

Année universitaire 2016-2017
Université de Caen Normandie

Rapport sur les snippets

Théo Sarrazin
L2 Informatique

Table des matières

1	Snippets	2
1.1	Les snippets : explication	2
1.2	Intergration des bouts de code dans l'IDE	2
1.2.1	Utilisation du JSON	2
1.2.2	Connection entre le JSON et l'IDE	2
2	Compilateur	3
2.1	Compilateurs utilisés	3
2.2	Integration de GCC à notre IDE	3
3	Cache	4
3.1	Utilité du cache	4
3.2	Intégration du cache dans l'IDE	4

2 Compilateur

2.1 Compilateurs utilisés

Pour notre IDE, nous avons utilisés le compilateur **GCC**, pour compiler les projets de type **C**. Mais nous utilisons aussi les interpréteur de python afin d'interpréter les projets de type **Python**. Par la suite, pour chaque nouveaux languages il nous suffit d'utiliser le bon compilateur ou le bon interpréteur.

2.2 Integration de GCC à notre IDE

Pour utilisé **GCC** avec notre IDE nous avons créé une fenêtre de configuration afin de permettre à l'utilisateur de choisir la configuration adéquate à la compilation de son projet.

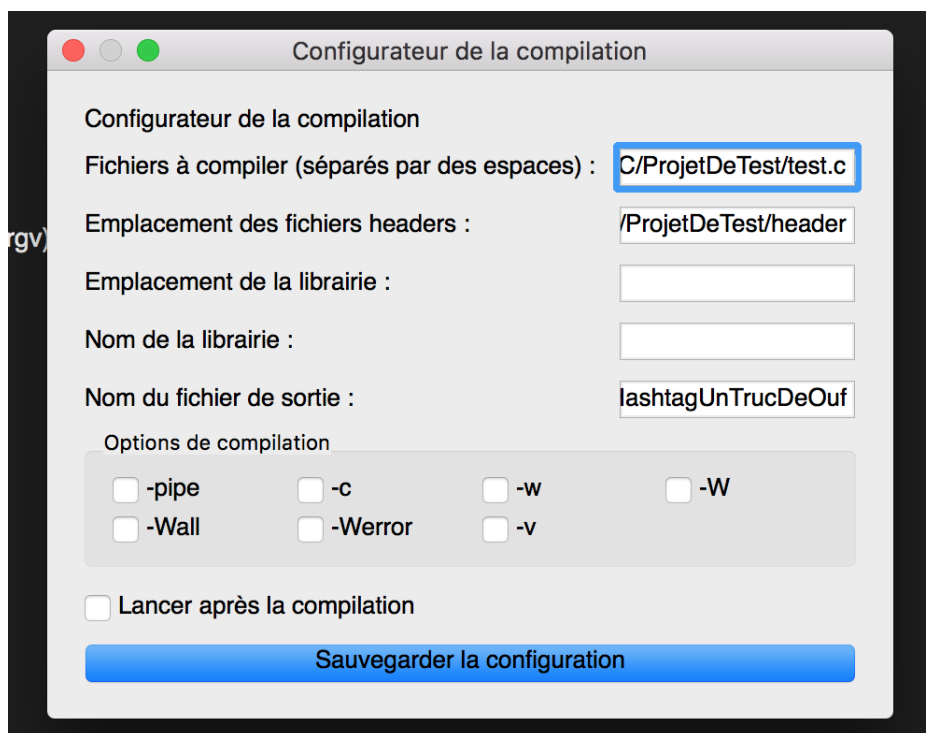


FIGURE 3 – Fenêtre de configuration de GCC

Par la suite, chaque information de cette fenêtre est convertie en une chaîne de caractère du type :

```
gcc /Users/theosarrazin/workplaceC/ProjetDeTest/test.c -I /Users/theosarrazin/workplaceC/ProjetDeTest/header -o MonExecutable
```

Cette chaîne de caractère est stockée dans le fichier XML de configuration du projet.

Par la suite on exécute cette ligne de commande, puis on récupère sur la sortie d'erreurs les eventuelles erreurs afin de pouvoir les afficher à l'utilisateur après les avoir parsé pour avoir un bon formatage.

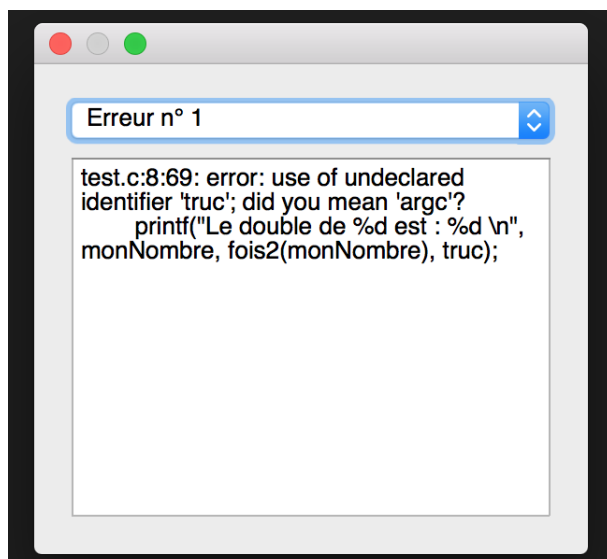


FIGURE 4 – Affichage des erreurs de GCC dans l’IDE

Pour la partie interpréteur la démarche est identique.

3 Cache

3.1 Utilité du cache

Afin de pouvoir faire de la coloration, nous avons utilisés **Lex** mais pour les fichiers de plusieurs 100^{ème} de lignes **Lex** prend plusieurs secondes pour nous donner la listes des tokens du fichier. Nous avons donc décidé d’utiliser un fichier de cache afin de ne pas utiliser **Lex** pour colorer des lignes que nous avons déjà coloré de manière ultérieure.

3.2 Intégration du cache dans l’IDE

Pour le cache, nous avons décidé d’utiliser un fichier JSON pour l’enregistrer. Le fonctionnement est assez simple la clé est le bout de code que l’on donne à **Lex** et la valeur est la réponse de ce dernier. Par la suite il nous reste plus qu’à utiliser les valeurs du fichier JSON à la place de **Lex**

```
{
  "\tint monNombre;": [{"INT", "int"}, {"IDENTIFIER", "monNombre"}, {"SEMICOLON", ";"}],
  "\treturn 2*a;": [{"RETURN", "return"}, {"CONSTANT", "2"}, {"TIMES", "*"}, {"IDENTIFIER", "a"}, {"SEMICOLON", ";"}],
  "\tprintf(\"Le double de %d est : %d \\n\", monNombre, fois2(monNombre), truc);": [{"IDENTIFIER", "printf"}, {"L_BRACKET", "("}, {"STRING_LITERAL", "\"Le double de %d est : %d \\n\""}, {"COMMA", ","}, {"IDENTIFIER", "monNombre"}, {"COMMA", ","}, {"IDENTIFIER", "fois2"}, {"PARENTHESIS", "("}, {"IDENTIFIER", "monNombre"}, {"PARENTHESIS", ")"}, {"COMMA", ","}, {"IDENTIFIER", "truc"}, {"SEMICOLON", ";"}]
```

FIGURE 5 – Extrait d’un JSON contenant le cache

Grâce au cache nous avons un gain de temps considérable sur l’ouverture des documents.