**一元稀疏多项式计算器**

题目：编制一个可以进行一元稀疏多项式计算的程序

姓名：王湘峰 学号：PB19030861

**一、需求分析**

一元稀疏多项式的功能如下：

1.输入并建立多项式

2.输出多项式，输出格式为整数序列：n,c1,e1,c2,e2……n代表多项式的项数，c代表系数，e代表指数。序列按指数降序排列。

3.多项式a和b相加，建立多项式a+b

4.多项式a和b相减，建立多项式a-b

**【测试数据】**

(x+x^100)+(x^100+x^200)=x+2x^100+x^200;

x+x^2+x^3+0=x+x^2+x^3;

x^3+x+(-x-x^3)=0

**二、概要设计**

本程序的实现需要两个数据类型：链表和数组。其中链表用于存储多项式的信息，而数组用于保存每个多项式的头结点，即保存所有生成的多项式。

抽象数据类型一元多项式的定义如下：

**ADT** Polynomial{

数据对象D={ai|ai∈Termset,i=1,23……}

数据关系R1={<ai-1,ai>|ai-1,ai∈D,且ai-1的指数小于ai}

基础操作：

CreatePoly(&P,m)

操作结果：输入m项的系数和指数，建立一元多项式P

DisplayPoly(P)

操作结果：打印输出一元多项式P

AddPoly(Pa,Pb,&Pc)

操作结果：将多项式Pa和Pb相加的结果存入Pc中

SubstractPoly(Pa,Pb,&Pc)

操作结果：将多项式Pa和Pb相减的结果存入Pc中

DeletePoly（n）

操作结果：把多项式数组中的第n项删除

}ADT Polynomial

**三、详细设计**

#include<stdio.h>

#include<malloc.h>

#define ERROR -1

#define OK 0

#define OVERFLOW -2

#define SIZE 20

typedef int Status;

typedef struct node{

float c;

int e;

struct node\* next;

}Node, \* Poly; //**结点类型**

Status Initlist(Poly& L);

void Createlist(Poly& L, int n);

int Display(Poly L);

void Polysub(Poly A, Poly B,Poly&C);

void Polyadd(Poly A, Poly B,Poly &C);

void Displayall(Node\* Polynomial[20]);

void Delete(Node\* A);

Node\* Polynomial[SIZE]; //**功能类型**

Status Initlist(Poly& L) { //**初始化一个带头结点的链表**

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

if (L == NULL) {

printf("error");

return ERROR;

}

L->next = NULL;

L->c = 32767;

L->e = 32767;

return OK;

}

void Createlist(Poly& L, int n) { //**输入创建一个多项式链表**

Initlist(L);

Node\* p, \* q;

q = L;

printf("input Polynomial\n");

for (int i = 1; i <= n; i++) {

q = L;

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

if (p) {

scanf("%f%d", &p->c, &p->e);

if (i == 1) {

p->next = q->next;

q->next = p;

}

else {

while (q->next) {

if (q->next->e > p->e)

q = q->next;

else break;

}

if (q->next) {

if (q->next->e == p->e) {

q->next->c = q->next->c + p->c;

}

else if (q->next->e < p->e) {

p->next = q->next;

q->next = p;

}

}

else {

p->next = q->next;

q->next = p;

q = p;

}

}

}

}

}

void Delete(Node\* A) { //**删除多项式所占的内存并释放空间**

Poly a, b;

a = A;

if (a) {

while (a->next) {

b = a->next;

a->next = b->next;

free(b);

}

free(a);

}

}

int Display(Poly L) { //**输出指定多项式并按照手写格式进行优化**

Node\* p;

int i=0;

p = L;

if (p) {

while (p->next) {

if (p->next->c) {

if (i) {

if (p->next->e == 0) {

if (p->next->c > 0) {

printf("+%f", p->next->c);

}

else {

printf("%f", p->next->c);

}

p = p->next;

i++;

}

else if (p->next->c == 1) {

if (p->next->e == 1)

printf("+x");

else

printf("+x^%d", p->next->e);

p = p->next;

i++;

}

else if (p->next->c == -1) {

if (p->next->e == 1) {

printf("-x");

}

else {

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

}

p = p->next;

i++;

}

else if (p->next->c > 0) {

if (p->next->e == 1) {

printf("+%fx", p->next->c);

}

else {

printf("+%fx^%d", p->next->c, p->next->e);

}

p = p->next;

i++;

}

else if (p->next->c < 0) {

if (p->next->e == 1) {

printf("%fx", p->next->c);

}

else {

printf("%fx^%d", p->next->c, p->next->e);

}

p = p->next;

i++;

}

}

else {

if (p->next->e == 0) {

printf("%f", p->next->c);

p = p->next;

i++;

}

else if (p->next->c == 1) {

if (p->next->e == 1)

printf("x");

else

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

p = p->next;

i++;

}

else if (p->next->c == -1) {

if (p->next->e == 1) {

printf("-x");

}

else {

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

}

p = p->next;

i++;

}

else if (p->next->c > 0) {

if (p->next->e == 1) {

printf("%fx", p->next->c);

}

else {

printf("%fx^%d", p->next->c, p->next->e);

}

p = p->next;

i++;

}

else if (p->next->c < 0) {

if (p->next->e == 1) {

printf("%fx", p->next->c);

}

else {

printf("%fx^%d", p->next->c, p->next->e);

}

p = p->next;

i++;

}

}

}

else {

p->next = p->next->next;

}

}

if (L->next == NULL) printf("0 1");

else printf(" %d", i);

}

else printf("poly not exist!\n");

return OK;

}

void Displayall(Node\* Polynomial[SIZE]) {

int i;

for (i = 0; Polynomial[i] != NULL; i++) {

Display(Polynomial[i]);

printf("\n");

}

}

void Polyadd(Poly A, Poly B, Poly& C) { //**进行多项式的加法**

Node\* pa;

Node\* pb;

Node\* pc;

Node\* t;

pa = A; pb = B; pc = C;

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

while (pa->next->e > pb->next->e) {

t = (Node\*)malloc(sizeof(Node));

t->next = pc->next;

t->c = pa->next->c;

t->e = pa->next->e;

pc->next = t;

pc = pc->next;

pa = pa->next;

}

if (pa->next->e == pb->next->e) {

t = (Node\*)malloc(sizeof(Node));

t->next = pc->next;

t->c = pa->next->c + pb->next->c;

t->e = pa->next->e;

pc->next = t;

pc = pc->next;

pa = pa->next;

pb = pb->next;

}

while (pa->next && pb->next && pa->next->e < pb->next->e) {

t = (Node\*)malloc(sizeof(Node));

t->c = pb->next->c;

t->e = pb->next->e;

t->next = pc->next;

pc->next = t;

pc = pc->next;

pb = pb->next;

}

}

while (pb->next) {

t = (Node\*)malloc(sizeof(Node));

t->c = pb->next->c;

t->e = pb->next->e;

t->next = pc->next;

pc->next = t;

pc = pc->next;

pb = pb->next;

}

while (pa->next) {

t = (Node\*)malloc(sizeof(Node));

t->c = pa->next->c;

t->e = pa->next->e;

t->next = pc->next;

pc->next = t;

pc = pc->next;

pa = pa->next;

}

Display(C);

printf("\n");

}

void Polysub(Poly A, Poly B,Poly &C) { //**进行多项式的减法**

**//由于核心算法与add相同，只有符号上的区别，故不再赘述**

int main() {

int i,j,n;

Node\* Polynomial[SIZE];

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

Polynomial[i] = NULL;

while (1){

printf("1 for create Polynomial\n2 for display\n3 for dispaly all\n4 for add\n5 for substract\n6 for delete\n0 for exit\n");

scanf("%d", &i);

if (i == 1) { **//对于用户的操作需求进行判断**

Poly L;

for (j = 0; Polynomial[j] != NULL; j++);

printf("input the number of poly\n");

scanf("%d", &n);

Initlist(L);

Createlist(L, n);

Polynomial[j] = L;

}

else if (i == 2) {

int n;

printf("display Poly x?\n");

scanf("%d", &n);

Display(Polynomial[n - 1]);

printf("\n");

}

else if (i == 3) {

Displayall(Polynomial);

}

else if (i == 4) {

printf("input two poly you want to add\n");

scanf("%d%d", &i, &j);

for (n = 0; Polynomial[n] != NULL; n++);

Initlist(Polynomial[n]);

Polyadd(Polynomial[i - 1], Polynomial[j - 1], Polynomial[n]);

}

else if (i == 5) {

printf("input two poly you want to substract\n");

scanf("%d%d", &i, &j);

for (n = 0; Polynomial[n] != NULL; n++);

Initlist(Polynomial[n]);

Polysub(Polynomial[i - 1], Polynomial[j - 1], Polynomial[n]);

}

else if (i == 6) {

printf("which poly do you want to delete?\n");

scanf("%d", &j);

Delete(Polynomial[j - 1]);

for (n = j - 1; n < SIZE-1; n++) {

Polynomial[n] = Polynomial[n + 1];

}

Polynomial[SIZE-1] = NULL;

}

else return OK;

}

return OK;

}

**四、调试分析**

1.起初由于忽视了一些变量参数的标识&，使程序出现了部分bug。今后应重视确定参数的变量以及赋值属性的区分与标识。

2.在编写多项式的减法子函数的过程中，采用的是复制加法函数，并把系数的符号变相反，这样做效率比较低下，并且容易出现错误。更高效稳定的做法是将被减多项式变成相反，然后将其与减数多项式相加。

3.本程序的数组的设置比较合理，一来方便了查找，二来又便捷了对已使用过的多项式进行计算。

4.算法的时间复杂度

CreatePoly、DisplayPoly、DeletePoly函数的时间复杂度均为O(n)，n为多项式的项数。InitPoly的时间复杂度为O(1)；Polyadd、Polysubstract的时间复杂度为O(lengthA+lengthB).

本次实验采用数据抽象的程序设计方法，将程序划分为四个层次：元素结点、有序链表、有序表和主控模板，使得程序设计思路清晰，实现时调试顺利，确实得到了一次良好的程序设计训练。

**五、测试结果**

x^2+x-x-x^2=0

1+x+x^100+0=1+x+x^100