**《数据结构》**

**实验报告**

**实验名称：线性数据结构实现与应用**

|  |  |
| --- | --- |
| **学院** | **网络空间安全学院** |
| **班级** | **2020级软件工程专业软卓2班** |
| **学号** | **202041404130** |
| **姓名** | **张 瀚** |
| **指导教师** | **周 坤 晓** |
| **时间** | **2021/04/02** |

# 需求分析

通过实验达到：

⑴ 理解和掌握线性结构的概念及其典型操作的算法思想；

⑵ 熟练掌握基本线性结构的顺序存储结构、链式存储结构及其操作的实现；

⑶ 理解和掌握受限线性结构——堆栈、队列的概念及其典型操作的算法思想、实现。

# 实验题目1

实验题目：一元多项式的操作

实验要求：

设有两个一元多项式：

p(x)=p0+p1x+p2x2+···+pnxn

q(x)=q0+q1x+q2x2+···+qmxm

多项式项的系数为实数，指数为整数，设计实现一元多项式的下列操作：

① 多项式链表建立：以（系数，指数）（例如（3,0）（-1,1））方式输入项建立多项式，返回所建立的链表的头指针；在输入一个多项式时如果遇到两项指数相同，可以选择舍弃、覆盖或者系数相加操作；

② 多项式排序：将所建立的多项式按指数非递减（从小到大）进行排序（也可以在建立时保证有序，创建一个有序链表）；

③ 多项式相加：实现两个多项式相加操作。操作生成一个新的多项式，原有的两个多项式不变，可以返回生成的多项式的头指针；

④ 多项式相减：实现两个多项式相减操作。操作生成一个新的多项式，原有的两个多项式不变，可以返回生成的多项式的头指针；

⑤ 多项式的输出：按照p0+p1x1+p2x2+···+pnxn格式输出多项式；

⑥ 主函数通过调用多项式链表建立函数，通过文件读取或者控制台输入两个多项式并分别输出；输出排序后的两个多项式；调用多项式相加函数实现多项式相加、相减操作，输出操作结果。

测试数据：自行设计2组测试数据，两个多项式均不少于4项，并且需要有同类项，至少一个同类项系数相同，另一个同类项系数相反。

## 数据结构设计

定义带头结点单链表的结构体如下：

typedef struct Node

{

double a; //系数

int b; //指数

struct Node \*next;

}SLNode;

## 主要操作算法设计与分析

**多项式的输出算法描述：**

void ListPrint(SLNode \*head)

**返回类型：**无返回值

**是否含参数：**是，含有一个结构体类型的指针

**步骤：**

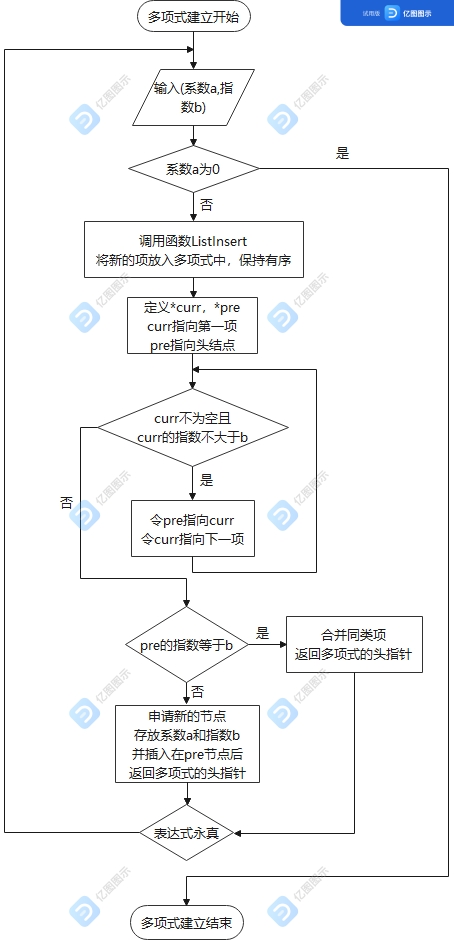
A.定义结构体指针\*p,整数i=0，i用于判断多项式是否为0；

B.使用for(p=head->next;p!=NULL;p=p->next)循环：①首先判断当前项的系数是否为0，如果为0，跳过此次后面步骤，进入下一次循环，否则继续；②判断i是否等于1，若不等于1，则说明该项是第一个非零项，不输出加号，否则输出加号；③令i=1，此多项式不为0；④输出系数，并判断系数的正负性，系数为负时需带上括号；⑤判断指数是否为0，若不为0输出xn（n为指数），否则不输出；

C.若i等于0，说明多项式为0，输出“0”；

**算法时间复杂度：**O(n)

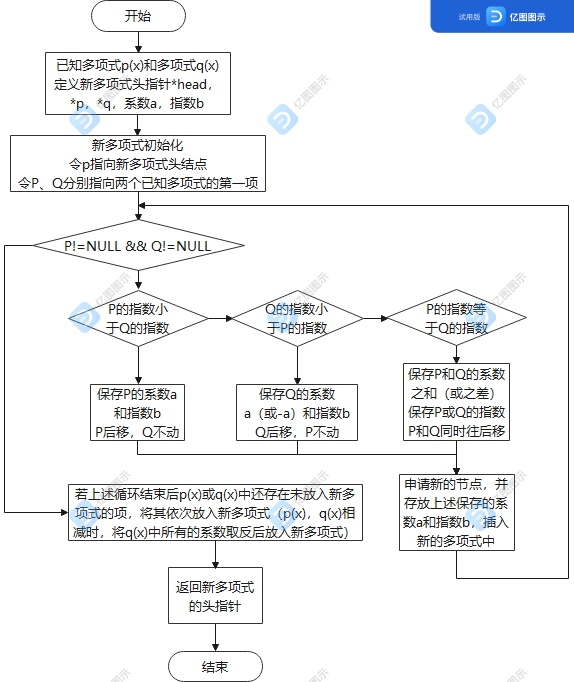
**（有序）多项式链表建立算法流程图：**



**算法时间复杂度分析：**

最好的情况：每次新的项的指数b都比原来多项式所有项小，算法中内层while循环的循环次数每次均为0，外层while循环共循环n次，时间复杂度为O(n)；最坏的情况：每次新的项的指数b都比原来多项式所有项大，内层while循环共循环0+1+2+···+(n-2)次，外层while循环共循环n次，时间复杂度为O(n²)。

**有序多项式相加（相减）操作算法流程图：**

****

**算法时间复杂度分析：**

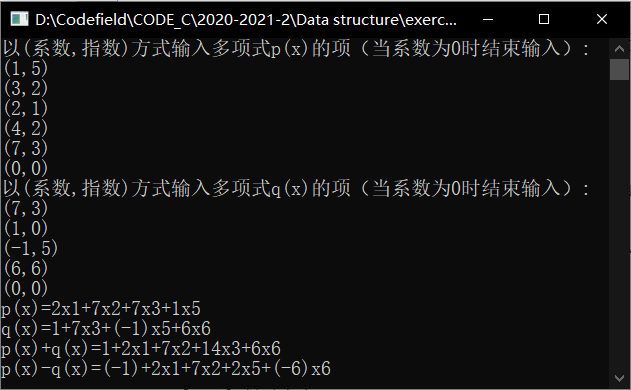
算法开始时，指针P和Q指向两个有序多项式的第一项。第一个while循环，每次循环指针P或Q都会后移一次（或P和Q同时后移一次）；第一个while循环结束后，将p(x)或q(x)还未放入新多项式中的元素依次放入新多项式中，每放入一个，P或Q后移一次；算法结束时，P和Q最终都指向了两个多项式链表末尾的空节点，因此算法时间复杂度为O(n)，n=n1+n2，n1、n2分别为两个多项式的项数。

## 程序运行过程及结果（截屏粘贴）

测试数据：

p(x):(1,5)\n(3,2)\n(2,1)\n(4,2)\n(7,3)\n(0,0)

q(x):(7,3)\n(1,0)\n(-1,5)\n(6,6)\n(0,0)



# 实验题目2

实验题目：顺序循环队列的基本操作

实验要求：

设计采取少用一个存储单元的方法解决队列满和队列空判断。设队列的元素类型为DataType，实现顺序循环队列的各种基本操作：

① 初始化队列Q；

② 判断队列Q是否为空；

③ 入队操作；

④ 出队操作；

⑤ 输出队列元素个数；

⑥ 输出队列序列；

⑦ 编写一个测试主函数，要求将若干个元素（不少于8个）入队；出队若干个元素（同时输出出队元素）；输出队列序列；通过函数调用实现以上各项操作。

测试数据：

字符型数据：假设队列的元素类型为char，建议用自己姓名的全拼。

## 数据结构设计

定义顺序循环队列的结构体如下：

typedef struct

{

// MaxQueueSize为队列的最大存储单元个数

//可存放MaxQueueSize-1个数据元素

DataType queue[MaxQueueSize];

int rear; //队尾指针

int front; //队头指针

}SeqCQueue;

## 主要操作算法设计与分析

**判断队列Q是否为空算法描述：**

int QueueEmpty(SeqCQueue Q)

**返回类型：**int型

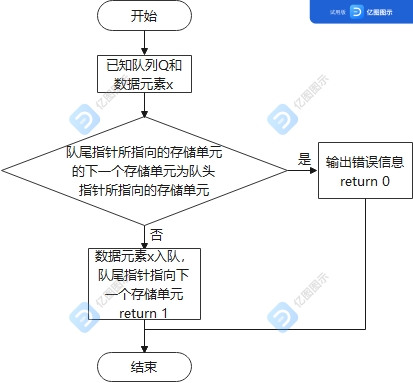
**是否含参数：**是，含有一个结构体

**步骤：**

判断Q.rear是否等于Q.front，即队头指针和队尾指针是否指向同一存储单元。若是返回1，否则返回0；

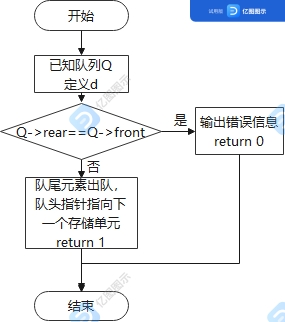
**算法时间复杂度：**O(1)

**入队操作算法流程图：**



**算法时间复杂度：**O(1)

**出队操作算法流程图：**

****

**算法时间复杂度：**O(1)

**输出队列元素个数算法描述：**

void QueueLength(SeqCQueue Q)

**返回类型：**无返回值

**是否含参数：**是，含有一个结构体

**步骤：**

A.判断Q.rear和Q.front的大小，若Q.rear大于等于Q.front,队列元素个数为Q.rear-Q.front；否则队列元素个数等于Q.rear+MaxQueueSize-Q.front（MaxQueueSize为队列的最大存储单元个数）；

B.输出队列元素个数；

**算法时间复杂度：**O(1)

**输出队列序列算法描述：**

void QueuePrint(SeqCQueue Q)

**返回类型：**无返回值

**是否含参数：**是，含有一个结构体

**步骤：**

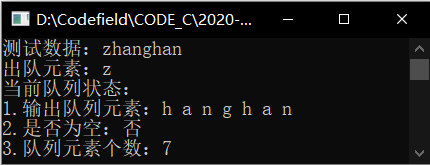
A.计算队列元素个数；

B.如果元素个数为0，输出“队列为空”；否则使用for(i=0;i<len;i++)循环，队列中每个元素可用Q.queue[(Q.front+i)%MaxQueueSize]表示（其中MaxQueueSize为队列最大存储单元个数），从队头元素开始依次输出队列中所有元素；

**算法时间复杂度：**O(n)

## 程序运行过程及结果（截屏粘贴）

测试数据：zhanghan



# 实验题目3

实验题目：有序顺序表的设计

实验要求：

1．有序顺序表的操作，包括初始化，求数据元素个数，插入，删除和取数据元素。放在头文件中（建议尝试用动态数组实现有序顺序表；注意有序顺序表的操作与课本上的操作有所不同，需要重写一些操作，如ListInsert(L,x)，不需要参数i）；

2．设计合并函数ListMerge（L1,L2,L3），其功能是把有序表L1和L2中的数据合并到L3中，要求L3中的数据依然保持有序。(要求时间复杂度O(n), n= n1+n2，n1、n2分别为两个顺序表的长度)；

3．设计一个测试主函数实际验证所设计有序表的各项操作以及合并函数的正确性。

测试数据：

字符型或者整形：z,h,o,u, k,u,n,x,i,a,o(同学们自己名字的拼音)

## 数据结构设计

定义有序顺序表的结构体如下：

typedef struct

{

DataType \*list; //动态数组

int size; //数组长度

}SeqList;

## 主要操作算法设计与分析

**删除操作算法描述：**

int ListDelete(SeqList \*L,int i,DataType \*x)

**返回类型：**int型

**是否含参数：**是，含有一个结构体类型的指针，一个int型数据，一个数据元素类型的指针

**步骤：**

A.判断顺序表是否为空和参数i是否合法，出错时给出对应出错信息，并返回0；

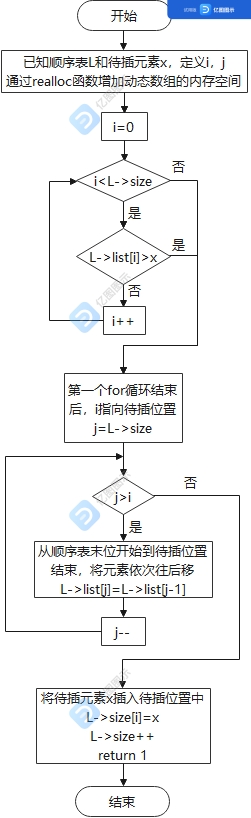
B.当顺序表非空且参数i正确时，把list[i]存放到参数x中，然后从前向后依次前移从list[i+1]到list[size-1]中的数据元素；

C.数据元素个数减1；

**算法时间复杂度分析：**

最坏的情况：i=0，需要移动size-1个数据元素；最好的情况：i=size，不需要移动数据元素。平均时间复杂度为O(n)。

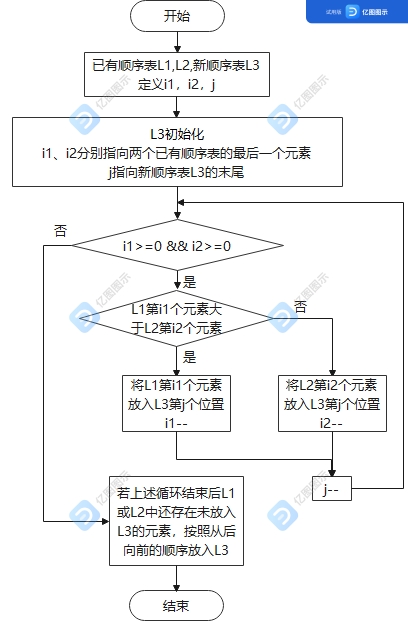
**插入操作算法流程图：**



**算法时间复杂度分析：**

每次插入需要经过寻找待插位置和将元素后移两个步骤，需要访问顺序表中的所有元素，因此算法时间复杂度为O(n)。

**合并函数ListMerge算法流程图：**

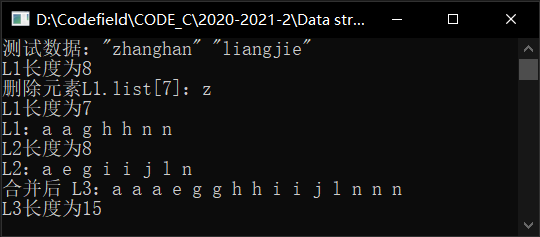
****

**算法时间复杂度分析：**

算法开始时，i1和i2分别为两个有序顺序表的最后一个元素的下标。第一个while循环，每次循环结束i1减1或i2减1；第一个while循环结束后，将L1或L2还未放入新顺序表L3中的元素按照从后向前的顺序依次放入L3中，每放入一个，i1减1或i2减1；算法结束时，i1和i2均等于-1，因此算法时间复杂度为O(n)，n=n1+n2，n1、n2分别为两个顺序表的长度。

## 程序运行过程及结果（截屏粘贴）

测试数据：zhanghan liangjie



# 实验题目4

实验题目：栈操作及应用：将一个非负十进制整数转换成d进制，并输出

算法思想：十进制数N转换成其他d进制数的转换公式 N=(N DIV d)\*d+N MOD d,其中：DIV为整除运算，MOD为求余运算（取模），d为进制数。

计算过程是按照低位到高位的顺序依次产生二进制的各个数位，对打印输出来说，一般应该按照从高位到低位的顺序进行，而这恰好和计算过程相反。因此，如果设置一个顺序栈，将计算过程中得到的二进制数的各个数位顺序进栈，然后再按照出栈序列打印输出，这样就可以得到与输入相对应的二进制数。

测试数据：自己学号的后3位

## 数据结构设计

定义链式堆栈的结构如下：

typedef struct snode

{

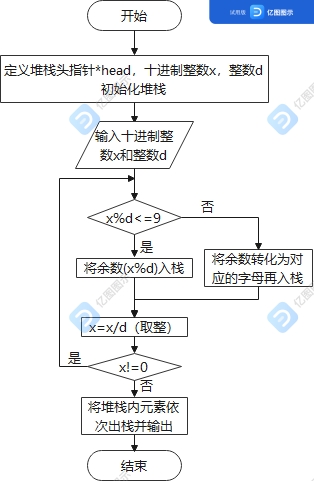
DataType data; //数据元素

struct snode \*next;

}LSNode;

## 主要操作算法设计与分析

**算法流程图：**



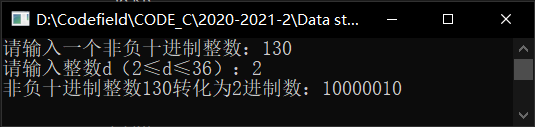
**算法时间复杂度分析：**

每次参与循环的x等于上一次循环中x/d的商，循环结束的条件为商等于0，可以计算出，循环总次数小于等于()。因此算法时间复杂度是O()。

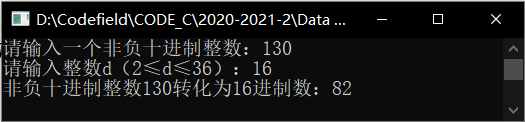
## 程序运行过程及结果（截屏粘贴）

测试数据：

十进制整数：130 d：2



十进制整数：130 d：16



# 问题与总结

1.实验题目1的多项式加减法操作和实验题目3的合并顺序表操作类似，都是在两个线性表有序的前提下合并成一个新的线性表。其中涉及到的排序问题，普通算法可能需要多重循环实现，时间复杂度高；但是利用这个问题的前提，可以设计出时间复杂度为O(n)的算法，大大提高了效率；

2.实验题目1，通过(系数,指数)的方式输入多项式的项时，最初仅使用到了循环和scanf函数实现多次输入、建立多项式，但运行存在问题，输入第一项回车之后会进入死循环；解决方法是，在scanf函数后写上" getchar()"用于清除缓存区。

3.实验题目1的多项式建立操作和实验题目3的插入操作类似，都是在插入时，找到合适的位置插入，从而保证线性表的有序性；

4.实验题目2，通过画图可以方便、形象地看出队满的状态，计算出队头指针和队尾指针的数量关系，和队列元素个数与队头、队尾指针的数量关系，从而实现判断队空队满、输出队列元素个数、输出队列序列等操作；

5.类比十进制数转化为二进制数的过程，可以推广出十进制数转化为更多其他进制数的过程，注意当转化进制超过十时，大于9的余数需要使用字母A~Z表示；同时借助堆栈，可以方便地输出转化后地结果；

6.制作实验报告的过程中，学会制作流程图，可以方便、形象地看出算法的结构、运行步骤，比文字描述更容易理解。

# 附录：源代码

## 题目1源代码：

/\* 多项式操作头文件1.1.h \*/

typedef struct Node

{

double a; //系数

int b; //指数

struct Node \*next;

}SLNode;

void ListInitiate(SLNode \*\*head) //初始化

{

\*head=(SLNode \*)malloc(sizeof(SLNode));

(\*head)->next=NULL;

}

//多项式链表建立，保证有序

SLNode \*ListInsert(SLNode \*head,double a,int b)

{

SLNode \*curr,\*pre,\*q;

curr=head->next; //curr指向第一项

pre=head; //pre指向头结点

//定位，从第一项开始寻找第一个大于或等于b的元素

while(curr!=NULL && curr->b<=b)

{

pre=curr;

curr=curr->next;

}

//如果找到与b相等的元素，合并同类项

if(pre->b==b)

{

pre->a=pre->a+a;

return head; //返回头指针

}

//插入新的项

q=(SLNode \*)malloc(sizeof(SLNode));

q->a=a;

q->b=b;

q->next=pre->next;

pre->next=q;

return head;

}

SLNode \*ListPlus(SLNode \*P,SLNode \*Q) //多项式相加

{

SLNode \*head,\*p,\*q;

int a,b;

head=(SLNode \*)malloc(sizeof(SLNode)); //申请新多项式的头结点

head->next=NULL;

//循环初始化

p=head;

P=P->next;

Q=Q->next;

//建立新多项式，并保证有序

while (P!=NULL && Q!=NULL)

{

//从两个多项式的第一项开始比较大小

if(P->b<Q->b)

{

a=P->a;

b=P->b;

P=P->next; //P后移，Q不动

}

else if(Q->b<P->b)

{

a=Q->a;

b=Q->b;

Q=Q->next; //Q后移，P不动

}

else if(P->b==Q->b) //合并同类项

{

a=P->a+Q->a;

b=P->b;

//P,Q均后移

P=P->next;

Q=Q->next;

}

q=(SLNode \*)malloc(sizeof(SLNode));

q->a=a;

q->b=b;

q->next=p->next;

p->next=q;

p=p->next;

}

//若上述循环结束后P或Q中还有元素未放入新多项式，依次放入

while (P!=NULL)

{

q=(SLNode \*)malloc(sizeof(SLNode));

q->a=P->a;

q->b=P->b;

q->next=p->next;

p->next=q;

p=p->next;

P=P->next;

}

while (Q!=NULL)

{

q=(SLNode \*)malloc(sizeof(SLNode));

q->a=Q->a;

q->b=Q->b;

q->next=p->next;

p->next=q;

p=p->next;

Q=Q->next;

}

return head; //返回新多项式的头指针

}

SLNode \*ListMinus(SLNode \*P,SLNode \*Q) //多项式相减

{

SLNode \*head,\*p,\*q;

int a,b;

head=(SLNode \*)malloc(sizeof(SLNode));

head->next=NULL;

p=head;

P=P->next;

Q=Q->next;

while (P!=NULL && Q!=NULL)

{

if(P->b<Q->b)

{

a=P->a;

b=P->b;

P=P->next;

}

else if(Q->b<P->b)

{

a=-Q->a; //将多项式Q中所有系数的相反数放入新多项式

b=Q->b;

Q=Q->next;

}

else if(P->b==Q->b)

{

a=P->a-Q->a; //同类项相减

b=P->b;

P=P->next;

Q=Q->next;

}

q=(SLNode \*)malloc(sizeof(SLNode));

q->a=a;

q->b=b;

q->next=p->next;

p->next=q;

p=p->next;

}

while (P!=NULL)

{

q=(SLNode \*)malloc(sizeof(SLNode));

q->a=P->a;

q->b=P->b;

q->next=p->next;

p->next=q;

p=p->next;

P=P->next;

}

while (Q!=NULL)

{

q=(SLNode \*)malloc(sizeof(SLNode));

q->a=-Q->a; //将多项式Q中所有系数的相反数放入新多项式

q->b=Q->b;

q->next=p->next;

p->next=q;

p=p->next;

Q=Q->next;

}

return head;

}

void ListPrint(SLNode \*head) //多项式的输出

{

SLNode \*p;

int i=0; //i用于判断多项式是否为0

for(p=head->next;p!=NULL;p=p->next)

{

if(p->a==0) continue; //如果系数为0，继续下一次循环

//第一个非零项前不用带加号

if(i==1 && p!=head->next) printf("+");

i=1; //i=1，多项式不为0

//若系数小于0，系数需带上括号

if(p->a<0) printf("(%.1lf)",p->a);

if(p->a>0) printf("%.1lf",p->a);

if(p->b!=0) printf("x%d",p->b); //若指数为0，不输出x

}

if(i==0) printf("0"); //若多项式为0，输出0

printf("\n");

}

int Destroy(SLNode \*\*head)

{

SLNode \*p,\*p1;

p=\*head;

while (p!=NULL)

{

p1=p;

p=p->next;

free(p1);

}

\*head=NULL;

}

/\* 实验1.1 多项式 \*/

#include <stdio.h>

#include <stdlib.h>

#include "1.1.h"

void main()

{

SLNode \*P,\*Q,\*plus,\*minus;

double a;

int b;

ListInitiate(&P); //多项式链表初始化

ListInitiate(&Q);

printf("以(系数,指数)方式输入多项式p(x)的项（当系数为0时结束输入）\n");

do

{

scanf("(%lf,%d)",&a,&b); //以(a,b)的方式输入

getchar();

if(a==0) break; //当系数为0时结束输入

ListInsert(P,a,b); //有序插入，建立有序链表

}while(1);

printf("以(系数,指数)方式输入多项式q(x)的项（当系数为0时结束输入）\n");

do

{

scanf("(%lf,%d)",&a,&b);

getchar();

if(a==0) break;

ListInsert(Q,a,b);

}while(1);

plus=ListPlus(P,Q); //多项式相加

minus=ListMinus(P,Q); //多项式相减

printf("p(x)=");

ListPrint(P); //多项式输出

printf("q(x)=");

ListPrint(Q);

printf("p(x)+q(x)=");

ListPrint(plus);

printf("p(x)-q(x)=");

ListPrint(minus);

Destroy(&P);

Destroy(&Q);

Destroy(&plus);

Destroy(&minus);

}

## 题目2源代码：

/\* 顺序循环队列操作头文件1.2.h \*/

typedef struct

{

DataType queue[MaxQueueSize];

int rear;

int front;

}SeqCQueue;

void QueueInitiate(SeqCQueue \*Q) //初始化

{

Q->rear=0;

Q->front=0;

}

int QueueEmpty(SeqCQueue Q) //判断队列Q是否为空

{

return(Q.rear==Q.front); //若为空返回1，否则返回0

}

int QueueAppend(SeqCQueue \*Q,DataType x) //入队

{

if((Q->rear+1)%MaxQueueSize==Q->front)

{

printf("队列已满无法插入！\n");

return 0;

}

else

{

Q->queue[Q->rear]=x;

Q->rear=(Q->rear+1)%MaxQueueSize;

return 1;

}

}

int QueueDelete(SeqCQueue \*Q,DataType \*d) //出队

{

if(Q->rear==Q->front)

{

printf("队列已空无数据元素出队列！\n");

return 0;

}

else

{

\*d=Q->queue[Q->front];

Q->front=(Q->front+1)%MaxQueueSize;

return 1;

}

}

void QueueLength(SeqCQueue Q) //输出队列元素个数

{

int len;

if(Q.rear>=Q.front) len=Q.rear-Q.front;

else len=Q.rear+MaxQueueSize-Q.front;

printf("%d\n",len);

}

void QueuePrint(SeqCQueue Q) //输出队列序列

{

int i,len;

if(Q.rear>=Q.front) len=Q.rear-Q.front;

else len=Q.rear+MaxQueueSize-Q.front;

if(len==0) printf("队列为空");

else

{

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

{

printf("%c ",Q.queue[(Q.front+i)%MaxQueueSize]);

}

}

printf("\n");

}

/\* 实验1.2 顺序循环队列 \*/

#include <stdio.h>

#define MaxQueueSize 10 //最大可存放元素个数为10-1=9

typedef char DataType;

#include "1.2.h"

void main()

{

SeqCQueue myQueue;

int i;

DataType x;

char str[]="zhanghan"; //测试数据

printf("测试数据：%s\n",str);

QueueInitiate(&myQueue); //初始化

for(i=0;str[i]!='\0';i++)

{

QueueAppend(&myQueue,str[i]); //入队

}

QueueDelete(&myQueue,&x); //出队

printf("出队元素：%c\n",x); //输出出队元素

printf("当前队列状态：\n");

printf("1.输出队列元素：");

QueuePrint(myQueue); //输出队列序列

//判断是否为空

printf("2.是否为空：%s\n",(QueueEmpty(myQueue)?"是":"否"));

printf("3.队列元素个数：");

QueueLength(myQueue); //输出队列元素个数

}

## 题目3源代码：

/\* 有序顺序表操作头文件1.3.h \*/

typedef struct

{

DataType \*list; //动态数组

int size;

}SeqList;

void ListInitiate(SeqList \*L) //初始化

{

L->size=0;

L->list=NULL;

}

int ListLength(SeqList L)

{

return L.size;

}

int ListInsert(SeqList \*L,DataType x) //插入并保持升序

{

int i,j;

//增加动态数组的元素个数

L->list=(DataType \*)realloc(L->list,(L->size+1)\*sizeof(DataType));

for(i=0;i<L->size;i++) //定位，找到第一个比x大的元素

{

if(L->list[i]>x) break;

}

for(j=L->size;j>i;j--) //从定位元素开始依次往后移

{

L->list[j]=L->list[j-1];

}

L->list[i]=x; //插入x

L->size++; //元素个数加1

return 1;

}

int ListDelete(SeqList \*L,int i,DataType \*x) //删除

{

int j;

if(L->size<=0)

{

printf("顺序表已空无数据元素可删！\n");

return 0;

}

else if(i<0 || i>L->size-1)

{

printf("参数i不合法\n");

return 0;

}

else

{

\*x=L->list[i];

for(j=i+1;j<=L->size-1;j++)

{

L->list[j-1]=L->list[j];

}

L->size--;

return 1;

}

}

int ListGet(SeqList L,int i,DataType \*x) //取数据元素

{

if(i<0 || i>L.size-1)

{

printf("参数i不合法！\n");

return 0;

}

else

{

\*x=L.list[i];

return 1;

}

}

/\* 实验1.3 有序顺序表的操作和合并 \*/

#include <stdio.h>

#include <stdlib.h>

typedef char DataType;

#include "1.3.h" //包含有序顺序表的操作函数

void main()

{

void ListMerge(SeqList,SeqList,SeqList \*);

SeqList L1,L2,L3;

int i;

DataType x;

char str1[]="zhanghan",str2[]="liangjie"; //测试数据

printf("测试数据：\"%s\" \"%s\"\n",str1,str2);

//初始化顺序表

ListInitiate(&L1);

ListInitiate(&L2);

for(i=0;str1[i]!='\0';i++)

{

ListInsert(&L1,str1[i]); //有序插入

}

printf("L1长度为%d\n",ListLength(L1));

ListDelete(&L1,7,&x); //删除

printf("删除元素L1.list[7]：%c\n",x);

printf("L1长度为%d\n",ListLength(L1));

for(i=0;str2[i]!='\0';i++)

{

ListInsert(&L2,str2[i]);

}

printf("L1：");

for(i=0;i<ListLength(L1);i++) //求元素个数

{

ListGet(L1,i,&x); //取数据元素

printf("%c ",x);

}

printf("\n");

printf("L2长度为%d\n",ListLength(L2));

printf("L2：");

for(i=0;i<ListLength(L2);i++)

{

ListGet(L2,i,&x);

printf("%c ",x);

}

printf("\n");

ListMerge(L1,L2,&L3); //合并

printf("合并后 L3：");

for(i=0;i<ListLength(L3);i++)

{

ListGet(L3,i,&x);

printf("%c ",x);

}

printf("\n");

printf("L3长度为%d\n",ListLength(L3));

free(L1.list);

free(L2.list);

free(L3.list);

}

void ListMerge(SeqList L1,SeqList L2,SeqList \*L3) //合并函数

{

int i1,i2,j;

//动态申请内存空间

L3->size=L1.size+L2.size;

L3->list=(DataType \*)calloc(L3->size,sizeof(DataType));

i1=L1.size-1; //i1指向数组L1的最后一个元素

i2=L2.size-1; //i2指向数组L2的最后一个元素

j=L3->size-1; //j指向数组L3的最后一个元素

while (i1>=0 && i2>=0)

{

//对比L1和L2数组的最后一个元素

if(L1.list[i1]>L2.list[i2])

{

//将较大者从L3末尾开始放入

L3->list[j]=L1.list[i1];

//i1往前移，i2不变，继续比较L1.list[i1]和L2.list[i2]

i1--;

}

else

{

L3->list[j]=L2.list[i2];

i2--;

}

j--; //j往前移

}

if(i1>=0) //将剩下没有参与比较的元素依次放入L3

{

while (i1>=0)

{

L3->list[j]=L1.list[i1];

i1--;

j--;

}

}

if(i2>=0)

{

while (i2>=0)

{

L3->list[j]=L2.list[i2];

i2--;

j--;

}

}

}

## 题目4源代码：

/\* 实验1.4 十进制转换为d进制 \*/

#include <stdio.h>

#include <stdlib.h>

typedef char DataType;

#include "LinStack.h" //包含堆栈的操作函数

int main()

{

LSNode \*head;

int x,d;

char t;

StackInitiate(&head); //初始化堆栈

printf("请输入一个非负十进制整数：");

scanf("%d",&x);

printf("请输入整数d（2≤d≤36）：");

scanf("%d",&d);

printf("非负十进制整数%d转化为%d进制数：",x,d);

do

{

//当余数为0~9时，将余数入栈

if(x%d<=9) StackPush(head,'0'+x%d);

//当余数大于9时，用字母A~Z表示余数，入栈

else StackPush(head,'A'+x%d-10);

x=x/d; //取整

} while (x!=0);

while(StackNotEmpty(head))

{

StackPop(head,&t); //出栈

printf("%c",t);

}

Destroy(head);

}

/\* 头文件LinStack.h \*/

typedef struct snode

{

DataType data;

struct snode \*next;

}LSNode;

void StackInitiate(LSNode \*\*head)

{

\*head=(LSNode \*)malloc(sizeof(LSNode));

(\*head)->next=NULL;

}

int StackNotEmpty(LSNode \*head)

{

if(head->next==NULL) return 0;

else return 1;

}

void StackPush(LSNode \*head,DataType x)

{

LSNode \*p;

p=(LSNode \*)malloc(sizeof(LSNode));

p->data=x;

p->next=head->next;

head->next=p;

}

int StackPop(LSNode \*head,DataType \*d)

{

LSNode \*p=head->next;

if(p==NULL)

{

printf("堆栈已空出错！");

return 0;

}

head->next=p->next;

\*d=p->data;

free(p);

return 1;

}

int StackTop(LSNode \*head,DataType \*d)

{

LSNode \*p=head->next;

if(p==NULL)

{

printf("堆栈已空出错！");

return 0;

}

\*d=p->data;

return 1;

}

void Destroy(LSNode \*head)

{

LSNode \*p,\*p1;

p=head;

while(p!=NULL)

{

p1=p;

p=p->next;

free(p1);

}

}