24点游戏求解

问题描述

24点游戏是一种使用扑克牌来进行的益智类游戏，游戏规则为：从一副扑克中抽去大小王，在剩下52张牌（1-13各4张）中任意抽取4张牌，把牌面上的数字运用四则运算得出24，每张牌必须且仅能使用一次。
本程序使用遍历方法，利用**Fortran90**语言编写代码，实现了24点游戏**全部解**的输出（对于任意一组牌面，输出其得出24的全部方法）。

方法描述

- 程序有两种模式：随机生成或用户从键盘手动输入1-13的4个整数。最终程序输出对于给定牌面的所有组成24点的方式。用户可以在程序最后决定继续运算或退出程序。
- 求解该问题的难点之一在于寻找一种算法来穷尽4个数的所有运算方式。程序主体采用类似迭代的思想：数组开始时有4个数，从4个数中挑选两个数，再挑选一种运算，得到结果；把结果和未被挑选的2个数组合起来形成新的数组，重复上述过程（挑选两个数做运算，和未被挑选的数形成新的数组），最终数组只剩下一个数，即为运算结果。
- 该问题的难点之二在于如何**在程序对于每种牌面不是按照特定的数字顺序运算**的情况下实现结果的输出。本程序采取的方式是，把上一条所述算法运行过程中所有的数（包括初始牌面、过程中的中间数以及结果）都与一个字符串表示的算式相对应（初始牌面对应以数字为内容的字符串，中间数对应的字符串是**从初始牌面起始的得到该中间数的算式**），在每一步运算的同时拼接字符串，最终就能得到期望的输出效果。

伪代码

```
function operate(num,pokerf,pokerl,oper,i,j,segment1,segment2,segmentmid)
//进行运算和字符串拼接，从pokerf数组生成pokerl数组。
//ij是被挑选的两个操作数，oper是运算符号
//segment1,2是两个操作数对应的字符串，segmentmid是字符串拼接结果
    switch(oper)
        case(1 to 6)    //减法和除法不满足交换律，于是两个数的运算有6种情况
            step ← operation result of pokerf(i) & pokerf(j)    //运算
            segmentmid ← "the combination of segment1, operator & segmemt2"    //拼接字符串
    end
    pokerl ← step,poker(1 to num except i & j)    //把运算结果和pokerf中没有用到过的数字存到pokerl中
end

for(i = 1 to 4) do    //第一轮运算：数组中有4个数字
    for(j = i+1 to 4) do
        for(k = 1 to 6) do  //两个数的6种运算
            segment1,segment2 ← string(poker1(i)),string(poker1(j))   //生成两个操作数对应的字符串
            operate(4,poker1,poker2,k,i,j,segment1,segment2,segmentmid1)    //运算与字符串拼接并生成新数组

            for(m = 1 to 3) do    //第二轮运算：数组中有3个数字
                for(n = m+1 to 3) do
                    for(p = 1 to 6) do  //两个数的6种运算
                        if(poker2(m) is the variable 'step' in the last called function 'operate()')
                            then segment1 ← segmentmid1
                            //若这一轮选中的第一个操作数poker2(m)是上一轮的运算结果而非初始牌面中的数，
                            //则其字符串应为上一轮的拼接结果
                        else
                            segment1 ← string(poker2(m))
                        end
                        segment2 ← string(poker2(n))
                        operate(3,poker2,poker3,p,m,n,segment1,segment2,segmentmid2)

                        for(c = 1 to 6) do    //第三轮运算：数组中有2个数字；两个数的6种运算
                            segment1 ← segmentmid2    //第一个操作数对应的字符串是上一轮的拼接结果
                            if(poker2(m) is the variable 'step' in the last called function 'operate()')
                                then segment2 ← string(poker3(2))
                            else
                                segment2 ← segmentmid1
                            end
                            operate(2,poker3,poker4,c,1,2,segment1,segment2,segmentmidfinal)
                            if(poker4(0)==24)
                                then print segmentfinal    //判断并输出
                            end
                        end

                    end
                end
            end

        end
    end
end
```

I/O示例

```
lbx@lbx-virtual-machine:~/compu_phys/week1/TwentyFourPoints_dir$ gfortran -c TwentyFourPoints.f90
lbx@lbx-virtual-machine:~/compu_phys/week1/TwentyFourPoints_dir$ gfortran TwentyFourPoints.o -o TwentyFourPoints
lbx@lbx-virtual-machine:~/compu_phys/week1/TwentyFourPoints_dir$ ./TwentyFourPoints
 Do you want to input by yourself or generate randomly?(I/R)
I
 Please input four integers in the range of [1,13]:
1 5 5 5
 ((   5.00000000-(   1.00000000/   5.00000000))*   5.00000000)=   24.0000000
 ((   5.00000000-(   1.00000000/   5.00000000))*   5.00000000)=   24.0000000
 ((   5.00000000-(   1.00000000/   5.00000000))*   5.00000000)=   24.0000000
 ((   5.00000000-(   1.00000000/   5.00000000))*   5.00000000)=   24.0000000
 ((   5.00000000-(   1.00000000/   5.00000000))*   5.00000000)=   24.0000000
 ((   5.00000000-(   1.00000000/   5.00000000))*   5.00000000)=   24.0000000
 Do you want to continue?(y/n)
y
 Do you want to input by yourself or generate randomly?(I/R)
I
 Please input four integers in the range of [1,13]:
4 4 10 10
 (((   10.0000000*   10.0000000)-   4.00000000)/   4.00000000)=   24.0000000
 (((   10.0000000*   10.0000000)-   4.00000000)/   4.00000000)=   24.0000000
 Do you want to continue?(y/n)
y
 Do you want to input by yourself or generate randomly?(I/R)
I
 Please input four integers in the range of [1,13]:
6 9 9 10
 ((   10.0000000/(   6.00000000/   9.00000000))+   9.00000000)=   24.0000000
 (((   9.00000000/   6.00000000)*   10.0000000)+   9.00000000)=   24.0000000
 ((   10.0000000/(   6.00000000/   9.00000000))+   9.00000000)=   24.0000000
 (((   9.00000000/   6.00000000)*   10.0000000)+   9.00000000)=   24.0000000
 ((   9.00000000/(   6.00000000/   10.0000000))+   9.00000000)=   24.0000000
 ((   9.00000000/(   6.00000000/   10.0000000))+   9.00000000)=   24.0000000
 (((   10.0000000/   6.00000000)*   9.00000000)+   9.00000000)=   24.0000000
 (((   10.0000000/   6.00000000)*   9.00000000)+   9.00000000)=   24.0000000
 (((   9.00000000*   10.0000000)/   6.00000000)+   9.00000000)=   24.0000000
 (((   9.00000000*   10.0000000)/   6.00000000)+   9.00000000)=   24.0000000
 Do you want to continue?(y/n)
n
```

输入内容参考维基百科-24点中较有难度的几种初始情况，依次分别为在过程中有分数出现、过程中涉及较大的数字、以及过程中涉及奇数的情况，可以看到程序均给出了全面但有所重复的结果。经过与网络上的24点求解器对比，本程序的重复度也在一个较低的水平。

源代码

完整源代码见附件。

```
real poker1(4),poker2(3),poker3(2),poker4(1)
character(len=100) segment1,segment2,segmentmid1,segmentmid2,segmentfinal
```

这是源代码第5行的定义语句。数字均定义为实数是因为整数除法精度不够，四个实数数组分别为3轮运算的初始数组和生成数组；`segment1`和`segment2`分别为每轮运算两个操作数对应的字符串，`segmentmid1`，`segmentmid2`和`segmentfinal`分别为三轮运算过程的表达式字符串。

```
do i=1,4
    call random_number(temp)
    poker1(i)=1+int(temp*13)
end do
```

这是源代码第14行随机生成初始牌面的代码。`random_number()`这一外部例程只能生成[0,1)的随机数，我们把该区间分成13份，以此生成1-13的整数随机数。

```
segment1=trim(segmentmid2)
if(m==1) then
    write(segment2,*)poker3(2)  !第一个操作数对应的字符串是上一轮的拼接结果
else
    segment2=trim(segmentmid1)
end if
```

这是源代码第41行初始化第三轮运算两个操作数对应的字符串的代码。第一个操作数的字符串必为第二轮调用`operate()`的运算结果而非初始牌面中的数字，于是`segment1`赋为`segmentmid2`；若第二轮的操作数中包含第一轮调用`operate()`的运算结果而非均为初始牌面中的数字，则第三轮的第二个操作数为初始牌面中的的字符串，`segment2`赋为`poker3(2)`，否则第三轮的第二个操作数为第一轮调用`operate()`的运算结果，`segment2`应赋为`segmentmid1`。