CARLIER Nathan
GUILBERT Max

INT1

# C PROJECT

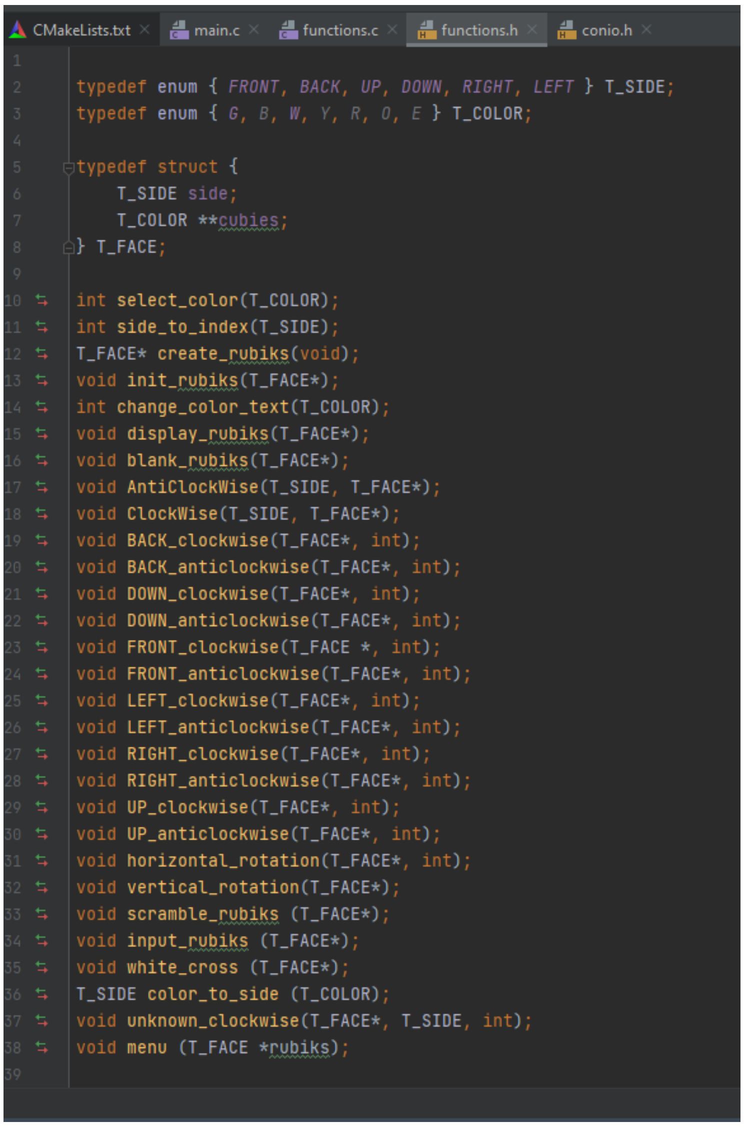
## Intro:

Here is our project to solve the Rubik's Cube in C. Our project includes almost all the functions of the specifications, and, apart from the solve function, these functions work all. We can solve the beginning of the cube but unfortunately not the end. We knew how to realize entirely the algorithm of resolution but unfortunately the lack of time and the important workload in the other subjects prevented us from finishing in time this algorithm of resolution which takes a lot of time because there are a lot of conditions to check. Actually, it does an infinite loop at the step of the perfect cross.

But apart from that we propose you a complete and functional project for its user which respects the requested conditions. We are going to explain to you how we realized this project.

## Functionality and interface:

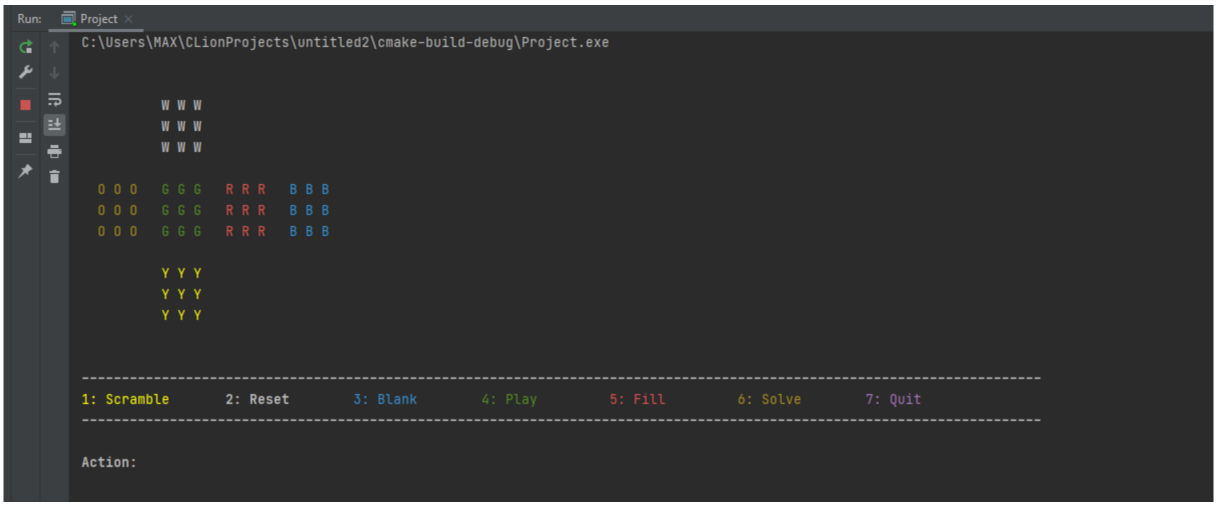
As we said in the introduction our project is almost complete, excepting the algorithm of resolution of the Rubik's cube which is not complete, but a part of the algorithm of resolution works (the beginning with the white cross). The Rubik's Cube can be partially to be solved for the moment. We have made a lot of research and we knew how to solve the Rubik's Cube with a function. but unfortunately, the Rubik's Cube is a complex case with a lot of possibilities that takes a lot of time to encode all the possibilities, and unfortunately the time was insufficient for us. but to show you that we were able to solve the cube, we decided to make the beginning of the algorithm of resolution, which took us a lot of time. and it is necessary to know that the beginning is very long because when we apply the Rubik's scramble, there are a lot of possible possibilities to start the Rubik's cube, and thus to analyze all these possibilities and to be able to realize the white cross it took us a lot of time. So, this is the only project function that is incomplete but works in part, because we can solve the beginning of the Rubik's Cube, from a random mix.



```c

typedef enum { FRONT, BACK, UP, DOWN, RIGHT, LEFT } T_SIDE;
typedef enum { G, B, W, Y, R, O, E } T_COLOR;

typedef struct {
    T_SIDE side;
    T_COLOR **cubies;
} T_FACE;

int select_color(T_COLOR);
int side_to_index(T_SIDE);
T_FACE* create_rubiks(void);
void init_rubiks(T_FACE*);
int change_color_text(T_COLOR);
void display_rubiks(T_FACE*);
void blank_rubiks(T_FACE*);
void AntiClockWise(T_SIDE, T_FACE*);
void ClockWise(T_SIDE, T_FACE*);
void BACK_clockwise(T_FACE*, int);
void BACK_anticlockwise(T_FACE*, int);
void DOWN_clockwise(T_FACE*, int);
void DOWN_anticlockwise(T_FACE*, int);
void FRONT_clockwise(T_FACE *, int);
void FRONT_anticlockwise(T_FACE*, int);
void LEFT_clockwise(T_FACE*, int);
void LEFT_anticlockwise(T_FACE*, int);
void RIGHT_clockwise(T_FACE*, int);
void RIGHT_anticlockwise(T_FACE*, int);
void UP_clockwise(T_FACE*, int);
void UP_anticlockwise(T_FACE*, int);
void horizontal_rotation(T_FACE*, int);
void vertical_rotation(T_FACE*);
void scramble_rubiks (T_FACE*);
void input_rubiks (T_FACE*);
void white_cross (T_FACE*);
T_SIDE color_to_side (T_COLOR);
void unknown_clockwise(T_FACE*, T_SIDE, int);
void menu (T_FACE *rubiks);

```

A function that has been modified but that is not necessarily visible through the menu. This is the horizontal function. indeed the function proposed in the subject allowed us only to make a 180° with the cube, thus passed from the face "Front" to the face "Back", what thus was complicated to manage the face "right" and the face "left". we thus modified this function so that it can make with the cube a half-turn of 90 degrees and not 180, one can thus pass from the face "Front" to the face "Right" (for example) easily. what allowed us to more easily manage the face "right" and the face "left". And if we want to make 180 degrees so passed from the face "Front" to the face "Back" just apply twice the horizontal function.

We have thus realized in part the whole of what is asked in the subject. The user who wants to play with his Rubik's Cube can have access to the 6 features requested in the topic.



```
C:\Users\MAX\CLionProjects\untitled2\cmake-build-debug\Project.exe



          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        7: Quit
------------------------------------------------------------------------------------------------------------------------

Action:
```

the user who wants to play with his Rubik's Cube can have access to the 6 functions requested in the subject. and these different functions are all functional. even if "solve" does not go all the way.

And as you can see, we have added a feature that was not requested. function 7 called "Quit" which allows the user to quit when he doesn't want to play anymore. we thought it was a good idea to add this option because we thought it was closer to a real interface. because when a player is tired of playing or has finished solving the Rubik's Cube or simply does not want to play, he should have the possibility to leave. That's why we added this option because we thought it was elementary for our interface and it made our program more professional and thoughtful.

**<u>Compliance with the specifications:</u>**

For the conformity of the schedule of conditions, we respected it rather well since we made all the functions which we had planned to make. we even added and modified some (horizontal function). The first things we tried to do were the initialization and display functions of the Rubik's cube. These functions seemed to us essential to do at the beginning. Because already to see the operations which we carry out on the Rubik's Cube for example to test the movements it was essential to have the display. That's why we made the display a priority in order to be able to test the different modifications and new functions added to our program.

Then we made the initialization function which is also essential for the tests that we were going to make, because indeed if we test to make different movements on the Rubik's Cube, it would have been mixed, the function of initialization thus allows to put all at zero in order to test new things. The fact of making these functions at the beginning allowed us to be able for example to test the functions of movement, which made us save time because we could thus test what we made and what we added and thus to be able to see our errors or what was good.

if you want to test our Rubik's Cube, you just have to follow the instructions in the menu. For example, if you want to test the movements, you just have to choose the action "Play", which corresponds to 4 in the actions, and then enter the movements you want to do for example " B' " to do a "BACK anticlockwise"

## <u>**Technical Aspect:**</u>

We have 1 function module, with a header file containing the prototypes of the functions.

We've chosen to modelize our cube as an array of T_FACE structure. We define T_FACE as a structure composed of 2 elements: a face name of enume type T_SIDE, called side and a 3x3 array of T_COLOR called cubies representing the cubbies states.

Also, our scramble_rubiks function works by calling randomly a movement function 30 times in a row, allowing us to have a shuffled rubik's cube that we can solve. That wouldn't be the case if we had put a random value in each cubbie.

Our horizontal_rotation function is a bit different from the prototype given in the subject. Actually, it does a quarter of turn instead of a half-turn, allowing us to put the RIGHT or ORANGE face in the FRONT place. It takes a parameter type that, in the same way that for the faces's movements, allows us to perform directly a half-turn instead of calling twice the function.

We have added a color_to_side function, allowing us to know at which face belongs a cubbie of color T_COLOR. It has allowed us to implement later an unknown_clockwise function,  that will rotate an unknown face. The color_to_side and the unknown_clockwise combined allow us to rotate faces that we haven't predetermined. It is very useful in the perfect cross algorithm, in which we want to rotate the lower cubbie of a face until it is under the corresponding middle cubbie, and then rotate this new face. Because we can't know the new face in advance, the unknown_clockwise function is very useful.

We have also implemented some functions to help us to switch easily from enum type to string and conversely. These functions are select_color, that returns the color code of a T_COLOR parameter, and allows us to print this color with a string list containing the string color values for the same indices than T_COLOR, change_color_text that is used to convert our color code into the conio color code to print our display with the corresponding colors, and finally ask_color, that asks the user to enter a color for a cubbie in a specific row and a specific column of a specific face. This last function is recursive, to be able to recall itself until the condition is reached.

All the functions that take a type as parameter are recursives. It’s more elegant and clearer. Also, most of our functions do not return anything, since we modify our rubik’s cube with it’s pointer it’s a modified parameter.

## <u>**Conclusion:**</u>

During this project we communicated a lot in order to share our tricks to realize more easily the Rubik's Cube, and to be able to adapt them in algorithm. the fact of making a project on the Rubik's Cube was amusing for us because it is a known game of which one suspects the difficulties. because it is a puzzle and like the other puzzles the realization is rather intuitive, it is thus rather complicated to adapt it in global algorithm, because there is enormously of possible arrangements. So, although we knew how to make the algorithm of resolution, the fact of translating it in program, took a lot of time, that's why we could not finish the algorithm of resolution. In spite of this, all the other functions work and allow the user to use the Rubik's Cube of the program as if it were a real one. Ce projet nous a pris beaucoup de temps et nous avons évidemment travailler hors des sessions de lab. Communication was our strong point, we communicated a lot in order to share the tasks and therefore save time.