# Rapport du TP 6 d'IGI-2001

Rémi Nicole

# Chapitre 1

# Exercice 1

Listing 1.1 – Programme

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

int main() {
    char nom[20] = "", prenom[20];
    int age;
    FILE* f = fopen("exo1.f.out", "w");

    if(f == NULL) {
        printf("Could not open file\n");
        return 1;
    }

    while(strcmp(nom, "FIN") != 0) {
        printf("Quel est votre nom ?\n");
        scanf("%s", nom);
        if(strcmp(nom, "FIN") == 0)
            break;
        printf("Quel est votre prénom ?\n");
        scanf("%s", prenom);
        printf("Quel est votre age ?\n");
        scanf("%i", &age);

        fprintf(f, "Nom: %s , Prénom: %s , Age: %i\n", nom, prenom, age);

    }

    fclose(f);

    return 0;
}
```

# Chapitre 2

# Exercice 2

Listing 2.1 – Programme

```c
#include <stdio.h>

int main() {
    FILE* f = fopen("exo1.f.out", "r");

    if(f == NULL) {
        printf("Could not open file\n");
        return 1;
    }

    char nom[20], prenom[20];
    int age;

    while(fscanf(f, "Nom: %s , Prénom: %s , Age: %i\n", nom, prenom, &age) == 3)
        printf("Nom:\t%s\nPrénom:\t%s\nAge:\t%i\n", nom, prenom, age);

    fclose(f);
    return 0;
}
```

## 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>

int main() {
    FILE* f = fopen("exo1.f.out", "r");
    char c = 0;

    if(f == NULL) {
        printf("Could not open file\n");
        return 1;
    }

    do {
        c = fgetc(f);
        printf("%c, %i\n", c, c);
    } while(c != EOF);

    return 0;
}
```

# Chapitre 4

# Exercice 4

Listing 4.1 – Programme

```c
#include <X11/Xlib.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sysexits.h>

#ifndef DEFAULT_HEIGHT
#define DEFAULT_HEIGHT 200
#endif

#ifndef DEFAULT_WIDTH
#define DEFAULT_WIDTH 200
#endif

struct pixel {
    unsigned char blue;
    unsigned char green;
    unsigned char red;
};

void affiche(Display *dpy, Window w, GC gc, struct pixel* tab,
             unsigned int width, unsigned int height) {
    unsigned int i , j ;

    for(i = 0 ; i < width ; i++) {
        for(j = 0 ; j < height ; j++) {
            struct pixel* p = &tab[i + (j*height)];

            Colormap cmap = DefaultColormap(dpy, 0);
            GC colorGC = XCreateGC(dpy, w, 0, 0);
            char colorHexCode[8];
            sprintf(colorHexCode, "#%X%X%X", p->red, p->green, p->blue);

            XColor color;
            XParseColor(dpy, cmap, colorHexCode, &color);
            XAllocColor(dpy, cmap, &color);

            XSetForeground(dpy, colorGC, color.pixel);
```

```
39              XDrawPoint(dpy, w, colorGC, i, j);
40          }
41      }
42
43      XFlush(dpy);
44  }
45
46  void blackout(unsigned char* tab, unsigned int width, unsigned int height) {
47      unsigned int i;
48      for(i = 0; i < width*height; i++)
49          tab[i] = 0;
50  }
51
52  void initFromBMP(struct pixel* tab, FILE* f,
53                       unsigned int pixelStart, unsigned int pixelEnd,
54                       unsigned short int bpp, unsigned short int paddingSize,
55                       unsigned int tabWidth, unsigned int tabHeight) {
56
57      fseek(f, pixelStart, SEEK_SET);
58
59      // BMP starts at bottom left
60      unsigned int row = tabHeight - 1, col = 0, i = 0;
61      while (ftell(f) < pixelEnd) {
62          // Store the pixel into the tab at the right place
63          fread(&tab[col + (row*tabHeight)], bpp, 1, f);
64
65          // Then go until the last column
66          col = (col == tabWidth - 1)? 0 : col + 1;
67          // And after the last column, go one row up and restart at column 0
68          if(col == 0) {
69              --row;
70              // And skip the padding
71              fseek(f, paddingSize, SEEK_CUR);
72          }
73          ++i;
74      }
75
76  }
77
78  int main (int argc, char const* argv[]) {
79      // ====== Check command-line usage ====== //
80      if(argc != 2) {
81          fprintf(stderr, "Usage: %s file.bmp\n", argv[0]);
82          return EX_USAGE;
83      }
84
85      FILE* f = fopen(argv[1], "r");
86
87      // ====== Check if readable file ====== //
88      if(f == NULL) {
89          fprintf(stderr, "Error: Could not load the file \"%s\"\n", argv[1]);
90          return EX_NOINPUT;
91      }
92
93      // ====== BMP file magic check ====== //
94      // If it really is a BMP, the first two bytes are the ASCII code of "BM"
```

```
    char magic[3];
    // Read the first two bytes one time starting from the 'magi'c memory block
    fread(&magic, 2, 1, f);
    if (!(magic[0] == 'B' && magic[1] == 'M')) {
        fprintf(stderr, "Error: Not a BMP file\n");
        return EX_DATAERR;
    }

    // ====== Pixel array offset ====== //
    unsigned int pixelStart, pixelEnd;
    fseek(f, 0xA, SEEK_SET);
    fread(&pixelStart, 4, 1, f);
    // Go to the end
    fseek(f, 0, SEEK_END);
    pixelEnd = ftell(f);
    printf("Pixel array starts at offset: %X and ends at offset: %X\n",
            pixelStart, pixelEnd);

    // ====== Width and Height ====== //
    unsigned int width, height;
    // width and height located at offsets 0x12 and 0x12
    fseek(f, 0x12, SEEK_SET);
    // Load the width and height into the corresponding variables (4 bytes each)
    fread(&width, 4, 1, f);
    fread(&height, 4, 1, f);

    printf("Height = %u, Width = %u\n", height, width);

    // ====== Bytes per pixels ====== //
    unsigned short int bpp;
    // The number of bytes per pixels located at offset 0x1C
    fseek(f, 0x1C, SEEK_SET);
    fread(&bpp, 2, 1, f);
    // Convert to bytes
    bpp /= 8;

    printf("Number of bytes per pixels: %hu\n", bpp);
    if(bpp == 4)
        fprintf(stderr, "Warning: Alpha channel is not supported.\n");
    else if(bpp != 3) {
        fprintf(stderr, "Error: Only RGB and RGBA file supported.\n");
    }

    // ====== Padding Size ====== //

    // There is a padding column until the row reaches a multiple of 4 bytes
    // bytesPerRow = rowSize × bytesPerPixel
    // paddingSize = { 0                    if bytesPerRow%4 = 0
    //               { 4 − (bytesPerRow%4)  else
    unsigned short int paddingSize = (4 - ((width * bpp) % 4) ) % 4;

    struct pixel pic[width*height];

    // ====== Read pixel array ====== //
    initFromBMP(pic, f, pixelStart, pixelEnd, bpp, paddingSize, height, width);
```

```
149     fclose(f);
150
151     /* // Picture initialization */
152     /* blackout(pic, width, height); */
153
154     XEvent e;
155     Display *dpy = XOpenDisplay(NULL); //pointeur sur un ecran
156     int noir = BlackPixel(dpy, DefaultScreen(dpy));
157     // creation fenetre : taille, couleur... :
158     Window w = XCreateSimpleWindow(dpy, DefaultRootWindow(dpy), 0, 0,
159             width, height, 0, noir, noir);
160     XMapWindow(dpy, w); // Affiche la fenetre sur l'ecran
161     GC gc = XCreateGC(dpy,w,0,NULL); //On a besoin d'un Graphic Context pour dessiner
162     // Il faut attendre l'autorisation de dessiner
163     XSelectInput(dpy, w, StructureNotifyMask);
164
165     while (e.type != MapNotify)
166         XNextEvent(dpy, &e);
167
168     affiche(dpy, w, gc, pic, height, width);
169     sleep(10); //on attend 10s avant de quitter
170
171     return 0;
172 }
```


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 :

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

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`.

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

et comme $1 - \dfrac{1}{r^2} \in [-1, 1]$ 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 $1 - \dfrac{x^2}{2}$. Cela donne :

$$1 - \frac{s^2}{2} \geq 1 - \frac{1}{r^2}$$
$$\Leftrightarrow \qquad \frac{s^2}{2} \leq \frac{1}{r^2}$$

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.