**第一部分 第1.2题：一元稀疏多项式计算器**

**实验报告**

题目：设计一个一元稀疏多项式简单计算器。

班级：F1702128 姓名：赵伟基 学号：517021910883 完成日期：2018-10-24

1. **需求分析**
2. 本实验要求输入端口输入的多项式包括多项式项数，以及各项的系数和指数，系数和指数之间需要用空格字符隔开。输入不允许包含非法字符。输入格式如下：

n

c1 e1

c2 e2

…

cn en

其中 n 是多项式的项数，ci 和 ei分别是第 i 项的系数和指数。

1. 多项式输出按照指数降序排列，输出形式为类数学表达式。具体要求为：省略系数和x之间的乘号。系数为±1 的非零次项的输出形式中略去1。
2. 多项式a和b相加（减），建立多项式 a+b（a-b），该多项式a和b的乘积记为ab。程序中通过类成员函数进行上述操作。
3. 程序执行的命令应包括：
4. 构建多项式a；（2）构建多项式b； （3）求多项式之和a+b；

（4） 求多项式之差a-b； （5）求多项式a的导数；

（6） 求多项式之积ab； （7）结束。

5. 测试数据

(1）(2x+5x8-3.1x11)+(7-5x8+11x9) = (-3.1x11+11x9+2x+7)

（2）(6x-3-x+4.4x2-1.2x9)-(-6x-3+5.4x2-x2+7.8x15)=

(-7.8x15-1.2x9+12x-3-x)

（3）(1+x+x2+x3+x4+x5)+(-x3-x4) = (1+x+x2+x5)

（4）(x+x3)+(-x-x3) = 0

（5）(x+x100)+(x100+x200) = (x+2x100+x200)

（6）(x+x2+x3)+(0) = (x+x2+x3)

（7）互换上述测试数据中的前后两个多项式

1. **概要设计**

多项式（polynomial）是指由变量、系数以及它们之间的加、减、乘、幂运算（非负整数次方）得到的表达式。多项式的每一项具有一下特征：（1）确定性：多项式的每一项系数和指数必须是确定的。（2）无序性：多项式各项之间么有先后要求。

题目要求实现多项式的加、减、乘法以及求导运算，这些运算都涉及到项的构建与删除，为方便项的删除和构建，选择用带头节点的单链表来储存多项式。

1. 程序模块
2. 主程序模块

int main()

{

构造多项式链表exp1,exp2;

调用add函数构建exp3=exp1+exp2；

调用sub函数构建exp4=exp1-exp2;

调用multi函数构建exp5=exp1\*exp2;

调用der函数构建exp6=exp1’;

打印；

清理；

}

1. 结点结构模块：存储多项式各项系数指数；
2. 运算函数模块：
   1. add( ) 加法实现函数：

初始条件：输入两个已存在多项式链表的头节点地址；

操作结果：构造出一个多项式链表，并将输入的多项式列表对应项相加，并将对应项的系数和指数填入该新建多项式链表，返回新构建的链表头节点地址。

* 1. sub( ) 减法实现函数

初始条件：输入两个已存在多项式链表头节点地址；前为被减项，后为减项

操作结果：构造出一个多项式链表，并将输入的多项式列表对应项相减，并将对应项的系数和指数填入该新建多项式链表，返回新构建的链表头节点地址。

* 1. multi( ) 乘法实现函数

初始条件：输入两个已存在多项式链表头节点地址；

操作结果：构造出一个多项式链表，并将输入的多项式列表对应项相乘并合并同类项，并将对应项的系数和指数填入该新建多项式链表，返回新构建的链表头节点地址。

* 1. der( ) 求导函数模块

初始条件：输入一个已存在多项式链表头节点地址；

操作结果：构造出一个多项式链表，并将输入的多项式进行求导，并将导后对应项的系数和指数填入该新建多项式链表，返回新构建的链表头节点地址。

1. 功能类型函数：
   1. create()

初始条件：多项式项数确定；

操作结果：构建多项式，用户与程序的交互，输入多项式项数与各项系数和指数。

* 1. clear\_list()

初始条件：输入已存在多项式链表头节点；

操作结果：链表空间被释放，防止内存泄漏。

* 1. view()

初始条件：输入已存在多项式列表头节点；

操作结果：打印多项式。

1. **详细设计**
2. Node.h

#ifndef node\_h

#define node\_h

#include <iostream>

using namespace std;

class Node //结点类

{

public:

double coef; //各项系数

int exp; //各项指数

Node \*next;

Node(){next=NULL;} //默认构造函数

Node(double n1,int n2,Node\*p=NULL):coef(n1),exp(n2),next(p){}

};

#endif

1. main.cpp

#include "Node.h"

#include <iostream>

Node\* create(int n); //创建列表

Node\* add(Node\*exp1,Node\* exp2); //加法函数

void view(Node\*exp); //输出函数

void clear\_list(Node\*exp); //清理函数

Node\* sub(Node\*exp1,Node\* exp2); //减法函数

Node\* der(Node\*exp1); //导数函数

Node\* multi(Node\*exp1,Node\* exp2); //乘法函数

int main()

{

Node\* exp1,\*exp2,\*exp3,\*exp4,\*exp5,\*exp6;

Node\* tmp;

int n1;

int n2;

cin>>n1;

exp1=create(n1);

cin>>n2;

exp2=create(n2);

exp3=add(exp1,exp2);

exp4=sub(exp1,exp2);

exp5=multi(exp1,exp2);

exp6=der(exp1);

view(exp3);cout<<" ";

view(exp4);cout<<" ";

view(exp5);cout<<" ";

view(exp6);

clear\_list(exp1);

clear\_list(exp2);

clear\_list(exp3);

clear\_list(exp4);

clear\_list(exp5);

clear\_list(exp6);

return 0;

}

1. Node.cpp

#include "Node.h"

#include <iostream>

Node\* create(int n) //输入数据 构建链表

{

Node\*p ,\*res,\*q;

double coef=1;

int expn;

q=res=p=new Node;

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

{

p->next=new Node;

p=p->next;

cin>>coef>>expn;

p->coef=coef;

p->exp=expn;

}

p->next=NULL;

for(p=res->next;p!=NULL;p=p->next) //检索同类项并合并

for(q=p->next;q!=NULL;q=q->next)

{

if(p->exp==q->exp)

{p->coef=p->coef+q->coef;

p->next=q->next;

delete q;

q=p;

}

}

return res;

};

Node\* add(Node\*exp1,Node\* exp2) //链表加法

{

Node \*res,\*p,\*tmp;

res=p=new Node() ;

exp1=exp1->next;

exp2=exp2->next;

while(exp1!=NULL&&exp2!=NULL){ //链表归并

if((exp1->exp==exp2->exp)&&(exp1->coef+exp2->coef==0))

{exp1=exp1->next;

exp2=exp2->next;

} //相加完系数为零直接跳过

else if(exp1->coef==0&&exp1->exp==0) exp1=exp1->next；

else if(exp2->coef==0&&exp2->exp==0) exp2=exp2->next;

//零项跳过

else

{

if(exp1->exp<exp2->exp) // 复制该项exp1的值

{

p->next=new Node(exp1->coef,exp1->exp);

exp1=exp1->next;

}

else if(exp1->exp>exp2->exp){ // 复制该项exp2的值

p->next=new Node(exp2->coef,exp2->exp);

exp2=exp2->next;

}

else if (exp1->coef+exp2->coef!=0) //归并同次项

{p->next=new Node(exp1->coef+exp2->coef,exp2->exp);

exp1=exp1->next;

exp2=exp2->next;}

p=p->next;

}

}

if(exp1==NULL) tmp=exp2; //将尚未结束的表达式归入结果表达式

else tmp=exp1;

while (tmp!=NULL){

p->next=new Node(tmp->coef,tmp->exp);

tmp=tmp->next;

p=p->next;

}

return res;

};

void view(Node\*p1) //排序，打印

{

Node\*p,\*q;

q=p=p1->next;

int expn1;

double coef1;

//冒泡排序

for(p = p1 -> next; p != NULL; p = p -> next)

for(q = p -> next; q != NULL; q = q -> next)

if((p->exp) < (q -> exp)){

int s = q -> exp; q -> exp = p -> exp; p -> exp = s;

double z = q->coef;q->coef=p->coef;p->coef=z;}

p=p1->next;

while (p!=NULL) //规范打印

{

if(p->exp!=0&&p->exp!=1&&p->coef!=1&&p->coef!=-1)

cout<<p->coef<<"x^"<<p->exp; // ax^b

else if(p->exp==0&&p->coef!=0) cout<<p->coef; //a

else if((p->exp==1)&&(p->coef!=1)&&(p->coef!=- 1))

cout<<p->coef<<"x"; //ax

else if(p->coef==1&&p->exp!=1) cout<<"x^"<<p->exp; //x^b

else if(p->coef==1&&p->exp==1) cout<<"x"; //x

else if(p->coef==-1&&p->exp!=1) cout<<"-x^"<<p->exp; //-x^b

else if(p->coef==-1&&p->exp==1) cout<<"-x"; //-x

p=p->next;

if (p&&p->coef>0)

cout<<"+"; //加号规范

}

if((p1->next==NULL)||(p1->next->next==NULL&&p1->next->coef==0)) cout<<0; //若未构造或全项为空 则直接输出0

}

void clear\_list(Node\*exp) //释放空间

{

Node\*tmp;

while(exp)

{

tmp=exp->next;

delete exp;

exp=tmp;

}

}

Node\* sub(Node\*exp1,Node\* exp2) //减法函数

{

Node\*p,\*res,\*q;

double coef=1;

int expn;

res=p=new Node;

for(q=exp2->next;q!=NULL;q=q->next)

//减法逻辑：令减项系数变为相反数，再调用加法函数

{

p->next=new Node;

p=p->next;

p->coef=-q->coef;

p->exp=q->exp;

}

p->next=NULL;

p= add(exp1,res);

clear\_list(res);

return p;

}

Node\* der(Node\*exp1)

{

Node\*p,\*res,\*q,\*tmp;

double coef=1;

int expn;

res=p=new Node;

for(q=exp1->next;q!=NULL;q=q->next)

{

p->next=new Node;

p=p->next;

p->coef=q->coef;

p->exp=q->exp;

}

p->next=NULL;

for(p=res->next;p!=NULL;p=p->next)

{

if(p->exp!=0)

{

p->coef=p->coef\*p->exp;

p->exp=p->exp-1;

}

else {

p->coef=0; //常数项标记为0

p->exp=0;

}

}

return res;

}

Node\* multi(Node\*exp1,Node\* exp2)

{

Node \*res,\*p,\*q,\*tmp1,\*tmp2;

res=p=new Node();

for(tmp1=exp1->next;tmp1!=NULL;tmp1=tmp1->next)

//利用循环使两个多项式项项相乘

{

if(tmp1->coef!=0)

for(tmp2=exp2->next;tmp2!=NULL;tmp2=tmp2->next)

{

if(tmp2->coef!=0)

{

p->next=new Node;

p=p->next;

p->coef=tmp1->coef\*tmp2->coef;

p->exp=tmp1->exp+tmp2->exp;

}

else continue;

}

else continue;

}

for(p=res->next;p!=NULL;p=p->next) //合并同类项

for(q=p->next;q!=NULL;q=q->next)

{

if(p->exp==q->exp)

{p->coef=p->coef+q->coef;

p->next=q->next;

delete q;

q=p;

}

}

return res;

}

1. **调试分析**
2. 该程序没有应用课本中链表类作为基类来构建新的多项式类，对多项式类的包装不成熟，导致程序中多项式的数据安全没有保证。
3. 未对运算符进行重载，也未重新定义析构函数，导致主函数内容臃肿。
4. 一开始未对view函数进行详尽的讨论，导致一些0项无法显示。
5. 算法复杂度分析

（1）：时间复杂度

由于采取带头节点的链表表示多项式，所以各算法的操作都比较简洁。

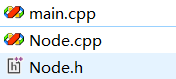
其中add(),sub(), view(),clear\_list(),der()需要遍历链表，复杂度为O(n); create(),multi()需要检索重复项，所以复杂度为O（n^2）;

（2）：空间复杂度

在add(),sub(),der(),create(),multi()都需要构建新链表。其中multi()的空间复杂度为O(n^2),其余的复杂度均为O(n)。

1. **用户手册**

1．本程序使用的 Code::Blocks 16.01 IDE，程序以项目（project）方式组织，如图 1 所示



2．依次点击菜单“Build” -> “Build and run”;键入多项式项数及各项系数和指数，便可进行加法运算。如果需要其他运算，在main函数中调用对应sub（），der（），multi（）即可。按下回车后即可输出相应结果。

1. **测试结果**
2. 调用add函数

输入：

3

2 1

5 8

-3.1 11

3

7 0

-5 8

11 9

输出：-3.1x^11+11x^9+2x+7

1. 调用sub函数

输入：

4

6 -3

-1 1

4.4 2

-1.2 9

4

-6 -3

5.4 2

-1 2

7.8 15

输出：-7.8x^15-1.2x^9-x+12x^-3

1. 输入：

6

1 0

1 1

1 2

1 3

1 4

1 5

2

-1 3

-1 4

输出：x^5+x^2+x+1

1. 输入：

2

1 1

1 3

2

-1 1

-1 3

输出：0

1. 输入：2

1 1

1 100

2

1 100

1 10000

输出：x^10000+2x^100+x

1. 输入

3

1 1

1 2

1 3

1

0 0

输出：x^3+x^2+x

1. **附录**

Node.h //节点结构单元

Node.cpp //计算函数与功能函数实现模块

Main.cpp //主函数 执行函数