Projet Programmation en C.

Le Rubik's Cube.

Groupe C.

SENECHAL Morgan.

SEBBANE Ryan.

Professeur : NACEUR Maha et CHABCHOUB Kamel.

I Les Algorithme :

1) Algorithme de la Fonction select_color :

```
Fonction select_color(**rubiks:FACES,i,j,k:entier)
Paramétre copies: rubiks
Parametre modifies:vide
Variable local:vide
Début:
        Si((*rubiks+i)->face[j][k]==0) alors
                c_textcolor(LIGHTGRAY)
        Si((*rubiks+i)->face[j][k]==1)alors
                c_textcolor(WHITE)
        Si((*rubiks+i)->face[j][k]==2)alors
                c_textcolor(YELLOW)
        Si((*rubiks+i)->face[j][k]==3) alors
                c_textcolor(BLUE)
        Si((*rubiks+i)->face[j][k]==4) alors
                c_textcolor(GREEN)
        Si((*rubiks+i)->face[j][k]==5) alors
                c_textcolor(RED)
        Si((*rubiks+i)->face[j][k]==6) alors
                c_textcolor(LIGHTRED
```

Cet algorithme est constitué de 7 conditions « Si ». Ces conditions nous permettent d'attribuer un chiffre à chacune des couleurs. La fonction selecte_color ne retourne rien.

2) Algorithme de la Fonction side_to_index :

```
Fonction side_to_index(**rubikscube:T_Rubiksface, T_side: coté)

Parametre copies: coté

Parametre modifies: vide

Variable local : i :compteur entier

Début :

        Pour i->0 a 6 faire

                Si((*rubikscube+i)->typeface==coté)

                        Retourner i

        Retourner -1
```

Cet algorithme est constitué d'une boucle pour et d'une condition « Si » qui nous permettent d'attribuer un chiffre pour chaque côté du rubiks cube. La fonction side_to_index nous retourne le chiffre attribuer soit i et retourne à la fin -1.

3) Algorithme de la Fonction create_rubiks :

```
Fonction create_rubiks(**rubikscube :T_Rubiksface)
Paramètre modifies : **rubikscube
Paramètre copies : **rubikscube
Variable local :i,j :compteur entier
Début :
        Rubiks=(FACES*) malloc (*6sizeof (FACES))
        Pour i->0 a 6 faire
                (rubiks+i)->type_face=i+1
                (rubiks+i)->face=(int**)malloc(3*sizeof(int*))
                Pour j->0 a 3 faire
                        (rubiks+i)->face[j]=(int*)malloc(3*sizeof(int))
        Retourner rubiks.
```

Cet algorithme nous permet de crée notre rubiks cube en initialisant l'espace mémoire avec malloc. La fonction create_rubiks nous retourne notre rubiks cube initialisé.

4) Algorithme de la Fonction free_rubiks :

```
Fonction free_rubiks(rubiks :FACES)
Paramètre copies : FACES
Paramètre modifies :vide
Variable local : i,j :compteur entier
Début :
        Pour i->0 a 6 faire
                Pour j->0 a 3 faire
                        free((*rubiks+i)->face[j])
                free((*rubiks+i)
        free(*rubiks)
```

Cet algorithme nous permet d'initialiser le rubisk de sorte que l'ordre des couelurs soit respecté. La fonction free_rubiks ne retourne rien.

5) Algorithme de la Fonction init_rubiks :

```
Fonction init_Rubiks(**rubikscube :T_Rubiksface)
Parametre copies : **rubikscube
Parametre modifies : **rubikscube
Variable local : i,j,k :compteur entier
Début :
        Pour i->0 a 6 faire
                (*rubikscube+i)->typeface=i+1 ;
                Pour j->0 a 3 faire
                        Pour k->0 a 3 faire
                                (*rubikscube+i)->FACE[j][k]=i+1
```

Cet algorithme nous permet de mettre en gris notre rubikscube pour l'initialiser sans les couleurs. La fonction init_rubiks nous retourne rien.

6) Algorithme de la Fonction blank_rubiks :

Cet algorithme nous permet d'afficher le rubiks. La fonction blank_rubiks ne nous retourne rien.

```
Fonction blank_rubiks(rubiks :FACES)
Paramètre copies : FACES
Paramètre modifies :vide
Variable local : i,j ,k:compteur entier
Début :
        Pour i->0 a 6 faire
                Pour j->0 a 3 faire
                        Pour k->0 a 3 faire
                                Si(j==1 ET k==1)alors
                                        (*rubiks+i)->face[j][k]=i+1
                                Sinon:
                                        (*rubiks+i)->face[j][k]=0
```

7) Algorithme de la Fonction display_rubiks :

Cet algorithme nous permet de remplir les cases du rubiks cube avec les couleurs. La fonction ne nous retourne rien.

```
Fonction display_rubiks(**rubikscube :FACE)
Parametre copies : **rubikscube

Parametre modifies : **rubikscube

Variable local : i,j :compteur entier

Début :

        Pour i->0 a 3 faire

                Afficher(« »)

                Pour j->0 a 3 faire

                        Select_color_conio(rubiks,up-1,i,j)

                        Afficher((*rubiks+up-1)->face[i][j]

                Afficher("\n")

        Pour i->0 a 3 faire

                Afficher(« »)


                Pour j->0 a 3 faire


                        Select_color_conio(rubiks,LEFT-1,i,j)


                        Afficher((*rubiks+LEFT-1)->face[i][j])


                Afficher("")
```

```
Pour j->0 a 3 faire
        Select_color_conio(rubiks,FRONT-1,i,j)
        Afficher((*rubiks+FRONT-1)->face[i][j])
Afficher(“”)
Pour j->0 a 3 faire
        Select_color_conio(rubiks,RIGHT-1,i,j)
        Afficher((*rubiks+RIGHT-1)->face[i][j])
Afficher(“”)
Pour j->0 a 3 faire
        Select_color_conio(rubiks,BACK-1,i,j)
        Afficher((*rubiks+BACK-1)->face[i][j])
Afficher(“\n”)
Pour i->0 a 3 faire
        Afficher(“”)
        Pour j->0 a 3 faire
                Select_color_conio(rubiks,DOWN-1,i,j)
                Afficher((*rubiks+DOWN-1)->face[i][j])
        Afficher(“\n”)
```

8) Algorithme de la Fonction blank_rubiks :

```
Fonction blank_rubiks(**rubikscube :FACE)
Parametre copies :rubikscube
Parametre : modifies : rubikscube
Variable local :i,j,k:compteur entier
Début :
        Pour i->0 a 6 faire
                Pour j->0 a 3 faire
                        Pour k->0 a 3 faire
                                Si(j=1 ET K=1) alors
                                        (*rubikscube+i)->Face[j][k]=i+1
                                Sinon:
                                        (*rubikscube+i)->Face[j][k]=0
        Retourner 0
```

9) Algorithme de la Fonction fill_rubiks :

```
Fonction fill_rubiks(**rubikscube :FACE)
Parametre copies :**rubikscube
Parametre modifies :**rubikscube
Variable local : i,j,o,k :compteur entier
Début :
        Pour o->0 a 27 faire
                Afficher(« saisir la face »)
                i=choose_face()
                afficher(« choisir la ligne et colonne »)
                faire
                        verif=sasir(j)
                        verif2=saisir(k)
                        fflush(stdin)
                tant que (j,k<-1 ou j,k>3 ou verif,verif 2==0)
                (*rubisk+i-1)->face[j][k]=choose_color(rubiks)
                Si((check_rubiks_color(rubiks)==0))
                        Faire
                                Afficher(“couleur impossible veuillez ressaisir un
nouveau \n »)
                                (*rubiks+i-1)->face[j][k]=choose_color(rubiks)
                                A=(check_rubiks_colors(rubiks))
                        Tant que (a==0)
```

Cet algorithme nous permet d’affecté manuellement les couleurs aux différentes cases. La fonction fill_rubiks ne retourne rien.

10)Algorithme de la Fonction choose_face :

```
Fonction choose_face()
Paramètre copies :vide
Paramètre modifies :vide
```

Cet algorithme nous permet de choisir la face de notre rubiks. La fonction choose_face retourne la face que l’utilisateur choisie.

```
Variable local : face,verif :entire
Début :
    Afficher(« 1 : \n 2 :BACK\n 3 :UP\n 4: DOWN\n 5:RIGHT\n 6:LEFT”)
    Faire
        Verif=saisir(face)
        Fflush(stdin)
    Tant que (face>6 ou face<verif==0)
    Si(face==1) alors
        Retourner FRONT
    Sinon si(face==2) alors
        Retourner BACK
    Sinon si(face==3) alors
        Retourner UP
    Sinon si(face==4) alors
        Retourner DOWN
    Sinon si(face==5) alors
        Retourner RIGHT
    Sinon si(face==6)
        Retourner LEFT
```

11)Algorithme de la fonction choose_color :

Cet algorithme nous permet de ne pas choisir les couleurs du rubiks. La fonction choose_color nous retourne notre rubiks après la modification des couleurs.

```
Fonction choose_color(**rubiks :FACES)
Paramètre copies:**rubiks
Paramètre modifies:vide
Variable local:color,verif:entire
Début:
    Display_rubiks(rubiks)
    Afficher(“\n”)
    Afficher(“1:BLANC”)
    Afficher(“”)
    Afficher(“2:JAUNE”)
    Afficher(“”)
    Afficher(“3:BLEU”)
    Afficher(“”)
    Afficher(“4:VERT”)
    Afficher(“”)
    Afficher(“5:ROUGE”)
```

Afficher(“”)

Afficher(“6:ORANGE”)

Afficher(“”)

Afficher(“\n\n”)

Faire

Verif=saisir(color)

Fflush(stdin)

Tant que(color>6 ou color<1 ou verif==0)

Retourner color

12)Algorithme check_rubiks_color :

Cet algorithme nous permet de ne pas depasser la limite de couleurs qui est de 9 cubes par couleurs. La fonction rubiks_color nous retourne 0 si le nombre de couleurs et >9 ou nous retourne 1 si le nombre de couleurs et <=9.

Fonction check_rubiks_color(**rubiks:FACES)

Parameter copies: **rubiks

Paramètre modifies: vide

Variable local: I,j,k:compteur entier, color_count:tableau d’entier

Début:

```
Color_count[]={0,0,0,0,0,0}
Pour i->0 a 6 faire
    Pour j->0 a 3 faire
        Pour k->0 a 3 faire
            Color_count[((*rubiks+ià->face[i][k])-1)+1

Pour i->0 a 6 faire
    Si(color_count[i)]>9)
        Retourner 0
Retourner 1
```

13)Algorithme de la fonction scramble_rubiks :

Cet algorithme nous permet de mélanger notre rubiks cube. La fonction scramble_rubiks nous retourne rien.

Fonction scramble_rubiks(**rubiks :FACES)

Paramètre copies : **rubiks

Paramètre modifies : vide

Variable local : n,j :entire

Début:

```
n=rand()%3 #un nombre entre 0 et 3
j=rand()%7+1 #un nombre entre 1 et 6
Si(n==0)
    Anticlowise(rubiks,j)
Si(n==1)
    Horizontal_rotation(rubiks)
Si(n==2)
    Vertical_rotation(rubiks)
```

14) Algorithme de la fonction anticlockwise :

Cet algorithme nous permet de bouger notre rubiks cube dans le sens d'orientation anti horaire. La fonction anticlockwise ne retourne rien.

Fonction anticlockwise(**rubiks : FACES, face_to_turn: T_SID)

Paramètre copies: **rubiks

Paramètre modifies: vide

Variable local:i,j:compteur entier,turn_left[8][2] :tableau 2 dimension d'entier,tmp :entier

Début :

```
Turn_left=[8][2]={{0,0},{0,1},{0,2},{1,2},{2,2},{2,1},{2,0},{1,0}}
Pour j->0 a 2 faire
    Tmp=(*rubiks+face_to_tur-1)->face[turn_left[0][0]][turn_left[0][1]]
    Pour i->0 a 7 faire
        (*rubiks+face_to_turn-1)-
>face[turn_left[i][0]][turn_left[i][1]]=(*rubiks+face_to_turn-1)-
>face[turn_left[i+1][0]][turn_left[i+1][1]]
    *(rubiks+face_to_turn-1)->face[turn_left[7][0]][turn_left[7][1]]=tmp
```

15)Algorithme de la function horizontal_rotation:

Cet algorithme nous permet de faire une rotation horizontale dans laquelle la face BACK devient la face FRONT et la face RIGHT devient la face LEFT. La fonction horizontale_rotation ne retourne rien.

Fonction horizontal_rotation(**rubiks:FACES)

Paramètre copies:**rubiks

Paramètre modifies:vide

Variable local:i,j,k :compteur entier,valeur[3][3] :tableau 2 dimension d'entier

Début :

```
Pour i->0 a 5 faire
    Pour j->0 a 3 faire
        Pour k-> 0 a 3 faire
            Valeur[j][k]=(*rubiks+i)->face[j][k]
            (*rubkis+i)->face[j][k]=(*rubiks+i+1)->face[j][k]
            (*rubiks+i+1)->face[j][k]=valeur[j][k]
```

16)Algorithme de la fonction vertical_rotation :

Cet algorithme nous permet de faire une rotation vertical de la face UP qui divient DOWN et de la face FRONT qui devient BACK

Fonction vertical_rotation(**rubiks :FACES,type :entier)

Paramètre copies : **rubiks

Paramètre modifies : vide

Variable local : i,j,k :compteur entier, valeur[3][3] :tableau a 2 dimension d'entier

Début :

```
Pour i->1 a 6 faire
    Pour j->0 a 3 faire
        Pour k->0 a 3 faire
            Valeur[j][k]=(*rubiks+i)->face[j][k]
            (*rubiks+i)->face[j][k]=(*rubiks+i-1)->face[j][k]
            (*rubiks+i-1)->face[j][k]=valeur[j][k]
```

17)Algorithme de la fonction RIGHT_clockwize :

Fonction RIGHT_clockwize(**rubiks :FACES,type :entier)

Paramètre copies : **rubiks

Paramètre modifies :vide

Variable local : j : compteur entier, cpt :entier,char1[3],char2 :tableau de caractère

Début :

Cpt->0

Tant que(cpt<type)

Pour j->0 a 3 faire

Char1[j]=rubiks[up-1]->face[j][2]

Pour j->0 a 3 faire

Rubiks[UP-1]->face[j][2]=rubiks[FRONT-1]->face[j][2]

Pour j->0 a 3 faire

Rubiks[FRONT-1]->face[j][2]=rubiks[DOWN-1]->face[j][2]

Pour j->0 a 3 faire

Rubiks[FRONT-1]->face[j][0]=char1[2-j]

Char2=rubiks[RIGHT-1]->fac[j][0]=char[2-j]

Rubiks[RIGHT-1]->face[0][0]=rubiks[RIGHT-1]->face[2][0]

Rubiks[RIGHT-1]->face[2][0]=rubiks[RIGHT-1]->face[2][2]

Rubiks[RIGHT-1]->face[2][2]=rubiks[RIGHT-1]->face[0][2]

Char2=rubiks[RIGHT-1]->face[0][1]

Rubiks[RIGHT-1]->face[0][1]=rubiks[RIGHT-1]->face[1][0]

Rubiks[RIGHT-1]->face[1][0]=rubiks[RIGHT-1]->face[2][1]

Rubiks[RIGHT-1]->face[2][1]=rubiks[RIGHT-1]->face[1][2]

Rubiks[RIGHT-1]->face[1][2]=char2

Cpt++

Les difficultés rencontrées :

L'ors de ce projet, nous avons rencontrée diverses difficultés : La première difficultés à était de s'avoir par où commencer le projet. En effet au départ nous avons eu un peu de mal avec les fichiers .h et .c mais nous avons réussi à comprendre leurs utilisations après plusieurs minutes de réflexion. La deuxième à était de faire les fonctions de mouvement sur lesquelles nous avons passé énormément de temps même si toute fois nous avons réussi à les faires. La dernière difficulté que nous avons rencontrée a était de percevoir le rubiks cube en 3 dimensions afin de pouvoir crée les différente fonction adéquate à celui-ci. Nous n'avons pas réussi à terminer à 100% le projet dû à un manque de temps, la résolution de l'algorithme n'a pas pu être fait et nous avons un problème dans notre le code de nôtre menu.

Les enseignements de ce projet :

Ce projet nous à permit de mieux comprendre et de nous entrainer sur les tableaux et les structure de donnée. En effet grâce aux différentes fonctions demandées qui demandé d'utiliser les tableaux et les structure de donnée, nous avons pue nous entrainer et mieux comprendre diverses notions sur la structure de donnée notamment avec les allocations mémoire de malloc.

Les perspectives d'amélioration :

Nous pensons que le code pourrait être mieux optimisé. De plus nous avons mis énormément de temps pour réaliser ce projet, nous pensons donc qu'il serait nécessaire d'améliorer nos connaissance pratique et théorique sur les structures et les tableaux.