ROUMEGOUS Adrien
Si Frédéric

«Projet Compilation»

Rapport Compilation.

Sommaire

Introduction.

Ce projet a pour but de reproduire au mieux un analyseur syntaxique du langage **C**, sous la forme de fichiers **tcp**, avec les outils **flex** et **bison**. Le fruit de notre travail doit pouvoir détecter une quelconque erreur de syntaxe en pointant la ligne et le caractère concerné. Ainsi, pour un fichier mis en argument, l'analyseur renvoie 0 si tout s'est bien passé, 1 en cas de soucis, avec une erreur, avec son numéro de ligne et de caractère, pointée par une flèche verticale. Ce projet nous a permis de revoir en amont les langages **flex** et **bison**, la gestion de conflits ou encore la manipulation de grammaires.

Architecture.

Voici comment est composé notre analyseur syntaxique :

- Un fichier **flex** ***as.lex*** qui regroupe l'ensemble des lexèmes reconnus par le langage dans lequel est défini notre grammaire. C'est autrement dit l'analyseur lexical.
- Un fichier **bison** ***as.y*** où est définie la grammaire **bison**, ainsi que les fonctions lisant le fichier et repérant les erreurs. Une grammaire de base nous a été fournie, assez similaire à l'actuelle.
- En dehors du programme, des jeux test de type **tpc** ont été stockés dans le dossier **Test** de notre programme. Un script **bash** crée un rapport regroupant les résultats de ces tests. Enfin un **makefile** compile le projet et crée un exécutable **as**.

Choix d'implémentation.

Nous avons commencé par le dossier **flex**. Les *tokens* de la grammaire ont été regroupés dans l'analyseur syntaxique (**as.y**) de pouvoir être considéré comme des lexèmes par l'analyseur lexical. Nous nous sommes ensuite adapté à la grammaire pour chaque lexème. Nous avons également rajouté des balises commentaires pour les spécifier, et aucun lexème n'est pris en compte.

Ensuite, notre objectif était d'implémenter la fonction d'erreur **yyerror** ainsi que le **main**. Il a suffit d'appeler **yyparse**. Par pur choix personnel, nous avons décidé de regrouper l'ensemble des erreurs d'un programme. Un compteur **parse** s'incrémente à chaque appel de **yyerror**. Le **main** affiche le nombre total d'erreurs et renvoie **0** ou **1** si le programme est respectivement correct ou non.

Par la suite, l'objectif était d'écrire la ligne où se situe l'erreur, avec une flèche désignant le caractère problématique. De plus, nous avons affiché les numéros de ligne et de caractère correspondant. Pour cela, nous incrémentons le numéro de ligne à chaque retour chariot **\n** (avec ou sans commentaires). Le numéro de colonnes est dans la plupart des cas incrémenté de **yyleng**. Il est remis à 0 à chaque retour chariot. Leurs données sont récupérées dans le **bison**, en tant que variables externes.

Afin de pouvoir stocker la ligne en cas d'erreur, tout en continuant de détecter la grammaire, nous utilisons la technique du **REJECT**. La ligne est copiée dans une variable **ident** (affiliée à **yylval**). La ligne est ensuite relue normalement lexème par lexème. En cas d'erreur, on affiche dans le

yyerror le **yylval.ident**. Afin de préciser le caractère, il a suffit d'afficher une flèche après un nombre d'espaces correspondant au numéro de caractère de l'erreur.

La grammaire initiale comprenait un conflit de décalage/réduction. Dans ce cas de figure, nous n'avons pas jugé utile, ni même tout simplement faisable de résoudre ça autrement que par un **%expect 1**.

Enfin, nous avons implémenté la détection de structures. Nous avons pour cela grossièrement ajouté un lexème **STRUCT** qui détecte le mot « struct ». Nous avons ajouté leur grammaire et modifié les cas impactés.

Au fur et à mesure, nous avons ajouté des fichiers test pour vérifier le véracité de la grammaire. Ceux-ci sont regroupés dans le dossier **Test**.

Difficultés rencontrées et expérience d'apprentissage.

Ce programme nous a permis de créer un analyseur syntaxique qui se rapproche d'un original. Nous avons cependant rencontré quelques difficultés. La lecture des tabulations (qui devraient faire 4 caractères) semble assez différente lors de l'exécution sur le terminal. La flèche est donc souvent non conforme au caractère. Dans le fichier généré *rapport.txt*, l'erreur n'a pas lieu, heureusement. Nous avons donc préféré (sauf dans un cas arbitraire) utiliser des espaces pour plus de lisibilité. Nous ne pouvons également pas contrôler la ligne où est détectée l'erreur par rapport à sa position réelle, dans le cas où les deux sont séparés par des retours chariots. L'erreur est bien détectée, mais la ligne retenue est la dernière à ce moment là. Non pas que ce soit incorrect, ce n'est pas non plus optimal. L'erreur attendue n'est pas non plus forcément la bonne dans le cas d'un docstring non fermé, ou bien d'un programme sans fonctions. Mais globalement le programme d'analyseur syntaxique est fonctionnel selon nous.

Nous nous sommes familiarisés en **flex** et en **bison** grâce à ce projet. Nous avons pu revoir nos notions et les approfondir, dans un cadre plus concret. Nous avons pu mieux comprendre les premières étapes produites par un compilateur.