#include <stdio.h>

#include <stdlib.h>

typedef struct PolyNode\* Polynomial;

struct PolyNode {

int coef; // 系数

int expon; // 指数

Polynomial next; //指向下一个node

};

Polynomial CreatePoly() {

Polynomial p, tail, temp; // 设置tail尾节点方便插入新项

int coef, expon;

int flag = 0; // flag用于辅助判断是否是第一个节点

p = (Polynomial)malloc(sizeof(struct PolyNode));

p->next = NULL;

tail = p;

scanf("%d %d", &coef, &expon);

while (expon != -1) { // 输入-1时结束多项式的创建

if (flag == 0 && coef == 0) { // 若第一个node系数为0，则退出循环

break;

}

temp = (Polynomial)malloc(sizeof(struct PolyNode));

temp->coef = coef;

temp->expon = expon;

temp->next = NULL;

tail->next = temp;

tail = temp;

flag = 1; //链表的第一个节点已经安置好，flag设置为1

scanf("%d %d", &coef, &expon); // 继续读取下一个node的系数与指数

}

return p;

}

void PrintPoly(Polynomial P) {

if (!P->next) { // 若链表为空，输出0

printf("0\n");

return;

}

int flag = 0; // 判断是否是第一个非零项

while (P->next) {

if (flag && P->next->coef > 0) {

printf("+"); // 系数为正时输出 "+" 符号

}

if (P->next->coef == 1) { //情况1：系数为1 省略掉1 避免输出如：1x^2的项

if (P->next->expon == 0) { //指数为0 仅输出系数

printf("%d", P->next->coef);

}

else if (P->next->expon == 1) {

printf("%dx", P->next->coef);

}

else {

printf("x^%d",P->next->expon);

}

}

else //其余情况正常处理

{

if (P->next->expon == 0) {

printf("%d", P->next->coef);

}

else if (P->next->expon == 1) {

printf("%dx", P->next->coef);

}

else {

printf("%dx^%d", P->next->coef, P->next->expon);

}

}

// 合并同类项

if (P->next->next && P->next->expon == P->next->next->expon) {

Polynomial temp = P->next->next;

while (temp && temp->expon == P->next->expon) {//循环向下遍历是否有同类项

P->next->coef += temp->coef;

temp = temp->next;

}

}

P = P->next;

flag = 1;

}

printf("\n");

}

Polynomial AddPoly(Polynomial p1, Polynomial p2) {

if (p1->next == NULL && p2->next == NULL) { //链表为空的情况

return NULL;

}

Polynomial front, tail, temp, t1, t2, t;

int sum;

t1 = p1->next; // 多项式1的第一个非零项

t2 = p2->next; // 多项式2的第一个非零项

front = (Polynomial)malloc(sizeof(struct PolyNode)); // 创建结果多项式的头结点

front->next = NULL;

tail = front;

while (t1 && t2) {

if (t1->expon < t2->expon) { //情况1：多项式1的项的指数 < 多项式2的项的指数 把指数小的往后移，输出时结果更美观

temp = (Polynomial)malloc(sizeof(struct PolyNode));

temp->coef = t1->coef;

temp->expon = t1->expon;

temp->next = NULL;

tail->next = temp;

tail = temp;

t1 = t1->next;

}

else if (t1->expon > t2->expon) {//情况2：多项式1的项的指数 > 多项式2的项的指数 把指数小的往后移，输出时结果更美观

temp = (Polynomial)malloc(sizeof(struct PolyNode));

temp->coef = t2->coef;

temp->expon = t2->expon;

temp->next = NULL;

tail->next = temp;

tail = temp;

t2 = t2->next;

}

else {// 情况3：多项式1的项的指数 = 多项式2的项的指数

sum = t1->coef + t2->coef; //指数相同，系数相加

if (sum != 0) {

temp = (Polynomial)malloc(sizeof(struct PolyNode));

temp->coef = sum;

temp->expon = t1->expon;

temp->next = NULL;

tail->next = temp;

tail = temp;

}

t1 = t1->next;

t2 = t2->next;

}

}

while (t1) {//找多项式1多出来的的项

temp = (Polynomial)malloc(sizeof(struct PolyNode));

temp->coef = t1->coef;

temp->expon = t1->expon;

temp->next = NULL;

tail->next = temp;

tail = temp;

t1 = t1->next;

}

while (t2) {//找多项式2多出来的的项

temp = (Polynomial)malloc(sizeof(struct PolyNode));

temp->coef = t2->coef;

temp->expon = t2->expon;

temp->next = NULL;

tail->next = temp;

tail = temp;

t2 = t2->next;

}

temp = front; // 释放头结点

front = front->next;

free(temp);

return front;

}

Polynomial MultiPoly(Polynomial p1, Polynomial p2) {

if (p1->next == NULL || p2->next == NULL) { //乘法运算只要有一个多项式为空 则无法进行 不同于多项式加法

return NULL;

}

Polynomial front, tail, temp, t1, t2, t, p;

int coef, expon;

t1 = p1->next; // 多项式1的第一个非零项

t2 = p2->next; // 多项式2的第一个非零项

front = (Polynomial)malloc(sizeof(struct PolyNode)); // 创建结果多项式的头结点

front->next = NULL;

tail = front;

while (t2) { //多项式2中的每一项与多项式中的第一项相乘

temp = (Polynomial)malloc(sizeof(struct PolyNode));

temp->coef = t1->coef \* t2->coef;

temp->expon = t1->expon + t2->expon;

temp->next = NULL;

tail->next = temp;

tail = temp;

t2 = t2->next;

}

t1 = t1->next; //从多项式1的第二项开始

while (t1) {

t = front;

t2 = p2->next;

tail = front;

while (t2) {

expon = t1->expon + t2->expon;//指数相加

coef = t1->coef \* t2->coef;//系数相乘

p = front;

while (p->next && p->next->expon >= expon) { // 找到正确的插入位置，并合并同类项

if (p->next->expon == expon) {

p->next->coef += coef;

if (p->next->coef == 0) { // 若系数和为0，则删除该节点

temp = p->next;

p->next = temp->next;

free(temp);

}

break;

}

p = p->next;

}

if (!p->next || p->next->expon < expon) { // 插入新节点(结果多项式链表没位置了or有新的指数)

temp = (Polynomial)malloc(sizeof(struct PolyNode));

temp->coef = coef;

temp->expon = expon;

temp->next = p->next;

p->next = temp;

}

t2 = t2->next;

}

t1 = t1->next;

}

return front;

}

void DestroyPoly(Polynomial p) {//释放多项式链表内存

Polynomial temp;

while (p) {

temp = p;

p = p->next;

free(temp);

}

}