Projet C++ : Graphes

Mathieu Boutolleau – Paul Fayoux

---

1 – Choix de réalisation, mise en œuvre des pratiques de Génie Logiciel :

- Gestion des versions des sources (SVN),
- Mise en place de tests unitaires pour les méthodes importantes,
- Relecture croisée de code, afin de détecter d'éventuelles erreurs,
- Écriture de commentaires pertinents,
- Respect d'une convention de nommage,
- Levage d'exception lors de situation anormales mais prévisible de fonctionnement,
- Affichage d'avertissement lors de situation supposé incohérente mais non critique,
- Développement favorisant la maintenabilité et l'évolution de l'application.

2 – Diagramme de Classe : (voir annexe)

Pour représenter en mémoire un graphe, nous avons suivi les consignes données dans le sujet et écrits 3 classes : `Cgraph`, `Cedges` et `Cvertex`. Nous avons aussi ajouté une classe `Cfile` responsable du parsage de fichiers textes représentant un graphe. La classe `Cexeption` est utilisée lors de la levée dans d'exceptions, dans des situations anormales mais prévisibles de fonctionnement.

Pour vérifier le bon fonctionnement de nos méthodes, et éviter la régression de code, nous avons mis en place des tests unitaires dans des classes distinctes. Les tests sont appelés uniquement en mode Debug. Les tests unitaires utilisent des fichiers textes représentant différents graphes, il est donc indispensable (en mode Debug) que le dossier *Tests_File* soit dans le même dossier que l'exécutable.

3 – Spécificités de l’application :

Pour l’importation de graphes, nous utilisons une méthode de `Cfile` appelée par un constructeur de `Cgraph`. Cette méthode de `Cfile` est écrite de telle sorte à pouvoir traiter n’importe quel fichier texte respectant une certaine syntaxe. Il s’agit d’avoir sur chaque ligne une information utile, représentée par un couple : `<nom_de_l’info><délimiteur><valeur>` (par exemple avec '`NBArcs=3`', “NBArcs” est le nom de l’info, '`=`' est le délimiteur et '`3`' la valeur).

Ainsi, seul le constructeur est dépendant aux informations du fichier parsé, `Cfile` retourne un tableau contenant chaque valeur trouvées, indépendamment de leur sens. La modularité de cette approche nous autorise à réutiliser `Cfile` pour d’autres types de fichiers, en adaptant le contenu des règles de parsage.

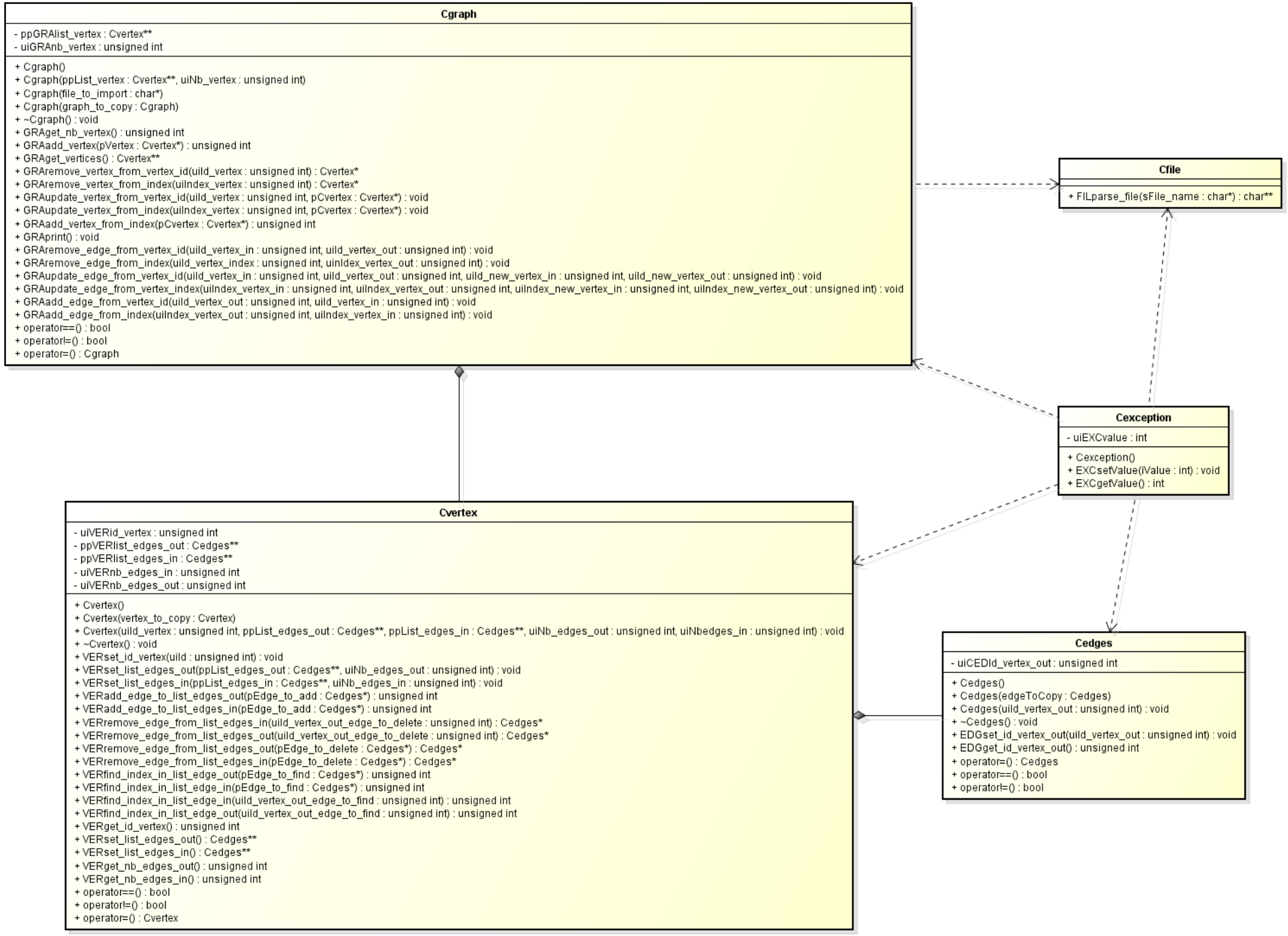
Lors de l’exécution il récupère un chemin vers un fichier passé en arguments par l’utilisateur. Un graphe est crée à partir de ce fichier (sous réserve que le fichier soit lisible et syntaxiquement correct), et ce graphe est affiché. Un autre graphe est crée, c’est l’inverse du premier, il est aussi affiché.

Nous avons traité les situations, qui nous semble incohérente, suivantes :

- L’utilisateur ne passe aucun arguments dans l’application, on affiche un avertissement et le programme se termine,
- L’utilisateur met plusieurs arguments, seul le premier est utilisé,
- Le fichier reçu en argument est non lisible par l’application,
- Le fichier ne respecte pas la syntaxe représentant un graphe,

D’autres exceptions peuvent être levées lors de l’exécution du programme (dépassement de tableau, taille de tableau différentes…)

4 – Annexes


Cgraph
- ppGRAlist_vertex : Cvertex**
- uiGRAnb_vertex : unsigned int
+ Cgraph()
+ Cgraph(ppList_vertex : Cvertex**, uiNb_vertex : unsigned int)
+ Cgraph(file_to_import : char*)
+ Cgraph(graph_to_copy : Cgraph)
+ ~Cgraph() : void
+ GRAget_nb_vertex() : unsigned int
+ GRAadd_vertex(pVertex : Cvertex*) : unsigned int
+ GRAget_vertices() : Cvertex**
+ GRAremove_vertex_from_vertex_id(uiId_vertex : unsigned int) : Cvertex*
+ GRAremove_vertex_from_index(uiIndex_vertex : unsigned int) : Cvertex*
+ GRAupdate_vertex_from_vertex_id(uiId_vertex : unsigned int, pCvertex : Cvertex*) : void
+ GRAupdate_vertex_from_index(uiIndex_vertex : unsigned int, pCvertex : Cvertex*) : void
+ GRAadd_vertex_from_index(pCvertex : Cvertex*) : unsigned int
+ GRAprint() : void
+ GRAremove_edge_from_vertex_id(uiId_vertex_in : unsigned int, uiId_vertex_out : unsigned int) : void
+ GRAremove_edge_from_index(uiId_vertex_index : unsigned int, uiIndex_vertex_out : unsigned int) : void
+ GRAupdate_edge_from_vertex_id(uiId_vertex_in : unsigned int, uiId_vertex_out : unsigned int, uiId_new_vertex_in : unsigned int, uiId_new_vertex_out : unsigned int) : void
+ GRAupdate_edge_from_vertex_index(uiIndex_vertex_in : unsigned int, uiIndex_vertex_out : unsigned int, uiIndex_new_vertex_in : unsigned int, uiIndex_new_vertex_out : unsigned int) : void
+ GRAadd_edge_from_vertex_id(uiId_vertex_out : unsigned int, uiId_vertex_in : unsigned int) : void
+ GRAadd_edge_from_index(uiIndex_vertex_out : unsigned int, uiIndex_vertex_in : unsigned int) : void
+ operator==() : bool
+ operator!=() : bool
+ operator=() : Cgraph
Cfile
+ FILparse_file(sFile_name : char*) : char**
Cexception
- uiEXCvalue : int
+ Cexception()
+ EXCsetValue(iValue : int) : void
+ EXCgetValue() : int
Cvertex
- uiVERid_vertex : unsigned int
- ppVERlist_edges_out : Cedges**
- ppVERlist_edges_in : Cedges**
- uiVERnb_edges_in : unsigned int
- uiVERnb_edges_out : unsigned int
+ Cvertex()
+ Cvertex(vertex_to_copy : Cvertex)
+ Cvertex(uiId_vertex : unsigned int, ppList_edges_out : Cedges**, ppList_edges_in : Cedges**, uiNb_edges_out : unsigned int, uiNbedges_in : unsigned int) : void
+ ~Cvertex() : void
+ VERset_id_vertex(uiId : unsigned int) : void
+ VERset_list_edges_out(ppList_edges_out : Cedges**, uiNb_edges_out : unsigned int) : void
+ VERset_list_edges_in(ppList_edges_in : Cedges**, uiNb_edges_in : unsigned int) : void
+ VERadd_edge_to_list_edges_out(pEdge_to_add : Cedges*) : unsigned int
+ VERadd_edge_to_list_edges_in(pEdge_to_add : Cedges*) : unsigned int
+ VERremove_edge_from_list_edges_in(uiId_vertex_out_edge_to_delete : unsigned int) : Cedges*
+ VERremove_edge_from_list_edges_out(uiId_vertex_out_edge_to_delete : unsigned int) : Cedges*
+ VERremove_edge_from_list_edges_out(pEdge_to_delete : Cedges*) : Cedges*
+ VERremove_edge_from_list_edges_in(pEdge_to_delete : Cedges*) : Cedges*
+ VERfind_index_in_list_edge_out(pEdge_to_find : Cedges*) : unsigned int
+ VERfind_index_in_list_edge_in(pEdge_to_find : Cedges*) : unsigned int
+ VERfind_index_in_list_edge_in(uiId_vertex_out_edge_to_find : unsigned int) : unsigned int
+ VERfind_index_in_list_edge_out(uiId_vertex_out_edge_to_find : unsigned int) : unsigned int
+ VERget_id_vertex() : unsigned int
+ VERset_list_edges_out() : Cedges**
+ VERset_list_edges_in() : Cedges**
+ VERget_nb_edges_out() : unsigned int
+ VERget_nb_edges_in() : unsigned int
+ operator==() : bool
+ operator!=() : bool
+ operator=() : Cvertex
Cedges
- uiCEDId_vertex_out : unsigned int
+ Cedges()
+ Cedges(edgeToCopy : Cedges)
+ Cedges(uiId_vertex_out : unsigned int) : void
+ ~Cedges() : void
+ EDGset_id_vertex_out(uiId_vertex_out : unsigned int) : void
+ EDGget_id_vertex_out() : unsigned int
+ operator=() : Cedges
+ operator==() : bool
+ operator!=() : bool