INFO-F-202
Langages de programmation 2
Université libre de Bruxelles
Laboratoire 1

Justin Dallant John Iacono
Yannick Molinghen Alexis Reynouard

22 Septembre 2021

0 À propos du cours

Pour des informations générales sur la structure du cours, regardez la première vidéo (plus tard !). Chaque semaine, il sera indiqué quelles vidéos vous devriez regarder avant de venir au labo.

Liens utiles :
— Teams : Vidéos et fichiers (voir les onglets)
— UV : Quel vidéo à voir pour quand ?
— UV : Références

1 Introduction

Dans les labos, le but est de programmer ! Formez des groupes et travaillez ensemble pour résoudre les tâches de programmation ci-dessous. Posez vos questions, soit directement à l'assistant, soit via le canal Teams "Lab".

Rien n'est noté ici, alors aidez-vous les uns les autres et n'hésitez pas à partager du code. La chose la plus importante est que vous compreniez.

Les solutions seront publiées après le laboratoire. Assurez-vous de bien comprendre les solutions, car chaque semaine s'appuiera sur la semaine précédente.

2 Objectif

Cette semaine, vous devez vous familiariser avec les bases de la bibliothèque FLTK et vous rappeler comment coder. Pas de programmation orientée objet cette semaine.

Fltk est une bibliothèque pour C++ qui vous permet d'écrire des programmes graphiques. C'est simple et fonctionne avec Windows, Linux et MacOS. Il est installé sur les ordinateurs du laboratoire et nous vous encourageons à l'installer sur vos propres ordinateurs. Nous l'utiliserons dans chaque laboratoire et vous l'utiliserez pour vos projets. Nous n'utiliserons qu'un très petit nombre des fonctionnalités de Fltk. Bien que fltk lui-même ne soit pas très populaire, il a été choisi en raison de sa portabilité et de sa simplicité. Ce que vous apprendrez peut être facilement appliqué à d'autres bibliothèques de développement graphique.

3 Compilation à l'aide de FLTK

Compiler avec fltk

Vous pouvez compiler un ficher fltk en utilisant le flag `-lfltk` comme dans l'exemple ci-dessous.

```
g++ input_file.cpp -o output_file -lfltk
```

Si cela ne fonctionne pas, vous pouvez demander à fltk de choisir lui-même le compilateur et les flags à utiliser.

```
fltk-config --compile -g input_file.cpp
```

Il est cependant conseillé d'utiliser votre propre commande pour la compilation, pour avoir un meilleur contrôle. En outre vous devriez utiliser le standard C++ le plus récent (–std), activer tous les avertissements (-W) et activer les informations de débuggage (-g) pour pouvoir utiliser un débugger.

```
g++-10 --std='c++20' -g -Wall -Wextra
g++-9 --std='c++17' -g -Wall -Wextra
```

Vous pouvez également utiliser fltk-config pour vous donner tous les flags nécessaires pour la compilation, comme dans l'exemple ci-dessous.

```
g++ -std='c++17' `fltk-config --cxxflags` input_file.cpp `fltk-config --ldflags`
-o output_file -Wall
```

4 Idée générale

Vous avez reçu un code qui contient les classes et fonctions suivantes :
— `main` : Ne touchez pas pour l'instant
— `MainWindow` : Ne touchez pas pour l'instant
— `init()` : Celui-ci est appelé une fois au début du programme. Mettez ici n'importe quel code nécessaire pour initialiser quoi que ce soit. Ne mettez pas de code de dessin ici.
— `draw()` : Ce code est appelé à plusieurs reprises, jusqu'à 60 fois par seconde. Ici, vous devriez avoir tout le code de dessin. Chaque fois que `draw` est appelé, vous devez tout redessiner, à partir d'un écran vide.
— `mouseMove(int x, int y)` : Ceci est appelé chaque fois que la souris se déplace, avec l'emplacement actuel de la souris. Ne mettez pas de code de dessin dans cette fonction.
— `keyPressed(int keyCode)` : Ceci est appelé chaque fois qu'une touche est enfoncée, et passe la clé en ASCII (c'est-à-dire que vous pouvez écrire `if keyCode=='a'`) et pour les clés spéciales voir https://www.fltk.org/doc-1.3/enumerations.html fpour les constantes.

5 Drawing

Voici tout ce que vous devez savoir pour aujourd'hui :
— `Fl_Color` : Le type utilisé pour stocker une couleur
— `fl_rgb_color(r,g,b)` : Renvoie une couleur en fonction des valeurs rouge, verte et bleue comprises entre 0 et 255.
— `fl_draw_box(FL_FLAT_BOX,x,y,w,h,boxColor)` : Dessine un rectangle plein avec un coin en (x, y), une largeur de w, une hauteur de h et de la couleur donnée.

Pour une référence complète sur d'autres fonctions de dessin, voir : https://www.fltk.org/doc-1.3/group__fl__drawings.html.

6 Buts

Effectuez ces tâches, une à la fois. Essayez d'aller le plus loin possible.
— Dessiner un carré
— Dessinez un carré qui se déplace, par exemple en formant un cercle.
— Dessine un carré exactement sous la souris
— Dessine un carré qui suit lentement la souris
— Dessine un carré qui suit lentement la souris et s'il le touche il réapparaît ailleurs au hasard sur l'écran

— Idem et ajouter, à chaque fois que la souris touche le carré, un “cratère” qui restera toujours sur l’écran à cet endroit.
— Permet à l’utilisateur d’appuyer sur la barre d’espacement pour effacer les cratères (et ne pas quitter le programme).
— Identique à ci-dessus mais ajouter une petite animation d’explosion chaque fois que la souris touche le carré.
— Permet à l’utilisateur d’appuyer sur 1-8 pour sélectionner laquelle des huit démonstrations ci-dessus exécuter.

Conseils :

— Utilisez des variables globales pour conserver tout état nécessaire entre les appels à dessiner. Nous verrons de meilleurs moyens que les variables globales pour maintenir l’état dans les futurs laboratoires.
— Pour dessiner quelque chose qui bouge comme dans la deuxième tâche, vous aurez souvent besoin de savoir quelle “heure” il est. Gardez donc une variable globale que vous incrémentez à chaque appel de draw.
— Utilisez cette définition : `struct Point {float x;float y;};`
— `rand()%500` vous donne un entier aléatoire dans la plage $[0, 499]$.
— Si vous dessinez les points $(\sin x, \cos x)$ avec tous les x, vous obtenez un cercle.