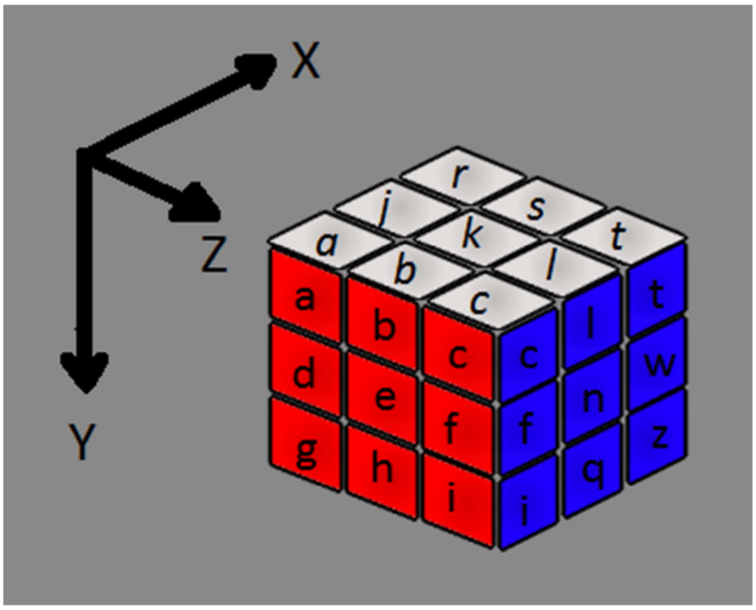
Objective

The aim of this implementation is to create a very basic Rubik's cube in C++ with a data structure which allows any user to rotate sides of the cube and play with it. It must tell the user whether the cube has been solved or not.

Data Structure

A cube of N*N*N elements has been modelled as a 3-dimensional matrix of small objects called "cubelets" or "cubies". Each of these cubelets corresponds to one of the small pieces that compose the real cube. In the case of a 3*3*3 cube, the model contains 27 cubelets.



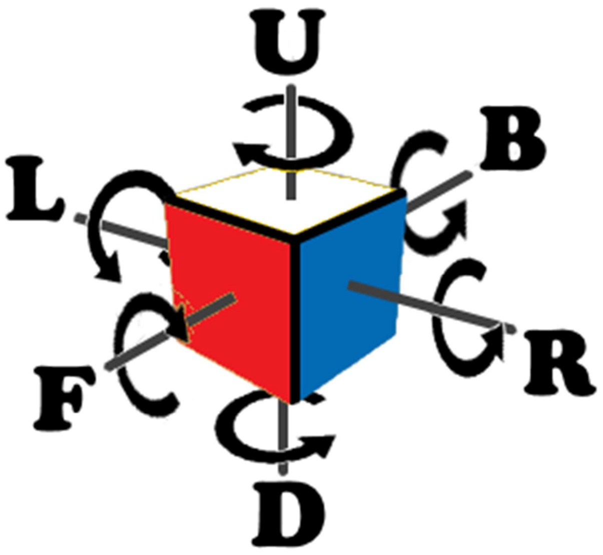
The axis X , Y and Z shown in the picture help to locate the position of each cubelet with the typical [X][Y][Z] approach. So for example if we assign a letter to each cubelet, cubie "a" is located at [0][0][0] while "h" is at [0][2][1] and "w" at [2][1][2].

Each of these cubelets is an object created as an array of 3 characters {colourX ; colourY ; colourZ}. The colours in the array correspond to the colour visible of each cubelet when looked from the axis X, Y and Z respectively. We need to take into account that we do not need to model the faces that are not shown and therefore with 3 dimensions is enough. The cubie does not care if shown in the face at the top or the one at the bottom of the Rubik's cube, they both are perpendicular to axis Y.

Therefore cubie "c" will be {red,white,blue}. If the cubie contains only 1 or 2 colours the array is filled with a dash characters. So cubelet "f" is{red, - ,blue} and cube "n" { - , - , blue}.

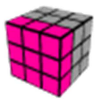
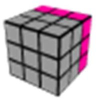
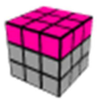
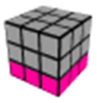
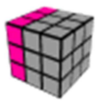
Rotations

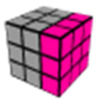
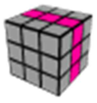
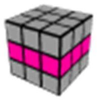
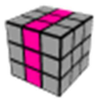
The rotations allowed in the cube have been implemented using the typical Rubik's nomenclature. Each of the faces receives a name corresponding to its position relative to the XYZ axis.



- Face F = Front, perpendicular to axis X
- Face B = Back, perpendicular to axis X
- Face U = Up, perpendicular to axis Y
- Face D = Down, perpendicular to axis Y
- Face L = Left, perpendicular to axis Z
- Face R = Right, perpendicular to axis Z

This implementation allows very easy movements by just rotating a face or layer which is in fact only a rotation into a 3x3 matrix of the cubelets corresponding to each face. Also it is necessary to rotate the colours inside each cubelet.

- Movement in layer F: 
- Movement in layer B: 
- Movement in layer U: 
- Movement in layer D: 
- Movement in layer L: 

- Movement in layer R: 
- Movement in layer S (layer Standing, between the Front and the Back layers): 
- Movement in layer E (layer Equator, between the Up and Down layers): 
- Movement in layer M (layer Middle, between the Left and the Right layers): 

For each of these movements, the rotation can be clockwise or counterclockwise. The clockwise / counterclockwise directions are always considered corresponding to the view of each face from outside of the cube. This means that rotating F clockwise is a rotation on the opposite direction of rotating B clockwise.

Display

For the case of a 3x3x3 cube, a 2D unrolled view of the cube has been implemented to show the colours in each face of the cube. This allows the program to easily run in any OS without the need of external libraries, but it has two main drawbacks:

1. No 3D view of the cube so the user can take some time to get used to it.
2. No colours showed in the console view. Instead of colours each cubelet show a character corresponding to the colour in that face:
    - R: Red
    - O: Orange
    - White: White
    - Y: Yellow
    - G: Green
    - B: Blue

Also the software shows a menu with the list of options for the user to choose. These options are:

- Exit the program.
- Rotate clockwise or counterclockwise any of the faces / layers.
- Print again the explanation of the 2D view.

After every movement, the software checks if the cube has been solved or not. This function checks that all the colours in the 6 faces are the same; it does take into account that all the layers could have been rotated in any of the X/Y/Z axis and the cube could still be solved even when face F has a different colour from the one at the starting position.