#include<stdio.h>

typedef struct {

int coef;

int expn;

}ElemType;

typedef struct LNode {

ElemType data;

struct LNode\* next;

}\*Link, \*Position;

typedef struct {

Link head, tail;//指向头结点和尾结点

int len;//线性表元素个数

}Linklist;

typedef Linklist polyn;

void InitList(Linklist &L) {

L.head = new LNode; L.head->next = NULL;

L.tail = L.head; L.len = 0;

}

Position GetHead(Linklist L) {

return L.head;

}

Position GetLast(Linklist L) {

return L.tail;

}

int cmp(ElemType a, ElemType b) {

if (a.expn < b.expn)return -1;

else if (a.expn == b.expn)return 0;

else return 1;

}

void InsFirst(Link h, Link s) {//h指向头结点，将s所指结点插入第一个结点之前

s->next = h->next;

h->next = s;

}

void DelFirst(Link h, Link &q) {//h指向头结点，删除链表第一个结点，并以q返回

q = h->next;

h->next = q->next;

}

void Append(Linklist &L, Link s) {//将指针s所指（彼此以指针相链）的一串结点接在线性表L的最后一个结点之后，并改变链表L的尾指针指向新的尾结点

L.tail->next = s;

while (s) {

s = s->next;

}

L.tail = s;

}

Position NextPos(Linklist L, Link p) {//已知p指向线性表L中的一个结点，返回p所指位置的直接后继位置，若无，则返回NULL

return p->next;

}

void FreeNode(Link &p) {//释放p所指的结点

delete p;

}

void MakeNode(Link &p, ElemType e) {//分配一个由p指向的值为e的结点

p = new LNode;

p->data = e;

}

ElemType GetCurElem(Link p) {//返回p所指结点的值

return p->data;

}

void SetCurElem(Link &p, ElemType e) {//用e更新p所指结点的值

p->data = e;

}

void nLen\_of\_File(int &n1, int &n2, const char\* path) {

FILE \*fp = nullptr;

fopen\_s(&fp, path, "rt");

char\* p = nullptr;

int nCount = 0; char ch;

while (ch = fgetc(fp) != '.') {

nCount++;

}

n1 = nCount;

nCount = 0;

while (ch = fgetc(fp) != EOF) {

nCount++;

}

n2 = nCount;

fclose(fp);

}

void CreatPolyn(polyn &P, int n1, int n2, const char \*path) {

InitList(P); Position head = GetHead(P);

Position current;

FILE \*fp = nullptr;

fopen\_s(&fp, path, "rt");

ElemType e;

for (int i = n1; i >0; i--) {

e.coef = fgetc(fp) - '0'; e.expn = -i;

current = new LNode; MakeNode(current, e);

current->next = NULL;

P.tail->next = current;

P.tail = current;

}

e.coef = fgetc(fp);

for (int i = 0; i < n2; i++) {

e.coef = fgetc(fp) - '0'; e.expn = i;

current = new LNode; MakeNode(current, e);

current->next = NULL;

P.tail->next = current;

P.tail = current;

}

fclose(fp);

P.len = n1 + n2;

}

void DestroyPolyn(polyn &P) {

Position ha = GetHead(P), qa = NextPos(P, ha);

while (qa) {

FreeNode(ha); ha = qa;

qa = NextPos(P, ha);

}

FreeNode(ha);

}

bool ListEmpty(Linklist L) {

if (L.head->next == NULL)return true;

return false;

}

void CarryBit(polyn &p) {//进位

Position h = GetHead(p);

int temp1, temp2;

Position q = NextPos(p, h);

while (q != NULL) {

temp1 = q->data.coef;

q->data.coef %= 10;

temp2 = temp1 / 10;

q = q->next;

if (q != NULL) q->data.coef += temp2;

}

if (temp2 != 0) {

p.tail->next = new LNode;

int expn = p.tail->data.expn;

p.tail = p.tail->next;

p.tail->data.coef = temp2;

p.tail->data.expn = expn + 1;

p.tail->next = NULL;

}

}

void AddPolyn(polyn &Pa, polyn &Pb) {

//多项式加法：Pa=Pa+Pb，利用两个多项式的结点构成“和多项式”

Position ha = GetHead(Pa), hb = GetHead(Pb);

Position qa = NextPos(Pa, ha), qb = NextPos(Pb, hb);

while (qa&&qb) {

ElemType a = GetCurElem(qa), b = GetCurElem(qb);

switch (cmp(a, b)) {

case -1:

ha = qa; qa = NextPos(Pa, qa); break;

case 0:

ElemType sum;

sum.coef = a.coef + b.coef; sum.expn = a.expn;

SetCurElem(qa, sum); ha = qa;

DelFirst(hb, qb); FreeNode(qb); qb = NextPos(Pb, hb); qa = NextPos(Pa, ha);

break;

case 1:

DelFirst(hb, qb); InsFirst(ha, qb);

qb = NextPos(Pb, hb); ha = NextPos(Pa, ha); break;

}

}

if (!ListEmpty(Pb)) { Append(Pa, qb); Pa.tail = Pb.tail; }

CarryBit(Pa);

FreeNode(hb);

}

void MultiplyPolyn(polyn &Pa, polyn &Pb) {

Position ha = GetHead(Pa), hb = GetHead(Pb);

polyn result; InitList(result);

Position qb = NextPos(Pb, hb);

for (int i = 0; i < Pb.len; i++) {

Position qa = NextPos(Pa, ha);

polyn temp; InitList(temp);

Position ht = GetHead(temp);

for (int j = 0; j < Pa.len; j++) {

Link qt = new LNode;

ElemType e;

ElemType ea = GetCurElem(qa), eb = GetCurElem(qb);

e.coef = ea.coef \* eb.coef; e.expn = ea.expn + eb.expn;

MakeNode(qt, e); InsFirst(ht, qt); ht = qt;

temp.tail = qt;

qa = NextPos(Pa, qa);

}

AddPolyn(result, temp);

qb = NextPos(Pb, qb);

}

CarryBit(result);

DestroyPolyn(Pa); DestroyPolyn(Pb);

Pa = result;

}

void main() {

char\* path1 = "infile1.txt"; char\* path2 = "infile2.txt"; char\* path3 = "outfile.txt";

int alen1, alen2, blen1, blen2;

nLen\_of\_File(alen1, alen2, path1); nLen\_of\_File(blen1, blen2, path2);

polyn Pa, Pb;

CreatPolyn(Pa, alen1, alen2, path1); CreatPolyn(Pb, blen1, blen2, path2);

AddPolyn(Pa, Pb);

FILE\* fp = nullptr;

fopen\_s(&fp, path3, "wt");

Position ha = GetHead(Pa);

Position qa = NextPos(Pa, ha);

fprintf\_s(fp, "pa + pb = ");

while (qa) {

fprintf\_s(fp, "%d", qa->data.coef);

if (qa->data.expn == -1)fprintf\_s(fp, ".");

qa = qa->next;

}

fprintf\_s(fp, "\n");

CreatPolyn(Pa, alen1, alen2, path1); CreatPolyn(Pb, blen1, blen2, path2);

MultiplyPolyn(Pa, Pb);

ha = GetHead(Pa);

qa = NextPos(Pa, ha);

fprintf\_s(fp, "pa \* pb = ");

while (qa) {

fprintf\_s(fp, "%d", qa->data.coef);

if (qa->data.expn == -1)fprintf\_s(fp, ".");

qa = qa->next;

}

printf\_s("Succeeded!\n");

fclose(fp);

DestroyPolyn(Pa);

}

心得：

建立在整数加法乘法的基础上，将代码稍加修改。

将小数点前后分别计数，然后保存在一个链表中。输出时，在指数为-1的地方输入’.’即可。