink(LINK *p,LINK *pf,LinkList *L);
int GetNumber(LinkList *L);//返回多项式项数
void PrintPolyn(LinkList *L);//打印多项式
LinkList *polyn_add(LinkList *L1,LinkList *L2,LinkList *L3);//多项式加法
void polyn_multi(LinkList *L1,LinkList *L2,int n);
int getmin(int a,int b);

int main()
{
	int i,n1,n2;
	char choice;
	term temp1,temp2;
	LinkList Pa_Linklist,Pb_Linklist,Pc_Linklist;

	InitialLink(&Pa_Linklist);
	printf("请输入第一个多项式的项数:\n");
	scanf("%d",&n1);
	for(i=0;i<n1;i++)
	{
		printf("第%d 项的系数：",i+1);
		scanf("%d",&temp1.coef);
		printf("第%d 项的指数：(x^)",i+1);
		scanf("%d",&temp1.expn);
		InsertLink(temp1,&Pa_Linklist,compare);
	}
	printf("Now output the first polynomial:");
	PrintPolyn(&Pa_Linklist);
	getch();

	InitialLink(&Pb_Linklist);
	printf("请输入第二个多项式的项数:\n");
	scanf("%d",&n2);
	for(i=0;i<n2;i++)
	{
		printf("第%d 项的系数：",i+1);
		scanf("%d",&temp2.coef);
		printf("第%d 项的指数：(x^)",i+1);
		scanf("%d",&temp2.expn);
		InsertLink(temp2,&Pb_Linklist,compare);
```

```
    }
    printf("Now output the first polynomial:");
    PrintPolyn(&Pb_Linklist);
    getch();

    printf("请输入您要做的数学运算：\n1、加法\n2、乘法\n");
    fflush(stdin);
    choice=getchar();

    switch(choice)
    {
        case '1':
        {
            InitialLink(&Pc_Linklist);
            polyn_add(&Pa_Linklist,&Pb_Linklist,&Pc_Linklist);
            PrintPolyn(&Pc_Linklist);
            break;
        }
        case '2':
        {
            polyn_multi(&Pa_Linklist,&Pb_Linklist,getmin(Pa_Linklist.len,Pb_Linklist.len));
            break;
        }
        default:
        {
            printf("输入错误！\n");
            break;
        }
    }
    return 0;
}

void InitialLink(LinkList *L)
{
    LINK *p;
    p=(LINK*)malloc(sizeof(LINK));
    if(!p)
    {
        printf("分配内存失败！\n");
        getch();
        exit(1);
    }
    p->data.coef=0;
    p->data.expn=-1;
```

```
    p->next=NULL;
    L->head=L->tail=p;
    L->len=0;
}

int compare(term a,term b)
{
    if(a.expn>b.expn)
        return 1;
    else if(a.expn==b.expn)
        return 0;
    else
        return -1;
}

void InsertLink(ElemType e,LinkList *L,int (*cmp)(term,term))
{
    LINK *p,*pfront,*pnew;

    pnew=(LINK*)malloc(sizeof(LINK));
    if(!pnew)
    {
        printf("分配内存失败！\n");
        getch();
        exit(1);
    }
    pnew->data.coef=e.coef;
    pnew->data.expn=e.expn;
    pnew->next=NULL;

    p=L->head->next;
    pfront=L->head;
    while(p&&compare(p->data,e)<0)
    {
        pfront=p;
        p=p->next;
    }
    if(p==NULL)
    {
        //printf("到表尾！插在表末尾！\n");
        pfront->next=pnew;
        L->tail=pnew;
        L->len++;
    }
```

```
    else if(p&&compare(p->data,e)>0)
    {
        pnew->next=p;
        pfront->next=pnew;
        L->len++;
    }
    else if(p&&compare(p->data,e)==0)
    {
        p->data.coef+=e.coef;
        if(p->data.coef==0)
        {
            DeleteLink(p,pfront,L);
            //删除这个节点
        }
        free(pnew);
    }
}

void DeleteLink(LINK *p,LINK *pf,LinkList *L)
{
    if(!p->next)
    {
        L->tail=pf;
    }
    pf->next=p->next;
    free(p);
    L->len--;
}

int GetNumber(LinkList *L)
{
    return L->len;
}

void PrintPolyn(LinkList *L)
{
    LINK *p;
    p=L->head->next;

    printf("P(x)=%dx^%d",p->data.coef,p->data.expn);
    p=p->next;
    while(p)
    {
        printf("+%dx^%d",p->data.coef,p->data.expn);
```

```
            p=p->next;
        }
        printf("\n.It has %d number\n",GetNumber(L));
}

LinkList *polyn_add(LinkList *L1,LinkList *L2,LinkList *L3)
{
        LINK *p1,*p2,*p3,*pnew;
        p1=L1->head->next;
        p2=L2->head->next;
        p3=L3->head;

        while(p1&&p2)
        {
                pnew=(LINK*)malloc(sizeof(LINK));
                //p1=p1->next;
                //p2=p2->next;
                if(compare(p1->data,p2->data)>0)//第一个比第二个大
                {
                        pnew->data.coef=p2->data.coef;
                        pnew->data.expn=p2->data.expn;
                        pnew->next=NULL;
                        p3->next=pnew;//把小的放入新的链表
                        p3=p3->next;
                        p2=p2->next;
                        L3->len++;
                }
                else if(compare(p1->data,p2->data)<0)//第一个比第二个大
                {
                        pnew->data.coef=p1->data.coef;
                        pnew->data.expn=p1->data.expn;
                        pnew->next=NULL;
                        p3->next=pnew;//把小的放入新的链表
                        p3=p3->next;
                        p1=p1->next;
                        L3->len++;
                }
                else if(compare(p1->data,p2->data)==0)//指数一样大，系数相加
                {
                        if(p1->data.coef==-p2->data.coef)//系数互为相反数
                        {
                                free(pnew);//释放新的空间
                        }
                        else
```

```
                {
                    pnew->data.coef=p1->data.coef+p2->data.coef;
                    pnew->data.expn=p1->data.expn;
                    pnew->next=NULL;
                    p3->next=pnew;
                    p3=p3->next;
                    L3->len++;
                }
                p1=p1->next;
                p2=p2->next;
            }
        }
        if(!p1)//如果 p1 先到尾部
        {
            p3->next=p2;
            while(p2)
            {
                L3->len++;
                p2=p2->next;
            }
        }
        else if(!p2)//如果 p2 先到尾部
        {
            p3->next=p1;
            while(p1)
            {
                L3->len++;
                p2=p2->next;
            }
        }
        return L3;
}

void polyn_multi(LinkList *L1,LinkList *L2,int n)
{
        int i=0;
        LinkList *temp,*pnew[n],*preturn,kong;
        LINK *p1,*p2;
        ElemType e;
        InitialLink(&kong);
        if(L1->len>L2->len)
        {
            temp=L1;
            L1=L2;
```

```
            L2=temp;
        }
        p1=L1->head->next;
        p2=L2->head->next;
        while(p2&&p1)
        {
            pnew[i]=(LinkList*)malloc(sizeof(LinkList));
            InitialLink(pnew[i]);
            //p3=pnew->head->next;
            while(p2)
            {
                e.coef=p1->data.coef*p2->data.coef;
                e.expn=p1->data.expn+p2->data.expn;
                InsertLink(e,pnew[i],compare);
                p2=p2->next;
            }
            p2=L2->head->next;
            p1=p1->next;
            i++;
        }
        preturn=pnew[0];
        for(i=0;i+1<n;i++)
        {
            preturn=polyn_add(preturn,pnew[i+1],&kong);
        }
        PrintPolyn(preturn);
}

int getmin(int a,int b)
{
        return a<b?a:b;
}
```

2.5 链表拓展

1、**循环链表**

循环链表是另一种形式的链式存储结构。

特点：表中最后一个节点的指针指向头节点，整个链表形成一个环。

优点：可以从表中任一节点出发均可找到表中其它节点。

特殊注意：判断循环链表为空的条件是 **p(p->next)是否等于 phead**

用处：合并**两个只设置尾指针的循环链表**的操作，时间复杂度仅为 O(1)

使第一个尾等于第二个尾->next（头），使第二个尾=第一个尾->next(头)

（此处需要保存一下指针值再进行操作）

2、双向链表

双向链表的节点中有两个指针，一个指向前，一个指后。

```
typedef struct DuLNode
{
    ElemType data;
    struct DuLNode *prior;
    struct DuLNode *next;
}
```

详细见书 P35