PROJET DE COMPILATION :
Rapport

Auteurs:
Adrien Fievet
Claire D'Haene

2023-2024

Titulaire
Véronique BRUYERE
Alexandre DECAN

Etudiants
Adrien (220625)
Claire (220323)

Table des matières

1 Introduction

Dans le cadre de notre cours de compilation, nous devons réaliser un projet. Ce dernier a pour but de mettre en oeuvre les connaissances acquises durant le cours ainsi que notre capacité à nous documenter afin de concevoir un interpréteur pour le langage SPF (Simple Programme en Français). SPF est un langage interprété basique qui se distingue par l'utilisation de ses propres mots-clés en français, tout en ayant des caractéristiques similaires à celles du langage Python. Le projet sera implémenté en Python à l'aide du module Lark.

2 Grammaire implémentée

D'abord, nous avons conçu notre programme comme un nombre indéterminé d'instructions. Cette dernière se définissant par soit une assignation, un affichage, un ajout, une condition, une boucle ou encore une expression. Le ";" n'a pas été défini globalement mais au cas par cas car les conditions et les boucles sont délimitées par des accolades. Nous allons passer en revue chacun de ces instructions.

2.1 Les déclarations

Une déclaration est simplement définie par un type, une variable ainsi que potentiellement initialisée avec une expression. Le langage SPF contient quatres types :

— booléen
— entier
— texte
— liste

Une liste est une liste de valeur, quel que soit le type, ou elle est caractérisée par une séquence de nombres entiers. Une variable quant à elle doit respecter la grammaire suivante : Le nom d'une variable est composé au minimum d'un caractère, ne peut contenir que des lettres (majuscules et minuscules) accentuées ou non, des chiffres ou un tiret bas Ce dernier ne peut pas non plus débuter par un chiffre. Nous avons donc obligé le fait que le premier terme soit une lettre ou un underscore et ensuite un nombre indéterminé de lettre, de chiffres et d'underscores. Nous reparlerons de la grammaire d'une expression

2.2 Les assignations

Elles sont très similaires aux déclarations, sauf que le type n'est plus précisé, mais maintenant, il est obligatoire d'avoir une expression.

2.3 L'affichage

afficher prend simplement une suite d'au moins une expression séparée par des virgules. Nous nous sommes également permis de rajouter la possibilité d'afficher directement le résultat de l'instruction **ajout**.

2.4 L'ajout

Pour **ajout**, nous définissons une grammaire avec une expression et une variable dans laquelle sera rajoutée le résultat de l'expression.

2.5 Les boucles

boucle représente simplement les deux possibilités de boucle offertes par le langage SPF. Il faut prendre en compte que **tantque** prend une expression et une suite d'instructions. Et **pourchaque** nécessite aussi une variable et de son type.

2.6 Les conditions

Tout comme le point précédent, **condition** contient une règle **si** qui exécute les instructions en fonction de la condition. **sisinon** est une extension de **si** qui prend en plus d'autres instructions dans un autre corps d'accolades.

2.7 Les expressions

Les expressions sont de base décrites comme des littéraux ou des opérations, mais peuvent être entourées de parenthèses pour donner des priorités, ou être représentées par une variable.

2.7.1 Les littéraux

Les littéraux représentent toutes les valeurs possibles par rapport aux quatre types de ce langage.

Les booléens : valant soit *vrai* soit *faux*.

Les entiers : une séquence de chiffres ne commençant pas par 0, à moins que ce ne soit pour représenter 0. Il peut optionnellement avoir un *plus*.

Les textes : une suite de n'importe quels symboles entre deux *guillemets*.

Les listes : soit une liste d'expressions entre *crochets* séparées par des *virgules*, soit une séquence définie par deux expressions entre *crochets* séparées par un *deux-points*.

2.7.2 Les opérations

Mises à part les opérations **indice** et**taille**, les autres opérations sont toutes composées d'une ou deux expressions ainsi que de mots clés pour les différencier. Attention au niveau des opérations mathématiques, nous avons mis en place les règles **exp1** et **exp2** pour émettre des priorités dans les calculs. Rien de particulier du côté d'**indice**, ce dernier prend deux expressions dont la deuxième entre *crochets*.

Pour finir, **taille** peut recevoir soit **leslistes** (une **liste** ou une **sequence**) mais aussi une variable qui contiendra une des deux valeurs précédemment citées.

2.8 Autres

Nous avons également défini une règle pour ignorer tous les espaces blancs et une règle pour ignorer tous les caractères après un *dièse* sur une ligne.

3 Notre approche

3.1 Les variables

Nous avons d'abord créé une classe pour stocker les caractéristiques des variables à savoir, leur type, leur nom, leur valeur mais également les types des valeurs contenues dans la liste si la variable en est une. C'est la classe **Memory** qui a permis de facilement interagir avec les variables grâce aux méthodes *declare*, *get* et *set*. Utiliser les variables dans les expressions était facilent car il suffisait de remplacer les variables par leurs valeurs et leurs types quand on se trouvait sur un noeud **exp**. Les affichages liés au mode *debug* ont très rapidement été mis en place étant donné qu'il suffisait d'ajouter un affichage aux trois méthodes citées précédemment avec les informations adéquates. Dans le cas du mode *memory*, on parcourt simplement le dictionnaire contenant toutes les variables.

3.2 Le test conditionnel

La condition *si* n'a pas pris longtemps à être ajouté car à ce stade, on avait déjà compris comment utiliser l'arbre et donc compris comment accéder à la condition pour ensuite itérer sur le reste des instructions. Concernant *sisinon*, il fallait juste récupérer le résultat du *si* et ensuite exécuter ou non les instructions se trouvant dans le corps de *sisinon*.

3.3 Les boucles

La boucle *tantque* fût très rapide également car la condition fonctionne tout comme les tests conditionnels. Nous avons aussi compris qu'une fois l'arbre parcouru, il se transformait et qu'il fallait donc dans le cas des boucles, sauvegarder les instructions se trouvant dans le corps avant de les parcourir. La boucle *pourchaque* n'était pas beaucoup plus complexe sur le papier mis à part si la variable créée existe déjà. Une première approche à consister à utiliser un deuxième dictionnaire dans la mémoire qui contenait les variables ne pouvant pas être utilisées temporairement mais cela causait un problème. En effet, si on a créé une variable et qu'ensuite il y a deux boucles imbriquées *pourchaque* avec le même nom que la variable initiale alors on perdait notre valeur initial et on se retrouvait avec une erreur. La deuxième approche a donc été d'utiliser une pile au lieu d'un dictionnaire car c'est toujours la dernière variable ajoutée qui doit être retirée en première.

4 Les erreurs connues, les solutions envisagées et les difficultés rencontrées

Il y a actuellement aucune erreur connue, tous les teste ont fonctionné. Cependant, nous avons eu des difficultés pour la gestion des types dans les listes. En effet, nous n'étions pas parti sur la meilleure solution dès le départ ce qui nous a embrouillés dans le code. La solution finale a été de stocker les types des valeurs (contenues dans les listes) dans un tuple

avec le type **liste** dans la partie type des Tokens. Ce problème fut compliqué parce qu'il fallait prendre en compte qu'une liste pouvait avoir plusieurs dimensions.

Une autre difficulté du projet a été de se débrouiller avec une documentation bancale à propos du module lark. Par exemple, les attributs de l'erreur UnexpectedCharacters n'étaient pas présents dans la documentation. Il a donc fallu aller voir le code source.

5 Répartition du travail

En ce qui concerne la répartition des tâches, dans un premier temps, nous fonctionnons en se documentant chacun de notre côté et ensuite, nous rassemblons nos idées. Le projet étant peu conséquent en matière de code, nous avons décidé de travailler lors de l'implémentation de manière alternée avec des appels récapitulatifs entre chaque shift ainsi qu'avec des séances en présentiel. Ceci nous permettant donc d'avancer plus rapidement et efficacement.

6 Conclusion

En conclusion, ce projet peu conséquent était très intéressant. Au début du projet, l'utilisation du module Lark nous a peu rassurés, ce qui nous a fait un peu reculer. Néanmoins, une fois que le fonctionnement a été correctement assimilé, cela a été une partie de plaisir.