RAPPORT DE PROJET

Projet réalisé en binôme dans le cadre du module **Analyse Syntaxique**.

BINÔME

Année : 2025 – 2026 | Licence 3 Informatique

OT **OUMAOUCHE Taous**

MR **MOKHTARI Rayane**

INTRODUCTION

Ce projet s'inscrit dans le cadre du module **Analyse Syntaxique** et a pour objectif la réalisation d'un analyseur syntaxique pour le langage **TPC**, un sous-ensemble du langage C.
L'analyseur doit vérifier la conformité syntaxique des programmes fournis en entrée, signaler les erreurs éventuelles et produire un arbre syntaxique abstrait représentant la structure du programme.
Le projet repose sur l'utilisation des outils **Flex** pour l'analyse lexicale et **Bison** pour l'analyse syntaxique. Il constitue une base essentielle pour la suite du projet de compilation au second semestre.

EXTENSIONS STRUCTURES

Le langage TPC gère les structures de données de manière simplifiée mais expressive.
Notre analyseur prend en charge les fonctionnalités suivantes :

- déclaration de structures globales et locales ;
- champs de structures imbriqués (ex. struct B utilisée dans struct A) ;
- utilisation des structures comme paramètres ou valeurs de retour de fonctions ;
- accès aux champs simples et imbriqués (a.b.x) ;
- affectation de structures et de champs.

Les restrictions imposées par le sujet ont été respectées, notamment l'absence de structures anonymes et l'interdiction de certaines formes de déclaration non prévues par la grammaire

ANALYSE LEXICALE

L'analyse lexicale est réalisée à l'aide de Flex.
Le lexer reconnaît les éléments suivants :

- mots-clés du langage (**if, else, while, return, struct, etc.**) ;
- identificateurs respectant la syntaxe standard du C ;
- constantes numériques ;
- caractères littéraux simples et échappés **('a', '\n', '\t')** ;
- opérateurs arithmétiques, relationnels et logiques ;
- commentaires de type **//** et **/* ... */** (sur un seul niveau).

Les erreurs lexicales sont correctement détectées et signalées avant l'analyse syntaxique.

GRAMMAIRE ET PRÉCÉDENCES

La grammaire du langage TPC a été définie conformément au sujet fourni.
Les précédences et associativités des opérateurs ont été soigneusement respectées afin de garantir une évaluation correcte des expressions.
La hiérarchie suivante est utilisée :

```
Exp → Exp OR TB | TB
TB  → TB AND FB | FB
FB  → FB EQ M   | M
M   → M ORDER E | E
E   → E ADDSUB T| T
T   → T DIVSTAR F | F
```

Une difficulté importante concernait l'ambiguïté du dangling else.
Elle a été résolue en utilisant les directives de priorité de Bison (%right et %prec) afin de garantir que le else soit toujours associé au if le plus proche, conformément au comportement du langage C.

ARBRE SYNTAXIQUE ABSTRAIT

L'analyseur construit un arbre syntaxique abstrait (AST) pour chaque programme syntaxiquement valide.
Cet arbre respecte les contraintes suivantes :

- chaque opérateur est représenté par un nœud interne ;
- les opérandes sont représentées comme fils du nœud opérateur ;
- les listes (paramètres, expressions, déclarations) sont regroupées sous un même nœud parent ;
- les symboles purement syntaxiques (`;`, `{}`, `,`) ne sont pas représentés.

L'AST permet une représentation claire et exploitable de la structure du programme, en vue des étapes ultérieures de compilation.

INTERFACE UTILISATEUR

L'analyseur est exécuté en ligne de commande.
Par défaut, il lit un programme TPC depuis l'entrée standard et vérifie sa syntaxe.

Une option supplémentaire **--tree** permet d'afficher l'arbre syntaxique abstrait du programme analysé lorsque celui-ci est valide.
En cas d'erreur syntaxique, un message explicite indiquant la ligne concernée est affiché et le programme retourne un code de sortie égal à **1**.
En l'absence d'erreur, le code de sortie est **0**.

DIFFICULTÉS TECHNIQUES

Plusieurs difficultés ont été rencontrées au cours du projet :

- la résolution de l'ambiguïté du **if / else** nécessitant une gestion fine des priorités dans Bison ;
- la construction cohérente de l'arbre syntaxique abstrait, notamment pour les productions optionnelles et les listes ;
- la gestion des erreurs syntaxiques afin d'obtenir des messages clairs sans interrompre prématurément l'analyse.
  Ces difficultés ont permis d'approfondir la compréhension du fonctionnement interne des analyseurs syntaxiques générés automatiquement.

RÉSOLUTION DU *DANGLING ELSE*

Grammaire ambiguë :

```
if (1) {
  if (1)
    return;
  else
    return;
}
```

Conflit décalage/réduction : le parseur hésite entre associer le else au premier ou au second if. Le comportement attendu est de relier le else au if le plus proche.

Solution avec Bison :

utilisation des priorités pour lever l'ambiguïté :

```
%nonassoc THEN
%nonassoc ELSE
%prec THEN
```

→ le terminal **ELSE** possède une priorité supérieure, ce qui force l'association avec le if le plus proche et supprime le conflit.

JEU DE TESTS

100 % réussite bac à sable VPL

Couverture des règles : **88,65 %**

ANNEXE — COMPILATION, EXÉCUTION ET OPTIONS

Compilation du projet :

La compilation complète de l'analyseur est automatisée à l'aide du Makefile. À la racine du projet, la commande suivante permet de générer l'exécutable :

```
make
```

L'exécutable tpcas est alors produit dans le dossier bin/.

Exécution de l'analyseur :

```
./bin/tpcas < fichier.tpc
./bin/tpcas fichier.tpc
```

En cas d'erreur syntaxique, l'analyseur affiche la ligne et la colonne de la première erreur rencontrée et retourne un code de sortie égal à 1. Si le programme est valide, le code de sortie est 0.

Options disponibles :

```
./bin/tpcas --help

Usage : ./bin/tpcas [OPTIONS] < FILE
        ./bin/tpcas [OPTIONS] FILE

Options :
  -t, --tree     Affiche l'arbre abstrait d'un langage reconue par notre analyseur syntaxique.
  -h, --help     Affiche cette aide
```

L'option --tree permet d'afficher l'arbre syntaxique abstrait du programme uniquement lorsque celui-ci est syntaxiquement valide.

CONCLUSION

Ce projet a permis de mettre en pratique les concepts fondamentaux de l’analyse lexicale et syntaxique.
L’analyseur obtenu est conforme aux spécifications du langage TPC, robuste face aux erreurs syntaxiques et capable de produire une représentation abstraite exploitable des programmes analysés.

Il constitue une base solide pour les étapes ultérieures du projet de compilation.