数据结构实验报告

**实验一：一元多项式的加减**

姓 名： 彭 思 翔

学 号： 2017301500061

班 级： 计科二班

日 期：2018年10月21日

上机环境：Win10 VSCode

1. **程序设计相关信息**
2. **实验题目**

实验一：一元多项式的加减运算

【问题描述】

请编写一个程序，完成一元多项式的存储，并实现两个多项式的相加减及相乘运算。

【基本要求】

（1）随机从键盘上输入多项式，构建单链表存储医院多项式，其中单链表的数据域包括系数和指数两项

（2）最后的结果输出形式为result=3+4X^2+5X^2

**2．实验项目的目的**

深入理解单链表并进行应用，顺便进行字符串处理，学会合理地设置函数，使函数复用率高。

1. **实验项目程序结构**

printPolynomial

addPolynomial

subPolynomial

initPolynomial

init

main

mulPolynomial

insertPolynomial

readPolynomial

destoryPolynomial

1. **实验项目中各文件函数功能描述**

void init(); //读入数据，控制进程

void initPolynomial(Polynomial\*&); //初始化多项式单链表

void readPolynomial(Polynomial\*&); //读取多项式，进行字符串处理

void printPolynomial(Polynomial\*); //打印多项式

void destoryPolynomial(Polynomial\*&); //销毁多项式单链表

void insertPolynomial(Polynomial\*&, Polynomial\*&); //插入排序单式至多项式的正确位置

void addPolynomial(Polynomial\*&, Polynomial\*&, Polynomial\*&); //多项式加法，由第三个参数返回结果

void subPolynomial(Polynomial\*&, Polynomial\*&, Polynomial\*&); //多项式减法，由第三个参数返回结果

void mulPolynomial(Polynomial\*&, Polynomial\*&, Polynomial\*&); //多项式乘法，由第三个参数返回结果

1. **算法描述**

【数据结构】

有序单链表：用于存储和表示指数由高到低的多项式，链表中存储指数/系数和下一项的地址，只存储系数不为0的项。

【设计思路】

读取实现：对于一个多项式的字符串，以加号分割，加号后第一个为数字则为系数，否则系数为1,字符‘x’后若为‘^’则指数为其后面的数字，否则指数为1。对于数，遇见连续数字在小数点前则前面的数×10加上当前的数字，在小数点后则前面的数+当前的数字/小数位数。

时间复杂度O(Lenth)

插入实现：插入排序，从前至后找到比需要插入单式的指数更小的位置前一个为止，若指数相等则合并，若指数不相等则插入。

时间复杂度O(n)

加法实现：两个多项式归并，指数上若A[i]>B[j]则i++，若A[i]<B[j]则j++，若A[i]==B[j]则相加系数并i++j++。

时间复杂度O(m+n)

加法实现：两个多项式归并，指数上若A[i]>B[j]则i++，若A[i]<B[j]则j++，若A[i]==B[j]则相减系数并i++j++。

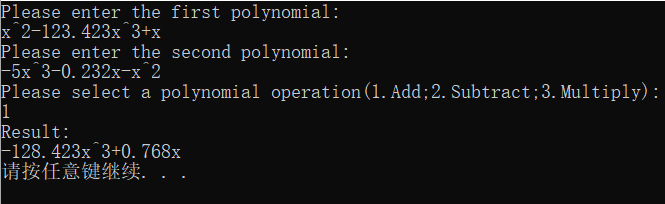
时间复杂度O(m+n)

乘法实现：暴力枚举两个多项式的每一位，并相乘，将结果项插入至新的多项式。

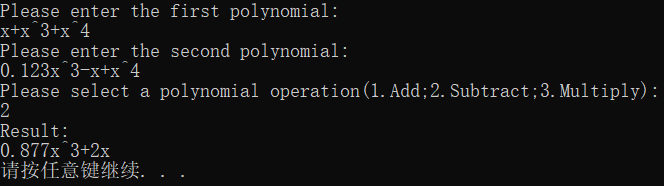
时间复杂度O(m\*n^2)

1. **实验数据和实验结果分析**

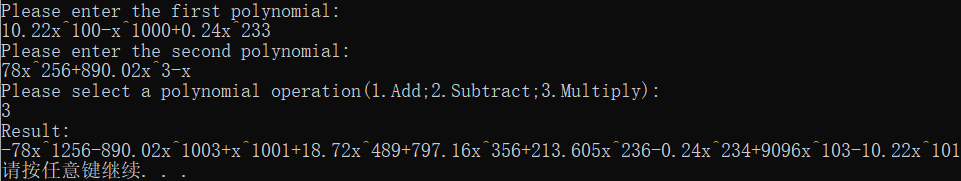
相加：



相减：



相乘：



运行结果良好。

1. **实验体会**

对单链表有了更加深入的认识，熟悉了二路归并等相关算法。运算的实现倒还算是简单，处理输入的字符串却废了不少功夫，比如系数指数省略，负数、小数的处理等，看来对字符串的操作要更熟悉一点才行。另外乘法运算过于暴力，暂时未想出以当前数据结构的更优解决方案。

**二． 源代码**

#include <cstdio>

#include <cstdlib>

using namespace std;

#define MaxLenth 100010

typedef struct Poly {

double coefficient, index;

struct Poly \* next;

}Polynomial;

void init(); //读入数据，控制进程

void initPolynomial(Polynomial\*&); //初始化多项式单链表

void readPolynomial(Polynomial\*&); //读取多项式，进行字符串处理

void printPolynomial(Polynomial\*); //打印多项式

void destoryPolynomial(Polynomial\*&); //销毁多项式单链表

void insertPolynomial(Polynomial\*&, Polynomial\*&); //插入单式至多项式的正确位置

void addPolynomial(Polynomial\*&, Polynomial\*&, Polynomial\*&); //多项式加法，由第三个参数返回结果

void subPolynomial(Polynomial\*&, Polynomial\*&, Polynomial\*&); //多项式减法，由第三个参数返回结果

void mulPolynomial(Polynomial\*&, Polynomial\*&, Polynomial\*&); //多项式乘法，由第三个参数返回结果

Polynomial \*polyA, \*polyB, \*polyC;

int main() {

init();

system("pause");

return 0;

}

void initPolynomial(Polynomial \*&L) {

L = (Polynomial \*)malloc(sizeof(Polynomial));

L->next = NULL;

L->coefficient = L->index = 0;

}

void destoryPolynomial(Polynomial \*&L) {

Polynomial \*pre = L, \*p = L->next;

while (p != NULL) {

free(pre);

pre = p;

p = p->next;

}

free(pre);

}

void insertPolynomial(Polynomial \*&L, Polynomial \*&x) {

Polynomial \*p = L;

while (p->next != NULL && p->next->index >= x->index)

p = p->next;

if (p->index == x->index && p != L) {

p->coefficient += x->coefficient;

free(x);

}

else

{

x->next = p->next;

p->next = x;

}

}

void printPolynomial(Polynomial \*L) {

Polynomial \*p = L->next;

if (p == NULL) {

puts("0");

return;

}

while (p != NULL) {

if (p != L->next && p->coefficient > 0) printf("+");

if (p->coefficient != 0) {

if (p->coefficient != 1 && p->coefficient != -1) printf("%g", p->coefficient);

if (p->coefficient == -1) printf("-");

if (p->index != 0)

if (p->index != 1) printf("x^%g", p->index);

else printf("x");

else if (p->coefficient == 1 || p->coefficient == -1) printf("1");

}

p = p->next;

}

puts("");

}

void readPolynomial(Polynomial \*&L) {

char poly[MaxLenth]={'\0'};

Polynomial \*p;

scanf("%s", poly);

for (int i = 0; poly[i]!='\0';) {

double c = 0, x = 0, f = 1, d = 1;

if (poly[i]== '-') i++, f=-1;

if (poly[i]== '+') i++, f=1;

if (poly[i] >= '0' && poly[i] <= '9') {

while (poly[i] >= '0' && poly[i] <= '9') {

c \*= 10;

c += poly[i] - '0';

i++;

}

if (poly[i] == '.') {

i++;

while (poly[i] >= '0' && poly[i] <= '9') {

d /= 10;

c += d \* (poly[i] - '0');

i++;

}

}

}

else c = 1;

if (poly[i] == 'x') {

i++;

if (poly[i]=='^') {

i++;

while (poly[i] >= '0' && poly[i] <= '9') {

x \*= 10;

x += poly[i] - '0';

i++;

}

}

else x = 1;

}

else x = 0;

if (c != 0) {

initPolynomial(p);

p->coefficient = c \* f;

p->index = x;

insertPolynomial(L, p);

}

}

}

void addPolynomial(Polynomial \*&A, Polynomial \*&B, Polynomial \*&C) {

Polynomial \*p = A->next, \*q = B->next, \*r = C, \*s;

while (p != NULL && q != NULL) {

if (p->index > q -> index) {

initPolynomial(s);

s->coefficient = p->coefficient;

s->index = p->index;

s->next = r->next;

r->next = s;

p = p->next;

r = r->next;

}

else if (p->index < q -> index) {

initPolynomial(s);

s->coefficient = q->coefficient;

s->index = q->index;

s->next = r->next;

r->next = s;

q = q->next;

r = r->next;

}

else if (p->index == q -> index) {

if (p->coefficient + q->coefficient != 0) {

initPolynomial(s);

s->coefficient = p->coefficient + q->coefficient;

s->index = q->index;

s->next = r->next;

r->next = s;

r = r->next;

}

p = p->next;

q = q->next;

}

}

while (p!=NULL) {

initPolynomial(s);

s->coefficient = p->coefficient;

s->index = p->index;

s->next = r->next;

r->next = s;

p = p->next;

r = r->next;

}

while (q!=NULL) {

initPolynomial(s);

s->coefficient = q->coefficient;

s->index = q->index;

s->next = r->next;

r->next = s;

q = q->next;

r = r->next;

}

}

void subPolynomial(Polynomial \*&A, Polynomial \*&B, Polynomial \*&C) {

Polynomial \*p = A->next, \*q = B->next, \*r = C, \*s;

while (p != NULL && q != NULL) {

if (p->index > q -> index) {

initPolynomial(s);

s->coefficient = p->coefficient;

s->index = p->index;

s->next = r->next;

r->next = s;

p = p->next;

r = r->next;

}

else if (p->index < q -> index) {

initPolynomial(s);

s->coefficient = -q->coefficient;

s->index = q->index;

s->next = r->next;

r->next = s;

q = q->next;

r = r->next;

}

else if (p->index == q -> index) {

if (p->coefficient - q->coefficient != 0) {

initPolynomial(s);

s->coefficient = p->coefficient - q->coefficient;

s->index = q->index;

s->next = r->next;

r->next = s;

r = r->next;

}

p = p->next;

q = q->next;

}

}

while (p!=NULL) {

initPolynomial(s);

s->coefficient = p->coefficient;

s->index = p->index;

s->next = r->next;

r->next = s;

p = p->next;

r = r->next;

}

while (q!=NULL) {

initPolynomial(s);

s->coefficient = q->coefficient;

s->index = q->index;

s->next = r->next;

r->next = s;

q = q->next;

r = r->next;

}

}

void mulPolynomial(Polynomial \*&A, Polynomial \*&B, Polynomial \*&C) {

Polynomial \*p = A->next, \*q = B->next, \*r = C, \*s;

while (p!=NULL) {

q = B->next;

while (q!=NULL) {

initPolynomial(s);

s->coefficient = p->coefficient \* q->coefficient;

s->index = p->index + q->index;

insertPolynomial(C, s);

q = q->next;

}

p = p->next;

}

}

void init() {

initPolynomial(polyA);

initPolynomial(polyB);

initPolynomial(polyC);

printf("Please enter the first polynomial:\n");

readPolynomial(polyA);

printf("Please enter the second polynomial:\n");

readPolynomial(polyB);

printf("Please select a polynomial operation(1.Add;2.Subtract;3.Multiply):\n");

int x;

scanf("%d", &x);

if (x == 1) addPolynomial(polyA, polyB, polyC);

else if (x == 2) subPolynomial(polyA, polyB, polyC);

else if (x == 3) mulPolynomial(polyA, polyB, polyC);

printf("Result:\n");

printPolynomial(polyC);

destoryPolynomial(polyA);

destoryPolynomial(polyB);

destoryPolynomial(polyC);

}