C PROJECT
"The Rubik's Cube"

Please read the subject carefully and entirely before you start the implementation.

1 Preamble & Objective

The Rubik's Cube is a 3-D combination puzzle invented in 1974 by Hungarian sculptor and professor of architecture Ernő Rubik. Marketing in toy stores in his country begins in 1977. The growing popularity of this game in Hungary allows the Rubik's Cube to be sold worldwide in 1980 : One hundred million Rubik's Cube were sold between 1980 and 1982. Since then, the Rubik's Cube has won the title of perfect puzzle, and today more than two hundred million copies have been sold.

However, far from the competitions organized around the Rubik's cube, several mathematicians find in this little toy an interesting optimization problem given the huge number of possible configurations if we had to find a solution at random. Thus, several resolution algorithms have emerged.

In this project, we will discover one of the simplest solving algorithms by performing the following two steps :

1. Implementation of the Rubik's cube and its different movements ;
2. Step by step resolution of the Rubik's cube.

But before that, let's see what are the characteristics of a Rubik's Cube ...

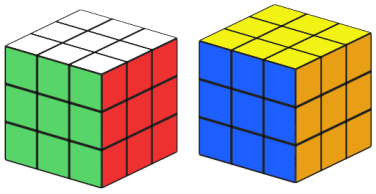
2 How the Rubik's cube works ?

There are a few standard designations for the Rubik's Cube which are essential for understanding solving methods. These standards are used in particular to represent the faces of the cube and the movements that must be produced on them.

2.1 The faces

The Rubik's cube is made up of 6 sides each having a different color : White, Orange, Yellow, Blue, Red and Green.
These colors are distributed as follows :

— Red faces Orange
— Blue faces Green
— White faces Yellow



Names have been chosen to designate each face according to its orientation with respect to the player. The following conventions are frequently found :

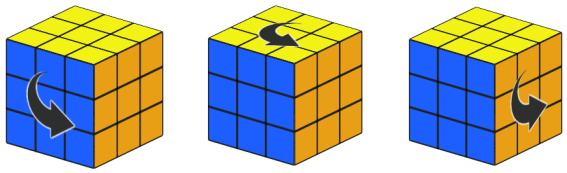
— the Blue face is called face **UP**, abbreviation **U** ;
— the White face is called face **FRONT**, abbreviation **F** ;
— the Red face is called face **RIGHT**, abbreviation **R** ;
— the Green face is called face **DOWN**, abbreviation **D** ;
— the yellow face is called face **BACK**, abbreviation **B** ;
— the Orange face is called face **LEFT**, abbreviation **L**

2.2 Movements

In our approach to solve the Rubik's cube, the entire faces will move. Indeed, each of the 6 faces can make two possible movements :

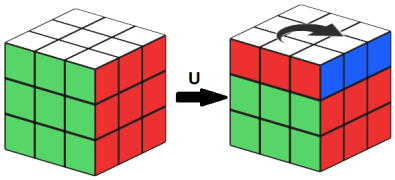
— Clockwise rotation ;

— Counterclockwise rotation. In this case, the movement will be indicated by a quote following the name of the face : U ', F', ...

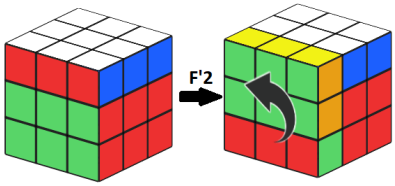


In addition, each rotation can be of two types :

— Quarter turn : make a single turn clockwise or counterclockwise.
Example : Rotate the **Up** face clockwise in a quarter-turn :

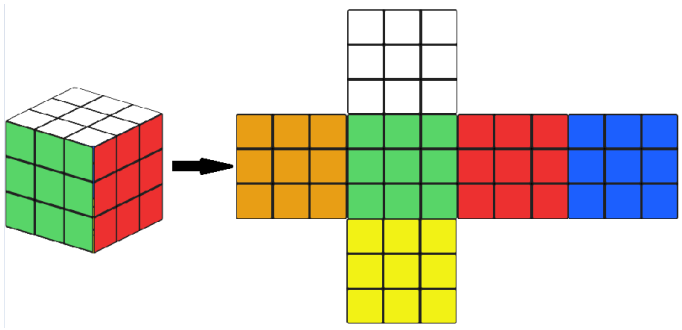


— Half turn : make two turns clockwise or counterclockwise. Half turn = 2 * Quarter turn.
Example : Rotate counterclockwise half a turn of the **Front** face :



3 Rubik's cube implementation in C language

The Rubik's cube will be represented in two dimensions as shown in the figure below :

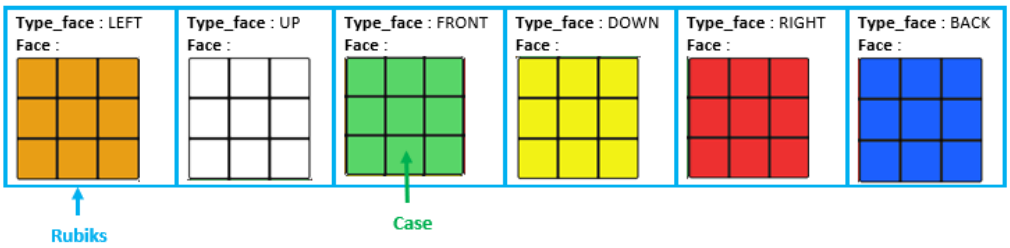


In memory, the Rubik's cube will be represented by a 3-dimensional array **"rubiks"** of size 6 corresponding to the number of its faces. Each position of this array is a structured type comprising two members :

— A 2-dimensional board of size 3 x 3 representing one side of the cube ;

— An enumerated variable of type **T_SIDE** designating the name of the face.

```
typedef enum { FRONT, BACK, UP, DOWN, RIGHT, LEFT } T_SIDE ;
```



Each cell of a face can also be of structured type containing at least one field of enumerated type **T_COLOR**.

```
typedef enum { R, B, G, W, Y, O, LG } T_COLOR ;
```

To help you handle your cube, here is the list of functions you need to implement :

3.1 Functions for the enum types

— **select_color** which takes a color in parameters and which returns its associated value.

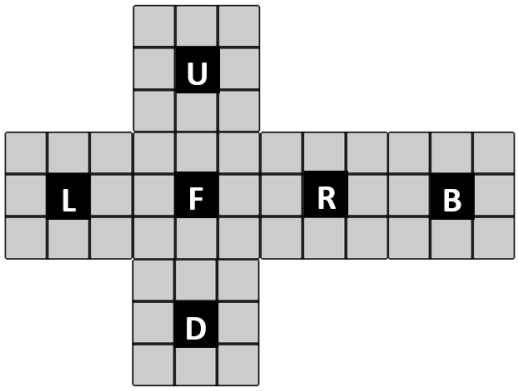
— **side_to_index** which takes a face name as a parameter and which returns its position in the rubiks array.

3.2 Cube representation functions

— **create_rubiks** which allows to create the general structure of the cube (3D array) dynamically.

— **init_rubiks** which takes a variable of type rubiks as a parameter and which initializes each face with a unique color. This function must take into account the color distribution rule described Section 2.1.

— **display_rubiks** which allows to display on the screen the Rubik's cube in 2D so that the order of the faces is as follows whatever their position in the array **rubiks**.



— **blank_rubiks** to allow all the boxes of the cube to be grayed out for possible manual initialization..

— **fill_rubiks** to allow manual assignment of colors to the different boxes. The function must ask the user to specify the face, the coordinates of the box to be modified as well as the color to be assigned. The function must be able to prevent prohibited combinations (to be determined).
Example : Two adjacent boxes (belonging to different faces) must not have the same color.

— **scramble_rubiks** to allow random movements to shuffle the cube.

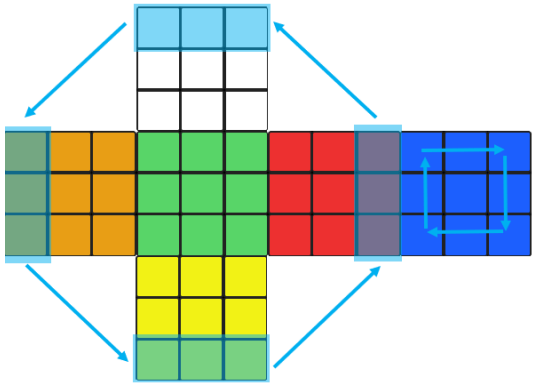
— **free_rubiks** to free the memory space occupied by the cube at the end of the program.

3.3 Movement functions

— **{side_name}_clockwise** with side_name ∈ {FRONT, BACK, UP, DOWN, RIGHT, LEFT} is a function with two parameters : the cube and the type of rotation. If type = 1, the function should perform a quarter-turn clockwise rotation on the side_name face. If type = 2, the rotation must be half a turn.

Indication :
A quarter-turn clockwise rotation of the BACK side (back_clockwise) consists in :

1. Rotate the face itself
2. Rotate the lines of index 0 of the faces around it : UP,RIGHT, LEFT, DOWN (Pay attention to the direction!!!)



— **{side_name}_anticlockwise** with side_name $\in \{FRONT, BACK, UP, DOWN, RIGHT, LEFT\}$ is a function with two parameters : the cube and the type of rotation. If type = 1, the function should perform a quarter-turn counterclockwise rotation on the side_name face. If type = 2, the rotation must be half a turn.

— **horizontal_rotation** allowing to make a horizontal rotation where the BACK face becomes FRONT (and vice versa) and the RIGHT face becomes LEFT (and vice versa).

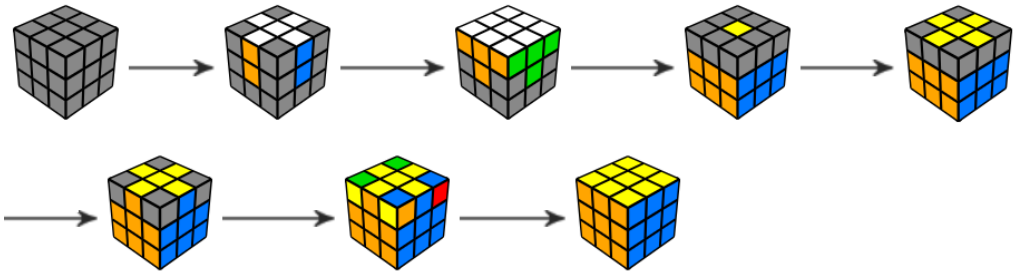
— **vertical_rotation** allowing you to make a vertical rotation where the UP side becomes DOWN (and vice versa) and the FRONT side becomes BACK (and vice versa).

— **move_rubiks** offering a menu to the user to be able to apply each of these movements manually.

4 Rubik's cube solving algorithm

To solve a Rubik's cube, several algorithms are available. The one we use here is the so-called "layer by layer" algorithm which is the simplest and the most used by beginners. It consists of the following steps :

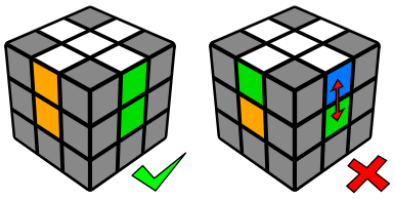
1. Realization of a perfect white cross ;
2. Realization of the white face entirely as well as the first crown of the cube ;
3. Creation of the second crown ;
4. Realization of the last face.



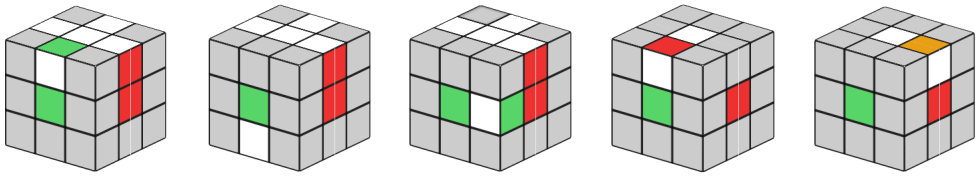
For certain positions of the boxes, and depending on the stage at which we are, algorithms exist to help move the box to its final position. However, for other cases, we do not have this advantage. Thus, for each of the steps below, you must implement the given algorithms according to the state of the cube. Then, at the end of each step, **you must offer the user the possibility to perform movements manually to complete or trigger a predefined configuration**.

4.1 Make a perfect cross

A perfect cross is one where we find the "+" sign on the UP side and where the upper edge boxes of each face have the same color as their centers.



To realize a perfect cross, apply one of the following algorithms (depending on the position of the white box to be moved) sequentially if one or more of these situations occurs. Then, propose to the user to complete it manually if necessary.

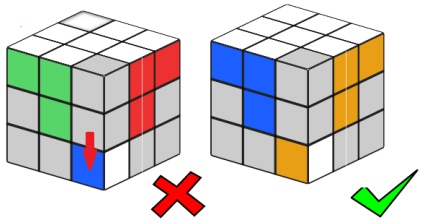


F U' R U F' R' D' R F2' R' D' R F2' F R R' F' U

4.2 Make the white corners and the first crown

This can be done in two steps :

1. Bring the corner under where it belongs. This involves testing the colors of the lower right corner (its FRONT, RIGHT, DOWN positions) :
   (a) If one of the colors in the corner is white and the other two colors match the faces around it, then go to the next step. Here, the orientation of the colors is not important.
   (b) Otherwise, you will have to make a movement on the DOWN face in order to position the corner in the right place.



2. Repeat the algorithm below until the white corner is positioned at the targeted location in the correct orientation (below are some examples of positions where the algorithm can be applied).

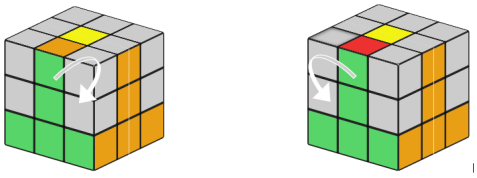
R' D' R D

4.3 Make the second crown

To solve this crown, it is necessary to use, essentially, two algorithms which we will call here : **right_move** and **left_move**. The resolution steps are as follows :

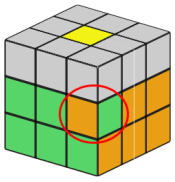
1. Make a vertical rotation of the cube : the white UP face already resolved at this stage must be in the DOWN position.
2. Bring the center edge box of the UP face to a position so that the two colors it carries match either the colors of the FRONT-RIGHT faces or the colors of the FRONT-LEFT faces.
3. Apply the algorithms **right_move** and **left_move** according to the state of the cube as shown below :

RIGHT_MOVE **LEFT_MOVE**



U R U' R' U' F' U F U' L' U L U F U' F'

If a situation of bad orientation arises, then the following algorithm must be applied :



U R U' R' U' F' U F U2 U R U' R' U' F' U F

4.4 Make the last crown

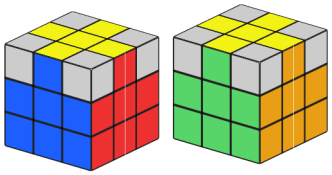
At this stage, we will aim to :

1. Make a yellow cross on the UP side
2. Make the entire yellow face

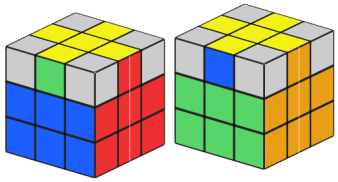
4.4.1 Make the yellow cross

At the end of the previous step, you should be in one of the situations below. Here are the algorithms to apply to reach the yellow cross.

1. **Case of a yellow cross**
   To validate a yellow cross, it must be perfect as for the white cross at the begining of the game. A perfect yellow cross is one where the 4 edges of the 4 faces are correctly placed.
   (a) If the perfect yellow cross is achieved, then go directly to the last resolution step presented in Section 4.4.2.

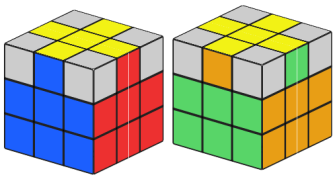


   (b) If it is a yellow cross with two well placed facing edges and two misplaced edges, then :
   — Designate as the face FRONT the one which contains one of the well placed edges and the BACK face the other face containing the other well placed edge.
   — Apply the following algorithm, then apply step c)



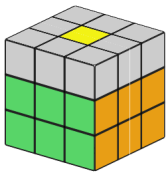
R U2 R' U' R U' R'

   (c) If it's a yellow cross with two well-placed side-by-side edges and two misplaced edges, then :
   — Designate as the FRONT face the one that contains one of the misplaced edges and the RIGHT face for the other misplaced edge. Thus, the two well placed edges will be on the LEFT and BACK faces.
   — Apply the following algorithm :



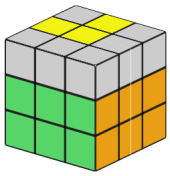
R U2 R' U' R U' R' U'

2. **Case of a single yellow box**
   Apply the following algorithm to obtain a cross then check if it is perfect.



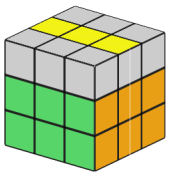
R' U' F' U F R F R U R' U' F'

3. **Case of L**
   To be able to apply the algorithms in this section, you must identify an L shape on the UP face of your cube following the same orientation shown in the figure below (an L starting with the LEFT face and oriented towards the BACK face). Once the yellow cross is obtained, check if it is perfect.



R' U' F' U F R

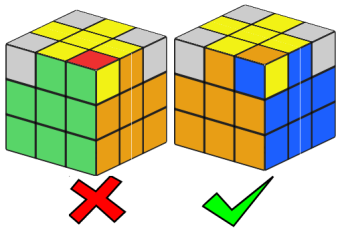
4. **Case of a yellow bar**
   To form a yellow cross from a yellow bar, the FRONT face of your cube should be the one that allows you to have the bar in front of you horizontally. According to the figure below, the two faces which allow to have the bar in front of us horizontally are GREEN and BLUE. Apply the following algorithm then check if the obtained yellow cross is perfect.



F R U R' U' F'

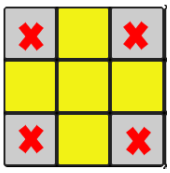
4.4.2 Place the corners correctly

Finally, all that remains is to place the corner boxes. To do this, you have to look at their positions and their orientations. A square is said to be well placed when it is at the intersection of the colors that compose it, otherwise, it is said to be misplaced.



1. If the 4 corners are well placed then, it is then necessary to apply section **??** to orient them well.

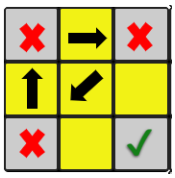
2. If the 4 corners are misplaced, then place the cube in any direction provided that the yellow boxes are oriented towards the UP side and apply the algorithm below before applying one of the two next algorithms.



L' U R U' L U R' U'

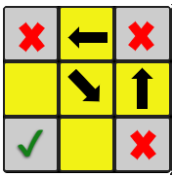
3. If only 3 corners are misplaced, then place the well-placed corner between the UP face and the FRONT face and observe which of the two situations arises :

One turn clockwise
will put them back in plac
The well-placed corner is between
the faces FRONT-UP-RIGHT



L' U R U' L U R' U'

One turn counterclockwise
will put them back in plac
The well-placed corner is between
the faces FRONT-UP-LEFT

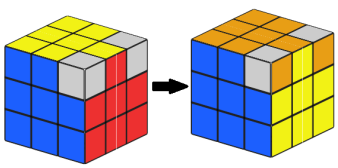


R U' L' U R' U' L U

4.4.3 Orient the corners correctly

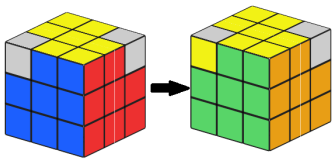
At this stage, the four corners are well placed, it will only remain to orient them. There are four possible cases :

1. Orient 2 corners side by side : Locate the two similar boxes and turn the cube so that they are oriented towards the RIGHT face, then apply the following algorithm :



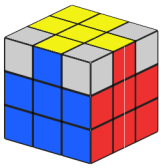
R U2 R' U' R U' R' L' U2 L U L' U L

2. Orient 2 corners diagonally : Identify the misplaced box which allows to have the yellow face in UP and which orients one of the misplaced yellow boxes towards the FRONT, then apply the following algorithm :

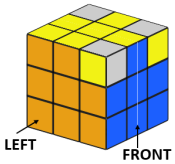


F R U2 R' U' R U' R' L' U2 L U L' U L F'

3. Orient 3 corners : In this case, you will have to pay attention to the orientation of the misplaced yellow boxes. The application of the algorithms of this level brings us to one of the two cases of the 2 mis-oriented corners. Two cases are possible :

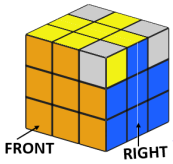


Orient a yellow box towards the LEFT side and so the other two are going to be on the FRONT and RIGHT boxes



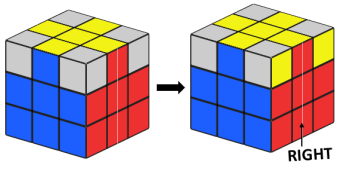
R U2 R' U' R U' R' L' U2 L U L' U L

Orient a yellow box towards the RIGHT side and so the other two are going to be on the BACK and LEFT boxes



R U2 R' U' R U' R' L' U2 L U L' U L

4. Orient 4 corners : locate two identical colors (not necessarily yellows) on the same face and designate this face as RIGHT, then apply the following algorithm which will bring you in the case of 2 corners side by side to orient.



R U2 R' U' R U' R' L' U2 L U L' U L

5 User interface

The user interface must continuously offer a menu allowing to :

1. Initialize a Rubik's Cube
2. Gray (empty) a Rubik's Cube
3. Randomly shuffle a Rubik's Cube

4. Manually assign colors to the boxes of the Rubik's Cube : A submenu must be displayed offering the user to specify :
   (a) The face to modify
   (b) The row number of the box to modify
   (c) The column number of the box to modify
   (d) The color to assign to it
5. Reset the Rubik's Cube at any time
6. Rotate the Rubik's vertically or horizontally
7. To be able to play manually by performing rotations on the faces
8. Solve the Rubik's Cube "layer by layer" :
   (a) Start by making a perfect white cross
   (b) Display the result and suggest to the user to go (or not) to the next step (realization of the white face).
   (c) Follow the same process for all the next steps, displaying the cube's state after each one, until it is completely resolved.

Display example :

```
        W W W
        W W W
        W W W

O O O   G G G   R R R   B B B
O O O   G G G   R R R   B B B
O O O   G G G   R R R   B B B

        Y Y Y
        Y Y Y
        Y Y Y

-------------------------------------------------------------------------------------
1: Scramble      2: Reset        3: Blank        4: Play         5: Fill         6: Solve
-------------------------------------------------------------------------------------
Action: 1
        Y W O
        W W Y
        R Y W

W R B   R G O   B R G   W G G
O O O   G G W   B R O   W B B
O O G   B R Y   W R O   R B B

        R G G
        Y Y B
        Y Y Y
```

6 Additional Resources

— For the display of colors, use the library "conio" ("conio.h" and "conio.c" files are on moodle) for Windows, or library "ncurse" (`https://developer.apple.com/library/archive/documentation/System/Conceptual/ManPages_iPhoneOS/man3/ncurses.3x.html`) for linux or mac OS.

— More details on how a Rubik's Cube works : `https://rubikscu.be/#cubesolver`https ://-rubikscu.be/#cubesolver

— Explanatory video : Résolution d'un Rubik's Cube

7 General guidelines

1. The source code must be divided into three files :
   — `rubiks.h` : it contains the definitions of the used sructures, enum types, constants, etc, and the prototypes of the used definition,
   — `rubiks.c` : it includes `rubiks.h` and contains the definitions of its functions,
   — `main.c` : it includes `rubiks.h` and contains the main function of your program.
2. The project is to be carried out in pairs. A single group of three students could be accepted in case the number of student in the group is odd.
3. The submission of your work must include :
   — A .zip file containing all the source files.
   — A 10 to 15 page report to containing :
      — your solutions (in algorithmic language) and your choices of data structures.
      — the encountered difficulties
      — the lessons of this project
      — prospects for improving your project
4. Any submitted file must be renamed as follows : NAME1_NAME2 (awher NAME1 and NAME2 are the names of the students that realized the project).
5. The project is to be submitted on Moodle.
6. The project must be submitted by 05/24/2021 at 11 :59 p.m. at the latest.
7. The final grade of the project will be composed of :
   (a) The code grade
   (b) The report grade
   (c) The defense grade.

8 Planning

— Friday 04/16/2021 : Project launch (publication on moodle)

— Week of 05/3/2021 : Project monitoring session

— Week of 05/24/2021 : Project defenses