

Année universitaire 2016-2017
Université de Caen Basse-Normandie

Rapport sur le premier semestre

Alexis Carreau
Thomas Lécluse
Emma Mauger
Théo Sarrazin
L2 Informatique

Table des matières

1	Introduction	2
1.1	Présentation	2
1.2	Objectifs	2
1.3	Déroulement et planning	2
2	Détails techniques	2
2.1	Bibliothèques utilisées	2
2.2	Techniques employées	3
2.3	Bibliographie	3
3	Structure générale	3
3.1	Répartition en modules	3
3.2	Module graphique	4
3.3	Module gestion de projets	6
3.4	Module gestion de fichiers	6
4	Conclusion	6
4.1	Améliorations	6

1 Introduction

1.1 Présentation

Nous avons choisi de réaliser l'IDE (Integrated Development Environment), car nous voulions créer un outil que nous pourrions utiliser par la suite. Ce sujet nous semblait donc intéressant à faire.

Un IDE fournit des facilités au programmeur pour le développement logiciel. Il a pour but de maximiser la productivité du programmeur. Il contient généralement :

- un éditeur de texte,
- un interpréteur,
- un debugger,
- un compilateur,
- des options avancées comme la recherche de termes, l'autocomplétion, la coloration syntaxique...

1.2 Objectifs

Nous devons d'ici à la fin de l'année réaliser un IDE qui contiendra les éléments cités ci-dessus. Pour la coloration lexicale et l'analyse syntaxique du code nous utiliserons les programmes Lex et Yacc.

1.3 Déroulement et planning

Nous avons réparti équitablement le travail et le temps de travail de chacun. Il faut préciser que deux membres du groupe (Thomas et Théo) connaissaient déjà la bibliothèque graphique utilisée dans l'IDE, et partaient donc avec un avantage non-négligeable.

Les deux autres membres du groupe ont donc été aidés afin de leur expliquer les bases et de pouvoir démarrer sans trop de problèmes.

Nous avons réparti les différentes tâches à effectuer pour le projet entre les membres du groupe. Certaines sont plus longues ou plus complexes, c'est pourquoi certains membres du groupe ont une liste moins remplie que d'autres. Il ne faut donc pas se fier uniquement à cette liste.

- Alexis : Navigation / Gestion des projets
- Emma : Découpage en modules / Architecture projet / Recherche bibliographique / Barre de menu
- Théo : Coloration syntaxique / Analyse lexicale / Interface graphique / Gestion (sauvegarde, ouverture) des fichiers
- Thomas : Documentation / Interface graphique / Thème, apparence texte et fenêtre / Barre de status
- Tout le groupe : Rapports et présentation

2 Détails techniques

2.1 Bibliothèques utilisées

Nous avons choisi la bibliothèque graphique QT (version 4.8.7, prévue à la base pour le C++) ce pourquoi nous avons aussi eu besoin de Pyside (version 1.2.4), qui fait le lien vers le

langage Python (version 3.4).

La raison de ce choix est le fait que QT est un outil plus puissant de par sa richesse de fonctionnalités. Sa documentation est de plus très fournie car les utilisateurs de QT sont plus nombreux que ceux de Tkinter.

2.2 Techniques employées

Nous avons choisi un style de programmation orienté objet, puisque plus logique dans un projet de cette envergure. L'objet nous permet de mettre en application les notions vues en cours. Cela nous permet de personnaliser en les ré-implémentant des classes de QT pour les adapter à nos besoins.

Lex est un programme qui permet de reconnaître des tokens qui sont des éléments d'une chaîne de caractères. Cela à l'aide de règles lexicales que l'on lui passe en entrée. Dans notre cas, nous utilisons Lex pour reconnaître les éléments que l'on écrit, afin de colorier ces derniers en fonction de leur rôle (identifiant, entier, déclaration...). Nous utilisons la sortie générée en la passant à un autre programme : Yacc.

Yacc permet de générer un arbre syntaxique abstrait qui nous permet de vérifier que la syntaxe de notre code est correcte. Aussi, cela permet de proposer une complétion automatique en fonction de ce que l'on écrit.

Nous devons également lui passer des règles, qui sont d'ordre grammaticales.

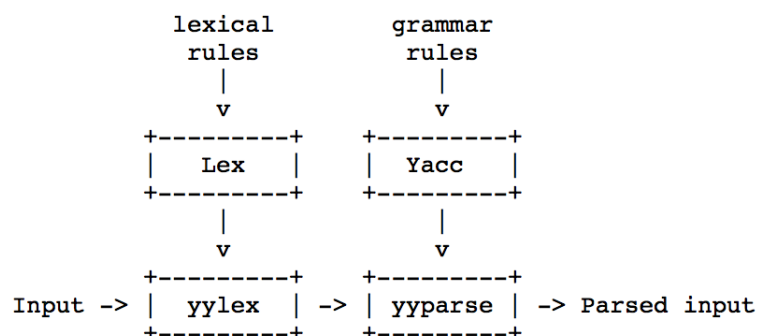


FIGURE 1 – Schéma résumant le fonctionnement de Lex et Yacc.

2.3 Bibliographie

Nous avons utilisé deux sites de documentation principalement sur QT et Pyside.

- <http://srinikom.github.io/pyside-docs/PySide/QtGui/>
- doc.qt.io

3 Structure générale

3.1 Répartition en modules

Les différents modules sont répartis par thèmes. Nous avons :

- Un module pour l'interface graphique

- Un module pour la gestion des projets
- Un module pour la gestion des fichiers
- Un module pour la coloration syntaxique
- Deux modules pour Lex et Yacc
- Un module pour l'analyse syntaxique
- Un module pour l'autocomplétion

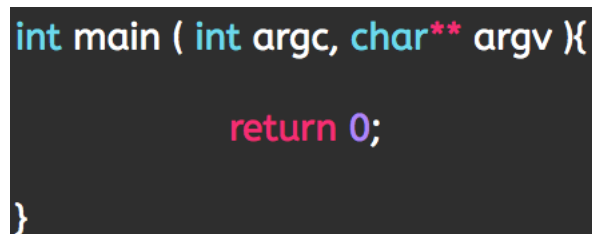
3.2 Module graphique

On retrouve dans ce module tout ce qui est relatif au GUI (l'interface graphique). C'est là qu'est créée la fenêtre principale de l'application, où l'on va pouvoir créer, ouvrir, sauvegarder des fichiers et des projets.

On y retrouve donc ce qui est nécessaire pour :

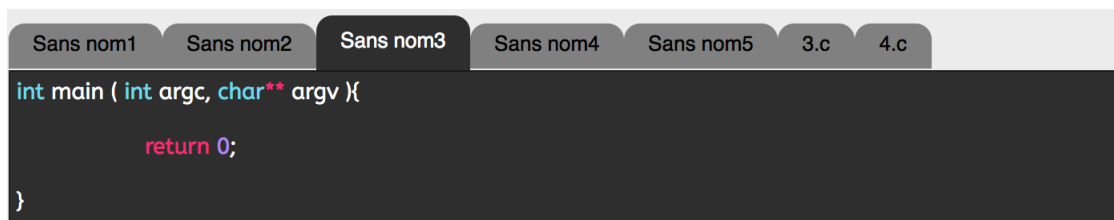
- Créer une zone de texte. Nous utilisons pour cela l'objet QTextEdit de QT, qui est un widget permettant d'avoir une zone de texte. Nous nous en servons comme fenêtre d'éditeur, c'est donc ici que l'on pourra écrire du code.

Le thème de l'application, (c'est à dire la couleur de fond et de la police ainsi que cette dernière) est relatif au QTextEdit. C'est également là que la couleur des éléments (tokens) sera modifiée par Lex en fonction de leur rôle.

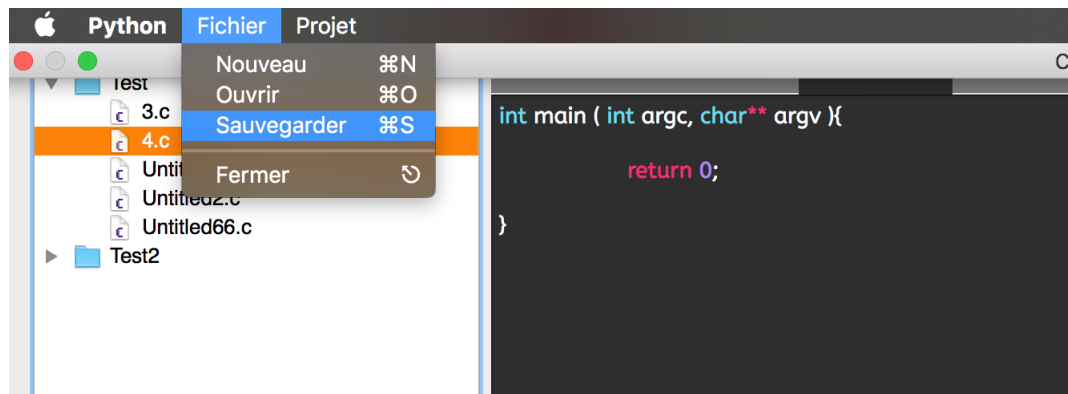


```
int main ( int argc, char** argv ){
    return 0;
}
```

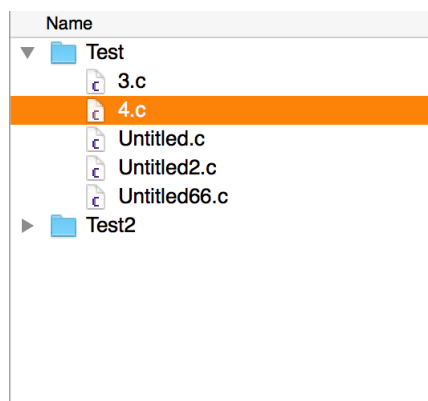
- Créer plusieurs onglets de code. C'est un QTabWidget que nous utilisons pour cela. Il va créer plusieurs QTextEdit (ou en supprimer si on ferme l'onglet), et possède des fonctions et raccourcis pour naviguer entre tous. Si aucun onglet n'est ouvert, le logo de notre groupe est affiché à la place.



- Créer une barre de menu. Cela permet d'avoir toutes les fonctionnalités et éventuellement des raccourcis. On utilise une QMenuBar pour cela, qui va elle-même utiliser une QAction pour créer une action sur le menu. Une action contient un nom, éventuellement un raccourci et une fonction à exécuter.

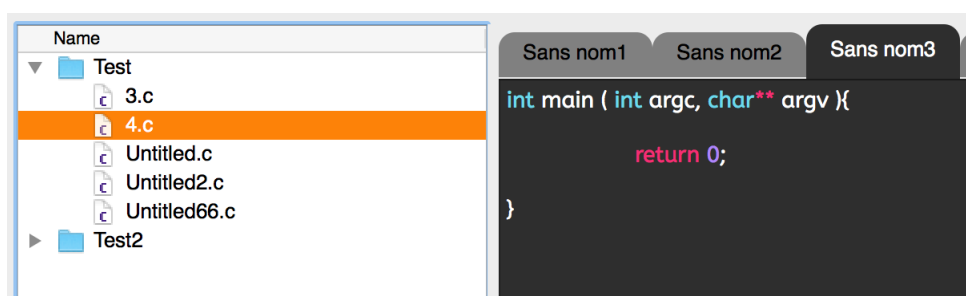
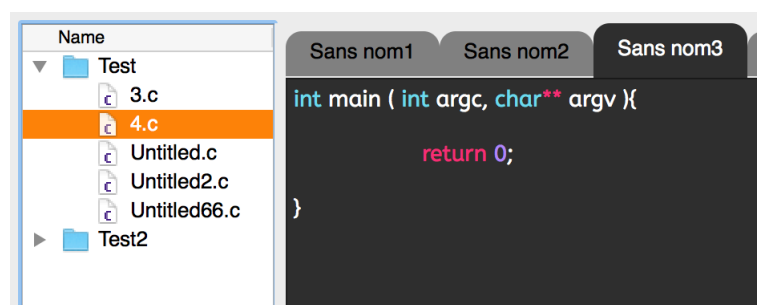


- Mettre en place le navigateur de fichiers et de projets. Nous utilisons un `QTreeView` qui nous permet d'afficher l'arborescence des fichiers et projets. C'est ce que l'on utilise pour ouvrir et naviguer dans les projets ouverts.

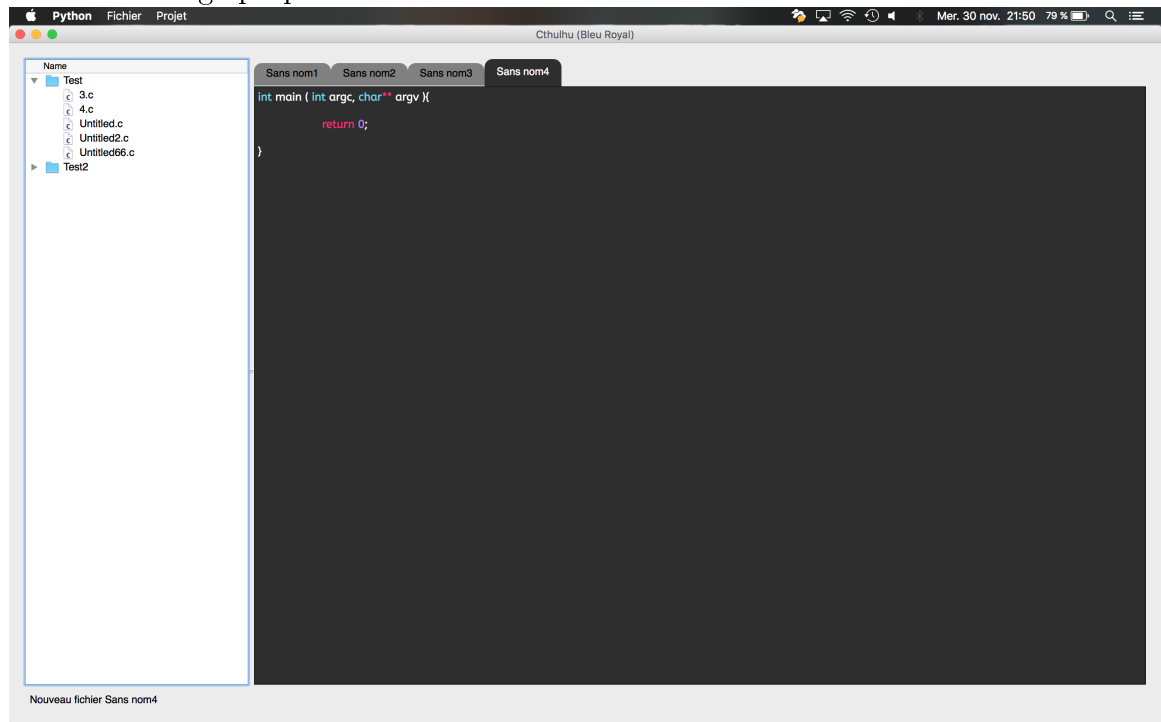


- Modifier la taille du navigateur de fichier/de l'éditeur de code. Pour cela, nous utilisons un `QSplitter`.

Cela peut permettre, en fonction de la taille du projet et des répertoires imbriqués dans d'autres, de visualiser complètement les fichiers. Au contraire, on peut réduire le navigateur de fichier pour avoir une zone de code plus grande.



- Créer la fenêtre principale. Nous utilisons un QWidget, qui est en fait une sorte de parent de tous les objets cités ci-dessus. C'est donc ici que l'on met tout en commun afin de faire un affichage propre.



!!! PAS VRAIMENT NÉCESSAIRE!!! Ce module est le plus imposant car la création d'un interface graphique nécessite beaucoup de lignes de codes.

3.3 Module gestion de projets

→ Déplacer les fonctions relatives dans un autre module que GUI

3.4 Module gestion de fichiers

4 Conclusion

4.1 Améliorations