TD 8 : compilation séparée, graphiques

Exercice 1 (Compilation séparée)**.**
Consulter les quatre fichiers suivants et préciser leurs rôles respectifs :

. factorielle.h

```
/** La fonction factorielle
 *  @param n un nombre entier positif
 *  @return n!
 **/
int factorielle(int n);
```

. factorielle.cpp

```
#include "factorielle.h"

int factorielle(int n) {
    int resultat = 1;
    for ( int k = 1; k <= n; k++ )
        resultat = resultat * k;
    return resultat;
}
```

. factorielle-exemple.cpp

```
#include <iostream>
#include "factorielle.h"
using namespace std;

int main() {
    int n;
    cout << "Entrez un entier n" << endl;
    cin >> n;
    cout << n << "! = " << factorielle(n) << endl;
    return 0;
}
```

. factorielle-test.cpp

```
#include <iostream>
#include "factorielle.h"
using namespace std;

/** Infrastructure minimale de test **/
#define ASSERT(test) if (!(test)) cout << "Test failed in file " << __FILE__ << " line " << __LIN

/** Les tests de la fonction factorielle **/
void factorielleTest() {
    ASSERT( factorielle(0) == 1  );
    ASSERT( factorielle(1) == 1  );
    ASSERT( factorielle(2) == 2  );
    ASSERT( factorielle(3) == 6  );
    ASSERT( factorielle(4) == 24 );
}

/** Cette fonction main ne sert qu'à lancer les tests **/
int main() {
    factorielleTest();
    return 0;
}
```

Correction : Ensemble, ces quatre fichiers forment une bibliothèque, qui implante une fonction factorielle, ainsi que des tests unitaires et un exemple d'utilisation de la fonction. Le fichier `factorielle.cpp` contient la définition (le code) de la fonction factorielle. Le fichier `factorielle.h` est le fichier d'entête : il contient la déclaration (l'entête) de la fonction factorielle, avec sa documentation. Ce fichier permet de pouvoir compiler des programmes qui utilisent la fonction factorielle, même sans avoir le code de la fonction factorielle. Pour cela il faut avoir mis la ligne `#include "factorielle.h"` dans le programme. Le fichier `factorielle-exemple.cpp` donne un exemple de programme utilisant la fonction factorielle. Enfin le fichier `factorielle-test.cpp` contient les tests de la fonction factorielle, ainsi qu'un programme qui lance ces tests.

Graphiques

En TP, nous utiliserons la bibliothèque MLV [1] qui permet de faire simplement du multimédia en C et C++, que l'on soit sous Linux, Windows, MacOS ou autre. Cette bibliothèque est développée par une équipe menée par Adrien Boussicault, originellement pour l'enseignement à l'Université de Marne-la-Vallée. Elle est basée sur la bibliothèque SDL [2].

Voici un exemple de programme utilisant cette bibliothèque :

. mlv-exemple.cpp

```
#include "MLV.h"
using namespace mlv;

int main() {
    // Crée et affiche une fenêtre de taille 640x480
    window_t mafenetre = window_t( "Premier essai", "essai", 640, 480 );
    // Dessine un point rouge de coordonnées x=120 et y=50
    mafenetre.draw_point({120, 50}, color::red);
    // Actualise la fenêtre
```

1. http://www-igm.univ-mlv.fr/~boussica/mlv/
2. https://www.libsdl.org/

```
    mafenetre.update();
    // Attend 10 secondes
    mafenetre.wait_seconds( 10 );
}
```

Noter dans cet exemple le nouveau type `window_t` qui représente une fenêtre, et les fonctions `draw_point`, `update` et `wait_seconds`. La syntaxe pour appeler ces fonctions est similaire à celle pour appeler les fonctions `size` ou `push_back` sur un tableau. Il s'agit en fait de *méthodes* que l'on appelle sur un *objet*.

Exercice 2 (Premiers dessins)**.**
En vous inspirant de l'exemple fourni, écrire des fragments de programme qui, respectivement,

(1) dessine un point noir (*black*) de coordonnées $(418, 143)$;

Correction :

```
window.draw_point({418, 143}, color::black);
```

(2) dessine un segment marron (*saddlebrown*) reliant les points $(100, 200)$ et $(200, 200)$;

Correction :

```
for(int x = 100; x <= 200; x++)
    window.draw_point({x, 200}, color::saddlebrown);
```

(3) dessine un segment marron reliant les points $(200, 300)$ et $(200, 400)$;

Correction :

```
for(int y = 300; y <= 400; y++)
    window.draw_point({200, y}, color::saddlebrown);
```

(4) dessine un rectangle marron horizontal plein dont les sommets diagonaux sont $(400, 150)$ et $(500, 200)$;

Correction :

```
for ( int x = 400; x <= 500; x++ )
    for ( int y = 150; y <= 200; y++ )
        window.draw_point({x, y}, color::saddlebrown);
```

(5) dessine un segment marron reliant les points $(400, 300)$ et $(500, 400)$;

Correction :

```
for (int t = 0; t <= 100; t++)
    window.draw_point({400+t, 300+t}, color::saddlebrown);
```

(6) dessine le rectangle vertical vide dont les sommets diagonaux sont $(200, 200)$ et $(400, 300)$;

Correction :

```
for ( int x = 200; x <= 400; x++ ) {
    window.draw_point({x,200}, color::saddlebrown);
    window.draw_point({x,300}, color::saddlebrown);
}
for ( int y = 200; y <= 300; y++ ) {
    window.draw_point({200,y}, color::saddlebrown);
    window.draw_point({400,y}, color::saddlebrown);
```

```
    }
```

(7) dessine un cercle marron de centre $(415, 145)$ et de rayon 10 ;
Indication : utiliser les fonctions `cos` et `sin` ;

Correction :

```
    double pi = 3.14159;
    point_t centre = {415, 145};
    int rayon = 10;
    double dt = .5 / rayon;
    for ( double t = 0; t <= 2*pi; t += dt )
        window.draw_point({centre.x + rayon*cos(t),
                           centre.y + rayon*sin(t)},
                           color::saddlebrown);
```

(8) dessine un disque rouge de centre $(700, 100)$ et de rayon 50 ;
Indication : utiliser la définition d'un disque.

Correction :

```
    centre = {700, 100};
    rayon = 50;
    for ( int x = -rayon; x <= rayon; x++ )
        for ( int y = -rayon; y <= rayon; y++ )
            if ( x*x + y*y <= rayon * rayon )
                window.draw_point({centre.x+x,
                                   centre.y+y},
                                   color::red);
```

Exercice ♣ 3.
Généraliser l'exercice 2 en implantant les trois fonctions suivantes. À noter qu'elles prennent la fenêtre où dessiner comme premier paramètre (`window`). Le symbole `&` indique que cette fenêtre est passée par référence afin que les fonctions puissent la modifier. Vous pouvez essentiellement ignorer ce détail technique dont vous verrez les tenants et aboutissants au deuxième semestre.

```
    // Dessine un cercle non rempli
    void cercle (window_t &window, point_t centre, int rayon);
    // Dessine un cercle rempli
    void disque (window_t &window, point_t centre, int rayon);
    // Dessine un segment
    void segment(window_t &window, point_t p1, point_t p2);
```

Correction :

```
void cercle(window_t & window, point_t centre, int rayon) {
    double pi = 3.14159;
    double dt = .5 / rayon;
    for ( double t = 0; t <= 2*pi; t += dt )
        window.draw_point({centre.x + rayon*cos(t),
                           centre.y + rayon*sin(t)},
                          color::red);
    window.update();
```

```
}
```

```
void disque(window_t & window, point_t centre, int rayon) {
    for (int x = -rayon; x <= rayon; x++)
        for (int y = -rayon; y <= rayon; y++)
            if ( x*x + y*y <= rayon * rayon )
                window.draw_point({centre.x+x, centre.y+y}, color::red);
    window.update();
}
```

```
void segment(window_t & window, point_t point1, point_t point2) {
    int xmin, xmax, ymin, ymax;
    if (point1.x < point2.x) {
        xmin = point1.x;
        xmax = point2.x;
    } else {
        xmin = point2.x;
        xmax = point1.x;
    }
    if (point1.y < point2.y) {
        ymin = point1.y;
        ymax = point2.y;
    } else{
        ymin = point2.y;
        ymax = point1.y;
    }

    int i, j;
    if ( xmin == xmax ) //droite verticale
        for ( j = ymin; j <= ymax; j++ )
            window.draw_point(point_t(xmin,j), color::red);
    if ( ymin == ymax ) //droite horizontale
        for(i = xmin; i <= xmax; i++)
            window.draw_point(point_t(i, ymin), color::red);

    float a, b, ii, jj;
    if ( (xmax-xmin >= ymin-ymax) and ymax  != ymin ) { // variation max en x et pas horizon
        a = (float) (point1.y-point2.y)/((float)(point1.x-point2.x)); // calcul de la pente
        b = point1.y - a*point1.x;
        for (i=xmin; i <= xmax; i++) {
            jj = a*i+b;
            j = jj; // j est la partie entière de jj
            if(jj-j > 0,5)
                j++;    // on prend le j le plus proche des coordonnées calculées
            window.draw_point(point_t(i,j), color::red);
        }
    }
    if ( (ymax-ymin >= xmin-xmax) and xmax  != xmin ) { // variation max en y et pas vertica
        //Le but est d'avoir une meilleure précision
        a = (float)(point1.y-point2.y)/((float) (point1.x-point2.x)); // calcul de la pente
        b = point1.y - a*point1.x;
        for (j=ymin; j <= ymax; j++) {
            ii = (j-b)/a;
            i = ii;
```

```
            if(ii-i > 0,5)
                i++;
            window.draw_point(point_t(i,j), color::red);
        }
    }
    window.update();
}
```

Exercice 4 (Souris et Clavier).
Voici maintenant un fragment de programme interactif utilisant la bibliothèque MLV pour réagir à des actions avec la souris ou le clavier.

```
// Crée et affiche une fenêtre de taille 640x480
window_t window = window_t( "Premier essai", "essai", 640, 480 );

// Attend que l'utilisateur clique sur le bouton gauche de la souris
// et stocke les coordonnées du point où à cliqué l'utilisateur dans
// la variable point.
point_t point = window.wait_mouse();

// Affiche les coordonnées du point
cout << point.x << " " << point.y << endl;

// Attend que l'utilisateur tape sur une touche
// et stocke la touche et d'autres informations dans
// la variable evenement
event::keyboard_t evenement = window.wait_keyboard();
```

Noter le nouveau type `point_t`. C'est un *type composite* : un point `point` est composé de ses deux coordonnées x et y, auxquelles on accède avec la syntaxe `point.x` et `point.y`. Techniquement parlant, il s'agit d'un *enregistrement* (*struct* en anglais) ; vous en verrez les détails au deuxième semestre.

On suppose de plus disposer d'une fonction permettant de dessiner un segment comme dans l'exemple suivant :

```
window.draw_line({20,20}, {500,580}, color::red);
```

(1) En vous inspirant du programme précédent, écrire un programme qui attend que l'utilisateur clique sur deux points de l'écran et trace le segment les reliants.

Correction :

```
    window_t window("Souris et clavier - 1 ", "Souris", 640, 480 );
    point_t point1 = window.wait_mouse();
    point_t point2 = window.wait_mouse();
    window.draw_line(point1, point2, color::red);

    window.update();
    window.wait_seconds( 10 );
    return 0;
}
```

(2) Écrire un programme qui attende que l'utilisateur clique sur quatre points et dessine le quadrilatère ayant ces quatre points comme sommets.

Correction :

```
    window_t window("Souris et clavier - 2 ", "Souris", 640, 480 );

    point_t point1 = window.wait_mouse();
    point_t point2 = window.wait_mouse();
    window.draw_line(point1, point2, color::red);
    window.update();

    point_t point3 = window.wait_mouse();
```

```
    window.draw_line(point2, point3, color::red);
    window.update();

    point_t point4 = window.wait_mouse();
    window.draw_line(point3, point4, color::red);
    window.draw_line(point4, point1, color::red);
    window.update();
```

(3) Écrire un programme qui dessine des polygones de la façon suivante : l'utilisateur clique sur des points successifs. À chaque clic, le programme relie les deux derniers points. Si l'utilisateur clique près du point initial, le polygone se ferme, et le programme commence un nouveau polygone.

Correction :

```
int main() {
    window_t window("Souris et clavier - 3 ", "Souris et clavier", 640, 480 );

    while (true) {
        point_t initial = window.wait_mouse();
        point_t point1 = initial;
        point_t point2;
        bool continuer = true;

        do {
            point2 = window.wait_mouse();
            if ( abs(point2.x-initial.x) <= 2 and abs(point2.y-initial.y) <= 2 ) {
                point2 = initial;
                continuer = false;
            }
            window.draw_line(point1, point2, color::red);
            window.update();
            point1 = point2;
        } while ( continuer );
    }
    return 0;
}
```