**Project3 实验报告**

组员 陈诗翰、易鸣奇、张倬豪

**总体功能描述**

该程序主要用于进行两个多项式之间的运算。在界面上输入形如3x5-2x的两个函数，输入完毕后程序会根据选择对从屏幕上输入的两个多项式进行加法，减法或者乘法运算，并将结果以次数降序排列输出。

**整体要求：**

运用链表，根据要求进行多项式的输入、运算和输出。

**设计关键思路方法**

int Create\_Calculator()

建立图形界面，并在合适的位置输出提示语句。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: // Create\_Calculator

Description: // 整个输入界面的建立

Calls: // initgraph(); setfillstyle(); setcolor(); outtextxy(); 等来自graphics.h的函数

Pick\_Out\_Nums(int num); 等本程序函数

Called By: // main(); 主函数

Input: // 无

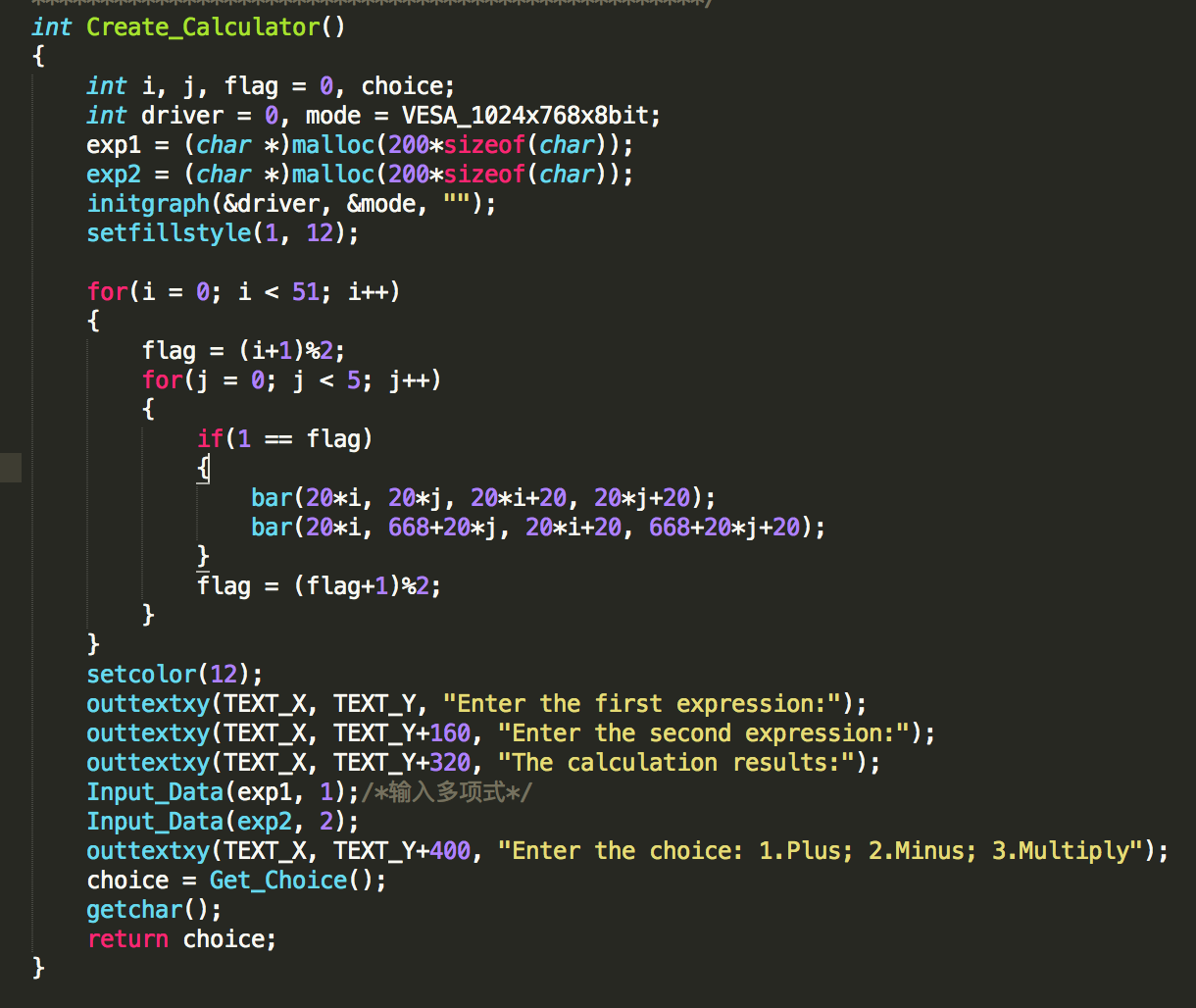
Output: // 输出整个输入界面的构件。

Return: // 返回选择的运算（加减乘）

Others: // 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

具体实现：该函数是本程序执行的第一个函数，创建了图形界面，没有过多可以描述的地方。



void Input\_Data(char \*p, int num)

该函数主要用于读取输入的字符（包括0~9十个数字，'+'，'-'，'\*'，'^'，'x'以及'X'），每读入一个字符，就在窗口的恰当位置重新输出一次。当打字出现错误时，可以用delete对前一个输入的字符用背景色进行覆盖，从而实现“删除”的功能。当函数读取到enter的键码时，自动跳出循环，结束当前多项式的输入。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: // Input\_Data

Description: // 利用bioskey读入键盘逐个输入的字符，并进行处理，同时处理回车、退格键的输入，达到输入字符串的效果

Calls: // bioskey(); setcolor(); outtextxy(); 等来自graphics.h的函数

Input\_Data(char \*p, int num); Get\_Choice(); 等本程序函数

Called By: // Create\_Calculator()

Input: // char \*p, int num p为储存字符串的指针，num为字符串编号，取值为1，2

Output: // 无

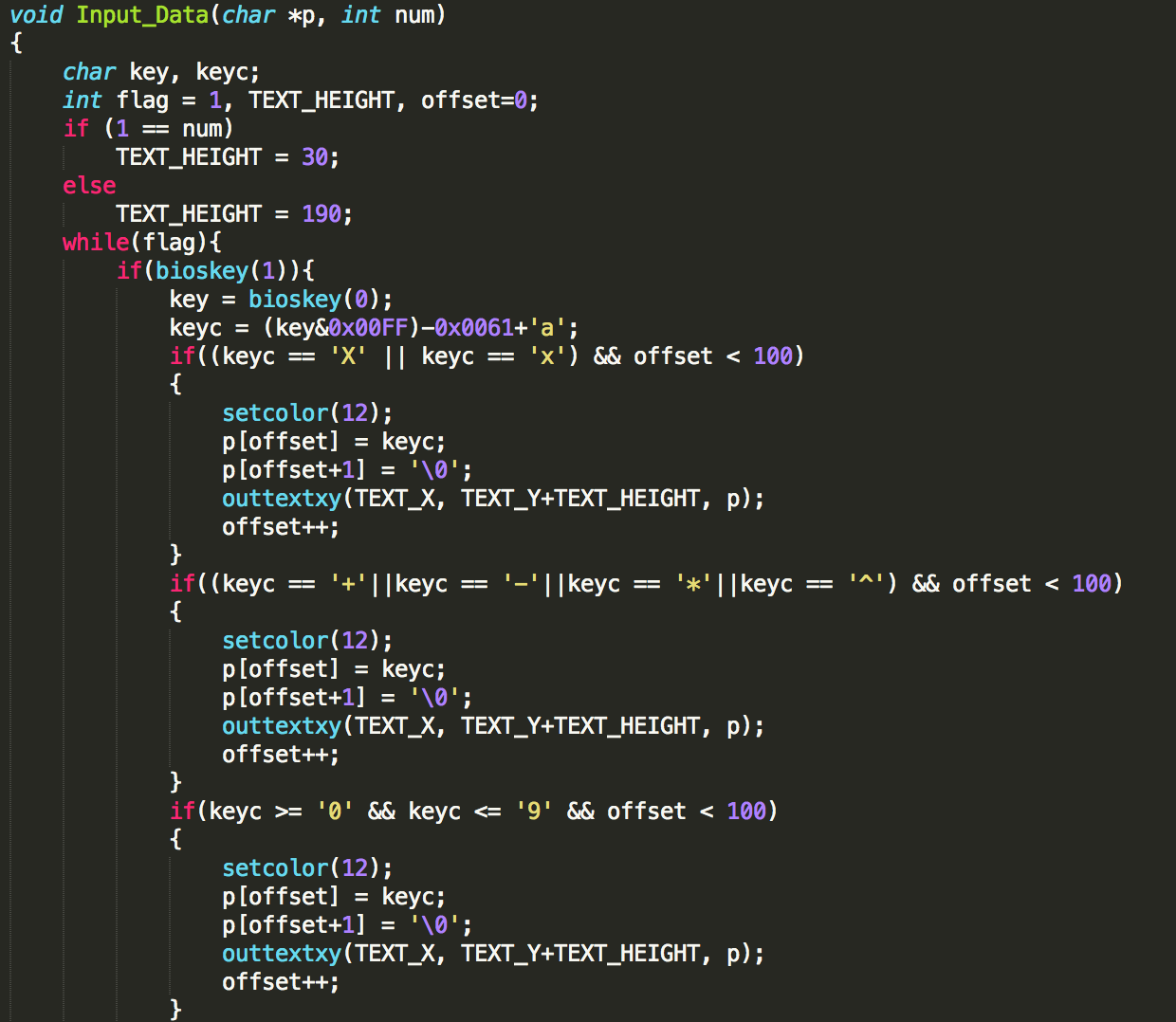
Return: // 无

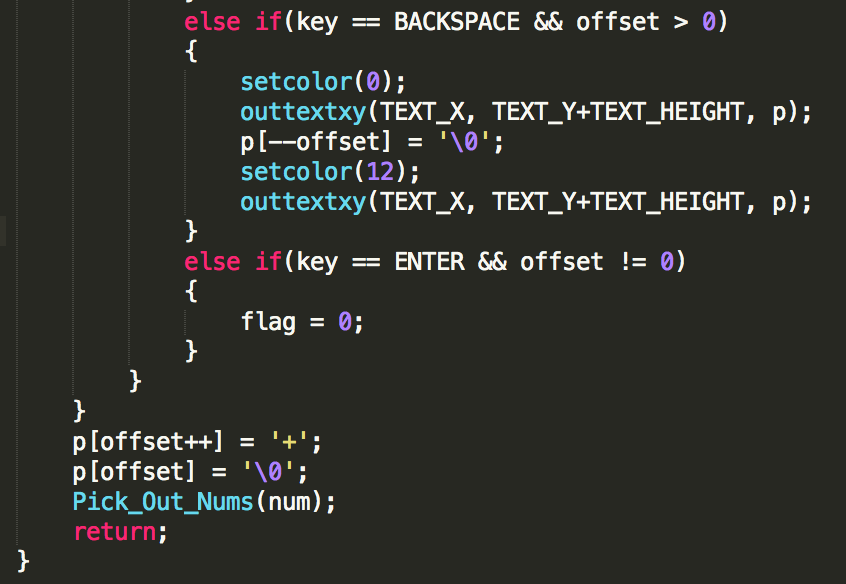
Others: // 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

此函数是本工程中较为困难的三个地方之一，需要以此读取键盘输入的字符，并对回车和退格均进行判断。

组员分别查阅了各键值，并且在退格键时进行刷新界面输出前一个字符时的状态。几乎模拟了正常的文本输入输出的界面。

}



void Pick\_Out\_Nums(int num)

对输入的字符串进行处理，提取出其中的系数和指数，每得到一个完整的数字，就调用一次Transform\_Nums，最后将所有数字按照顺序存储在一个数组之中。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: // Pick\_Out\_Nums

Description: // 处理输入的字符串，将含有x的多项式提取出系数和指数来

Calls: // Transform\_Nums();

Called By: // Input\_Data(int num);

Input: // int num 表示是第几个字符串，取值为1，2

Output: // 无

Return: // 无

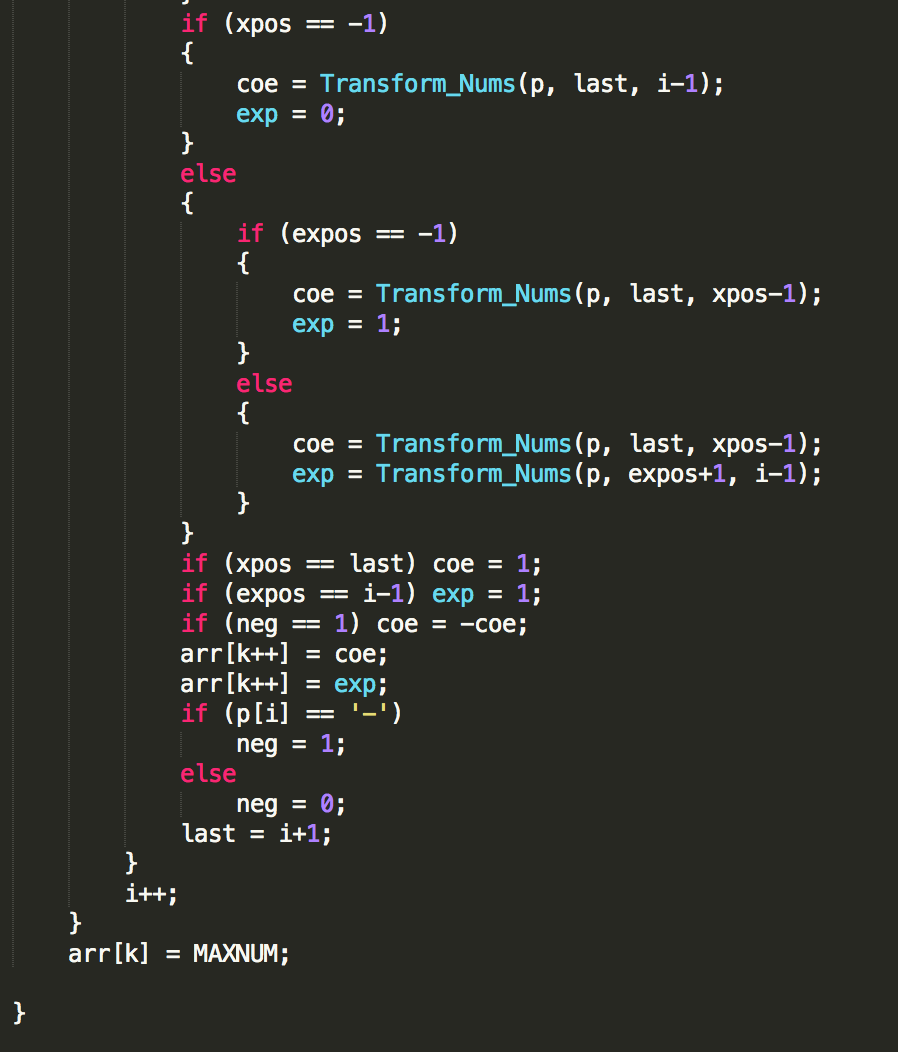
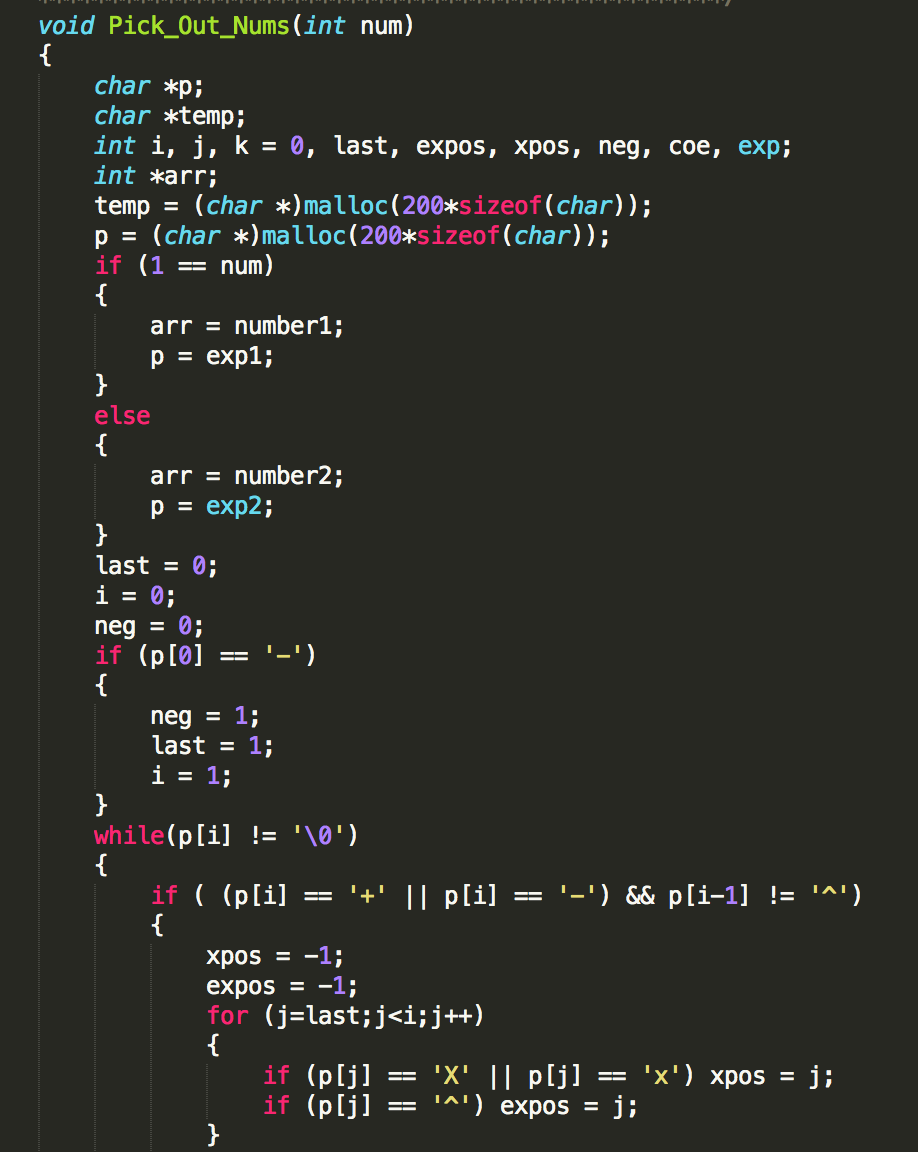
Others: // 结果存在全局变量number1,number2里面

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

这是本工程较为困难的地方之一，也是工程的输入处理核心所在。

需要特别考虑当指数为1，0（也就是常数），负数，以及系数为1，-1的情况。而其他形如2x^2这样的则比较简单，可以用一般的读入字符串处理。这段花了许多时间进行反复调试，从刚开始的只能识别1\*x^1+4\*x^0这样的多项式，到改进到可以实现x+4这样的多项式。

此处还调用了下面的Transform\_Nums函数，进行部分全是数字的字符串的转换，转换成数字存入数组，数组中最后存储的是系数和指数间隔存放，下面有函数对此进行处理，最后一个数据结尾是一个非常大的数字（程序不允许这么大的数字因为进行乘法运算以后可能会溢出）用于标识。



int Transform\_Nums(char \*p, int start, int end)

将字符串p中从start到end的数字字符提出来，并转换为int类型的一个数字。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: // Transform\_Nums

Description: // 将字符串转换成数字

Calls: // 无

Called By: // Pick\_Out\_Nums(int num);

Input: // char \*p, int start, int end 字符指针，从start到end位置的转换成数字

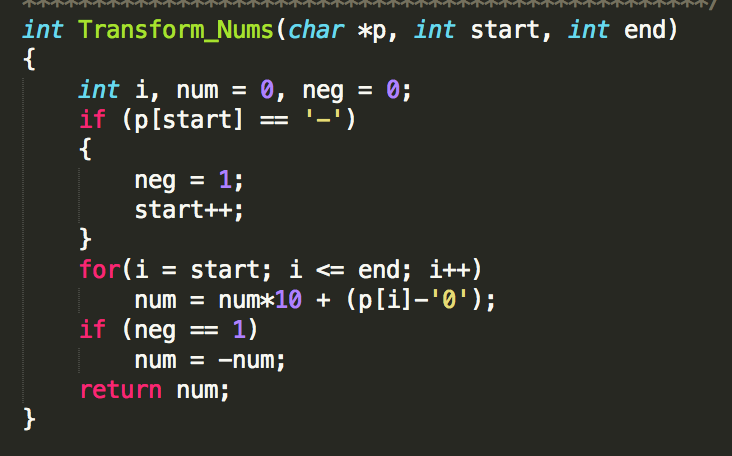
Output: // 无

Return: // num，转换的数字

Others: // 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

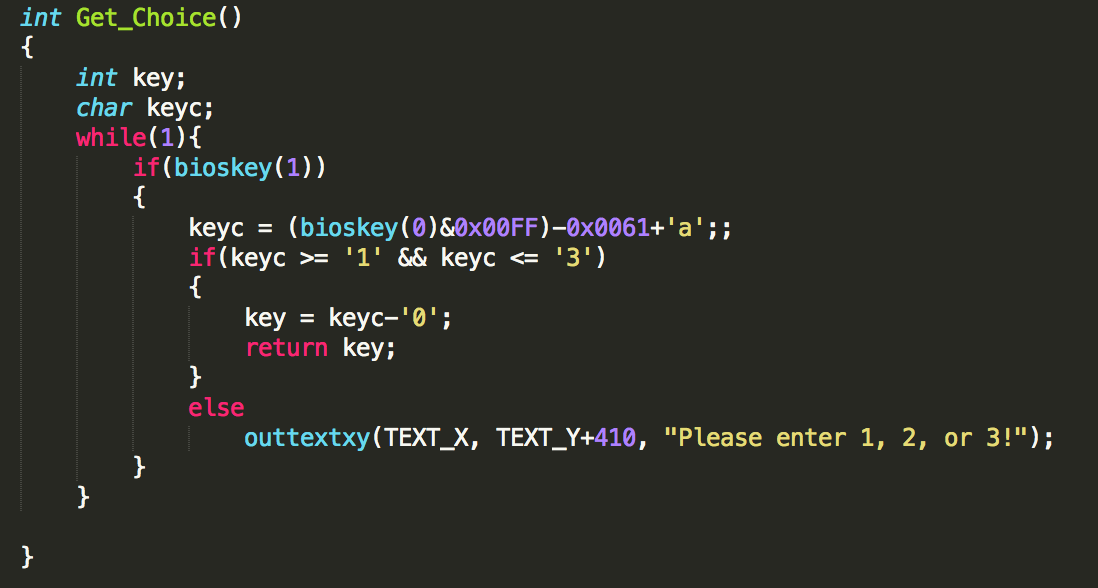
具体实现：这个函数比较简单，被处理字符串的函数Pick\_Out\_Nums调用，进行部分全是数字的字符串的转换，转换成数字存入数组。



int Get\_Choice()

读入所要进行的运算，1代表加，2代表减，3代表乘。

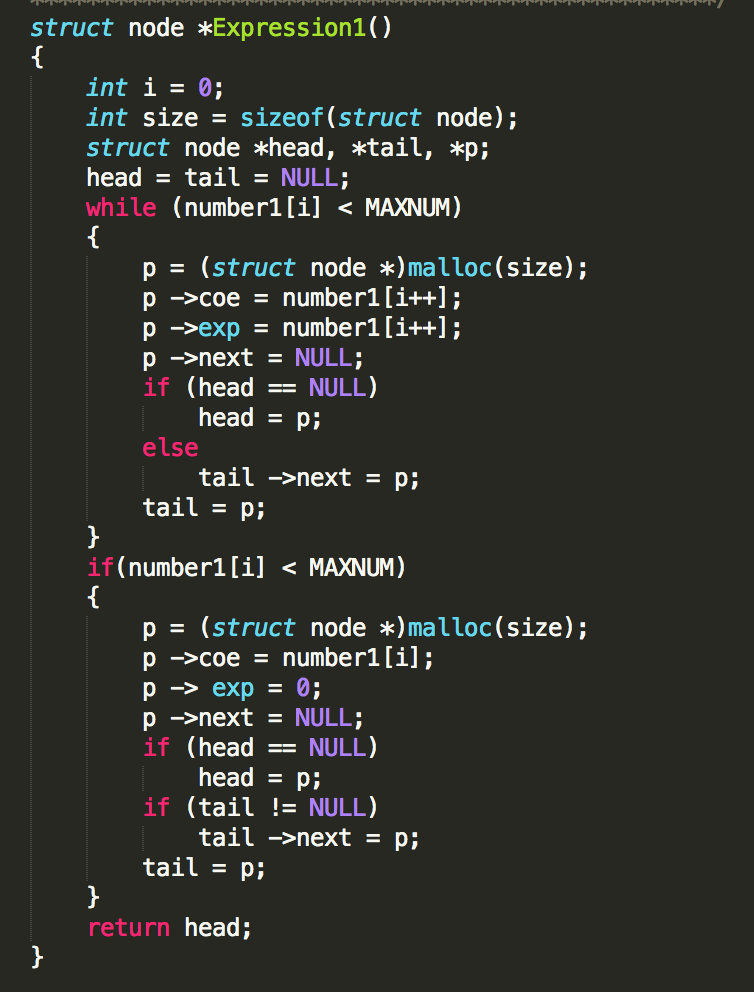
此函数比较简单，不做过多描述。



struct node \*Expression1()

将表达式1的系数和指数存储在链表当中，方面运算调用。

此处用到了基本的链表操作，这些是这次project的基础。此处需要注意头结点的处理，需要多加一个if语句，当然这句也是所有建立链表操作的基础。



struct node \*Expression2()

将表达式2的系数和指数存储在链表当中，方面运算调用。同上。

**void Plus(struct node \*head1, struct node \*head2)**

**关键函数：加法**

**具体实现：**

对链表一进行操作，将链表二加入链表一，先进行指数从高到低排序（Sort函数）然后进行从高次到低次搜索，如果碰到指数相等，则加入系数值，如果碰到下一个结点指数比要加的结点小，而当前结点比要加的结点大，则插入要加的结点到当前结点和下一个结点的中间。若“下一个结点”是头结点，则插入为第一个结点（此处要特殊考虑）

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: // Plus

Description: // 加法，最后结果通过函数Print输出

Calls: // Merge\_Same(); Delete\_Zero(); Print();

Called By: // 主函数

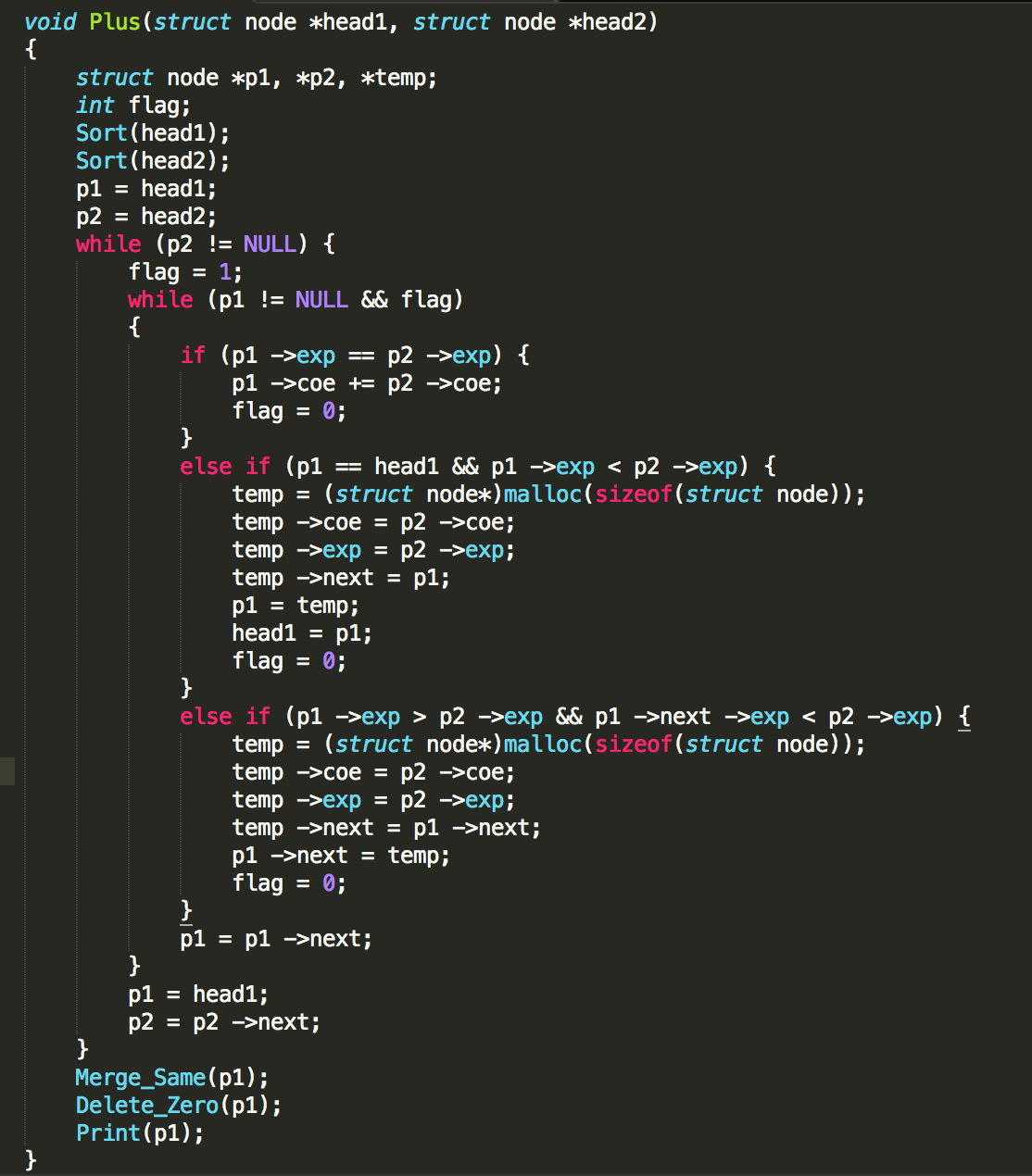
Input: // struct node \*head1, struct node \*head2 两条链表

Output: // 无

Return: // 无

Others: // 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/



**void Minus(struct node \*head1, struct node \*head2)**

同plus，只是把中间的加法运算改成减法，其他的基本链表操作完全一致

void Multiply(struct node \*head1, struct node \*head2)

**关键函数：乘法**

**具体实现：**

不同于加减函数，乘法函数再进行了运算以后再进行排序，这是非常明显的。这里是进行各项扫描进行乘法并插入链表当中，最后进行排序。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: // Multiply

Description: // 乘法，最后结果通过函数Print输出

Calls: // Merge\_Same(); Print();

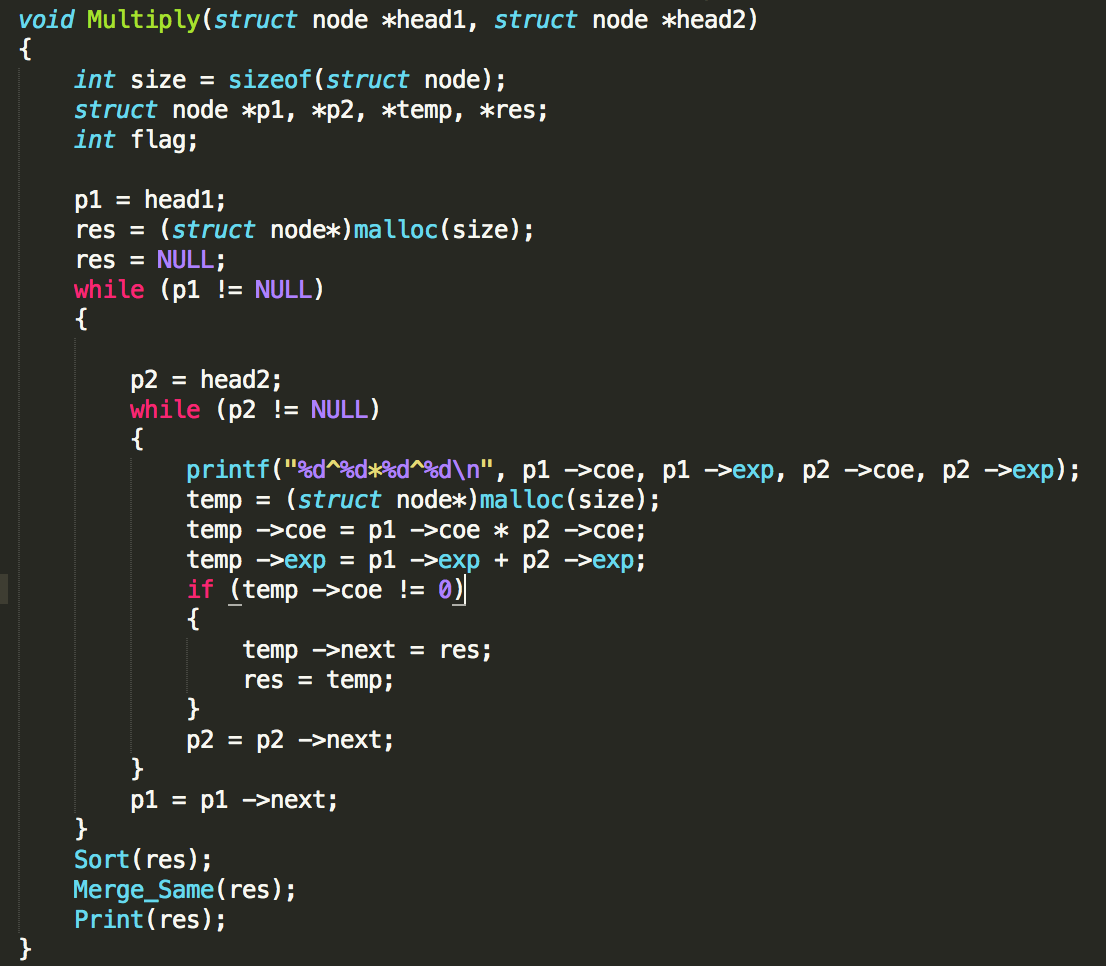
Called By: // 主函数

Input: // struct node \*head1, struct node \*head2 两条链表

Output: // 无

Return: // 无

Others: // 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void Sort(struct node \*head)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: // Sort

Description: // 将head开头的多项式链表按次数降序排序

Calls: // 无;

Called By: // Plus(); Minus(); Multiply();

Input: // struct node \*head 一条链表

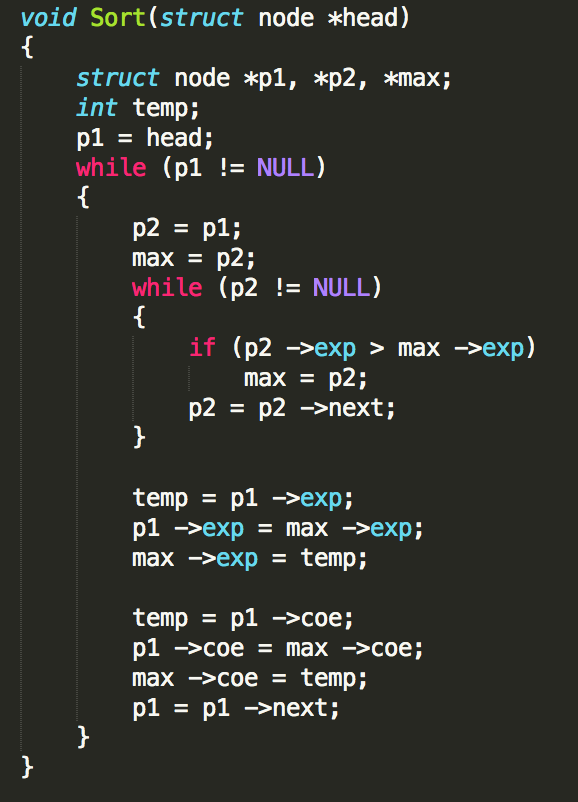
Output: // 无

Return: // 无

Others: // 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

具体实现：算法核心是选择算法，此处将多项式链表按照指数降序排列。实现难度不大。



void Merge\_Same(struct node \*head)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: // Merge\_Same

Description: // 将head开头的有序链表合并同类项

Calls: // 无;

Called By: // Plus(); Minus(); Multiply();

Input: // struct node \*head 一条链表

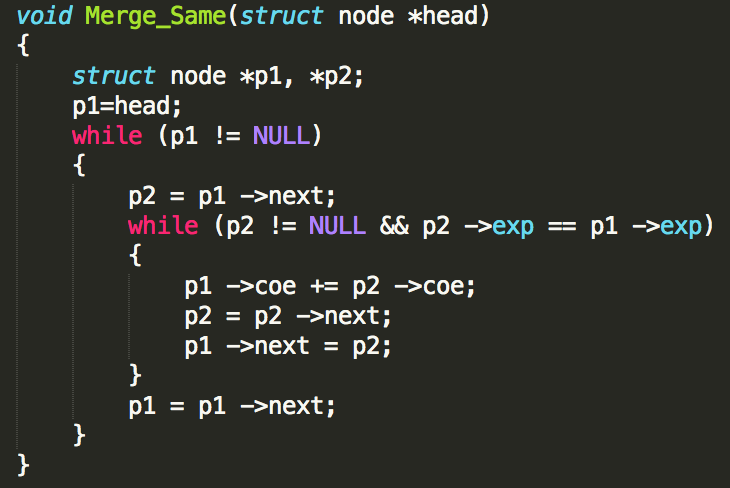
Output: // 无

Return: // 无

Others: // 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

具体实现：类似于归并排序的两个有序数列合并排序，时间复杂度仅O(n)，实现也较为简单。



void Delete\_Zero(struct node \*head)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: // Delete\_Zero

Description: // 去除链表中的0项

Calls: // 无;

Called By: // Plus(); Minus();

Input: // struct node \*head 一条链表

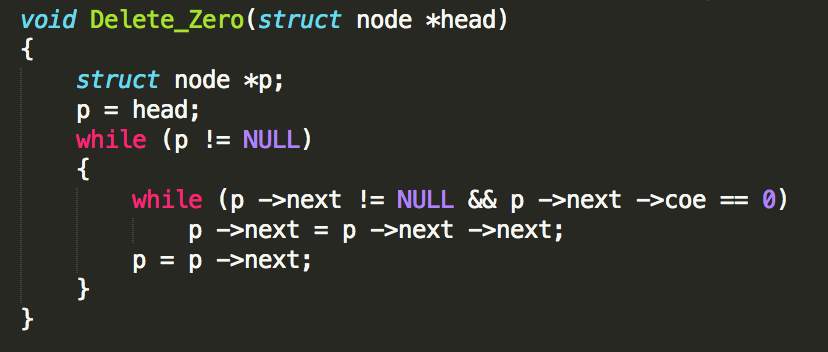
Output: // 无

Return: // 无

Others: // 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

具体实现：此处为删除零项的函数，防止后续输出的函数造成困扰。



void Print(struct node \*head)

以多项式的形式输出链表中的数据。在常规形式之外，还需要分出整体为0，系数为0、+1、-1，指数为0、+1、-1等多种情况讨论。非常复杂，容易出错。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function: // Print

Description: // 打印链表中代表的多项式

Calls: // 无;

Called By: // Plus(); Minus(); Multiply();

Input: // struct node \*head 一条链表

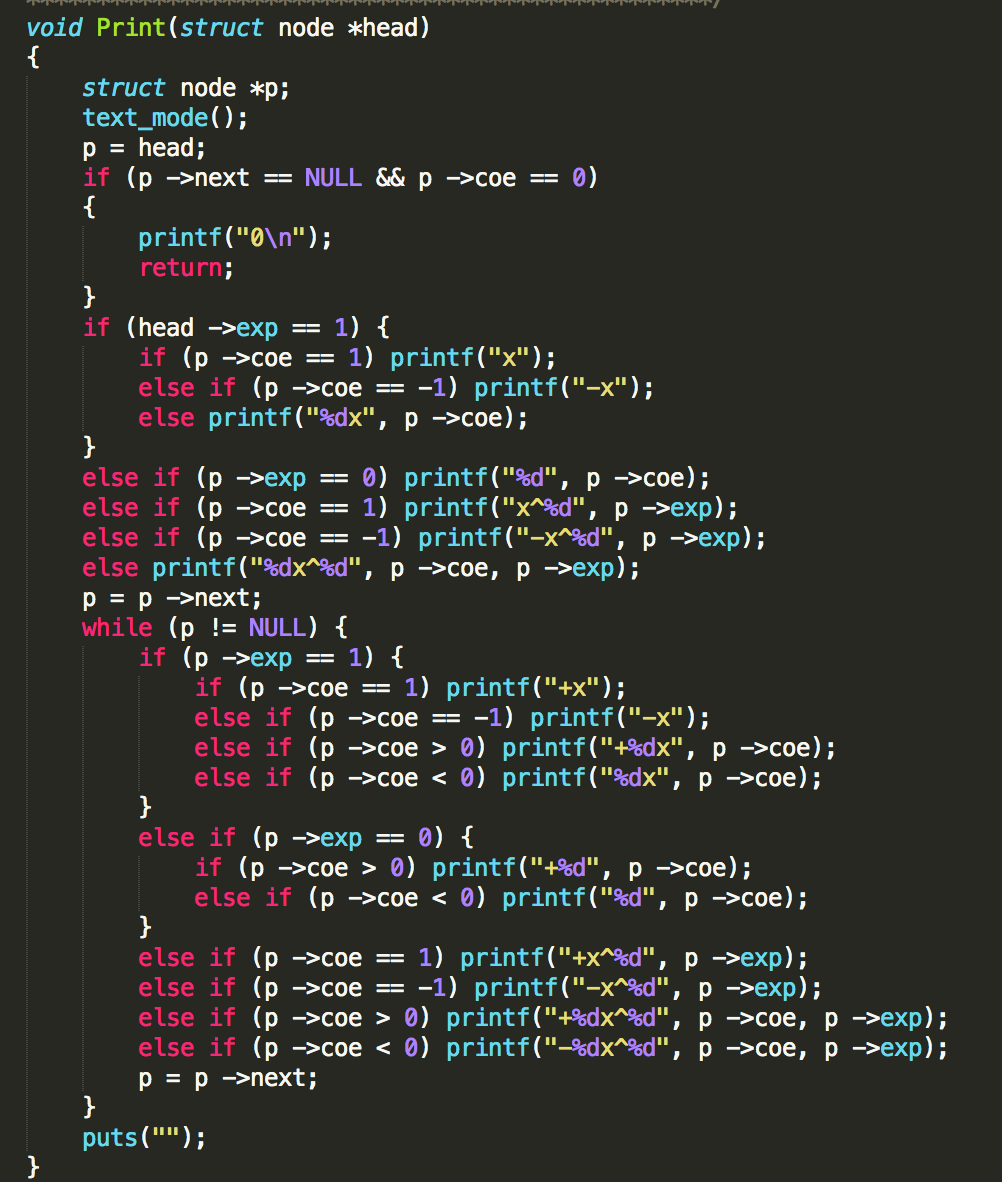
Output: // 链表代表的多项式，指数和系数由链表给出

Return: // 无

Others: // 无

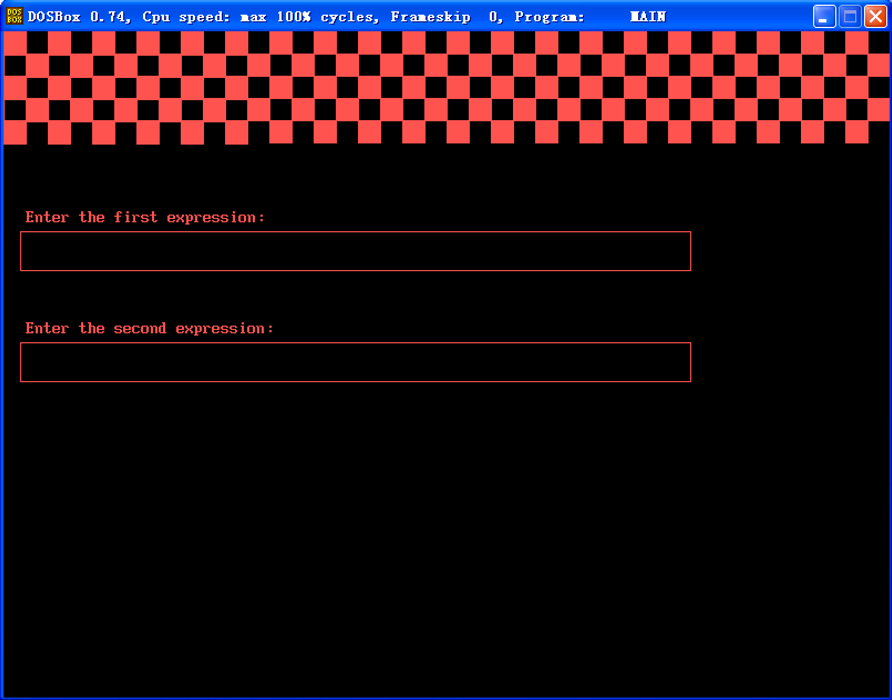
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

具体实现：此处为本工程难点之一，按照正常的数学习惯输出多项式比较复杂，需要进行许多许多if语句判断，判断指数为0，1的情况，系数为1，-1的情况，复杂程度不亚于读入处理字符串的函数。

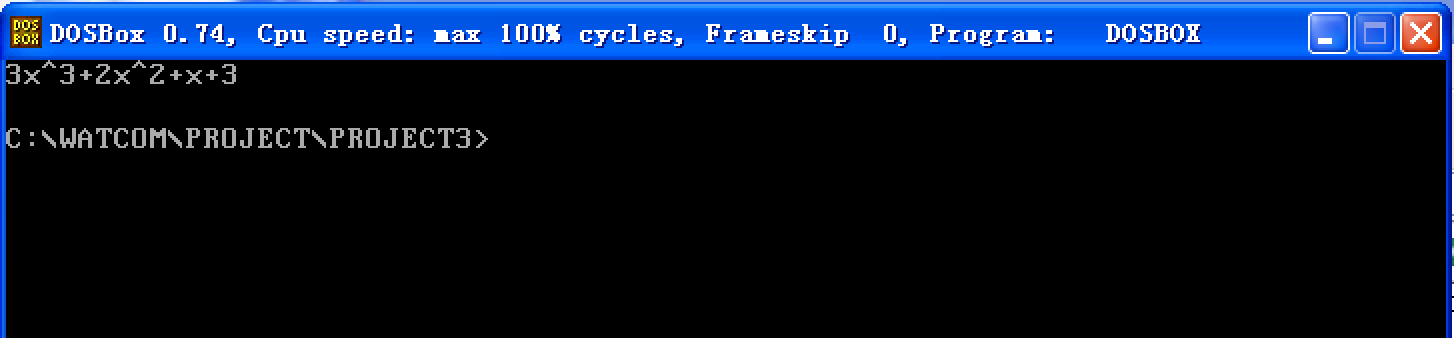
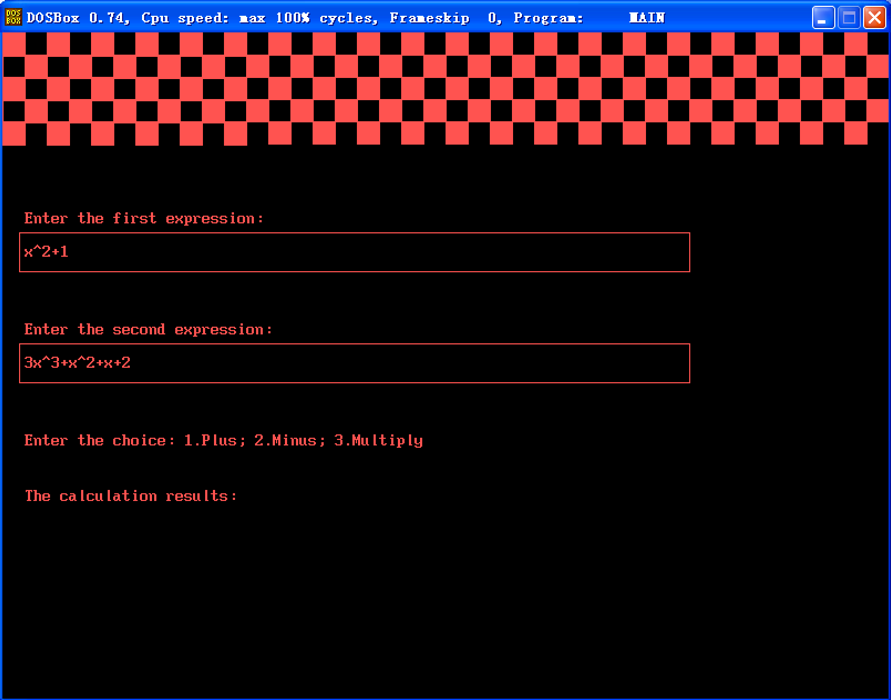


**测试报告**

此为开始界面

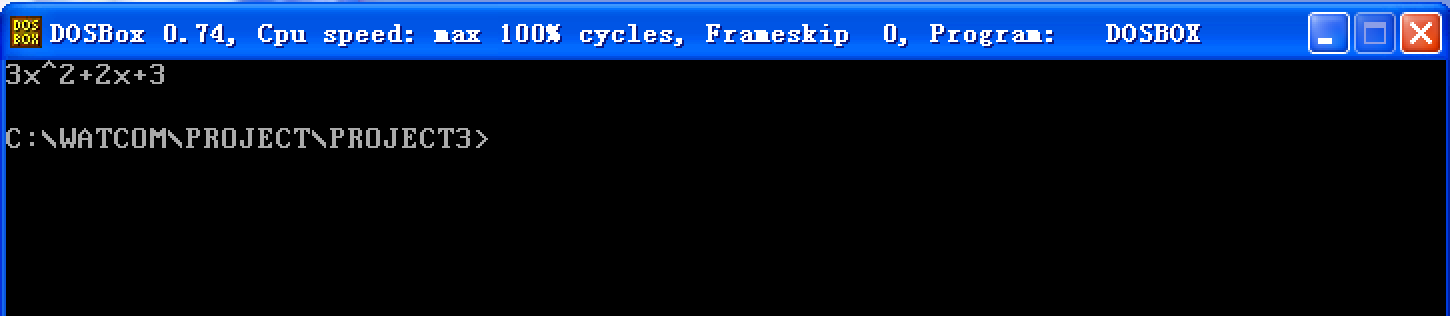
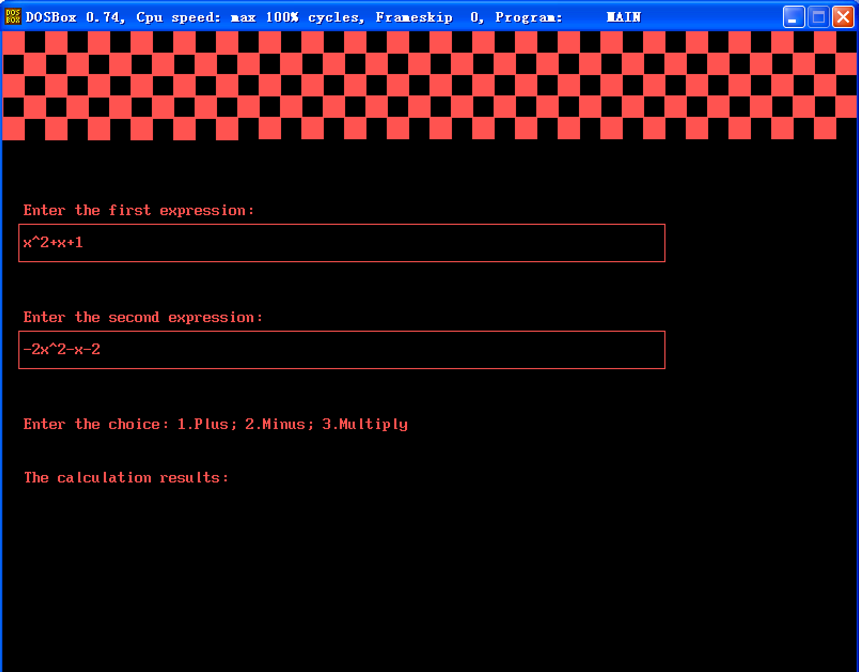


此为加法实例

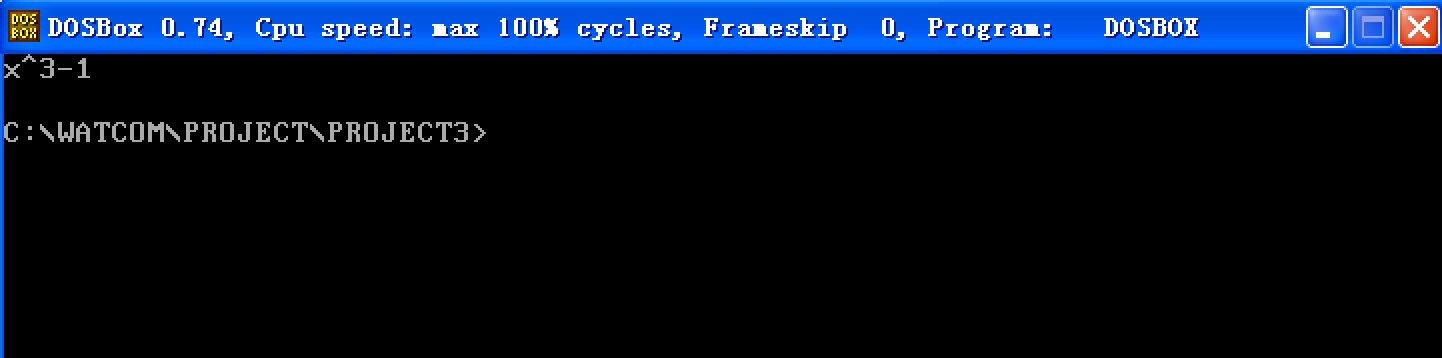
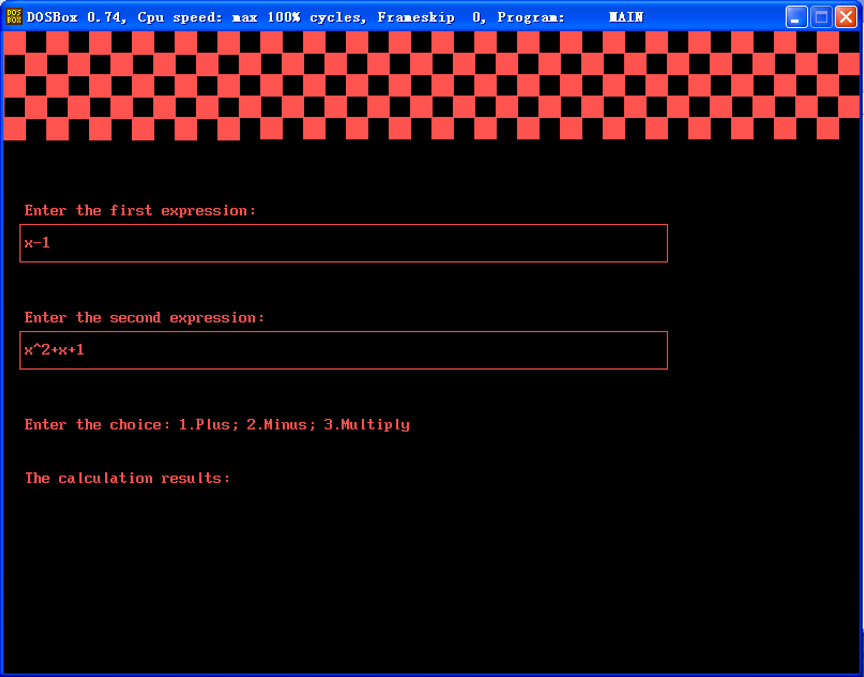


这里可以看到结果是正确的

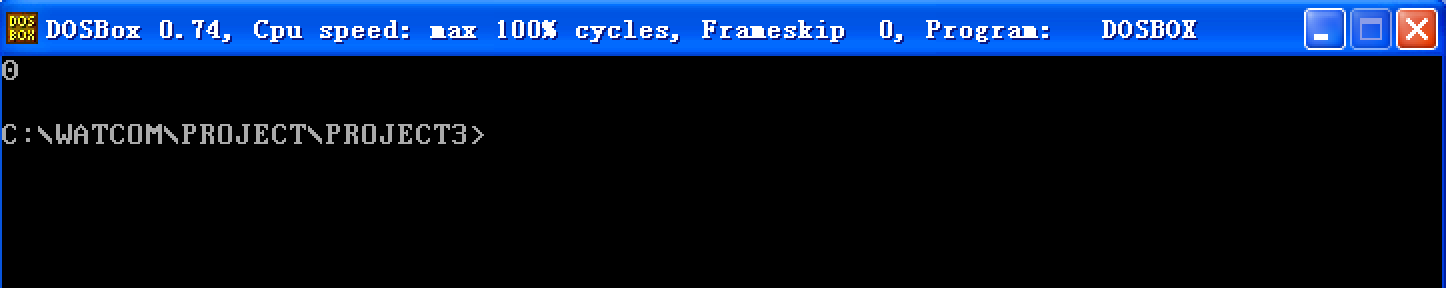
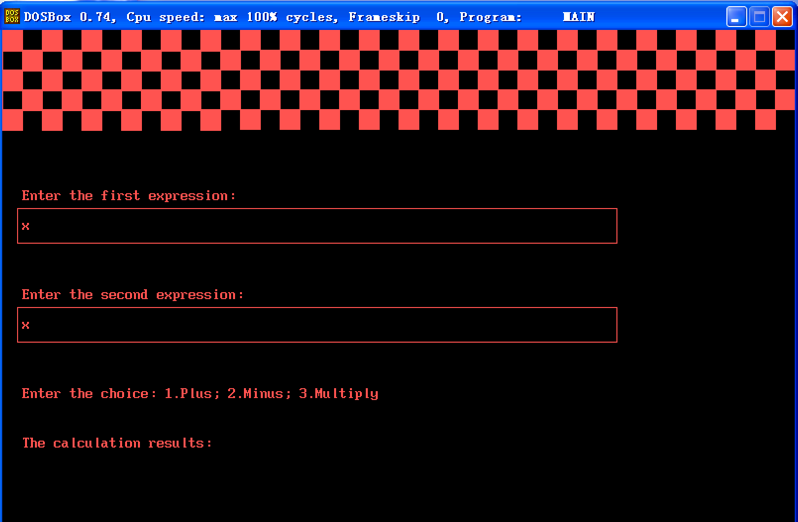
此为减法实例



此为乘法实例



特殊情况实例



**开发体会**

这一次的project中，我们在完成代码的主要功能之外，还试图加上一些额外的功能。比如，将程序做成图形界面，使画面看上去更加美观简洁。我们在完成的过程中遇到的困难主要集中在实现图形界面指定位置的输入删除和怎样妥善处理读取多项式时出现的多种输入与输出情况上。这些问题对我们的思考能力与缜密程度都是一种锻炼提高。在这一次的团队合作中，小组成员们的配合工作能力也有了显著的增强。

**反思回顾**

这次的project仍有很多不足的地方，比如最后的输出并没有放在图形界面中，因为采取图形界面输出必须先把输出的转成字符串，而把链表中的系数和指数变成规范简洁的多项式非常繁琐，考虑到开发时间成本，并没有这么做。还有，在加减法的编写时，可以改进，取消第二个大的if语句判断（插入跟当前没有指数相同可以合并的项，此处可以直接插入，最后再调用sort函数进行，而不是先排序然后进行小心翼翼地插入。

**任务职责及互评（不分先后）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 角色 | 学号 | 姓名 | 分数 | 职责 |
| 组长 | 3150102418 | *张倬豪* | *3* | * 总体策划，负责协调活动 * 负责召集组内成员开会讨论 * 组内成员任务分工 * 代码实现 * 测试 * 撰写《project报告》 |
| 组员 | 3150102390 | *易鸣奇* | *3* | * 信息收集 * 记日志、拍照、视频 * 代码实现 * 测试 * 撰写《project报告》 |
| 组员 | 3150103690 | *陈诗瀚* | *3* | * 信息收集 * 记日志、拍照、视频 * 代码实现 * 测试   撰写《project报告》 |