**综合实验报告样例**

#### 一、实验题目

一元多项式的加法运算

#### 二、实验内容

设计一个简单的一元稀疏多项式加法运算器。

一元稀疏多项式简单计算器的基本功能包括：

1. 输入两个稀疏的一元n次多项式A与B，建立多项式的存储结构。
2. 计算多项式A与B的和，即建立一个和多项式A+B。

多项式相加的运算规则为：对于两个一元多项式中指数相同的子项，对应的系数相加，若和非零，将其构成和多项式中的一个子项；对于两个一元多项式中指数不同的子项，则分别复制到和多项式中去。

③输出多项式A、B以及A+B。

#### 三、实验目的和要求

**1．目的**

训练链表的应用，掌握链表的创建、结点的插入和删除、链表的输出等操作。

练习在文本界面下，采用类似于数学表达式的方式输出多项式。

**2．输入输出的要求：**

①输入多项式：

按照指数递增的顺序和指定的输入格式输入各个系数不为0的子项：“系数、指数对”，直至遇到输入结束标志（“系数、指数对”为“0、0”）的时候停止。

②输出多项式：

在文本界面下，采用类似于数学表达式的方式输出多项式。例如：

多项式A=3+6X3−2X8+12X20，可在屏幕显示为：A=3+6X∧3−2X∧8+12X∧20

格式要求：

(a) 系数值为1的非零次项的输出形式中略去系数1，如子项“1x8”的输出形式为“x8”，项“－1x3”的输出形式为“−x3”。

(b) 指数为0的项输出形式中略去指数部分，如“3x0”的输出形式应为“3”；指数为“1”的项输出时略去指数“1”。

(c) 多项式的第一项的系数符号为正时，不输出“+”，其它项要输出“+”、“−”符号。

#### 四、程序的框架结构

1．主要数据类型定义：

对于一元n次多项式P(x,n)=P0+P1X1+P2X2+…+PnXn而言，每一个子项都是由“系数”和“指数”两部分来组成的，因此可将多项式抽象成一个由“系数、指数对”构成的线性表，其中多项式的每一系数非零的子项作为线性表中的一个元素。

因此，定义一个带有头结点的单链表来表示一元n次多项式。

typedefstructpnode

{

intcoef; /\*系数域\*/

intexp; /\*指数域\*/

structpnode \*next; /\*指针域，指向下一个系数不为0的子项\*/

}PolyNode,\*PolyLink;

2．本程序包括以下四个模块：

①主程序模块：

main()

{

输入多项式A；

输入多项式B；

计算A与B的和多项式C；

输出和多项式C；

}

②输入模块：

PolyLinkinput\_polynm ( )

{

初始化带头结点的单链表（多项式链表）；

按指数递增的顺序，逐对输入多项式的各个子项，并插入到多项式链表中去；

}

③加法模块：

PolyLinkadd\_polynm (PolyLink A, PolyLink B)

{

对于两个多项式中指数相同的子项，其系数相加，若系数的和非零，则构成

“和多项式”中的一项；对于指数不同的项，直接构成“和多项式”中的一项。

}

④输出模块：

voidoutput\_polynm (PolyLink H)

{

从单链表H的第一个结点开始，逐个输出各结点所表示的子项的值。

}

3．各程序模块之间的层次关系。

主程序模块

加法模块m

输出模块

输入模块nm

#### 五、源代码

#include <stdio.h>

#include <stdlib.h>

typedefstructpnode

{

intcoef; /\*系数域\*/

intexp; /\*指数域\*/

structpnode \*next; /\*指针域，指向下一个系数不为0的子项\*/

}PolyNode,\*PolyLink;

int main() /\*主函数\*/

{

PolyLinkinput\_polynm();

PolyLinkadd\_polynm(PolyLink A, PolyLink B);

voidoutput\_polynm(PolyLink H);

PolyLink A,B,C; /\*单链表存储的多项式A、B、C\*/

printf("\n\n\t\t\*\*\*\*\*\*多项式加法运算\*\*\*\*\*\n");

printf("\n请输入多项式A：\n");

A=input\_polynm(); /\*输入多项式A\*/

printf("\n请输入多项式B：\n");

B=input\_polynm(); /\*输入多项式B\*/

printf("\n多项式:\t\tA = ");

output\_polynm(A); /\*输出多项式A\*/

printf("\n多项式:\t\tB = ");

output\_polynm(B);

C=add\_polynm (A, B); /\*计算多项式A与B的和多项式C\*/ printf("\n和多项式:\tC = ");

output\_polynm(C); /\*输出和多项式C\*/

return 0;

}

PolyLinkinput\_polynm() /\*输入多项式，创建多项式链表\*/

{

PolyLinkinit\_polylink();

int c, e;

PolyNode \*H, \*p, \*r;

H=init\_polylink(); /\*初始化多项式H\*/

r=H;

scanf("%d,%d",&c,&e);

while(!(c==0&&e==0))

{

p=(PolyNode \*)malloc(sizeof(PolyNode)); /\*申请结点\*/

p->coef=c;

p->exp=e;

p->next=NULL; /\*填装结点\*/

r->next=p; /\*将新结点插入到链表尾\*/

r=p;

scanf("%d,%d",&c,&e);

}

return(H);

}

PolyLinkadd\_polynm (PolyLink A, PolyLink B) /\*计算多项式A、B的和\*/

{

PolyLink C;

int x;

PolyNode \*pa,\*pb,\*rc,\*q;

C=A; /\*以A的头结点作为C的头结点\*/

rc=C; /\*rc指向C的尾结点\*/

pa=A->next; pb=B->next; /\*pa、pb指向A、B的第一个结点\*/

free(B); /\*释放B的头结点\*/

while(pa&&pb) /\*当pa、pb皆非空\*/

{

if(pa->exp<pb->exp)

/\*若pa所指结点的指数小于pb所指结点的指数，将pa插入到C的链表尾\*/

{

rc->next=pa;

rc=pa;

pa=pa->next;

}

else if(pa->exp>pb->exp)

/\*若pa所指结点的指数大于于pb所指结点的指数，将pb插入到C的链表尾\*/

{

rc->next=pb;

rc=pb;

pb=pb->next;

}

else /\*若pa所指结点指数等于pb所指结点指数\*/

{

x=pa->coef+pb->coef; /\*计算系数和\*/

if(x==0) /\*系数和为零\*/

{

q=pa;

pa=pa->next;

free(q);

q=pb;

pb=pb->next;

free(q);

}

else /\*系数和非零\*/

{

pa->coef+=pb->coef;

rc->next=pa;

rc=pa;

pa=pa->next;

q=pb;

pb=pb->next;

free(q);

}

}

}

rc->next=pa?pa:pb; /\*插入剩余段\*/

return(C);

}

void output\_polynm(PolyLink H) /\*输出多项式H\*/

{

PolyNode \*firstp,\*p;

if(H->next==NULL)

{

printf("0\n");

return;

} /\*多项式H为空，输出0\*/

firstp=H->next; /\*firstp指向多项式的第一项\*/

p=H->next;

while(p)

{

if(p!=firstp&&p->coef>0) printf("+"); /\*若p不是第一项，当系数为正时输出“+”号\*/

if(p->exp==0) printf("%d", p->coef); /\*指数为0\*/

else if(p->exp==1) /\*指数为1\*/

{

if(p->coef==1) printf("x");

else if(p->coef==-1) printf("-x");

elseprintf("%dx",p->coef);

}

else /\*其它情况\*/

{

if(p->coef!=1) printf("%d",p->coef);

printf("x^%d", p->exp);

}

p=p->next;

}

printf("\n");

}

PolyLinkinit\_polylink() /\*初始化一个空的多项式链表\*/

{

PolyNode \*H;

H=(PolyNode\*)malloc(sizeof(PolyNode));

H->next=NULL;

return(H);

}

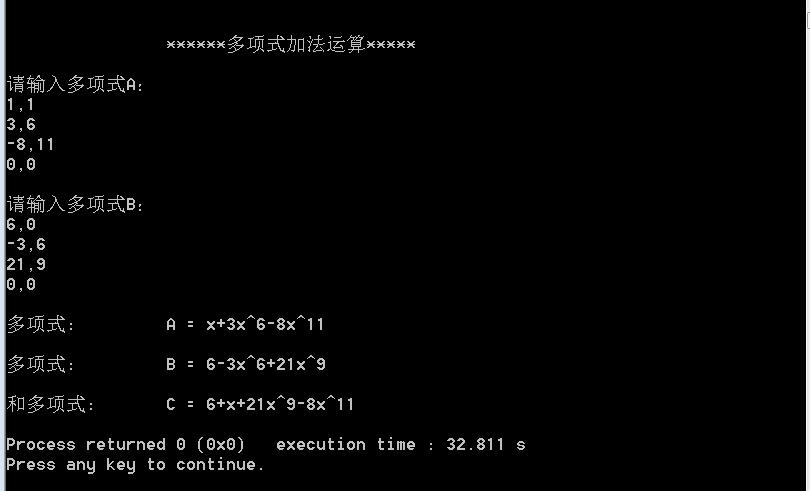
#### 六、实验结果

**1．使用说明：**

①输入多项式时，按指数递增的次序输入。

②指数和系数都为０时输入结束。

③程序运行界面如下（以第一个测试数据为例）：



**2．其他测试数据：**

①输入A：x+3x6－8x11

输入B：(6－3x6+21x9)

输出C：6+x+21x9－8x11

②输入A：3x－3－x+4x2－12x9)

输入B：―3x―3－5x2 +7x12

输出C：－x－x2－12x9+7x12

③输入A：x+x3

输入B：―x―x3

输出C：0

④输入A：x+x100

输入B：x100+x200

输出C：x+2x100+x200

⑤输入A：x+x2+x3

输入B：0

输出C：x+x2+x3

#### 七、遇到的问题及解决方法

**1、遇到的问题：**

①在输入多项式时，输入了实型的系数，忽略了程序中指定的类型是整型，从而产生了错误的输出结果。

②程序运行的结果出现了形如“0x^3”的项，不符合习惯。

③输出的多项式不符合数学上的表示方法。

**2、解决方法：**

①输入整型的系数。也可考虑修改程序中系数的类型，但因为时间原因，没有修改。

②在多项式加法函数中，对于两多项式中指数相等的项系数相加时，要考虑系数和为0的情况，此时应删除结点，不要将其作为和多项式中的一项。

③在输出函数的设计中，考虑以下几点：

多项式的第一项，系数为正时不显示“+”号，其他项要显示“+”、“-”号。

指数为0时，不输出指数部分。

指数为1时，不输出指数1。

系数为1时，不输出系数。

还要考虑以上几种综合的情况，如输出“1”，“-1”，“x”，“-x”的正确方法等。