魔方大作业：代码写作过程分析

组号：5　　组长：常晟 2017080064　　组员：陈育凯 2017080066

一．实验思路

本组的魔方解法紧扣参考材料的解法。本组的代码可分成一下几部分：转魔方函数（Moves Functions）, 辅助函数 （Utility Functions）, 解法步骤函数（Algorithm Functions）

二．代码实现

数据组织方案

本组选择以二维的’char’数组收藏魔方的坐标。二维数组为rubic[6][10]，每个一维数组代表魔方的某一面，每个相应的二维数组代表其面的坐标。每个坐标的char是某个颜色的首个大写字母，例如，红色（red）的坐标以R表出、蓝色（blue）的坐标以B表出等。

转魔方函数

显然，本组把十一个的魔方转法写成一一对应的函数：R, D, L, B, F, U，Ri，Di，Li，Bi，Fi 和 Ui。由于每个的转法的逆转法是相反的，为了缩短代码以及减少编译时间，每个逆转法写成相应的转法的三倍。详细而言，本组把每个转法分成两部分：一、turn_clockwise的函数把边上的面的坐标按顺时针方向旋转。二、其余的函数把边缘的面的坐标转换。

```
22
23 void turn_clockwise(char rubic[6][10],char RUB[6][10],int opnum)
24 {
25     //swap sides clockwise
26     rubic[opnum][8]=RUB[opnum][2];
27     rubic[opnum][7]=RUB[opnum][5];
28     rubic[opnum][6]=RUB[opnum][8];
29     rubic[opnum][3]=RUB[opnum][7];
30     rubic[opnum][0]=RUB[opnum][6];
31     rubic[opnum][1]=RUB[opnum][3];
32     rubic[opnum][2]=RUB[opnum][0];
33     rubic[opnum][5]=RUB[opnum][1];
34 }
```

辅助函数

为了实时记录魔方的坐标，本组写了getcorner和getedge的函数，分别引入魔方的角落坐标（corners）以及边缘坐标（edges）。

接着，为了辅助编程排除故障过程，本组写了print_corners与print_edges函数，为了可视化并显示魔方的最新坐标。

其次，为了测试转法列表（输入）正确，test_moves函数引入原本的魔方坐标，并对其进行列表相应的转法操作，再显示魔方的坐标结果。从此可见结果是否正确，并认证转法列表无错误。

最后，由于以上所述的逆转法表示为相应转法的三倍，本组必须把转法的列表写成两步。首先，当程序在执行魔方解法的步骤函数时，把转法现存储于presteps数组，但是，presteps组数只包括正转法（R，D, L, B, F, U）。然后，在print_moves函数，本组把presteps组数简化并存储于actualsteps组数。其操作是找连续重复三次的转法转化为转法的逆转法。最终的输入为actualsteps组数。（在print_moves当中，我们主要是先调用solve，这个函数就是用来解魔方的一个函数。然后之后就会在循环里查找一下prestep里有没有出现连续三个一样的步骤，如果有的话把那三个步骤直接改成一个逆向的步骤。比如说数组里有一个地方出现连续三个R的话，直接换成RI 就好了，然后把这些新的步骤放到actualstep当中。然后在主程序里就用cin输入然后直接调用print_moves就好了。）

```
corner getcorner(char rubic[6][10])
{
    corner ans;
    //front right top
    ans.cor[0][0]=rubic[0][2];
    ans.cor[0][1]=rubic[3][0];
    ans.cor[0][2]=rubic[4][8];
    //right back top
    ans.cor[1][0]=rubic[3][2];
    ans.cor[1][1]=rubic[1][0];
    ans.cor[1][2]=rubic[4][2];
    //back left top
    ans.cor[2][0]=rubic[1][2];
    ans.cor[2][1]=rubic[2][0];
    ans.cor[2][2]=rubic[4][0];
    //left front top
    ans.cor[3][0]=rubic[2][2];
    ans.cor[3][1]=rubic[0][0];
    ans.cor[3][2]=rubic[4][6];
    //front right down
    ans.cor[4][0]=rubic[0][8];
    ans.cor[4][1]=rubic[3][6];
    ans.cor[4][2]=rubic[5][2];
    //right back down
    ans.cor[5][0]=rubic[3][8];
    ans.cor[5][1]=rubic[1][6];
    ans.cor[5][2]=rubic[5][0];
    //back left down
    ans.cor[6][0]=rubic[1][8];
    ans.cor[6][1]=rubic[2][6];
    ans.cor[6][2]=rubic[5][6];
    //left front down
    ans.cor[7][0]=rubic[2][8];
    ans.cor[7][1]=rubic[0][6];
    ans.cor[7][2]=rubic[5][0];
    return ans;
}
```

```
edge getedge (char rubic[6][10])
{
    edge ans;
    //front top
    ans.ed[0][0]=rubic[0][1];
    ans.ed[0][1]=rubic[4][7];
    //right top
    ans.ed[1][0]=rubic[3][1];
    ans.ed[1][1]=rubic[4][5];
    //back top
    ans.ed[2][0]=rubic[1][1];
    ans.ed[2][1]=rubic[4][1];
    //right top
    ans.ed[3][0]=rubic[2][1];
    ans.ed[3][1]=rubic[4][3];
    //front right
    ans.ed[4][0]=rubic[0][5];
    ans.ed[4][1]=rubic[3][3];
    //right back
    ans.ed[5][0]=rubic[3][5];
    ans.ed[5][1]=rubic[1][3];
    //back left
    ans.ed[6][0]=rubic[1][5];
    ans.ed[6][1]=rubic[2][3];
    //left front
    ans.ed[7][0]=rubic[2][5];
    ans.ed[7][1]=rubic[0][3];
    //front down
    ans.ed[8][0]=rubic[0][7];
    ans.ed[8][1]=rubic[5][1];
    //right down
    ans.ed[9][0]=rubic[3][7];
    ans.ed[9][1]=rubic[5][5];
    // back down
    ans.ed[10][0]=rubic[1][7];
    ans.ed[10][1]=rubic[5][7];
    // left down
    ans.ed[11][0]=rubic[2][7];
    ans.ed[11][1]=rubic[5][3];
    return ans;
}
```

解法步骤函数

魔方的解法总共有八个步骤。step0到step7，他们的作用是完成第一步到第七步的步骤。

在step0和step1里，我们首先会检查顶面是否形成了白的十字架，没有的话就执行while语句。while语句里，我们会首先检查第一层有没有白色的，如果有白色方块而且上面的十字架还没完全拼好的话，把那个白色块转上来，然后检查中间层，然后检查最后一层和最后一面，一直到形成白色十字架为止。

step2是用来拼完顶面的白色。我们先检测一下上面是否已经全是白色方块，如果不是的话，solve=0，执行while语句。while语句的前半部分就是查找魔方中的白色块在哪里，然后把那些白色块转上去。然后后面的for语句是用来纠正一下放错位置的白色块。

step3的功能就是把第二层拼完。step3的算法要比别的要相对简单一点，step3中，我先检验第二层是否已经拼完，如果没有的话，就开始执行while语句。while语句里就是会找符合这个位置上的edge的颜色，然后按照说明书的步骤来把那个颜色放到这个正确的位置上。执行这个while语句直到第二层跟第一层有一样的颜色而且已经变成拼好的状态。每个while循环的下面都有一次检验的循环，检验一下是否已经拼好，如果拼好的话直接把solve的值变为0从而可以跳出while循环。接下来就是step4。

step4就是用来拼地面的十字。函数一开始跟别的函数一样，检验一下是否拼完，没有的话执行while语句来把十字拼完。我在for循环里找哪里不是黄色的edge，然后按照说明书上的步骤来转魔方，一旦转好后，直接跳出for循环。之后while循环结束之前，会检验一下是否拼完，如果没有的话继续执行while循环直到solve的值为1。

step5，step6，step7针对解魔方的第三层。

首先，step5完成黄色的面。本函数先检查黄色面的四角在面上的颜色是否是黄色，并对非黄色的角落做相应的转置操作。

然后，step6针对第三层角落的边缘。

最后，step7针对第三层中间的边缘。

结语

人类在解魔方的时候，经常会变换魔方的握法，从不同的视角看待及解魔方。然而，为了使程序最优化，本组决定以固定的视角看待魔方。虽然其方案的难度较高，这使得程序的执行表现大幅度提升，并符合测试的运行时间和占用空间要求。同时，通过程序个部分的模块化，如turn_clockwise、解法步骤的循环机构等，本组把代码减至最低数量，在1150行以内把程序完成。

参考风格分 100

测试点编号	状态	运行时间	占用空间
0	评测通过 (25)	1 ms	4240 KB
1	评测通过 (25)	1 ms	4240 KB
2	评测通过 (25)	1 ms	4240 KB
3	评测通过 (25)	1 ms	4240 KB

三. 实验分析与讨论

我们一开始在写这个代码的时候，我们不知道怎么把电脑的思考过程跟我们人的思考过程同等起来。比如说，当我转魔方时，我会找我想要的颜色，然后按照那个颜色块的位置来转魔方。但是我们发现把这个过程教给电脑的话是一件很难的事情。虽然拼魔方对于我们来说是一件蛮容易的事情因为我们有一些公式和步骤可以用，然后我们也不用考虑到转魔方时会影响到别的哪些颜色等等。但是在写代码时就不一样，要像一个老师教小朋友一样，一步一步解决这些问题。所以我们在魔方上面标记了各自的坐标，然后我们按照这些坐标的转换规则，把U,R,L,D等规律写到代码里面。然后我们也发现写魔方的代码的话没有简单的方法，只能用暴力的方法来教会电脑如何拼魔方，所以我们在代码里面有许多不同的方案执行不同的步骤等等。

四. 实验收获

我们写这个代码的时候，我们将尽花了差不多一个星期的时间，我们在上课的时候在不断研究魔方规律，有时候debug时很愤怒因为我们实在找不到bug，有时候会很有一种满足感因为我们成功过了一步一步的难关等等。徐明星老师在第一堂课的时候跟我们说过“计算机其实很笨，你要写很具体而且很复杂的代码才能教会计算机如何操作，但是一旦教会了以后，就会变成一个很powerful的东西”。我写完这次大作业以后我才意识到老师说的这句话原来是这个意思。我在写代码的一开始，计算机什么都不知道所以我要像一个老师一样一步一步的去教它，但是一旦教会了以后就会是一个非常强大的东西。还有一个收获就是我发现只要给我们充分的时间的话，我们都能够完成任何复杂的程序。

五. 小组分工

常晟：写了跟转魔方有关的函数以及辅助函数和算法思路等。
陈育凯：写了step0到step7的函数。