

## 「 Graphe visuel 」

En réalisant ce TP vous pourrez afficher des nœuds et des arêtes en utilisant LibGraph2. Par défaut vous afficherez des nœuds en cliquant!!!! Pour l'instant la solution compile et laisse une fenêtre blanche.

### └ Exercice 1 : Préparation à la programmation ▮

J'ai récupéré les fonctions utiles du TD/TP précédents. J'ai rajouté les fonctions que je trouve nécessaire. J'ai aussi ajouté presque toute la partie graphique excepté l'ajout d'arête.

**Je vous conseille dans la suite des TP de prendre des notes et d'avoir un brouillon papier à côté de vous!**

### └ Exercice 2 : CNode ▮

Commencez par compléter la classe CNode; J'ai ajouté une variable statique `m_NbNodes` qui totalisera les instances de `CNode`, elle est initialisée à 0 dans le header de `prog.cpp`. Cette variable qui s'incrémente dans le constructeur, permet d'affecter un nom différent à chaque nœud : P1, P2, P3, ...

Normalement les nœuds n'ont pas la connaissance des arêtes. Mais comme nous allons utiliser plus tard un algorithme de coloration qui devra trier les nœuds par degré décroissant. Il sera souhaitable de stocker dans la classe CNode cette information (le degré) qui sera mise à jour à chaque fois qu'on ajoute ou enlève une arête.

On ajoute ainsi une variable `m_Degre` initialisée à 0, avec ses accesseurs. On ajoute enfin les coordonnées du points.

```
size_t m_Degre = 0;
static unsigned m_NbNodes;
float m_PosX = 0;
float m_PosY = 0;
```

On ajoute les accesseurs sur ces variables et les opérateurs d'égalité.

```
size_t GetNbNodes() const;
float GetX() const;
float GetY() const;
void SetX(float X);
void SetY(float Y);
bool operator==(const CNode& node);
bool operator==(const CNode::pNode& pNode);
size_t GetDegre() const;
void AddDegre(size_t degreToAdd);
void SubDegre(size_t degreToSub);
```

### └ Exercice 3 : CEdge ▮

Ensuite vous poursuivez en complétant cette classe. La classe CEdge est très peu impactée par ce TP. Vous aurez juste à ajouter un opérateur == qui renverra vrai s'ils ont même poids et mêmes nœuds et les geteurs sur le milieu de l'arête.

```
bool operator==(const CEdge & edge);  
float GetMilieuX() const ;  
float GetMilieuY() const ;
```

### └ Exercice 4 : CGraph ▮

Une fois cette troisième classe complétée vous devriez pouvoir afficher des nœuds. Ajouter une fonction qui retourne un pointeur sur le nœud le plus proche du point  $(x, y)$  Ajouter une fonction qui retourne un pointeur sur l'arête la plus proche du point  $(x, y)$

```
const CNode::Pointer GetNodeNear(unsigned int x, unsigned int y) const;  
const CEdge::Pointer GetEdgeNear(unsigned int x, unsigned int y) const;
```

Un point  $(x, y)$  est le plus proche d'un nœud si sa distance euclidienne est la plus petite :

$$\sqrt{(x - \text{pNode->GetX}())^2 + (y - \text{pNode->GetY}())^2}$$

Un point est le plus proche d'une arête s'il est le plus proche de son milieu.

### └ Exercice 5 : CTP ▮

- La classe CTP va faire la liaison entre le graphe qu'elle possède en donnée membre et les fonctions de feedback.
- Elle va gérer toutes les fonctionnalités de base : Ajouter arête et nœud, et les supprimer.
- Elle permet aussi de dessiner le Graphe.

Dans ce TP vous ajouterez une fonction qui transforme un graphe à  $n$  nœuds, en son graphe complet  $K_n$ . Un graphe est complet si tous ses nœuds sont reliés les uns aux autres.

```
void CreateGraphFull();
```

### └ Exercice 6 : prog.cpp ▮

C'est le fichier contenant la fonction *main*. Il gère tous les événements. Le seul cas que vous aurez à programmer c'est l'ajout d'une arête.