Rapport

Compilateur réalisée en Python 2.7 par

- Léo MULLOT
- Mathieu HIREL

Listes des éléments implémenté

Analyse lexical

- Découpage des élemens en token
- Gestion des commentaire avec le symbole #

Analyse Syntaxique:

- Opération mathématique
    - `a + b`
    - `a - b`
    - `a * b`
    - `a / b`
    - `- a`
    - `a % b`
- Opération logique
    - `a && b`
    - `a || b`
    - `a > b`
    - `a >= b`
    - `a <= b`
    - `a < b`
    - `!a`
- Déclaration de variable
- Affectation d'une valeur sur une variable
- Lecture d'une variable
- Récupération de l'adresse d'une variable
- Récupéré la valeur a l'adresse i d'une variable (`a[i]`)
- if
- if else
- boucle while
- boucle do while

- boucle for
- return
- continue
- break

Analyse Sémantique

- Contrôle de si un `continue` ou un `break` sont appelée dans une boucle
- Affectation des adresses au déclaration de variables

> Ceci n'a pas était testé, car nous n'avons pas implémenté les fonction malloc et free

- Récupération de l'adresse et vérification que la variable est déclarée

> Ceci n'a pas était testé, car nous n'avons pas implémenté les fonction malloc et free

Optimisation de code

- Optimisation de l'opération : `- (a - b)`
- Optimisation de l'opération : `- (a + b)`
- Optimisation des opérations mathématique sur des constantes

Runtime

- `print(int i);` : Affiche la valeur i
- `println(int i);` : Affiche la valeur i suivis d'un retour a la ligne

Listes des éléments non fonctionnels

Utilisation de variable affecté

Nous n'avons pas rajouter dans l'analyser sémantique de vérification qu'une variable est bien initialisé en plus d'être déclaré avant d'être appelé.

Exemple de codes qui **devrait** ne pas compiler (mais qui compile):

```
int main(){
  int i;
  while(i<5) {
    println(i);
  }
}
```

```
int main(){
  int a; a  = 3;
  int b;
  int c; c = a * b;
}
```

Fonction `malloc` & `free`

Les fonctions `malloc` et `free` n'ont pas était ajouté

Listes des tests réalisée

Tests unitaires

Lors du développement du compilateur nous avons écrit des tests unitaires pour valider le fonctionnement des différents fonctionnalités.

Pour lancer les tests unitaires :

```
python -m unittest discover
```

Tests fournis

On a rajouter les tests fournis dans nos tests unitaires pour augmenter les tests

Tests complémentaires réalisée

En plus des tests unitaires nous avons fait des tests complet qui sont aussi lancés par nos tests unitaires pour verifier leur bon fonctionnement

Nom du test	Description	Localisation du fichier
vif	Test les if	tests_perso/pass/if.cmm
for	Test les boucles for	tests_perso/pass/for.cmm
do_while	Test les boucle do while	tests_perso/pass/do_while.cmm
while	Test les boucle while	tests_perso/pass/while.cmm
print10	Test l'affichage de 2 10	tests_perso/pass/print10.cmm
println	Test la fonction println	tests_perso/pass/println.cmm
function	Test les appelles aux fonction	tests_perso/pass/function.cmm
fibonnaci	Calcul de la suite de fibonnaci	tests_perso/pass/fibonnaci.cmm