Compte-rendu du projet C

« It's time to C Code (2) ! »

Membres du groupe :

- Anto BENEDETTI ;
- Antony DAVID ;
- Anthony JABRE.

Sommaire

Durant ce projet de programmation en langage C, nous avons cherché, pour chaque exercice :

- À avoir le code le plus lisible possible ;
- À ajouter des détails rendant l'expérience utilisateur plus agréable ;
- À sécuriser au maximum le programme, en prenant en compte le maximum de cas différents, lors de la saisie par l'utilisateur.

De plus, nous avons utilisé les types de variables nécessitant le minimum de mémoire pour l'optimisation, hormis certaines où une grande variable était préférable pour avoir une sécurité sur le maximum de cas (ex : `usr_inpt`).

Avant de commencer l'explication de chaque exercice, nous devons décrire trois fonctionnalités communes à tous :

❖ Boucle principale

Quasiment l'ensemble du code est à l'intérieur d'une boucle `do while`, où la condition n'est d'autre qu'une variable `running` valant soit 1, soit 0.

Elle représente donc un booléen, la rendant exploitable par la boucle. Sa valeur est initialisée à 1 au tout début du code, et est mise à 0 uniquement lorsque l'utilisateur choisi d'arrêter le programme, grâce au menu.

❖ Menu

Dans chaque programme, un menu est accessible pour lancer l'exercice, autant de fois que l'on veut, ainsi que pour l'arrêter.

Cette fonctionnalité nécessite, dans un premier temps, à afficher le menu, grâce à un `printf()`, et à récupérer le choix de l'utilisateur, puis dans un second temps, à utiliser un `switch`.

Le premier choix correspond à lancer le programme (*case 1* dans le *switch*), contenant l'algorithme de l'exercice. Le second choix correspond à l'arrêt du programme.

❖ Vérification des saisies

Chaque saisie par l'utilisateur est contrôlée par un algorithme en trois vérifications :

- Type de la variable respectée ;
- Inférieur au minimum ;
- Supérieur au maximum

Lorsqu'une étape est validée, un compteur *check_step* s'incrémente. Cette variable doit être égale à 3 pour que la saisie soit acceptée. Si la vérification échoue, le compteur est réinitialisé et la boucle *while* recommence.

Une variable *type_check* est utilisée pour vérifier si l'utilisateur a bien rentrer une valeur correspondant au type de variable demandé.

Si le programme nécessite une valeur numérique entière (*int* par exemple), mais que l'utilisateur entre une chaîne de caractères, la variable sera égale à 0.

Dans le cas où le type est respecté, la variable sera égale aux nombres de types respectés.

Exemple :

```
type_check = scanf("%d %c", &a, &b);
```

➔ Si la première saisie de l'utilisateur est un nombre entier, et que la deuxième est un caractère, *scanf()* retourne la valeur 2, qui sera affectée à la variable *type_check*.

➔ Si la première saisie est un nombre entier, mais que la deuxième est autre qu'un caractère, *scanf()* retourne la valeur 1, qui sera affectée à la variable *type_check*.

➔ Si aucun des saisies ne correspond au code format, *scanf()* retourne la valeur 0, qui sera affectée à la variable *type_check*.

Après le contrôle réussi, la variable *check_step* est de nouveau réinitialisée, afin que la prochaine vérification ne soit pas ignorée.

Exercices

❖ Exercice 1

Nous avons fixé le nombre maximum de lignes à 100, car nous avons jugé qu'avec cette valeur, l'image était suffisamment grande.

De plus, nous avons ajouté une fonctionnalité de génération de nombre aléatoire, entre 1 et 20 (inclus), rendant le processus légèrement plus automatisé.

```
119     // Cas où l'utilisateur veut générer un nombre aléatoire de lignes
120     if (n == 0)
121     {
122         n = rand() % 20 + 1;
123         printf("Nombre de lignes al%catoirement g%cn%cr%c : %d\n", 130, 130, 130, 130, n);
124     }
```

Ensuite, si le nombre de ligne choisi est égal à 1, il n'y a nullement besoin de réaliser tout un algorithme. Un simple *printf* suffit, puis nous ignorons le reste du code.

```
126     // Cas particulier où l'utilisateur veut qu'une seule ligne
127     if (n == 1)
128     {
129         printf("* *\n");
130         break; // Permet de passer les étapes ci-dessous
131     }
```

Viens maintenant le cœur de du programme. Pour réaliser le pattern demandé, nous avons diviser le dessin en deux parties si le nombre de lignes est pair, et en trois parties pour le cas impair.

Ceci explique pourquoi nous avons diviser la variable de la boucle, *i*, par deux. Par la suite, nous avons remarqué que chaque moitié est composée de 3 triangles : 2 triangles constitués d'étoiles, et un d'espace entre ces derniers. Nous avons donc utilisé une boucle pour la moitié, et trois autres pour les triangles.

```
133     // Partie haute
134     for (i = 0; i < n / 2; i++) // n / 2 -> nombre de ligne de la partie haute de la figure
135     {
136         for (j = 0; j <= i; j++) // Premier triangle
137             printf("* ");
138 
139         for (j = i + 2; i + j < n; j++) // "pyramide" d'espaces entre les 2 triangles
140             printf("  ");
141 
142         for (k = 0; k <= n - j; k++) // Deuxième triangle
143             printf("* ");
144 
145         printf("\n");
146     }
```

Ensuite, nous arrivons à la ligne centrale si *n* est impair.

```
148         // Ligne centrale uniquement si n est impair
149         if (n % 2 == 1)
150         {
151             for (i = 0; i < n; i++)
152                 printf("* ");
153 
154             printf("\n");
155         }
```

Et pour terminer, nous avons la partie basse, également dotée d'une grande boucle avec une pour chaque triangle.

```
157     // Partie basse
158     for (i = 0; i < n / 2; i++) // n / 2 -> nombre de ligne de la partie basse de la figure
159     {
160         for (j = 0; j + i < n/2; j++) // Premier triangle
161             printf("* ");
162 
163         for (j = n % 2 == 0 ? 1 : 0; j <= 2 * i; j++) // "pyramide" d'espace entre les 2 triangles
164             printf("  ");
165 
166         for (j = i; j < n/2; j++) // Deuxième triangle
167             printf("* ");
168 
169         printf("\n");
170     }
```

❖ Exercice 2

Notre programme peut se diviser en deux parties : une si l'utilisateur rentre un chiffre, et l'autre partie s'il saisit un nombre.

Pour affecter le nombre de barres nécessaires pour chaque chiffre, nous utilisons un *switch*. Les chiffres avec le même nombre de barres ne seront pas espacés par des *break*. Sans ce mot-clé, les cas se suivront jusqu'à en rencontrer un, où une instruction sera utilisable pour tous ces cas. Petit passage pour illustrer :

```
case 0:
case 6:
case 9:
    bars = 6;
    break;
```

Pour le cas du nombre, plus d'instructions sont nécessaires. Tout d'abord, nous devons extraire chaque chiffre.

```
digit1 = n / 10;
digit2 = n % 10;
```

Ensuite, nous avons une boucle permettant de dissocier les traitements du premier chiffre et du second. Cela reprend le *switch* précédent mais avec une addition des barres.

```
case 0:
case 6:
case 9:
    bars = 6;
    bars_sum += bars;
    break;
```

Le code se termine sur l'ensemble des *printf()*, avec tous les détails pour aider l'utilisateur à comprendre tout le processus de calcul (nombre de barres par chiffre, somme des barres et des chiffres et résultat final : magique ou non).

❖ Exercice 3

Dans l'exercice 3 il nous faut tout d'abord séparer 3 problèmes de l'exercice :

- Afin de trouver dans quelle zone on se trouve avec les cordonnées de l'utilisateur on parcourt les zones une par une en x et en y afin de trouver et de quadriller notre terrain. Une fois que l'on a trouvé dans quelle zone on se trouver on enregistre dans une variable les coordonnées en haut à droite pour retrouver plus tard dans notre « switch » dans quelle zone on se trouve.

Et afin de mettre en place toutes nos frontières on va mettre en place 4 booléens pour une zone. Chaque booléen va nous indiquer s'il y a une frontière en haut à droite à gauche et en bas. On utilise ensuite des boucles SI dans chacune de nos zone afin de traiter nos différents cas.

```
int find = 0; // Stopper le programme si on trouve
int zone_number_xb; // Sert à comparer avec la coordonnée xb à rechercher
int zone_number_yb; // Sert à comparer avec la coordonnée yb à rechercher

int is_frontier_x_R = 0; // Vaut 1 si le point recherché se trouve à la frontière droite d'une zone
int is_frontier_x_L = 0; // Vaut 1 si le point recherché se trouve à la frontière gauche d'une zone
int is_frontier_y_down = 0; // Vaut 1 si le point recherché se trouve à la frontière inférieure d'une zone

// Boucle afin de trouver dans quelle zone se situe les coordonnées à rechercher en X
for (int i = xa; i <= width * 2 + xa && find == 0; i = i + width)
{
    if (find_x <= i)
    {
        if (find_x == i)
            is_frontier_x_R = 1;

        if (find_x == i - width)
            is_frontier_x_L = 1;

        zone_number_xb = i;
        find = 1;
    }
}
find = 0;

// Boucle afin de trouver dans quelle zone se situe les coordonnées à rechercher en Y
for (int j = ya; j <= height * 2 + ya && find == 0; j = j + height) {
    if (find_y <= j)
    {
        if (find_y == j)
            is_frontier_y_down = 1;

        zone_number_yb = j;
        find = 1;
    }
}
```

- Pour afficher les coordonnées de chaque zone, il nous faut afficher les coordonnées en haut à gauche et celles en bas à droite. Nous récupérons l'ensemble de ces coordonnées via des calculs faits à partir des coordonnées de l'utilisateur et nous les stockons dans un switch dans laquelle les case corresponde respectivement à chaque zone.

Exemple zone 1 :

```
// Boucle principal parcourant les zones afin de calculer les coord des zones et vérifier si les coord sont dedans
for (int i = 1; i <= 9; ++i)
{
    printf("\n============================================================\nZone %d", i);

    // Chaque "case" correspond à une zone
    switch (i)
    {
        case 1:
            new_xa = 0;
            new_ya = 0;
            new_xb = xa;
            new_yb = ya;

            printf("\nCoordonn%ces du coin sup%crieur gauche : (%d, %d)", 130, 130, new_xa, new_ya);
            printf("\nCoordonn%ces du coin inf%crieur droit  : (%d, %d)", 130, 130, new_xb, new_yb);

            if (zone_number_xb == new_xb && zone_number_yb == new_yb)
            {
                if (is_frontier_x_R == 1 && is_frontier_y_down == 0)
                    printf("\n\nLe point recherch%c se trouve sur la fronti%cre zones %d & 2.", 130, 138, i);

                else if (is_frontier_x_R == 0 && is_frontier_y_down == 1)
                    printf("\n\nLe point recherch%c se trouve sur la fronti%cre zones %d & 4.", 130, 138, i);

                else if (is_frontier_x_R == 1 && is_frontier_y_down == 1)
                    printf("\n\nLe point recherch%c se trouve sur la fronti%cre zones %d, 2, 4 & 5.", 130, 138, i);

                else if (is_frontier_x_L == 1 && is_frontier_y_down == 1)
                    printf("\n\nLe point recherch%c se trouve sur la fronti%cre zones %d & 4.", 130, 138, i);

                else
                    printf("\n\nLe point recherch%c se trouve ici.", 130);
            }
            break;
```

❖ Exercice 4

Dans l'exercice 4 il nous faut tout d'abord séparer 2 cas :

- Celui des nombres positifs donc à partir de 2 octets. Il nous suffit pour ce cas de décomposer le nombre de l'utilisateur avec des modulos et des divisions puis de dénombres le nombre de 0 et de 1.

```
// Parcours et traitement des nombres entre a et b
for (i = a_dec; i <= b_dec; i++)
{
    actual_value = i;
    temp = i;

    // Cas spécial du nombre négatif --> initialisation du bool negative à 1 et transformation en nb positif
    if (i < 0)
    {
        negative = 1;
        i *= -1;
    }

    // Boucle decimal to binary
    // et récupération de la valeur dans bin (on calcul en même temps le nb de 0 et 1)
    while (i != 0)
    {
        rem = i % 2;

        if (rem) // Bit égal à 1
        {
            num_of_1++;
            num_of_0--;
        }

        i /= 2;
        bin += rem * j;
        j *= 10;
    }
```

- Celui des nombres négatifs. Nous avons voulu ajouter ce cas-là comme fonctionnalité en plus. Il faut tout d'abord récupérer le premier 1 du nombre en partant de la droite, une fois cela fait on inverse tous les bits qui suivent en gardant le premier 1.

Il reste à compléter s'il manque des bits pour atteindre 2 octets.

```c
// Traitement du cas négatif
if (negative)
{
    binary_negative = 0;
    j = 1;
    compteur = 0;
    // recherche du premier 1 à partir de la droite
    while (bin % 2 != 1)
    {
        j = j*10;
        compteur++;
        bin /= 10;
    }
    //si le premier nombre est directement un 1
    binary_negative = binary_negative + (bin%10)*j;
    bin /= 10;
    j = j*10;


    while(bin > 0)//On inverse les autres bit restants dans bin et on les stocks dans binary_negative
    {
        if (bin % 2 == 0)
        {
            rem = 1;
            binary_negative += rem * j;
            j = j * 10;
            compteur++;
        }
        else
        {
            rem = 0;
            binary_negative += rem * j;
            j = j * 10;
            compteur++;
        }
        bin /= 10;
    }
    while(compteur < 15)//on complète ce qu'il reste pour atteindre 2 octets dans binary_negative
    {
        binary_negative = binary_negative + j;
        j = j *10;
        compteur++;
    }

    // Calcul du nombre de 1 et 0 dans notre bianire final
    num_of_0 = 16;
    num_of_1 = 0;

    while (binary_negative > 0)// on compte le nombre de 1 et 0 dans notre nombre
    {
        if(binary_negative % 2 == 0) {
            num_of_1++;
            num_of_0--;
        }
        binary_negative /= 10;
    }
}
```

Une fois ces deux cas traités on compare le nb de 0 et de 1 et on affiche si le nombre est sympathique.

On doit ensuite procéder à une suite de test pour savoir s'il on affiche une virgule ou un « et » après notre nombre.

```c
//comparaison du nb de 0 et 1 pour savoir si on print ou non
if (num_of_1 == num_of_0 && goodCounter == 0)
{
    goodValue = actual_value; // Le premier bon nombre est dans goodValue
    goodCounter++;
}
else if (num_of_1 == num_of_0 && goodCounter == 1)
{
    printf("%d",goodValue); // Le premier bon nombre
    goodValue = actual_value;
    goodCounter++;
}
else if (num_of_1 == num_of_0 && goodCounter > 1)
{
    printf(", %d",goodValue); // On rajoute une virgule pour les prochains
    goodValue = actual_value;
}
// Si on est arrivé au dernier nombre et qu'il est bon
if (temp == b_dec && goodCounter > 1)
    printf(" et %d", goodValue);

    // Si on trouve qu'un seul bon qui est le dernier
else if (num_of_1 == num_of_0  && temp == b_dec && goodCounter == 1)
    printf("%d", goodValue);

    // Si on trouve qu'un seul bon qui n'est pas le dernier
else if (temp == b_dec && goodCounter == 1 && num_of_1 != num_of_0)
    printf("%d", goodValue);
```

❖ Exercice 5

Le programme commence par demander toutes les informations nécessaires concernant le commercial. Nous avons donc une grande partie du code composée de vérifications de saisie.

Une petite variation à lieu, si le commercial travaille à l'étranger avec le *if (abroad)*. Si cette condition est vraie, une nouvelle saisie est demandée, pour le nombre de jours passés à l'étranger.

Vient ensuite les différents calculs :

- Pour la commission sur le CA :

```
// Calcul de la commission
if (turnover == 0)
    commission = 100;

else if (turnover <= 13000)
    commission = turnover * 0.016;

else if (turnover <= 22000)
    commission = ((turnover-13000)*2.2)/100+(13000*0.016);

else
    commission = ((turnover-22000)*0.03)+(9000*0.022)+(13000*0.016);
```

- Pour l'indemnité de déplacement :

```
305     // Calcul de l'indemnité de déplacement
306     indemnity = 0.5 * commute;
307
308     if (indemnity < 50)
309         indemnity = 50;
310
311     else if (indemnity > 250)
312         indemnity = 250;
```

- Pour l'indemnité du travail à l'étranger :

```
314     // Calcul de l'indemnité reçue par les commerciaux à l'étranger
315     if (abroad)
316         indemnity += work_days * 100;
```

Le programme se termine avec la somme du salaire de base, de la commission et des indemnités.

❖ Exercice 6

Pour cet exercice nous avons séparé la fenêtre du menu et celle qui affiche la courbe voulue, cela facilite la gestion des évènements qui sont donc indépendant pour les deux fenêtres.

Pour cela nous avons 2 variables globales qui correspondent chacune à une fenêtre.

```
Ez_window win1; // Fenêtre menu
Ez_window win2; // Fenêtre courbe
```

Commençons par le **main** de cet exercice :

```
// Génération de la fenêtre
int main(int argc, char **arg)
{
    if (ez_init() < 0) exit(1);

    win1 = ez_window_create(800, 800, "Exercice 6", win1_on_event);
    ez_main_loop ();
    exit(0);
}
```

Ici on initialise la bibliothèque graphique « *ez-draw* » tout en vérifiant qu'il n'y a pas eu de problème durant l'initialisation.

On commence par créer notre première fenêtre « win1 » qui correspond à notre menu pour pouvoir lancer le programme ou l'arrêter., on définit notre taille, le nom de la taille fenêtre puis on lance l'exécution de la fonction « *win1_on_event* » qui comme son nom l'indique permet de lancer des instructions en fonction des évènements récupérés par ez-draw.

```
void win1_on_event(Ez_event *ev)
{
    switch (ev->type)
    {
        case Expose:    win1_on_expose    (ev); break;
        case KeyPress: win1_on_key_press(ev); break;
    }
}
```

En fonction de l'évènement récupéré, le switch case va exécuter une fonction différente.
Expose -> La fenêtre doit être générée.

Keypress -> Une touche du clavier a été saisie.

```c
void win1_on_expose(Ez_event *ev)
{
    // Dimension de la fenêtre
    int width, height;
    ez_window_get_size(ev->win, &width, &height);

    // Menu
    ez_set_nfont(3); // Taille de police
    ez_draw_text(win1, EZ_TC, width/2, height/2 - 30, "Exercice 6");
    ez_set_nfont(2);
    ez_draw_text(win1, EZ_TC, width/2, height/2, "[A] : Lancer");
    ez_draw_text(win1, EZ_TC, width/2, height/2 + 15, "[Q] : Quitter");
}
```

Une fois la fenêtre créer, l'événement « *Expose* » bien récupérer, la fonction « on_expose » est lancée, on commence par récupérer la taille de la fenêtre créée puis par définir une taille de police pour pouvoir placer les informations nécessaires pour le menu.

```c
void win1_on_key_press(Ez_event *ev)
{
    switch (ev->key_sym)
    {
        // Lorsque l'utilisateur appuie sur 'A' --> Génération de la courbe
        case XK_a:
        case XK_A:
            ez_window_show(win1, 0);
            win2 = ez_window_create(800, 800, "Courbe", win2_on_event);
            break;

        // Lorsque l'utilisateur appuie sur 'Q' --> Quitter
        case XK_q:
        case XK_Q:
            ez_quit();
            break;
    }
}
```

Pour le menu il y a 4 combinaisons de touches possibles (avec uppercase & lowercase) la touche « a » permet d'exécuter le programme pour cela on n'affiche plus le menu (mais la fenêtre n'est pas détruite pour pouvoir y revenir) et on créer la fenêtre pour afficher la courbe ce qui va en même temps exécuter « *win2_on_event* ». La touche « q » permet simplement de fermer le programme.

```c
void win2_on_event(Ez_event *ev)
{
    zoom = 50;
    switch (ev->type)
    {
        case Expose     : win2_on_expose     (ev); break;
        case ButtonPress  : win2_on_button_press (ev);  break;
        case KeyPress: win2_on_key_press(ev); break;
    }
}
```

Pour la fenêtre « *Courbe* » il y a 3 évènements possible. Nous avons déjà vu « *Expose* » et « *Keypress* » pour la fenêtre du menu, ici s'ajoute « *ButtonPress* » qui correspond au clic droit ou gauche de la souris.

Pour la suite nous avons besoin des 2 variables globales suivantes, une définie le zoom et la deuxième l'état du zoom (0 = Off / 1 = On).

```c
//variables globales
int zoom;
int bool = 0;
```

```c
void win2_on_key_press(Ez_event *ev)
{
    switch (ev->key_sym)
    {
        // Lorsque l'utilisateur appuie sur 'r' --> Retour menu
        case XK_r:
        case XK_R:
            ez_window_destroy(win2);
            ez_window_show(win1, 1);
            break;

        // Lorsque l'utilisateur appuie sur 'Z' --> Zoom avant
        case XK_z:
        case XK_Z:
            if(bool==1){break;}//pas besoin de refresh la page si on a déjà un zoom de 150
            zoom = 150;
            bool =1;
            ez_window_clear(ev->win);
            win2_on_expose(ev);
            break;

        // Lorsque l'utilisateur appuie sur 's' --> Zoom arrière
        case XK_s:
        case XK_S:
            if(bool==0){break;}//pas besoin de refresh la page si on a déjà un zoom de 50
            zoom = 50;
            bool =0;
            ez_window_clear(ev->win);
            win2_on_expose(ev);
            break;
    }
}
```

Pour la fenêtre de notre courbe, il y a 3 touches possibles au clavier :

R/r -> Retour au menu

Z/z -> Zoom avant

S/s -> Reset zoom

Pour Z et S, on vérifie avant l'état du zoom, s'il est déjà dans l'état voulu pas besoin de refresh la page inutilement.

```c
void win2_on_expose(Ez_event *ev)
{
    // Dimension de la fenêtre
    int width, height;
    ez_window_get_size(ev->win, &width, &height);
    int middleH = height / 2;
    int middleW = width / 2;

    // Équations
    double x, y;

    // Explications des touches
    ez_draw_text(ev->win, EZ_TC, middleW, 1, "[R] : Retour au menu principal");
    ez_draw_text(ev->win, EZ_TC, 100, 1, "[Z] : Zoom avant");
    ez_draw_text(ev->win, EZ_TC, 100, 30, "Clique gauche/droit -> coordonnees");
    ez_draw_text(ev->win, EZ_TC, width-100, 1, "[S] : Zoom arriere");

    // Détails equations
    ez_draw_text(ev->win, EZ_TC, 100, height-20, "x = cos(t) - cos(3t)");
    ez_draw_text(ev->win, EZ_TC, width-100, height-20, "y = sin(t) - sin(3t)");
    ez_draw_text(ev->win, EZ_TC, middleW, height-20, "t = [0, 2pi]");

    // Axe des ordonnées
    ez_draw_text(ev->win, EZ_TC, middleH-30, 50, "Y");
    ez_draw_line(ev->win,  middleH,  50, middleH, height-50);

    // Axe des abscisses
    ez_draw_text(ev->win, EZ_TC, width-50, middleH+30, "X");
    ez_draw_line(ev->win, 50,  middleH, width-50,middleH);
```

Tout d'abord on commence par stocker la taille puis la moitié en hauteur / largeur car cela va nous aider pour la suite.

Ensuite on affiche toutes les informations nécessaires puis on termine par afficher les axes des ordonnées et des abscisses.

```c
int secondI= middleH+zoom;
int secondJ = middleW+zoom;
int numberGrad = 0;

// Graduation de l'axe des ordonnées
for (int i = middleH; i <= height-50; i += zoom)
{
    ez_draw_line (ev->win, middleH-5, i, middleH+5, i);
    secondI -= zoom;
    ez_draw_line (ev->win, middleH+5, secondI, middleH-5, secondI);
    ez_draw_text(ev->win, EZ_TC, middleH-10, secondI-15, "%d",numberGrad);
    numberGrad++;
}
numberGrad = 0;

// Graduation de l'axe des abscisses
for (int j = middleW; j <= width-50; j += zoom){
    ez_draw_line(ev->win, j, middleH-5, j, middleH+5);
   if(numberGrad>0)
        ez_draw_text(ev->win, EZ_TC, j+10, middleH+5, "%d",numberGrad);
    secondJ -= zoom;
    ez_draw_line(ev->win, secondJ, middleH-5, secondJ, middleH+5);
    numberGrad++;
}
```

La partie la plus technique de l'affichage correspond aux graduations, pour cela nous utilisons 2 « *for* », nous allons détailler pour l'axe des ordonnées.

On commence du milieu et on incrémente à chaque fois en fonction du zoom (50 ou 150) le « *secondI* » lui permet de faire la même chose mais dans l'autre sens, ce qui permet en partant du milieu de tracer vers le haut et le bas dans un seul for.

« *numberGrad* » correspond à la valeur qu'on affiche pour chaque trait de graduations.

C'est la même chose pour les abscisses.

```
ez_set_color(ez_red); // Coloration de la courbe
ez_set_thick(1); // Épaisseur de la courbe

// Génération de la courbe
for (double i = 0.00; i < 2 * M_PI; i += 0.0001)
{
    x = (cos(i) - cos(3*i)) * zoom;
    y = (sin(i) + sin(3*i)) * zoom;
    ez_draw_point(ev->win, x+middleH, y+middleH);
}
```

Pour terminer il reste le for qui permet de calculer les coordonnées de chaque point, nous avons choisi d'avoir une forte précision comme on peut le constater avec une incrémentation de « 0.0001 » à chaque fois, il suffit ensuite de multiplier la valeur avec celle du zoom et d'ajouter le décalage de l'origine du graphique par rapport à l'origine d'ez-draw (TOP LEFT).

```
void win2_on_button_press(Ez_event *ev){

    int width,height;//taille de la fenêtre
    double x,y;//variable pour les calculs

    ez_window_get_size(ev->win, &width, &height);//on récupére la taille de la fenêtre
    ez_set_color(ez_white);
    ez_fill_rectangle (win2, 30, 50, 150, 80);
    ez_set_color(ez_red);
    height /=2;
    x = ev->mx-height; //on retire la hauteur/2 pour corriger le décalage
    y = ev->my-height;

    if(ev->my !=0) //on transforme en positif ce qui devient négatif avec la soustraction
        y *=-1;  //0 est un cas particulier, pas besoin de convertir

    x/=50;//on divise par 50 car 1 unité = 50 pixels
    y/=50;
    if(bool==1){//si le zoom == 150
        ez_draw_text(ev->win, EZ_TL, 30, 50, "x =%.2lf, y =%.2lf", x/3, y/3);//On divise par 3 car 1 unité est 3 fois plus grande avec le zoom 150
    }
    else{//zoom == 50
        ez_draw_text(ev->win, EZ_TL, 30, 50, "x =%.2lf, y =%.2lf", x, y);
    }
}
```

Cette fonction permet d'afficher les coordonnées du point où se situe la souris si on fait un clic gauche ou droit, sachant que par défaut « *ev->mx* » et « *ev->my* » renvoi les coordonnées du curseur en pixel, il a fallu adapter cette valeur à notre graphique.

On commence par retirer le décalage de l'origine qui correspond à la hauteur de la fenêtre divisé par 2.

Ensuite il faut transformer en positif ce qui est négatif à la suite de la soustraction (sauf en 0) après ça on vérifie le zoom car il change aussi la valeur, si le zoom est à 150 on divise une première fois par 50 (1 unité = 50 par défaut dans notre programme) mais il faut ensuite diviser à nouveau par 3 car lorsque le zoom est de 150, les unités sont 3 fois plus grandes qu'avec le zoom à 50.

Après tous ces calculs on obtient les coordonnées du point où l'utilisateur clique, qu'il soit sur la courbe ou non cela donne la bonne valeur correspondante.

Synthèse de travail

Nous avons utilisé l'outil GitHub pour travailler efficacement à trois. Nous avions une branche par exercice, et nous communiquions énormément sur Discord afin d'avancer ensemble. Nous utilisions une « todo list » pour savoir quelles tâches étaient à faire/terminées.

Le fondamental de chaque exo a été terminé assez rapidement. Nous avons passé la majorité de notre temps à ajouter des fonctionnalités, à améliorer et à sécuriser nos programmes.

Nous n'avons pas rencontré de problèmes, excepté pour l'installation de ez-draw sur Windows. Nous avons pu le régler assez rapidement, après avoir recommencer le processus de compilation plusieurs fois.