等价表达式

(equal.pas/c/cpp)

【问题描述】

明明进了中学之后，学到了代数表达式。有一天，他碰到一个很麻烦的选择题。这个题目的题干中首先给出了一个代数表达式，然后列出了若干选项，每个选项也是一个代数表达式，题目的要求是判断选项中哪些代数表达式是和题干中的表达式等价的。

这个题目手算很麻烦，因为明明对计算机编程很感兴趣，所以他想是不是可以用计算机来解决这个问题。假设你是明明，能完成这个任务吗？

这个选择题中的每个表达式都满足下面的性质：

1． 表达式只可能包含一个变量‘a’。

2． 表达式中出现的数都是正整数，而且都小于 10000。

3． 表达式中可以包括四种运算‘+’（加），‘-’（减），‘*’（乘），‘^’（乘幂），以及小括号‘(’，‘)’。小括号的优先级最高，其次是‘^’，然后是‘*’，最后是‘+’和‘-’。‘+’和‘-’的优先级是相同的。相同优先级的运算从左到右进行。（注意：运算符‘+’，‘-’，‘*’，‘^’以及小括号‘(’，‘)’都是英文字符）

4． 幂指数只可能是 1 到 10 之间的正整数（包括 1 和 10）。

5． 表达式内部，头部或者尾部都可能有一些多余的空格。

下面是一些合理的表达式的例子：

((a^1) ^ 2)^3，a*a+a-a，((a+a))，9999+(a-a)*a，1 + (a -1)^3，1^10^9……

【输入文件】

输入文件 **equal.in** 的第一行给出的是题干中的表达式。第二行是一个整数 n（2 <= n <= 26），表示选项的个数。后面 n 行，每行包括一个选项中的表达式。这 n 个选项的标号分别是 A，B，C，D……

输入中的表达式的长度都不超过 50 个字符，而且保证选项中总有表达式和题干中的表达式是等价的。

【输出文件】

输出文件 **equal.out** 包括一行，这一行包括一系列选项的标号，表示哪些选项是和题干中的表达式等价的。选项的标号按照字母顺序排列，而且之间没有空格。

【样例输入】

```
( a + 1) ^2
3
(a-1)^2+4*a
a  + 1+ a
a^2 + 2 * a * 1 + 1^2 + 10 -10 +a -a
```

【样例输出】
AC

【数据规模】
对于 30%的数据，表达式中只可能出现两种运算符‘+’和‘-’；
对于其它的数据，四种运算符‘+’，‘-’，‘*’，‘^’在表达式中都可能出现。

对于全部的数据，表达式中都可能出现小括号‘(’和‘)’。

【问题分析】
题意很明确：给定两个表达式，判断其是否等价。处理这个问题有两种办法：
(1) 展开。将给定的式子按照多项式运算的规则展开，最后合并同类项，判断两表达式是否相同。这一种方法无疑是严密的，但是对于如下数据：(a+1)^10^10^10，将导致最终项数过多，程序无法在规定时间、空间内出解。
(2) 代值。将数值代入并计算。当然，假如碰巧代入的值使两个多项式值相等，将会造成误判。当然，这是有解决途径的：多次代入不同数值、代入较大数值等等……虽然这个算法并不一定正确，但是随机多次后出错的概率已经可以忽略不计，而且实现又比较简单不失为一种好办法。

要想将值代入，就需要一个表达式求值的过程。表达式求值与人的思维相同，应用了一个递归的过程：每次找出优先级最低的一个符号，将它左边、右边的算式分别进行计算直到只剩一个数字时返回。这样的代码符合自然思维，实现简单明了，不易出错。
由于计算中出现了次方运算，所以求值过程中需要进行取模运算，这样避免了整数的溢出，保证了最后结果的准确性。

【代码清单】
```
#include <cstdio>
#include <cstring>
#include <cstdlib>
const int MaxN = 100;
const int MOD = 19880216;

struct node {
    char ch;
    int data;
} A[MaxN], B[MaxN];
int N, M, value;
char str[MaxN];
int Priority[256];

void prepear() {
    freopen("equal.in", "r", stdin);
    freopen("equal.out", "w", stdout);
```

```
    memset(Priority, 10, sizeof(Priority));
    Priority['+'] = 0;
    Priority['-'] = 0;
    Priority['*'] = 1;
    Priority['^'] = 2;
}

bool factor(node A[], int &N, char str[]) { //将读入字串分解
    int i = 0, top = 0, stk[MaxN], tmp;
    N = 0;
    while (i < strlen(str)) {
        if (str[i] == ' ') {
            i ++;
            continue;
        }
        if (str[i] == 'a' || str[i] == '+' || str[i] == '-' ||
str[i] == '*' || str[i] == '^') {
            A[N ++].ch = str[i ++];
            continue;
        }
        if (str[i] == '(') {
            stk[top ++] = N; //左括号的Data域表示与它匹配右括号的位置
            A[N].ch = '(';
            A[N ++].data = str[i ++];
            continue;
        }
        if (str[i] == ')') {
            if (top == 0) return false;
            A[stk[-- top]].data = N;
            A[N ++].ch = ')';
            i ++;
            continue;
        }
        tmp = 0; //处理数字
        while (i < strlen(str) && str[i] >= '0' && str[i] <= '9') {
            tmp = tmp * 10 + str[i] - 48;
            i ++;
        }
        A[N].data = tmp;
        A[N ++].ch = 'N';
    }
    if (top != 0) return false;
    else return true;
}
```

```
int OP(int Left, char op, int Right) { //数值运算
    __int64 Tmp;
    Tmp = Left;
    if (op == '+') Tmp += Right;
    else if (op == '-') Tmp -= Right;
    else if (op == '*') Tmp *= Right;
    else if (op == '^') {
        Tmp = 1;
        for (int j = 0; j < Right; j ++)
            Tmp = (Tmp * Left) % MOD; //乘方运算
    }
    Tmp %= MOD;
    return int(Tmp);
}

int F(node *A, int i, int j) { //递归求解
    if (A[i].ch == '(' && A[i].data == j) return F(A, i + 1, j -
1);
    if (i == j && A[i].ch == 'N') return A[i].data % MOD;
    if (i == j && A[i].ch == 'a') return value % MOD;
    int Min = 100, Pos, Bracket = 0;
    for (int k = i; k <= j; k ++) {
        if (A[k].ch == '(') Bracket ++;
        if (A[k].ch == ')') Bracket --;
        if (Bracket == 0 && Priority[A[k].ch] <= Min) {
            Min = Priority[A[k].ch]; //寻找优先级最低的算符
            Pos = k;
        }
    }

    //计算
    return OP(F(A, i, Pos - 1), A[Pos].ch, F(A, Pos + 1, j));
}

int main() {
    bool Equal;
    int n;
    char str[MaxN];
    prepear();

    gets(str);
    factor(A, N, str);
```

```
    scanf("%d\n", &n);
    for (int i = 0; i < n; i ++) {
        Equal = true;
        gets(str);
        if (!factor(B, M, str)) continue;

        for (int j = 0; j < 20; j ++) {
            value = rand() % MOD; //随机将数值代入
            if ((MOD + F(A, 0, N - 1)) % MOD != (MOD + F(B, 0, M -
1)) % MOD) { //注意取模后数字可能为负
                Equal = false;
                break;
            }
        }
        if (Equal) putchar('A' + i);
    }
    putchar('\n');
    return 0;
}
```

【小结】

这个问题本身是一个简单问题，只是有一定的编程难度。寻找最简洁有效的方法，才能在最短的时间内解决问题。