RAPPORT DU PROJET DE COMPILATION

Licence 3 Informatique

Membre du groupe : Dieunel MARCELIN 12207041

Professeur : Flavien BREUVART

Soumis le : 27/05/2024

Description du travail effectué

Ce rapport a pour objectif de préciser les fragments implémentés, de mentionner les déviations prises par rapport au sujet.

A la fin de mon travail, j'ai implémenté en intégralité 4 grands fragments. Ce sont les fragments 0, 1, 2 et 3. Et comme demande dans l'énoncé, chaque sous-fragment est indexe par un tag de la forme :

- pi.j pour le sous-fragment j du grand fragment i du "parser" (branche *parser*) ou du moins le fragment i.j
- ci.j pour le sous-fragment j du grand fragment i "compiler" (branche *main*) ou du moins le fragment i.j

Pour les branches, je reconnais que la correction se portera que sur 2 branches : *parser* et *main*. Je voudrais quand même attirer votre attention sur ma branche *ast*. Au tout début, j'ai rencontré pas de difficultés dans ma branche *ast*. Des problèmes de "merge" ou de "commit" dus au fait de mon manque d'expérience avec l'utilisation de *git*. Pour pallier cela, j'ai pris la liberté de créer une nouvelle branche appelée *ast2* qui remplace *ast* et qui fait le même travail. De ce fait, depuis le fragment 1, je travaille avec *ast2* et non *ast*.

Dans le fragment 3, pour implémenter les structures de contrôle "i*f_then_else*", "*while*" et "*do while*", il faut placer les expressions à évaluer entre parenthèses et les commandes à exécuter entres accolades. Même si le bloc d'instructions contient une commande, les accolades sont requises sinon ce sera une erreur de syntaxe.

Exemple :

```
if (x == 2.0){
        y=z + 1.0;
}
else{
        y= z - 1.0;
}
```

Les nombres traités dans ce projet sont des flottants. Ils contiennent un "." et une partie décimale. Je les ai traités comme demandés dans les fragments.

La fin du fichier est marquée par le fait qu'il n'y ait plus de caractères à lire, non pas par un retour à ligne. Le séparateurs '*espace*', '*retour à la ligne*', '*tabulation*' et autres sont traités. Dans le fichier source, quelqu'un est libre de mettre les séparateurs et les espacements où il veut comme dans un code normal.

Branche parser

Dans la branche *parser*, nous pouvons donner en entrée le fichier de code source JavaScript, et le *parser* répondra oui ou non si le code est valide.

Pour exécuter le programme, il faut s'assurer d'être dans la branche *parser*.

Exécuter la commande : ***make***

Et lancer l'exécutable avec le nom du fichier source : ***./main nom_fichier_javascript***

Branche main

Cette branche, en plus de faire ce que fait la branche *parser*, elle affiche le programme reçu en entrée et affiche également le code assembleur associé. Dans mon implémentation, chaque commande s'affiche sur une nouvelle ligne et les instructions imbriquées commencent par une tabulation dans un souci d'un affichage un peu plus propre. Ensuite s'affiche le code assembleur.