湖南大學

数据结构

课程实验报告

题　　目：线性表的应用

学生姓名　杨兰馨

学生学号　201708020305

专业班级　通信 1703 班

完成日期　2019/4/20

一、 需求分析

问题描述

在数学上，一个一元 n 次多项式 Pn(x)可按降序写成：

$$Pn(x)=PnX^n + Pn-1X^{n-1}+...+p1X+p0$$

它是由 n＋1 个系数唯一确定。因此，在计算机里它可以用一个线性表 P 来表示：

P=（Pn, Pn-1, …, P1, Po）

一元多项式的运算包括加法、减法和乘法，而多项式的减法和乘法都可以用加法来实现。本实验要基于线性表设计和实现两个一元多项式的加法运算。要用有序链表表示一元多项式。表内元素按照多项式的次数递减的次序排列。

1. 问题分析

(1) 设计一个有序链表 ADT，将输入数据按照幂次从高到底存储；

(2) 实现两个多项式的相加，对相同幂指数的项进行合并，对应的系数相加做和。并将相加得到的新表达式存储在一个新的有序链表中。

(3) 对有序链表中的表达式进行格式化处理，处理当系数为 1，-1，幂指数为 0，1 这些特殊情况，格式化后，对有序链表进行遍历输出。

2. 输入数据

exe 程序运行文件中有提示输入语句，总共要输入 A(X)和 B(X)两个多项式，输入格式如下：

（1） 每行输入两个数字，第一个表示幂次 index，第二个为该幂次的系数 coef。以空格分离。

（2） 以 0 0 结束每个多项式的输入

（3） 由题目中的要求，最小幂次为 0 次，所以要求 index>=0;

（4） 要求输入的系数不能为 0，即 coef != 0；

（5） 输入的顺序不一定要按照幂次由高到低排列，任意排列均可。

3. 输出数据

总共有三个输出：

（1） 在存储了 A(X)多项式之后，输出 A(X)多项式的内容；

（2） 在存储了 B(X)多项式之后，输出 B(X)多项式的内容；

（3） 将两个多项式相加之后，输出作和的多项式的内容；

（4） 输出的多项式的格式类似为：A(X)=6x^5+2x^3+9;其中零系数项不显示，系数为 1 或者-1 省略 1，幂指数为 1 也省略 1.

4. 测试样例设计

测试样例 1	
设计目的	按顺序输入
样例输入	5 6 3 3 1 4 0 0

	4 5 2 4 0 7 0 0
说明	按照幂指数由高到低输入。即为已经排好序的输入数据。并且这组数据中两个多项式中无幂指数相同的项。为一组最简单的数据，用来测试程序能否正常运行。
样例输出	6x^5+5x^4+3x^3+4x^2+4x+7

测试样例 2	
设计目的	打乱顺序的输入
样例输入	3 3 1 4 5 6 0 0 0 7 2 4 4 5 0 0
说明	验证有序链表的插入功能是否成功。
样例输出	6x^5+5x^4+3x^3+4x^2+4x+7

测试样例 3	
设计目的	输入带有相同幂指数的项
样例输入	5 6 3 3 4 5 1 6 0 0 1 6 3 2 4 2 0 0
说明	验证两个多项式在进行相加的时候，能否将相同幂指数的项进行合并，系数相加。
样例输出	6x^5+7x^4+5x^3+4x^2+12x

测试样例 4	
设计目的	合并同类项之后系数相加刚好等于 0 的情况
样例输入	5 6 0 6 1 4 3 3 0 0 1 -4 0 -6 2 4 5 -6 0 0
说明	验证当合并同类项之后系数相加为 0 不再输出这一项的情况
样例输出	3x^3+4x^2

测试样例 5	
设计目的	验证输出的规格化
样例输入	6 1 5 1 3 -1 0 6 0 0 5 1 4 -1 0 -6 3 1 0 0
说明	对系数为 1，或者系数为-1，幂指数为 0，幂指数为 1 这些情况进行规格化处理，并验证。
样例输出	X^6+2x^5-x^4

二、概要设计

1. 抽象数据类型

为实现上述程序的功能，创建有序线性表。用有序线性表表示一元多项式。输入的

数据是一组二元数对，在创建的有序链表中，每一个结点有三个域来存储数据。次数，系数，next 域。其中由于系数可以为小数，所以将系数的数据类型定义为 double。次数的数据类型为 int。

抽象数据类型设计：

数据对象：一组数对

数据关系：每个数对代表多项式的一个项（次数和系数）。满足线性特征 。

基本操作：

1. 定义一个 insert 函数，按照 index 值由高到低插入新的结点，实现有序链表；
2. 返回当前指针的次数值，系数值。
3. 定义 print 函数用来格式化输出多项式。
4. 定义 clear 函数清空链表。

ADT IntegerSet {

数据对象：D = {< index,coef> | index∈N,coef∈Q， i = 1,2，…， n，1≤n≤1000}

数据关系：R = {< index,coef> | index∈N,coef∈Q }

基本操作：

```
    void insert(double c,int i);//顺序链表的实现,插入的数对为（i，c）
    void next();//指针指向下一结点
    int getIndex();//获取幂指数
    double getCoef();//获取系数
    bool hasNext();//判断有无有下一项
    void print();//遍历输出
    void clear();//清空链表
}
```

2.算法的基本思想

1. **有序链表实现的算法思想：**如果链表为空，则新建一个结点，让空头结点的 next 域指向新建的结点。如果链表不为空，从第一个非空结点开始遍历，若当前结点的 index 域的值小于即将插入的 i 值，则将新的结点插入在该结点之前，如果当前结点的 index 域的值大于即将插入的 i 值，则继续遍历链表中的下一个结点进行比较。

2. **将两个多项式相加的算法思想：**新建一个有序链表 C，用来存储相加之后的多项式结果。分别从两个链表的第一个非空结点开始遍历，判断如果 A 的 index 和 B 的 index 相等，则将两者系数相加，再将新的系数和 A（或 B）的次数储存至有序链表 C 中，如果相加的系数等于 0，则这一项不显示。如果 A 的 index 和 B 的 index 不相等，则储存次数更大的那个。

3. **对链表进行规格化后遍历输出的基本思想：**处理特殊的系数和次数的情况。当系数为 1 的时候，若为头结点，直接输出 x 的指数形式，若不是头结点，则输出“+”后再输出 x 的指数形式。当系数为-1 时，与系数为 1 相似，只是符号变为“-”。当幂指数为 1 的时候，省略不写。

3.程序的流程

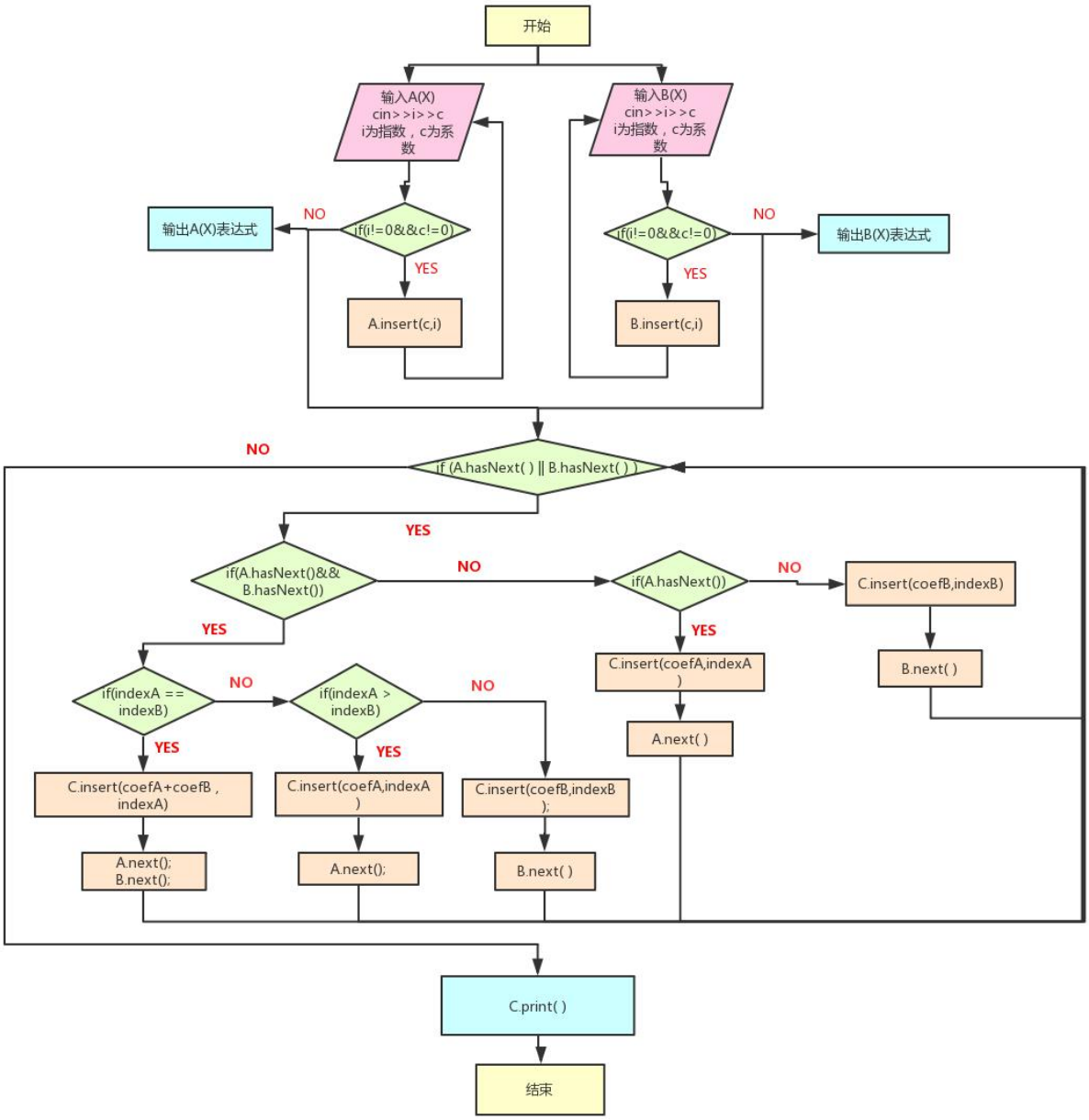
程序由三个模块组成：

（1） 输入模块：有提示语句，提示输入格式以及结束标志，总共需要输入两个多项式，并且根据输入的数据分别创建两个有序链表。

（2） 处理模块：计算两个多项式相加的和，并且创建一个新的有序链表来存储

（3） 输出模块：对多项式结果进行格式化处理之后进行遍历输出

程序流程图如下：



三、详细设计

1. 物理数据类型

输入的数据为两个一元多项式，系数用 double 类型，指数用 int 类型，由于多项式是由 n 个独立的因式构成的，满足线性特性，所以物理数据类型为线性结构，对于插入操作来说，链表的时间复杂度明显低于顺序表，所以逻辑实现上采用链表的形式。

定义的数据类型的伪代码：

```
class sxlist
{   private:
        Node *head;
```

```
    Node *curr;
    Node *tail;
    void init()  {
        head = curr = tail = new Node;
        tail = new Node(-1,-1);//因为多项式的最小幂数为 0
        head->next = tail;  }
    void removeall()
    {   while(head != NULL)
        {    curr = head;
             head = head->next;
             delete curr;
        }
    }
public:
    sxlist();//构造函数
    ~sxlist();//析构函数
    void insert(double c,int i);//顺序链表的实现
    void next();//指针指向下一结点
    int getIndex();//获取幂指数
    double getCoef();//获取系数
    bool hasNext();//判断有无有下一项
    void print();//遍历输出
    void clear();//清空 };
```

2. 输入和输出的格式

输入时有提示语句，说明输入的方法与结束条件，将每个多项式存储在不同的链表 A，B 中，多项式相加的结果存储在链表 C 中，最后格式化输出 C 中存储的内容。

3. 算法的具体步骤

1. 通过插入实现有序链表的算法：

假设要插入数据

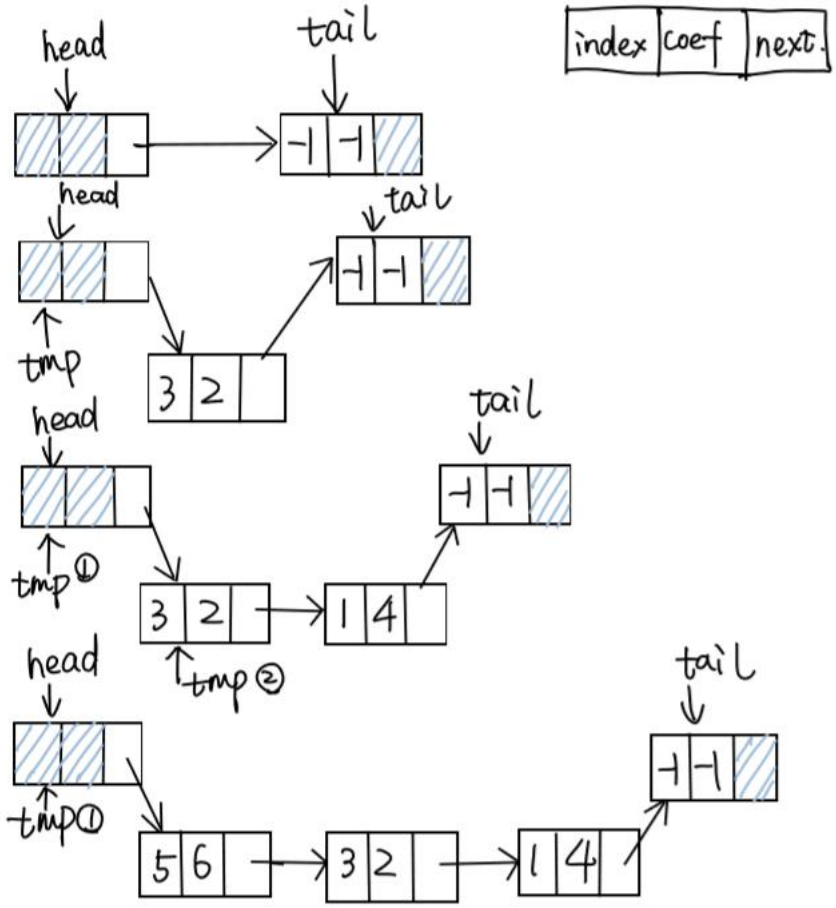
3 2

1 4

5 6

每每插入一个结点，tmp 指针都要从头指针开始遍历，如果 tmp->next 的 index 域大于 i 值的话，就说明该项比当前项的幂指数要小，所以 tmp 要继续向下遍历。如果 tmp->next 的 index 域小于 i 值，则说明该项比当前幂指数要大，当机立断，就在此位置插入。都插入完成之后就是一个有序的链表了。

示意图如图所示：



伪代码如下：

```
void sxlist::insert(double c,int i)
{
    Node *tmp = head;
    if(tmp->next == tail)  //如果链表为空链表，则直接插入新建结点
        head->next = new Node(c, i, tail);
    else {
        while(tmp->next->index > i) {//如果当前结点的指数域大于插入的 i 值
            tmp = tmp->next;//则向下继续遍历
        }
        tmp->next = new Node(c, i, tmp->next);//在合适位置插入新建的结点
    }
}
```

2. 实现两个多项式加法的算法思想：

新建两个结点分别指向两个有序链表的头结点，向下遍历，逐项进行比较：

（1） 两个多项式此时指针都指向某一项。

若两项的系数相等，则对系数进行相加，判断系数是否等于 0，如果等于 0，该项不输出，如果不等于 0，则将新的系数和 A 或 B 的指数作为新的结点插入到新建链表 C 中。

若两者系数不等，则一定有一个较大的项，则需将较大的一项先插入进链表 C 中，然后该多项式的指针继续向下遍历，再与另一多项式的该项进行比较进行同样的比较操作。

伪代码如下：

```
if(A.hasNext()==true && B.hasNext()==true)
{
    int indexA = A.getIndex();
    int indexB = B.getIndex();
    if(indexA == indexB)
    {
        double coefA = A.getCoef();
        double coefB = B.getCoef();
        coef = coefA + coefB;
        if((int)coef != 0)
        C.insert(coef,indexA);
        A.next();
        B.next();
    }
    else if(indexA > indexB)
    {
        coef = A.getCoef();
        C.insert(coef,indexA);
        A.next();
    }
    else
    {
        coef = B.getCoef();
        C.insert(coef,indexB);
        B.next();
    }
```

（2） 两个多项式中有一个多项式的指针已经指向了尾结点，则只需插入有剩余项的多项式的剩余的项进入新的有序链表即可。

伪代码如下：

```
else if (A.hasNext())
{
        int indexA = A.getIndex();
        coef = A.getCoef();
        C.insert(coef, indexA);
        A.next();
  }
  else
{
        int indexB = B.getIndex();
        coef = B.getCoef();
```

```
            C.insert(coef, indexB);
            B.next();
        }
```

这里有一个问题值得思考，既然已经将 C 应用已有的 ADT 建立为有序链表，为何还要按顺序比较之后将较大项先插入进链表 C 中？这是因为，当两个多项式中当前结点指向的两个项的指数不等时，先插入较大的那一项，剩下的幂指数较小的那一项还有可能与另一多项式的剩余项的幂指数相等的情况。若是先插入了小幂次的项，则可能导致合并同类项功能坏掉。

3. 清空链表的算法思想：

每次都让当前指针指向头结点，当头结点向后移动之后，删除掉当前结点，重复执行该过程，知道链表为空链表。

伪代码如下：

```
while(head != NULL)
{
    curr = head;
    head = head->next;
    delete curr;
}}
```

4. 遍历输出的算法思想：

这里主要注意规格化。幂指数为 1 时省略幂指数，幂指数为 0 的时候直接输出常数，系数为 1 的时候，若不是首项，则输出“+”再输出 x^n，若为首项，直接输出 x^n，若系数为 -1，与系数为 1 的时候相似，不过将“+”改为“-”。

4. 算法的时空分析

1）插入每一项的算法时间复杂度为 $\Theta(n)$，插入 n 项形成有序链表的时间复杂度为 $\Theta(n^2)$.

2）实现两个多项式的加法的算法的时间复杂度为 $\Theta(n)$.

3）实现清空链表的时间复杂度为 $\Theta(n)$；

4）实现遍历链表输出的时间复杂度为 $\Theta(n)$.

5）综上所述，程序的时间复杂度为 $\Theta(n^2)$.

四、调试分析

1. 调试方案设计

调试目的：

测试程序是否可运行，发现代码的语法错误、链接错误、逻辑错误、和运算错误，是否有不严谨之处。

测试样例：（设计一个包含了所有特殊情况的样例用来调试分析）

1 6
5 6
0 8
0 0
0 5
1 -6
5 4

0 0

调试计划：

设定好断点，在输入之前，单步执行，观察在 if 语句判断时的跳转是否正确。并 add watch 合适的变量值，观察程序执行的步骤。

设置断点：

在第一次输入命令前设置断点，之后单步执行，调用函数，跳转到对应函数中执行命令，观察对命令的执行情况（删除，插入）是否正确，及时进行修改，对代码的不完善之处进行修改。

2. 调试过程和结果，及分析

设置断点位置：

```
14      double c;
15      int i;
16      sxlist A,B,C;
17      while(cin>>i>>c)
18      {
19          if(c==0 && i==0) break;
20          A.insert(c,i);
21      }
```

单步进入，执行插入操作，首先输入 1 6 数对：

```
====输入A(X)====
1 6
```

判断最开始链表为空，则直接执行插入新建结点操作。

```
30  void sxlist::insert(double c,int i)
31  {
32      Node *tmp = head;
33      if(tmp->next == tail)
34          head->next = new Node(c, i, tail);
35      else {
36          while(tmp->next->index > i) {
37              tmp = tmp->next;
38          }
39          tmp->next = new Node(c, i, tmp->next);
40      }
41  }
```

新建结点需要执行 Node(double,int,Node *)进行传参

```
13  Node::Node(double c,int i,Node *ptr)
14  {
15      coef = c;
16      index = i;
17      next = ptr;
18  }
```

第一个数对输入操作执行完毕，输入下一组数对 5 6：

发现 5 6 数对的 i 值大于已经存储进链表中的 index=1，所以在符合 while 循环终止条件，不需执行 while 循环

```
30  void sxlist::insert(double c,int i)
31  {
32      Node *tmp = head;
33      if(tmp->next == tail)
34          head->next = new Node(c, i, tail);
35      else {
36          while(tmp->next->index > i) {
37              tmp = tmp->next;
38          }
39          tmp->next = new Node(c, i, tmp->next);
40      }
41  }
```

直接执行在 tmp 指针后新建结点的操作。

```
38          }
39          tmp->next = new Node(c, i, tmp->next);
40      }
```

类比地步骤执行下去，执行完 A 多项式的输入，此时 A 多项式输出：

```
====输入A(X)====
1 6
5 6
0 8
0 0
您构造的A(X)多项式为：A(X)=6x^5+6x+8
```

下面执行 B 多项式的输入，然后 B 多项式输出，步骤与 A 同理，这里不再赘述：

```
====输入B(X)====
0 5
1 -6
5 4
0 0
您构造的B(X)多项式为：B(X)=4x^5-6x+5
```

然后跳转到主函数的下一条命令：

```
34      cout<<"则 A(X)+B(X)=";
35      double coef;
36      while(A.hasNext()==true || B.hasNext()==true)
37      {
38          if(A.hasNext()==true && B.hasNext()==true)
39          {
40              int indexA = A.getIndex();
```

进入 while 循环体，执行两个多项式相加的操作

$A(X)=6x^5+6x+8;$

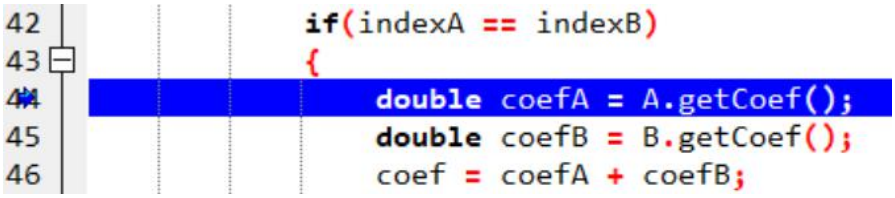
$B(X)=4x^5-6x+5;$

从两个多项式的头结点开始遍历,两个多项式当前结点指向的均不是尾结点，所以进入第一个 if 的分支语句中：

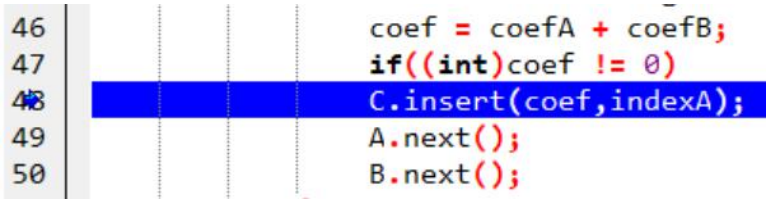
```
38          if(A.hasNext()==true && B.hasNext()==true)
39          {
40              int indexA = A.getIndex();
41              int indexB = B.getIndex();
```

由于两个链表的第一个结点的 index 域相等,即为两个多项式的第一项的幂指数相

等，可以进行合并，所以进入该 if 条件分支语句中执行操作：



并进行判断两个多项式的系数作和之后是否为 0：（此处不等于 0，所以执行插入操作）：



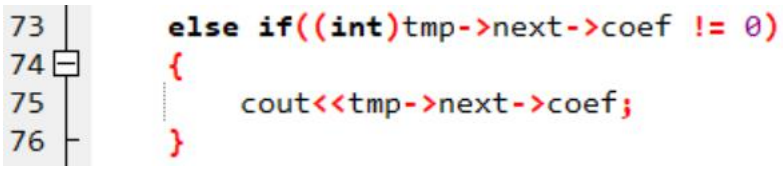
两个多项式的指针都向下挪一位，同时指向了下一项，这一项同样两个幂指数相等，但是两个系数相加等于 0，所以不进行插入操作，直接进行遍历下一项，这里不再进行展示。

最后两项为常数相加，即幂指数等于 0. 相加过程与前面执行的步骤同理，这里亦不再赘述。

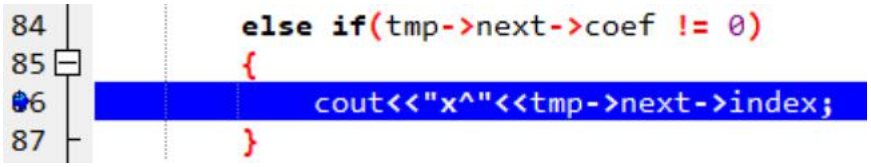
下面执行规格化输出操作，这两个表达式作和，即将输出的表达式为：

$$C(X)=10x^5+13;$$

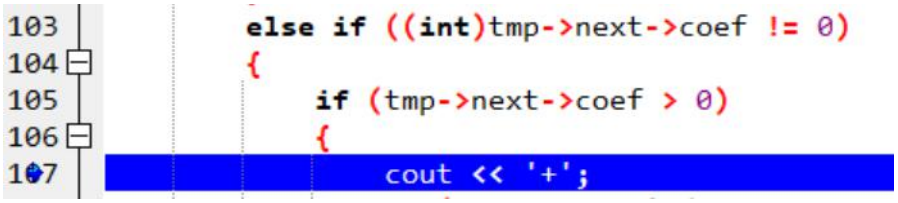
第一项输出系数操作：



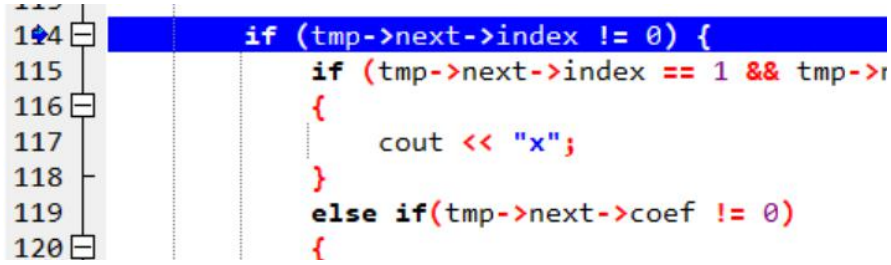
第一项输出 x^n 操作：



第二项输出系数操作：



第二项输出 x^n 操作：（由于幂指数为 0，所以不进入该 if 分支）



输出结束：

```
125     }
126     cout << endl;
```

而后执行清空链表操作：

```
82      A.clear();
83      B.clear();
84      C.clear();
```

执行完毕后程序结束。
最终程序输出结果为：

```
则 A(X)+B(X)=10x^5+13
```

五、测试结果

样例 1：按顺序输入，得到预期效果：

```
每行输入两个数字，第一个表示幂次，第二个表示该幂次的系数，以空格分离
以0 0结束一个多项式的输入
====输入A(X)====
5 6
3 3
1 4
0 0
您构造的A(X)多项式为：A(X)=6x^5+3x^3+4x

====输入B(X)====
4 5
2 4
0 7
0 0
您构造的B(X)多项式为：B(X)=5x^4+4x^2+7

则 A(X)+B(X)=6x^5+5x^4+3x^3+4x^2+4x+7
```

样例 2：打乱顺序的输入，得到预期效果：

```
每行输入两个数字，第一个表示幂次，第二个表示该幂次的系数，以空格分离
以0 0结束一个多项式的输入
====输入A(X)====
3 3
1 4
5 6
0 0
您构造的A(X)多项式为：A(X)=6x^5+3x^3+4x

====输入B(X)====
2 4
4 5
0 7
0 0
您构造的B(X)多项式为：B(X)=5x^4+4x^2+7

则 A(X)+B(X)=6x^5+5x^4+3x^3+4x^2+4x+7
```

样例 3：输入带有相同幂指数的项，得到预期结果：

```
每行输入两个数字，第一个表示幂次，第二个表示该幂次的系数，以空格分离
以0 0结束一个多项式的输入
====输入A(X)====
5 6
3 3
4 5
1 6
0 0
您构造的A(X)多项式为：A(X)=6x^5+5x^4+3x^3+6x

====输入B(X)====
1 6
3 2
4 2
0 0
您构造的B(X)多项式为：B(X)=2x^4+2x^3+6x

则 A(X)+B(X)=6x^5+7x^4+5x^3+12x
```

样例 4：合并同类项之后系数相加刚好等于 0 的情况，得到预期效果。

```
每行输入两个数字，第一个表示幂次，第二个表示该幂次的系数，以空格分离
以0 0结束一个多项式的输入
====输入A(X)====
5 6
0 6
1 4
3 3
0 0
您构造的A(X)多项式为：A(X)=6x^5+3x^3+4x+6

====输入B(X)====
1 -4
0 -6
2 4
5 -6
0 0
您构造的B(X)多项式为：B(X)=-6x^5+4x^2-4x-6

则 A(X)+B(X)=3x^3+4x^2
```

样例 5：验证输出的规格化，得到预期效果：

```
每行输入两个数字，第一个表示幂次，第二个表示该幂次的系数，以空格分离
以0 0结束一个多项式的输入
====输入A(X)====
6 1
5 1
3 -1
0 6
0 0
您构造的A(X)多项式为：A(X)=x^6+x^5-x^3+6

====输入B(X)====
5 1
4 -1
0 -6
3 1
0 0
您构造的B(X)多项式为：B(X)=x^5-x^4+x^3-6

则 A(X)+B(X)=x^6+2x^5-x^4
```

六、实验日志

2019 年 4 月 19 日星期五，我是一个拖延症晚期患者，现在才开始写关于实验二作业。我好悲哀。不过不论啥时候开始做，认真做还是不能改变的原则。这个实验是在实验一的基础上进行一些变化，最关键的就是一个有序链表的建立。关键就在 insert 插入部分，我刚开始想的思路是，每插入一个结点都要重新进行一次排序，那么就需要从头开始遍历，每次都要和每一个结点进行比较，如果比该结点的 index 值大，则就此插入，如果比该结点的 index 值小，则继续遍历。一个比较通俗易懂的思路形成，然而代码写出来的错误却层出不穷。不过这都是后话，因为刚写完这部分代码，编译通过，我还窃喜来着，没想到在写完主程序之后才发现这里有错误，当前指针指向不明确，导致程序输出不了。唉，我还是应该思维再谨慎一点。

2019 年 4 月 20 日星期六，我完成了代码，最麻烦的就是处理那个规格化输出，刚开始测试样例的时候，我总是输入系数为 0 的项，此时输出的时候系数为 0 总是还能显示 0，然后我就继续规格化输出，把系数为 0 的项直接去掉。但实际上，在输入要求中根本就不会输入系数为 0 的项，因为本来多项式中就不存在这一项，我又何必自找麻烦。

2019 年 4 月 21 日星期日，我完成了报告。这个报告，我呕心呖血制作了流程图，又呕心呖血完成了调试分析的一步一步分析。不过有一个地方我刚开始一直犹豫不决，就是我写的算法的时间复杂度的分析，每次的插入操作时间复杂度到底是 n 还是 1 呢，因为还需要每次从头开始遍历，比如说我插入的结点是第 3 个，就是已经有 2 个结点了，最好的情况就是不用往下查找，直接在第一个位置就插入，此时时间复杂度为 2，最坏的情况就是要往下遍历到最后一个，才能插入，此时时间复杂度为 5.所以经过一番思想斗争最终认为插入操作的时间复杂度为 n。插入 n 个结点形成一个链表的时间复杂度即为 n^2.至今我依旧不确定我分析的正不正确。...

通过这次实验吧，我觉得我把链表的应用又精进了一下，写实验的时候也多了更多的耐心，加油。