Université de Mons
Faculté des sciences
Département d'Informatique

Compilation : interpréteur Dumbo
Rapport de projet

Professeur :
Véronique Bruyère
Alexandre Decan

Auteur :
Thomas Lavend'Homme
Guillaume Proot

Année académique 2019-2020

Table des matières

1 Introduction

L'objectif de ce projet est de nous familiariser avec le fonctionnement d'un compilateur et pour celà il nous a été demandé d'en réalisé un en python interpretant du code *dumbo*.

2 Mode d'emploi

Pour faire fonctionner notre compilateur, il vous est demandé de vous placer dans le répertoire où se trouve le fichier *dumbo_interpreter.py* ainsi que vos fichiers à interpréter. Ensuite, dans l'invite de commande, rentrez la commande suivante : *python3 dumbo_interpreter.py data template output*.

3 Grammaire

Pour ce projet, nous avons utilisé la grammaire de base du langage dumbo à laquelle nous avons apporté quelques modifications et ajouts :

- Tout d'abord nous avons ajouté la règle : *dumbo_block* : ("{{" "}}") pour pouvoir gérer les cas où nous aurions affaire à un dumbo_block vide.
- Nous avons géré les différentes expressions individuelement afin de limiter les erreurs et simplifier la clarté du code. Pour ce qui est des expressions "print" nous l'avons séparée en deux cas distinct :
  - **print** affichera ce qui est attendu sans check supplémentaire. (Exemple : print 4 ⇒ 4)
  - **print !** affichera ce qui est attendu exepté que les entiers deviendront des booléens. (0 ⇒ False et le reste ⇒ True). (Exemple : print ! 4 ⇒ True)
- Afin de gérer l'expression "if < boolean > do < expressions list > endif", nous avons ajouté l'assignation d'un booléen à une variable.
- En plus de la grammaire de base de dumbo nous avons ajouté la possibilité de faire des opérations logiques (AND, OR, NOT, TRUE, FALSE) ainsi que des comparaisons logiques et arithmétiques ($=, <, >, <=, >=, !=$).
- Dans la grammaire de base de dumbo, il n'y a pas de distinctions entre l'instanciation de variable et l'appel de celle-ci. C'est pourquoi, nous avons décidé de séparer celà en en fonctions différentes *variable_set* et *variable_get*. Tel les set et get dans d'autres langages de programmation *variable_get* ne fonctionnera que si la variable sur laquelle la

fonction est appliquée existe déjà alors que *variable_set* n'impose pas cette contrainte.
- Enfin, nous demandons pour instancier une variable d'y ajouter son type (exemple : int x = 4)

4 Gestion d'expression

4.1 Gestion des variables locales et globales

Pour gérer les variables, nous avons décidé d'utiliser une liste chainée de dictionnaire où chaque maillon est un scope. Quand on détecte un nouveau dumbo bloc, nous créons un nouveau scope, on visite ensuite le bloc en insérent toutes les variables de ce bloc dans le scope nouvelement créé et lorsque le parcours du dumbo bloc est finis nous refusionnons le scope avec les variables globales.

4.2 Gestion du for

Nous gérons les boucles for de la manière suivantes :

1. Tout d'abord nous créons une variable locale (loop_variable). Nous l'assignons à la valeur correspondante dans la string_list ou par rapport à la variable donnée dans un nouveau scope.
2. On exécute le dumbo bloc à l'intérieur de la boucle.
3. On efface la variable locale et on recommence l'opérationtant qu'il y a des éléments dans la string_list.

4.3 Gestion du if

Nous avons décidé de gérer les if de la manière suivante :

1. On teste si la condition est vraie.
2. Si celle-ci est vérifiée, on crée un nouveau scope.
3. On exécute le dumbo bloc.
4. On supprime les variables locales.

5 choix personnel

Nous avons décidé d'utiliser la librairie LARK à la place de PLY car malgré le fait que nous ne l'ayons pas aborder en tp nous la trouvions plus

adaptée pour la réalisation de notre projet.

Il n'y a pas de booleen à la base avec notre compilateur. Quand nous gérons une condition logique, nous considérons $0 \Rightarrow False$ et $(int \neq 0) \Rightarrow True$.

Nous avons aussi choisi d'imposer un typage aux variables du fichier d'entrée. Cela nous permet de gérer plus facilement les erreurs de type lors de la compilation.

6 Problème survenus

Un problème que nous avons rencontré est la gestion de la boucle for. En effet, nous utilisions un "Transformer" de Lark qui fonctionne en bottom-up ce qui entrainait que l'on parcourait d'abord l'intérieur de la boucle avant l'expression en elle-même. Pour résoudre celà, nous avons revu notre implémentation en utilisant la classe "Interpreter" de Lark à la place de "Transformer" qui elle utilise une approche top-down.

7 Conclusion

Ce projet nous auras permis de mieux nous familiariser avec le processus de compilation et différents outils facilitant celui-ci. Si nous devions emettre une critique concernant notre projet, ça serait sur notre gestion des types. En effet, celle-ci est loin d'être vraiment optimisée.