《程序设计进阶与实践》实验报告

实验名称：24点问题的求解

姓名：吴越 学号：PB21000004 日期：2022.4.16

实验环境：CPU :Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz;内存：4.00 GB;操作系统：Windows 10;软件平台：VS code

1.问题分析与求解思路：

问题：对于给定的四个正整数，判断是否存在通过加减乘除四则运算得到结果24

思路1：穷举

一个简单的思路是采用穷举法，也就是把四个数字的所有运算式进行尝试计算:
一是四个整数位置的排列，二是四个运算符的变化，三是四个运算符的优先级，就是括号位置的变化。将这些情况结果全部枚举出来，再判断是否为24。

思路2：回溯（递归）

首先将思路打开，将问题抽象化：给定m个数，判断这m个数能否组成n?

对于这个问题我们发现：判断m个数能否组成n，其实就是判断其中一个数能否与其他m-1个数运算结果组成n。而其他m-1个数运算结果，又可以通过其中一个数能否与其他m-2个数运算结果组成……通过这样分析可以发现，这满足递归算法所适用的条件。

回溯（递归）的思路便是把大的问题变小，每一步将原先要求解的问题分解成与原问题相似的规模较小的问题更小的问题，直至最后达到一个满足退出条件的简单问题，这样再一层层向上返回，便解决了原先的大规模的问题。在代码实现中，程序不断调用自身，每次调用都使问题规模变小，最后达到退出条件一层层返回直至解决原先的问题。

回到具体问题：用四张牌求出24点，其实就是判断其中一个数能否与其他3个数运算结果组成n，也就是有加减乘除四种运算方式，判断其是否为24即可。再扩展到三个数：我们可以从其中任意挑选两个数(三种挑选方式)，例如选中a，b。之后通过加减乘除计算得到a与b组合后的数d。那么d确定了之后，问题便变成c与d两个数算24点的两数问题，这又回到了第一种情况。同理可以扩展到4张牌。那么总体思路也就很容易得到了：

(1)递归函数作用：判断当前几个数能否组成24点。

(2)递归的终止条件：当只剩下一个元素时，即为四个数的运算结果，我们判断其是否为24，返回判断结果。

(3)递归的操作：首先从集合中取出两个数，将尝试四种运算（加减乘除）得到的结果与剩下的数构成新的集合，对这个新的集合，递归调用自身，再次判断当前几个数能否组成24点。若当前数不能组成24点，那么我们将这个集合复原**(回溯)**，然后再次取两个数，直到遍历完所有情况为止。

总体的代码结构也与基本思路类似，但是要注意一些特殊情况（如除数为0等）

本实验中我们采用**回溯**解法。

2.核心代码说明：

(1)24点判断函数

```
bool Point24::JudgePoint24(vector<int>& cards) {//函数功能：判断能否组成24点
    vector<double> list;                        //将元素转化为浮点数储存
    for(int i = 0;i < cards.size();i++){
        list.push_back((double)cards[i]);       //将元素转化为浮点数压入列表
        answer.push_back(to_string(cards[i])); //将元素转化为字符串压入答案
    }
    return ListCanMake24(list);                 //判断列表能否组成24
}
```

算法分析： 这是程序的主体函数。我们将输入的几个整数转化为浮点数储存入list，将元素转化为字符串形式压入答案。这是由于在某些数的组合中，要通过分数运算来凑得24点（如：(5X(5-(1/5)))=24，(8/(3-(8/3)))=24 等）因此，我们要通过浮点数来计算这些特殊情况。此外，用字符串储存答案表达式是由于字符串类型易于操作，可以使用+来连接各字符串，从而达到与列表操作的一致性，更加方便运算。

(2)24点递归函数

```
bool Point24::ListCanMake24(vector<double>& list) {
    if (list.size() == 1)   //递归终止条件：当只有一个元素时，即为运算结果
        return IsPoint24(list[0]);          //如果是24，返回真
    for(auto num : list){
        double num1 = list[0];
        list.erase(list.begin());           //取出列表第一个数
        for (auto num : list) {
            double num2 = list[0];          //取出列表第二个数
            list.erase(list.begin());
            for (int i = 0;i < 4;i++) {
                double nNum;                //储存两数运算结果
                switch (i) {...}            //遍历四则运算(省略)
                list.push_back(nNum);       //将运算结果放入列表
                if (ListCanMake24(list))
                    return true;            //开始递归，判断此时列表能否组成24点
                list.pop_back();            //若不能组成，将列表复原
            }
            list.push_back(num2);           //复原列表
        }
        list.push_back(num1);               //复原列表
    }
    return false;
}
```

算法分析：

当(list.size() == 1)，即只剩下一个元素时，该元素即为四个数的运算结果，我们判断其是否为24，返回判断结果。这便是递归的终止条件。否则，我们使用两个for(auto num : list)用于遍历从集合中取出两个数的所有情况。将这两个数遍历四则运算可能的结果后放回列表，再递归调用函数本身，判断规模较小的问题，直至最后达到一个满足退出条件的简单问题后，再一层层向上返回，便解决了原先的大规模的问题。要特别注意，每遍历完一种情况，我们都要将这个集合复原**(也就是实现回溯)**，否则第二次遍历与之前状态不同，就会导致错误的结果。

(3)24点结果判断

```
bool Point24::IsPoint24(double num) {
    if (abs(num - TARGET) < EPSILON) {
        return true;            //当元素与24相差1e-6时，可以认为相等
    }
    return false;
}
```

算法分析：

由于我们在运算中采用了浮点数来储存运算结果，最终结果与24间有浮点数误差。因此在判断结果是否为24时，不能简单的用num==24来判断。实际上，当两数差的绝对值小于EPSILON（可容忍的区间，一般为1e-6)时，我们就可以人为两数相等。这一点在判断num2是否为0时也要特别注意。若简单的用（num!=0)来判断num是否为0，则会得到错误的结果，因此在判断浮点数的相等问题时要格外注意。

3.测试，运行与分析：

1.输入2 3 10 11 判断不能构成24点，输出false

```
2 3 10 11
false
```

2.输入1 9 12 13判断不能构成24点，输出false

```
1 9 12 13
false
```

3.输入5 5 5 1判断能构成24点，输出true并输出一组解

```
5 5 5 1
(5X(5-(1/5)))
true
```

4.输入3 3 8 8判断能构成24点，输出true并输出一组解

```
3 3 8 8
(8/(3-(8/3)))
true
```

省略了更多的测试用例，参考 在标准的扑克24点中，哪些牌组永远无法组成24点 可以判断结果正确。

4.备注

1.关于程序的扩展应用：

在本实验中，我们要求用四个数生成24点。实际上，通过修改宏定义中的参数，我们完全可以实现其他问题的解决（如"100点问题"），还可以使用更多的数（如5个数24点问题），而完全不用修改程序实际的代码，这就体现了该程序复用性强，可用于解决多种问题。若使用穷举法则需要修改一些代码，这也体现了递归算法在本题的优越性。

2.关于其他的解决方式：

在本实验中，由于本实验只要求判断是否能组成24点，针对此问题我们采取了回溯解法，但事实上，此题也可稍作拓展（如要求给出所有解），由于回溯法在探测到一组解后直接返回，因此要做较大改动。此时我们也可以采取其他算法，如 穷举法 等。这样可以成功获得全部的解，应用范围更广。

总结：

通过本次24点问题实验，我掌握了：

(1) C++语言文件读取写入的优化算法。
(2) 理解了回溯，递归的算法，加深了对递归的理解。
(3) 提高了编程与debug能力，加强了对C++语言的理解认识。
(4) 有了初步进行单元测试的能力。
(5) 提高了分解，抽象，概括问题的能力，提升了计算思维。

当然实验还有许多能改进的地方，比如在产生解时，我们不断清理与加入向量组list，且每遍历完一种情况，还要将整个向量都复原，消耗量大量时间。此外，在构造表达式时，表达式与list同步转化，耗费了更多的时间，显然，本算法还有很大的优化空间。

另附：

*交互方式：

输入四个数（用空格分开）并回车

若四个数能构成24点，则输出true，并且输出可行的一组解

若四个数不能能构成24点，输出false

源代码文件：

24test.cpp

参考资料：

1. https://www.zhihu.com/question/329559720
2. https://blog.csdn.net/weixin_43700732/article/details/108957268
3. https://blog.csdn.net/qq_42403733/article/details/82925006
4. https://blog.csdn.net/weixin_34236869/article/details/90126140
5. https://blog.csdn.net/LingXi__Y/article/details/74002500