Rapport du TP 6 d'IGI-2001

Rémi Nicole

Chapitre 1

Exercice 1

Listing 1.1 – Programme

```c
#include <stdio.h>
#include <stdlib.h>

unsigned long long int factoriel(int n) {
    if (n <= 1) {
        return 1;
    } else {
        unsigned long long int temp = 0;
        printf("Avant fact: temp=%llu, n=%i\n", temp, n);
        temp = factoriel(n-1) * n;
        printf("Après fact: temp=%llu, n=%i\n", temp, n);
        return temp;
    }
}

int main (int argc, char const* argv[]) {
    if (argc != 2) {
        printf("Usage: %s n\n", argv[0]);
    } else {
        printf("%llu\n",factoriel(atoi(argv[1])));
    }
    return 0;
}
```

Listing 1.2 – Résultat

```
Avant fact: temp=0, n=10
Avant fact: temp=0, n=9
Avant fact: temp=0, n=8
Avant fact: temp=0, n=7
Avant fact: temp=0, n=6
Avant fact: temp=0, n=5
Avant fact: temp=0, n=4
Avant fact: temp=0, n=3
Avant fact: temp=0, n=2
Après fact: temp=2, n=2
Après fact: temp=6, n=3
Après fact: temp=24, n=4
```

```
Après fact: temp=120, n=5
Après fact: temp=720, n=6
Après fact: temp=5040, n=7
Après fact: temp=40320, n=8
Après fact: temp=362880, n=9
Après fact: temp=3628800, n=10
3628800
```

Chapitre 2

Exercice 2

Listing 2.1 – Programme

```
#include <stdio.h>

void add_mul(int x, int y, int* pointeur_add, int* pointeur_mul) {
    printf("pointeur_add=%i, pointeur_mul=%i\n", *pointeur_add, *pointeur_mul);
    *pointeur_add = x+y;
    *pointeur_mul = x*y;
}

int main () {
    int a = 10, b = 20, res_addition = 0, res_multiplication = 0;
    printf("&a=%p, &b=%p\n", &a, &b);
    add_mul(a, b, &res_addition, &res_multiplication);
    printf("a=%i, b=%i, a+b=%i, a*b=%i\n",
            a, b, res_addition, res_multiplication);
    return 0;
}
```

Listing 2.2 – Résultat

```
&a=0x7fff8f69c138, &b=0x7fff8f69c13c
pointeur_add=0, pointeur_mul=0
a=10, b=20, a+b=30, a*b=200
```

2.1 Question 1

On doit constater un résultat égal à 0 car les valeurs ont été initialisée à la déclaration.

Chapitre 3

Exercice 3

Listing 3.1 – Programme

```c
#include <stdio.h>
#include <stdlib.h>

struct np_cmpx {
    float re;
    float im;
};

void affiche_complexe(const struct np_cmpx nb) {
    printf("%f + %fi\n", nb.re, nb.im);
}

struct np_cmpx add_cmpx(const struct np_cmpx x, const struct np_cmpx y) {
    // a + b = ℜ(a) + ℜ(b) + (ℑ(a) + ℑ(b)) i
    struct np_cmpx ret = {x.re + y.re, x.im + y.im};
    return ret;
}

struct np_cmpx mul_cmpx(const struct np_cmpx x, const struct np_cmpx y) {
    // (a0 + b0 i) × (a1 + b1 i) = a0 × a1 − b0 × b1 + (b0 × a1 + a0 × b1) i
    struct np_cmpx ret = {(x.re * y.re) - (x.im * y.im),
                          (x.im * y.re) + (x.re * y.im)};
    return ret;
}

void permute_complexe(struct np_cmpx* nb) {
    // Store the real value in temporary variable
    int temp = nb->re;
    nb->re = nb->im;
    nb->im = temp;
}

int main (int argc, char const* argv[]) {
    if (argc != 5) {
        printf("Usage: %s a b c d\n", argv[0]);
    } else {
        struct np_cmpx x = {atof(argv[1]), atof(argv[2])},
                       y = {atof(argv[3]), atof(argv[4])},
```

```c
                            z;

        affiche_complexe(x);
        affiche_complexe(y);
        affiche_complexe(z);

        affiche_complexe(add_cmpx(x, y));
        affiche_complexe(mul_cmpx(x, y));

        permute_complexe(&x);
        affiche_complexe(x);

    }
    return 0;
}
```

Listing 3.2 – Résultat

```
1.000000 + 2.000000i
3.000000 + 4.000000i
0.000000 + 0.000000i
4.000000 + 6.000000i
-5.000000 + 10.000000i
2.000000 + 1.000000i
```

Chapitre 4

Exercice 4

Listing 4.1 – Programme

```
#include <X11/Xlib.h>
#include <unistd.h>
#include <math.h>
#include <stdlib.h>

#ifndef HEIGHT
#define HEIGHT 200
#endif

#ifndef WIDTH
#define WIDTH 200
#endif

void dessine(Display* dpy, Window w, GC gc, int col) {
    XSetForeground(dpy, gc, col); //Couleur du stylo
    XDrawLine(dpy, w, gc, 0, 0, WIDTH, HEIGHT);
    XDrawPoint(dpy, w, gc, 10, 50);
    XDrawPoint(dpy, w, gc, 10, 51);
    XFlush(dpy); //Force l'affichage
}

void affiche(Display *dpy, Window w, GC gc, unsigned char * T) {
    int i , j ;
    int blanc = WhitePixel(dpy, DefaultScreen(dpy));
    int noir = BlackPixel(dpy, DefaultScreen(dpy));

    for(i = 0 ; i < WIDTH ; i++) {
        for(j = 0 ; j < HEIGHT ; j++) {
            XSetForeground(dpy, gc, (T[i + (j*HEIGHT)] == 1) ? blanc : noir);
            XDrawPoint(dpy, w, gc, i, j);
        }
    }

    XFlush(dpy);
}

void blackout(unsigned char* tab) {
    unsigned int i;
```

```c
    for(i = 0; i < HEIGHT*WIDTH; i++)
        tab[i] = 0;
}

void creeCarre(unsigned int x0, unsigned int y0,
               unsigned int x1, unsigned int y1, unsigned char* tab) {
    unsigned int i, j;
    // Check if square is inside boundaries
    if((x0 <= WIDTH) && (x1 <= WIDTH) && (y0 <= HEIGHT) && (y1 <= HEIGHT)) {
        for(i = x0 ; i <= x1 ; i++)
            tab[(y0*WIDTH)+i] = tab[(y1*WIDTH)+i] = 1;
        for(j = y0 ; j <= y1 ; j++)
            tab[(j*WIDTH)+x0] = tab[(j*WIDTH)+x1] = 1;
    }
}

void creeCercle(unsigned int x0, unsigned int y0,
                unsigned int r, unsigned char* tab) {
    int i;
    for(i = (int)x0-(int)r; i < (int)(x0+r); i++) {
        // If i is outside boundaries, no need to draw
        if(i < HEIGHT) {
            // (x - x_0)^2 + (y - y_0)^2 = r^2 <=> y = ±sqrt(r^2 - (x - x_0)^2) + y_0
            /* abs is necessary as pow fails with negative values */
            float racine = sqrt(powf(r, 2) - powf(abs(i-x0), 2));
            float yp = y0 + racine;
            float yn = y0 - racine;
            // Upper part and lower part of the circle

            // Check boundaries
            if(yp > 0 && WIDTH > yp)
                tab[(int)(i*HEIGHT) + (int)yp] = 1;
            if(yn > 0 && WIDTH > yn)
                tab[(int)(i*HEIGHT) + (int)yn] = 1;
        }
    }
}

void creeCercleTrigo(unsigned int x0, unsigned int y0,
                     unsigned int r, unsigned char* tab) {
    float i;
    // Δ(M(t), M(t+s)) <= 1 <=> s <= cos^-1(1 - 1/r^2) = 1/r
    float step = 1. / r;
    // t ∈ [0, 2π]
    for(i = 0; i < 6.3; i += step) {
        // (x, y) = (r × cos(t) + x_0, r × sin(t) + y_0)
        int x = (int)(r*cos(i) + x0);
        int y = (int)(r*sin(i) + y0);

        if(x > 0 && WIDTH > x && y > 0 && HEIGHT > y)
            tab[(int)(y*HEIGHT) + (int)x] = 1;
    }
}
```

```c

int main () {
    unsigned char pic[HEIGHT*WIDTH];
    // Picture initialization
    blackout(pic);

    XEvent e;
    Display *dpy = XOpenDisplay(NULL); //pointeur sur un ecran
    int noir = BlackPixel(dpy, DefaultScreen(dpy));
    int blanc = WhitePixel(dpy, DefaultScreen(dpy));
    // creation fenetre : taille, couleur... :
    Window w = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), 0, 0,
            WIDTH, HEIGHT, 0, noir, noir);
    XMapWindow(dpy, w); // Affiche la fenetre sur l'ecran
    GC gc = XCreateGC(dpy,w,0,NULL); //On a besoin d'un Graphic Context pour dessiner
    // Il faut attendre l'autorisation de dessiner
    XSelectInput(dpy, w, StructureNotifyMask);

    while (e.type != MapNotify)
        XNextEvent(dpy, &e);
    // On dessine:
    dessine(dpy, w, gc, blanc);

    creeCarre(10, 10, 42, 69, pic);
    creeCercle(100, 100, 100, pic);

    int i;
    // Concentric circles
    for (i = 10; i <= HEIGHT; i += 30) {
        creeCercle(150, 150, i, pic);
        creeCercleTrigo(50, 150, i, pic);
    }

    affiche(dpy, w, gc, pic);
    sleep(10); //on attend 10s avant de quitter
    return 0;
}
```

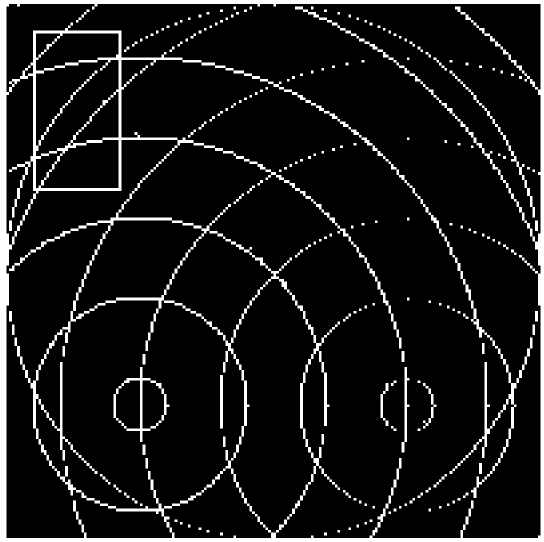

FIGURE 4.1 – Capture d'écran du résultat de l'exercice 4

À gauche les cercles créés avec la méthode trigonométrique et à droite avec la méthode cartésienne

4.1 Question 1

Ce code initialise le système d'affichage, les couleurs noir et blanc, la fenêtre, l'affiche, attends la possibilité de dessiner dessus, dessine la diagonale haut gauche, deux points aux coordonnées $\begin{pmatrix}10\\50\end{pmatrix}$ et $\begin{pmatrix}10\\51\end{pmatrix}$ et finalement "force" l'affichage (autrement dit les affiche).

4.2 Question 2

4.2.1 Solution cartésienne

Afin de dessiner un cercle, on utilise l'équation cartésienne du cercle :

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{4.1}$$

Avec $\begin{pmatrix}x_0\\y_0\end{pmatrix}$ les coordonnées du centre du cercle et r le rayon du cercle.

Cependant, ne pouvant tracer y sans connaître x ni tracer x sans connaître y, il suffit d'isoler la variable x ou la variable y de l'équation 4.1 pour obtenir une équation exploitable pour le programme. On obtient donc :

$$y = \pm\sqrt{r^2 - (x - x_0)^2} + y_0 \tag{4.2}$$

Le fait d'avoir un $\pm$ dans l'équation 4.2 n'est en aucun gênant car il est évidemment possible de calculer les deux possibilités et de les enregistrer dans le tableau qui sera affiché.

Malheureusement, la fonction `pow` de la STL du langage C gère mal les nombres négatifs à la base. Il faut donc utilise la fonction `abs` de la STL pour les valeurs à la base d'une puissance qui risquent d'être négatives, soit $x - x_0$. L'équation 4.2 deviendra donc dans le programme :

$$y_+ = y_0 + \sqrt{r^2 - |x - x_0|^2} \tag{4.3}$$

et

$$y_- = y_0 - \sqrt{r^2 - |x - x_0|^2} \tag{4.4}$$

4.2.2 Solution trigonométrique

Un autre solution serait d'utiliser l'équation paramétrique :

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r \times \cos(t) + x_0 \\ r \times \sin(t) + y_0 \end{pmatrix} \tag{4.5}$$

et de faire varier t sur l'intervalle $[0, 2\pi]$ via une boucle `for` et par étape déterminant la précision du tracé du cercle.

4.2.3 Avantages et inconvénients

Les avantages de la solution cartésienne sont d'abords le fait que l'on puisse déduire deux valeurs à partir d'une seule racine (à une addition près) et donc dessiner deux valeurs "en même temps" et le fait que la précision sur l'axe x est "parfaite" de par le fait que l'on incrémente pixel par pixel. Cependant, cela entraîne des points manquant par rapport à l'axe y, ce qui se remarque sur la capture d'écran.

Les avantages de la solution trigonométrique sont sa simplicité (8 lignes de moins que la solution cartésienne) et le fait qu'il y ait une précision constante sur les deux axes. Cela entraîne aussi le fait que pour avoir une précision correcte sur un cercle quelque soit son rayon, il faut calculer le pas de la boucle `for` en fonction du rayon. Pour cela, il faut que le différence entre deux valeurs adjacente soit inférieure à 1 (la distance minimale entre deux pixels). On a donc :

$$\sqrt{(x(t) - x(t+s))^2 + (y(t) - y(t+s))^2} \leq 1$$

avec $x(t)$ et $y(t)$ les fonctions d'abscisse et d'ordonnée de l'équation paramétrique 4.5 et s l'étape corrélée à la précision du tracé du cercle qui est utilisé dans la boucle `for`.

$$
\begin{aligned}
&\Leftrightarrow (r\times\cos(t)+x_0-r\times\cos(t+s)-x_0)^2+(r\times\sin(t)+y_0-r\times\sin(t+s)-y_0)^2 && \leq 2\\
&\Leftrightarrow (r\times(\cos(t)-\cos(t+s)))^2+(r\times(\sin(t)-\sin(t+s)))^2 && \leq 2\\
&\Leftrightarrow r^2\times\Big((\cos(t)-\cos(t+s))^2+(\sin(t)-\sin(t+s))^2\Big) && \leq 2\\
&\Leftrightarrow r^2\times\Big(\cos(t)^2+\sin(t)^2-2\times\big(\cos(t)\cos(t+s)+\sin(t)\sin(t+s)\big)+\sin(t+s)^2+\cos(t+s)^2\Big) && \leq 2\\
&\Leftrightarrow 2\times r^2\times\big(1-\big(\cos(t)\cos(t+s)+\sin(t)\sin(t+s)\big)\big) && \leq 2\\
&\Leftrightarrow r^2\times\Bigg(1-\Big(\cos(t)\big[\cos(t)\cos(s)-\sin(t)\sin(s)\big]+\sin(t)\big[\sin(t)\cos(s)+\cos(t)\sin(s)\big]\Big)\Bigg) && \leq 1\\
&\Leftrightarrow r^2\times\Big(1-\cos(s)\times\Big(\cos(t)^2+\sin(t)^2\Big)\Big) && \leq 1\\
&\Leftrightarrow r^2\times(1-\cos(s)) && \leq 1\\
&\Leftrightarrow \cos(s) && \geq 1-\frac{1}{r^2}
\end{aligned}
$$

et comme $1-\frac{1}{r^2}\in[-1,1]$ et que la fonction $\cos^{-1}$ est décroissante sur $[-1,1]$ on en déduit que :

$$s\leq\cos^{-1}\left(1-\frac{1}{r^2}\right) \tag{4.6}$$

Cependant, on sait que r est petit donc on peut faire un approximation de $\cos(x)$ en $1-\frac{x^2}{2}$. Cela donne :

$$
\begin{aligned}
& 1-\frac{s^2}{2}\geq 1-\frac{1}{r^2}\\
\Leftrightarrow\quad & \frac{s^2}{2}\leq\frac{1}{r^2}
\end{aligned}
$$

Et par approximation de la racine, on obtient donc :

$$s\leq\frac{1}{r} \tag{4.7}$$

Ce résultat s'explique aussi par le fait que la différentielle de la circonférence d'un cercle est égale à :

$$\partial\mathcal{C}=r\times\partial t\leq 1 \tag{4.8}$$

Avec $\mathcal{C}$ la circonférence du cercle et avec $\partial t=s$. Cela nous donne donc :

$$s\leq\frac{1}{r} \tag{4.9}$$

On utilise donc cette approximation afin d'avoir une étape convenable pour la précision du tracé du cercle. Cela fait donc encore un calcul de plus (mais approximé) par rapport à la méthode cartésienne qui est dans ce cas de figure.