CS 4300: Artificial Intelligence

Assignment: Rubik’s Cube Class

This assignment is the first in a series. By the end of the series, you will have created a program that can solve the 3x3x3 Rubik’s Cube from any starting state.

This assignment requires you to create a C++ class to represent the cube and provide some basic operations on the cube. Additionally, you will be required to write a wrapper program that can be used in acceptance testing of the class.

Required Class Elements

- protected or private data member(s) to represent the state of the cube. We will be using the layout shown in this image.
- public method to allow any of the 12 legal rotation moves. For each of the 6 faces, you can rotate the cube clockwise or counter-clockwise.
- \texttt{bool operator== (const RubiksCube& rhs) const}, comparison operator that returns true if the cubes have the same state.
- \texttt{bool operator< (const RubiksCube& rhs) const}, comparison operator that returns true if the current cube is less than the parameter cube.
- If your class uses dynamic memory, you must provide a copy constructor, an assignment operator and a destructor.
- Use \texttt{enum} or other constants where appropriate. \textit{No magic values!}

Required Wrapper Program

The wrapper program’s purpose is to facilitate a collection of acceptance tests. The program will read directly from standard input and write directly to standard output. An input file will have the following format:

```
initial
white white white
white white white
white white white
green green green
green green green
green green green
red red red
red red red
red red red
blue blue blue
blue blue blue
blue blue blue
orange orange orange
orange orange orange
orange orange orange
yellow yellow yellow
yellow yellow yellow
yellow yellow yellow
rotate red cw
rotate blue cw
rotate green ccw
rotate green ccw
rotate done
show
isequal
white white white
```
<table>
<thead>
<tr>
<th>white</th>
<th>white</th>
<th>white</th>
</tr>
</thead>
<tbody>
<tr>
<td>white</td>
<td>white</td>
<td>white</td>
</tr>
<tr>
<td>green</td>
<td>green</td>
<td>green</td>
</tr>
<tr>
<td>green</td>
<td>green</td>
<td>green</td>
</tr>
<tr>
<td>green</td>
<td>green</td>
<td>green</td>
</tr>
<tr>
<td>red</td>
<td>red</td>
<td>red</td>
</tr>
<tr>
<td>red</td>
<td>red</td>
<td>red</td>
</tr>
<tr>
<td>red</td>
<td>red</td>
<td>red</td>
</tr>
<tr>
<td>blue</td>
<td>blue</td>
<td>blue</td>
</tr>
<tr>
<td>blue</td>
<td>blue</td>
<td>blue</td>
</tr>
<tr>
<td>blue</td>
<td>blue</td>
<td>blue</td>
</tr>
<tr>
<td>orange</td>
<td>orange</td>
<td>orange</td>
</tr>
<tr>
<td>orange</td>
<td>orange</td>
<td>orange</td>
</tr>
<tr>
<td>orange</td>
<td>orange</td>
<td>orange</td>
</tr>
<tr>
<td>yellow</td>
<td>yellow</td>
<td>yellow</td>
</tr>
<tr>
<td>yellow</td>
<td>yellow</td>
<td>yellow</td>
</tr>
<tr>
<td>yellow</td>
<td>yellow</td>
<td>yellow</td>
</tr>
</tbody>
</table>

The first set of paragraphs describe the initial state of the cube. Note that the example is a solved cube. However, this will not be the case in all acceptance tests.

The next paragraph describes rotations in the desired order. The rotate command will describe the face to rotate by the color of the center tile. The direction will be clockwise (cw) or counter-clockwise (ccw). The `done` command will signal the end of rotation.

The show command indicates the program should show the current state. The faces should be shown in the order and format given in the example: white, green, red, blue, orange, yellow.

Finally, the `isequal` command indicates that another cube state will be described. The program should read it in, and compare to the current state with the `==` operator. The program must write `TRUE` or `FALSE` to the output.

**Passoff**

Submit your source code to Canvas along with a Makefile that will build it on an Ubuntu 16.04 system with the proper development environment installed. Your executable must be named `rubik_test`. Several acceptance tests will be executed. If the code passes all tests, it will be accepted. If not, it will be sent back for completion.