TP 8 : compilation séparée, graphiques

Exercice 1 (Préliminaires : compilation séparée)**.**

(1) Télécharger et extraire l'archive `Semaine8.zip` dans votre répertoire `Info111`

(2) Créer un nouveau projet `Code::Blocks` avec `File -> New -> Project`, puis :
- Catégorie : application console
- Langage : C++
- Titre du projet : `factorielle-exemple`
- Répertoire : `.../Semaine8/`
- Nom du fichier résultant : `.../Semaine8/factorielle-exemple.cbp`
  Attention : `CodeBlocks` propose par défaut le nom de fichier `.../Semaine8/factorielle-exemple/factorielle-exemple.cbp` ; il faut supprimer la création du sous-répertoire `factorielle-exemple/`.
- Options par défaut pour le reste

(3) Consulter le contenu des fichiers suivants et les ajouter au projet (`Project -> Add File`) : `factorielle.h`, `factorielle.cpp`, `factorielle-exemple.cpp`. Lorsqu'on vous le demande, cocher les cases « Debug » et « Release ».

(4) Supprimer `main.cpp` des sources du projet

(5) Compiler et lancer le programme.

(6) Créer un autre projet `factorielle-test` comme le précédent, mais en remplaçant le programme principal `factorielle-exemple.cpp` par `factorielle-test.cpp`.

Exercice 2 (Préliminaires : compilation séparée)**.**
Récupérer le fichier `fibonacci.cpp` que vous aviez implanté au TP 4, et le découper en plusieurs fichiers en suivant le modèle de l'exercice précédent (avec création de projets).

Exercice 3 (Préliminaires : graphiques avec MLV)**.**
Attention : dans les salles de TP, la bibliothèque MLV n'est installée que sous Linux. Pour ceux qui souhaitent travailler sur leur machine personnelle, voir les instructions d'installation [1] de la documentation de MLV.

(1) Créer un nouveau projet :
- Catégorie : application console
- Titre du projet : `mlv-exemple`
- Strictement comme précédemment pour le reste (y compris supprimer la création par défaut d'un sous-répertoire !)

(2) Supprimer `main.cpp` des sources du projet.

(3) Ajouter au projet les fichiers `mlv-exemple.cpp` et `MLV.h`.

(4) Ajouter la bibliothèque `MLV` (Projet → Build options → Linker settings → link libraries → Add, puis taper `MLV` et valider.)

(5) Consulter le contenu de `mlv-exemple.cpp`.

(6) Compiler et lancer le programme.

1. `http://www-igm.univ-mlv.fr/~boussica/mlv/api/French/html/installation.html`

(7) Alternativement, vous pouvez simplement compiler ce programme depuis le terminal avec :

```
g++ mlv-exemple.cpp -std=c++11 -lMLV -o mlv-exemple
```

puis le lancer avec :

```
./mlv-exemple
```

Il faut bien sûr, au préalable, être allé avec `cd` dans le dossier contenant ce programme.

Est-ce plus compliqué ?

Exercice 4 (Premier dessin)**.**
Reprendre l'exercice 2 du TD en complétant le programme fourni `graphisme-premier-dessin.cpp`. Pour cela, vous commencerez par créer un nouveau projet `graphisme-premier-dessin` comme dans l'exercice précédent. Puis vous implanterez chacun des items en vérifiant à chaque fois le résultat. N'hésitez pas à changer la valeur de la variable `delai` pour voir le résultat s'afficher plus longtemps.

Correction :

```
#include "MLV.h"
#include <cmath>
using namespace mlv;
using namespace std;  // Pour les exceptions dans la version distribuée

int main() {
    int delai = 1;

    window_t window = window_t("Premier dessin", "Premier dessin", 900, 480 );
    window.clear(color::lightgreen);

    // Dessine un point noir de coordonnées (418, 143)
    window.draw_point({418, 143}, color::black);


    // Dessine un segment marron entre les points (100,200) et (200,200)
    for(int x = 100; x <= 200; x++)
        window.draw_point({x, 200}, color::saddlebrown);
    window.update();
    window.wait_seconds( delai );

    // Dessine un segment entre les points (200,300) et (200,400)
    for(int y = 300; y <= 400; y++)
        window.draw_point({200, y}, color::saddlebrown);
    window.update();
    window.wait_seconds( delai );

    // Dessine un segment entre les points (400,300) et (500,400)
    for (int t = 0; t <= 100; t++)
        window.draw_point({400+t, 300+t}, color::saddlebrown);
    window.update();
    window.wait_seconds( delai );

    // Dessine un rectangle vertical plein de sommets diagonaux (400,150) et (500,200)
    for ( int x = 400; x <= 500; x++ )
```

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

    // Dessine un rectangle vertical vide de sommets diagonaux (200,200) et (400,300)
    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);
    }
    window.update();
    window.wait_seconds( delai );

    // Dessine un cercle marron de centre (415,145) et de rayon 10
    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);
    window.update();
    window.wait_seconds( delai );

    // Dessine un disque rouge de centre (700, 100) et de rayon 50
    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);
    window.update();
    window.wait_seconds( 10 );
    return 0;
}
```

Exercice 5.
Pour vous donner une idée des primitives graphiques de MLV, consulter l'exemple fourni `mlv-exemple3.cpp`.

Exercice 6 (Souris et clavier)**.**
Reprendre l'exercice 4 du TD.

Correction : Voir la correction du TD.

Exercice ♣ 7 (Jeu du démineur)**.**
Reprendre le jeu du démineur du TP 6 en rajoutant une petite interface graphique.

On dessinera à chaque étape la grille dans la fenêtre graphique plutôt que de l'afficher dans le terminal. Pour aller plus loin, l'utilisateur pourra marquer une case comme comportant une bombe avec un clic droit de la souris et démasquer une case avec un clic gauche (utiliser par exemple `wait_mouse()`).

À vous de concevoir les fonctions à introduire pour décomposer le problème.