TD 10 : compilation séparée, graphiques

Exercice 1 (Compilation séparée)**.**
Annoter les quatre fichiers suivants en précisant leurs rôles respectifs et en indiquant où se trouvent entête, définition, documentation, tests, et utilisation de la fonction `factorielle`.

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-test.cpp

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

/** Infrastructure minimale de test **/
#define CHECK(test) if (!(test)) cout << "Test failed in file " \
         << __FILE__ << " line " << __LINE__ << ": " #test << endl

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

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

factorielle-exemple.cpp

```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;
}
```

Graphiques

En TP, nous utiliserons la bibliothèque `SFML` [1], une bibliothèque largement utilisée facilitant le développement de jeux ou d'applications multimédias grâce à une grande panoplie de modules (fenêtrage, graphismes, audio, réseau, ...), principalement développée par Laurent Gomila. Nous allons nous en servir pour créer une fenêtre et dessiner dedans.

La `SFML` n'étant pas immédiatement intuitive à utiliser, nous avons écrit une micro bibliothèque composée de quelques primitives pour vous simplifier les premiers pas. Cette bibliothèque, documentée dans `primitives.h`, a vocation à être la plus transparente possible. À vous de la lire en détail pour vous l'approprier et, par la suite, vous en passer.

Voici un exemple de programme utilisant la `SFML` et notre bibliothèque `primitives` :

exemple-graphisme1.cpp

```cpp
#include <SFML/Graphics.hpp>
using namespace sf;

#include "primitives.h"

int main()
{
    // Crée une fenêtre de taille 640x480
    RenderWindow window(VideoMode(640, 480), "Ma super fenêtre");

    // Remplit la fenêtre de blanc
    window.clear(Color::White);

    draw_point(window, {120, 5}, Color::Red);

    // Actualise la fenêtre
    window.display();

    // Attend 10 secondes
    sleep(seconds(10));

    return 0;
}
```

Noter dans cet exemple le nouveau type `RenderWindow` représentant une fenêtre, la fonction `clear`, qui permet de vider la fenêtre, `sleep` qui permet d'attendre un temps donné et enfin `draw_point` (qui vient de `primitives.cpp`) permettant de dessiner un point.

1. `https://www.sfml-dev.org`

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

(1) dessinent un point noir (*Black*) de coordonnées $(418, 143)$;

(2) dessinent un segment blanc (*White*) reliant les points $(100, 200)$ et $(200, 200)$;

(3) dessinent un segment rouge (*Red*) reliant les points $(200, 300)$ et $(200, 400)$;

(4) dessinent le rectangle horizontal vide de contour rouge dont les sommets diagonaux sont $(200, 200)$ et $(400, 300)$;

(5) dessinent un rectangle horizontal plein noir dont les sommets diagonaux sont $(400, 150)$ et $(500, 200)$;

(6) dessinent un segment rouge reliant les points $(400, 300)$ et $(500, 400)$;

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

(8) dessinent un disque jaune (*Yellow*) de centre $(700, 100)$ et de rayon 50 ;
Indication : utiliser la définition d'un disque.

Exercice ♣ 3 (Fonctions de dessin)**.**
Notre bibliothèque `primitives` contient entre autres les fonctions suivantes :

```
/** Affiche une ligne de couleur entre deux positions données
 * @param w une fenêtre ouverte dans laquelle dessiner
 * @param pos1 les coordonnées du premier point de la ligne
 * @param pos1 les coordonnées du dernier point de la ligne
 * @param color la couleur de la ligne
 */
void draw_line (RenderWindow& w, Point pos1, Point pos2, Color color);

/** Affiche un cercle coloré vide
 * @param w une fenêtre ouverte dans laquelle dessiner
 * @param center la position du centre du cercle
 * @param r le rayon du cercle
 * @param color la couleur du trait
 */
void draw_circle (RenderWindow& w, Point center, int r, Color color);

/** Affiche un cercle coloré plein
 * @param w une fenêtre ouverte dans laquelle dessiner
 * @param center la position du centre du cercle
 * @param r le rayon du cercle
 * @param color la couleur du trait et du remplissage
 */
void draw_filled_circle(RenderWindow& w, Point center, int r, Color color);
```

À noter qu'elles prennent la fenêtre où dessiner comme premier paramètre (`w`). 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 les aboutissants au deuxième semestre.

Notez aussi les variables de type `Point`. Ces dernières représentent des coordonnées. On peut créer un point et accéder à ses coordonnées comme suit :

```
Point p = {42, 2713};
int x = p.x; // 42
int y = p.y; // 2713
```

Techniquement parlant, il s'agit d'un *enregistrement* (*struct* en anglais) ; vous en verrez les détails au deuxième semestre.

Dans notre bibliothèque, ces fonctions sont implémentées via des appels directs à des fonctions de la `SFML`. Par exemple :

```
void draw_filled_circle(RenderWindow& w, Point center, int r, Color color) {
    CircleShape shape(r);
    shape.setPosition(center);
    shape.setOutlineThickness(0.f);
    shape.setFillColor(color);
    w.draw(shape);
}
```

(1) Réimplantez ces fonctions, en n'utilisant que des appels à `draw_point`.

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

exemple-graphisme2.cpp

```
    // Crée une fenêtre de taille 640x480
    RenderWindow window(VideoMode(640, 480), "Ma super fenêtre");
    window.clear(Color::White);
    window.display();

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

    // 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::KeyEvent evenement = wait_keyboard(window);

    // Affiche bonjour si la touche est o
    if ( evenement.code == Keyboard::Key::O )
        cout << "Bonjour" << endl;
```

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

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

(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.