INFO-F-202
Langages de programmation 2
Université libre de Bruxelles
Laboratoire 5

Justin Dallant John Iacono
Yannick Molinghen Alexis Reynouard

1 Goal

Le but de cette classe est de créer correctement un constructeur, un destructeur, un constructeur de copie et un opérateur d'assignation pour une classe simple utilisant la mémoire dynamique.

2 Code fourni

On vous donne un nouvelle classe Tree.

Dans Canvas, un vecteur de pointeurs vers des arbres est créé dans le constructeur, et dans Canvas::draw, draw est appelé sur chaque arbre vers lequelle le vecteur pointe.

Il existe d'autres classes : Rectangle, Texte, Traduction, Rotation et une nouvelle classe Line. Vous n'avez pas besoin de modifier ces classes.

Il vous suffit de changer Tree et le constructeur de Canvas

3 Tree

On vous donne le code d'une classe Tree qui dessine un arbre.

```
class Tree{
  Tree *left,*right;
  Line line;
public:
  Tree();
  Tree(Tree *left,Tree *right);
  void draw();
  void setColor(Fl_Color newColor);
};
```

Chaque arbre a une `Line` et des pointeurs vers deux autres `Tree`, qui pourraient être `nullptr`. Draw fonctionne en traçant une ligne, puis en dessinant les deux arbres en haut, légèrement inclinés à gauche et à droite. Vous n'aurez pas besoin de changer la méthode draw.

```
void Tree::draw(){
  // DO NOT CHANGE THIS
  line.draw();
```

```
  Translation t({0,-30});
  if (left) {
    Rotation r({0,0},-10);
    left->draw();
  }
  if (right) {
    Rotation r({0,0},10);
    right->draw();
  }
}
```

Il existe également une méthode setColor, qui modifie la couleur d'un arbre. Il change la couleur de la ligne, puis appelle setColor sur les arbres de gauche et de droite, s'ils existent. Vous n'aurez pas besoin de changer la méthode setColor.

```
void Tree::setColor(Fl_Color newColor){
  // DO NOT CHANGE THIS
  line.setColor(newColor);
  if (left) left->setColor(newColor);
  if (right) right->setColor(newColor);
}
```

4 Canvas

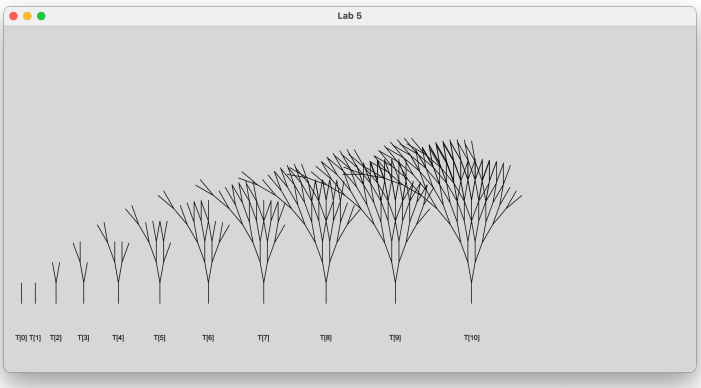
Canvas a une variable d'instance T de type `vector<Tree *> T;`.

Le constructeur de canvas crée deux arbres construits par défaut, ajoute leurs pointeurs au T, puis continue à créer et ajouter à T d'autres arbres en utilisant des pointeurs vers des arbres existants dans T.

```
Canvas::Canvas(Fl_Window *fltkWindow):fltkWindow{fltkWindow} {
  T.push_back(new Tree);
  T.push_back(new Tree);
  for (int i=0;i<9;i++)
    T.push_back(new Tree{T[i],T[i+1]});
 }
```

(Ne vous inquiétez pas pour la fltkWindow, elle est utilisée pour stocker un pointeur vers une classe de fltk, où nous appelons fltkWindow->hide() pour terminer le programme)

Le résultat du code initial est le suivant :



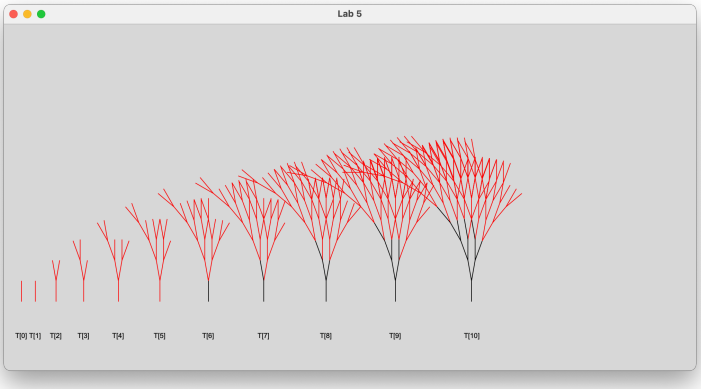
5 Exercice

Le seul code que vous devez écrire est : constructeurs, destructeur et opérateur = pour Tree, et décommentez le code de test dans le constructeur de Canvas. Ne changez rien d'autre.

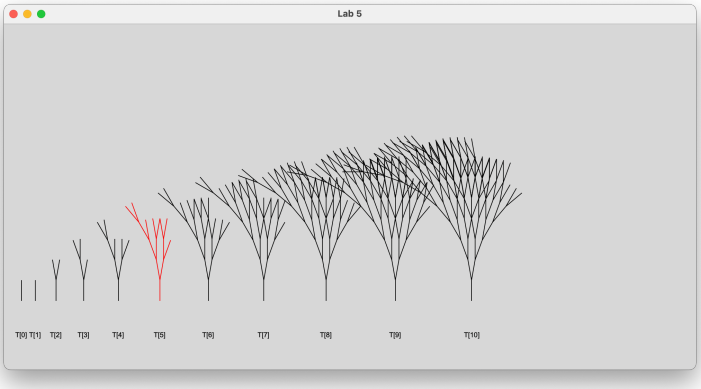
Décommentez la ligne suivante dans le constructeur de Canvas :

```
1 T[5]->setColor(FL_RED);
```

Cela a l'effet suivant :



Ce n'est pas ce que nous voulons. Nous voulons que le cinquième arbre soit rouge et les autres inchangés, comme ceci :



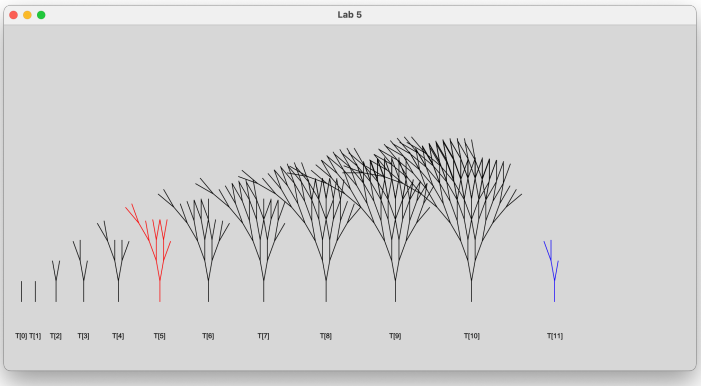
Le problème est que les pointeurs vers toutes les instances d'arbres sont entrelacés. Ceci est similaire aux problèmes que nous avons rencontrés avec la classe Stack dans la leçon 11.

Votre tâche est de résoudre ce problème ; chaque ligne que vous voyez à l'écran doit correspondre à une instance distincte de Tree. Tout comme dans la leçon 11, vous devrez corriger le constructeur et ajouter un constructeur de copie.

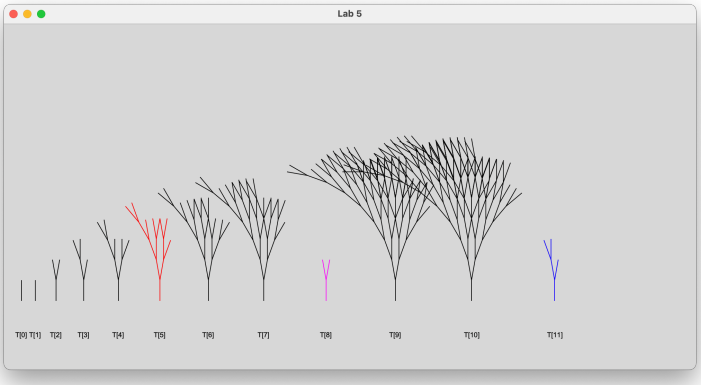
Continuez ensuite à décommenter les tâches supplémentaires dans le constructeur de Canvas, en vous assurant que le résultat est correct.

L'écran doit ressembler à ceci après chaque tâche :

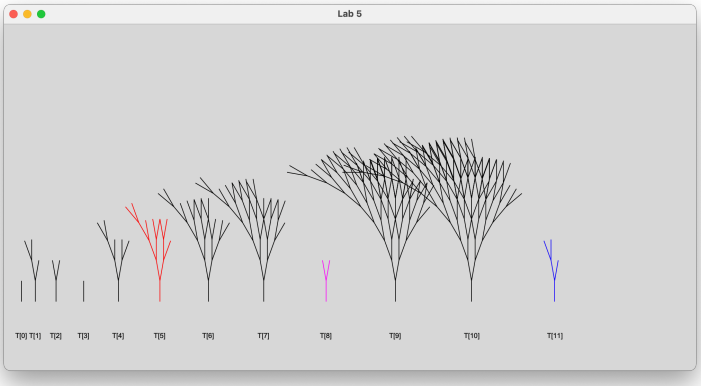
Tâche 2 : (constructeur de copie)



Tâche 3 : (operator =)



Tâche 4 : (échange, devrait fonctionner si vous avez effectué correctement les tâches précédentes)



Tâche 5 : Enfin, assurez-vous que vous avez codé un destructeur pour Tree qui appelle delete pour supprimer toute mémoire allouée dynamiquement. Lorsque vous appuyez sur q, l'instance de Canvas est détruite par le système fltk qui provoquera l'appel du destructeur de Canvas :

```
Canvas::~Canvas() {
  // Do not change this
  while (!T.empty()) {
    delete T.back();
    T.pop_back();
  }
}
```

Cela appellera le destructeur sur chaque arbre que nous avons créé. Assurez-vous que lorsque vous appuyez sur q pour quitter le programme ne plante pas.