Université Gustave Eiffel

Projet d'Analyse syntaxique

Licence d'informatique

—2020-2021—

Le but du projet est d'écrire un analyseur syntaxique en utilisant les outils `flex` et `bison`.

Le langage source est un petit langage de programmation appelé TPC, qui ressemble à un sous-ensemble du langage C.

Le projet est à faire en binôme ou seul. Si vous n'avez pas de binôme, contactez Eric Laporte. Les dates limite de rendu sont :

- le dimanche 14 novembre 2020 à 23h55 pour une première version de l'analyseur [1] avec makefile, 2 jeux de tests (au moins 1 fichier dans chaque jeu), script de test, affichage des numéros de ligne en cas d'erreur, valeur de retour de l'analyseur syntaxique égale à 0 si et seulement si l'entrée est correcte (1/4 de la note de projet) ;
- le mercredi 30 décembre 2020 à 23h55 pour l'analyseur complet, y compris les types structures, la documentation et la flèche verticale pour indiquer les erreurs de syntaxe (3/4 de la note de projet).

1 Définition informelle du langage source

Un programme TPC est une suite de fonctions. Chaque fonction est constituée de déclarations de variables (locales à la fonction), et d'une suite d'instructions. Les fonctions peuvent être récursives. Il peut y avoir des variables de portée globale. Elles sont alors déclarées avant les fonctions.

Les types de base du langage sont `int` (entier signé) et `char`. Le mot clé `void` est utilisé pour indiquer qu'une fonction ne fournit pas de résultat ou n'a pas d'arguments.

Le langage TPC utilise `print` pour afficher un entier ou un caractère. Le mot-clé `readc` permet d'obtenir un caractère lu au clavier ; `reade` permet de lire un entier.

2 Définition des éléments lexicaux

Les identificateurs sont constitués d'une lettre, suivie éventuellement de lettres, chiffres, symbole souligné ("_"). Vous pouvez fixer une longueur maximale pour un identificateur. Il y a distinction entre majuscule et minuscule. Les mots-clés comme `if`, `else`, `return`, etc., doivent être écrits en minuscules. Ils sont reconnus par l'analyseur lexical et ne peuvent pas être utilisés comme identificateurs.

Les éléments lexicaux pour les constantes numériques sont des suites de chiffres.

Les caractères littéraux dans le programme sont délimités par le symbole `'`, comme en C. Dans les caractères littéraux, la barre oblique inverse ("\") est utilisée pour déspécialiser `'` et pour spécialiser `n` et `t` : `\n` et `\t` sont le caractère fin de ligne et la tabulation.

Les commentaires sont délimités soit par `/*` et `*/`, soit par `//` et la fin de la ligne, et ne peuvent pas être imbriqués.

Les différents opérateurs et autres éléments lexicaux sont :

`=`	: opérateur d'affectation
`+`	: addition ou plus unaire
`-`	: soustraction ou moins unaire
`*`	: multiplication
`/` et `%`	: division et reste de la division entière
`!`	: négation booléenne
`==`, `!=`, `<`, `>`, `<=`, `>=`	: les opérateurs de comparaison
`&&`, `||`	: les opérateurs booléens
`;` et `,`	: le point-virgule et la virgule
`(`, `)`, `{` et `}`	: les parenthèses et les accolades

(1). Déposez votre projet sur la plateforme elearning dans la zone prévue à cet effet, sous la forme d'une archive tar compressée de nom "ProjetASL3_NOM1_NOM2.tar.gz", qui, au désarchivage, crée un répertoire "ProjetASL3_NOM1_NOM2" contenant le projet.

Chacun de ces éléments sera identifié par l'analyse lexicale qui devra produire une erreur pour tout élément ne faisant pas partie du lexique du langage.

3 Notations et sémantique du langage

Dans ce qui suit,

- **CHARACTER** et **NUM** désignent respectivement un caractère littéral et une constante numérique ;
- **IDENT** désigne un identificateur ;
- **TYPE** désigne un nom de type simple qui peut être `int` ou `char` ;
- **EQ** désigne les opérateurs d'égalité ('`==`') et d'inégalité ('`!=`') ;
- **ORDER** désigne les opérateurs de comparaison '`<`', '`<=`', '`>`' et '`>=`' ;
- **ADDSUB** désigne les opérateurs '`+`' et '`-`' (binaire ou unaire) ;
- **DIVSTAR** désigne les opérateurs '`*`', '`/`' et '`%`' ;
- **OR** et **AND** désignent les deux opérateurs booléens '`||`' et '`&&`'.
- Les mots-clés sont notés par des unités lexicales qui leur sont identiques à la casse près.

L'instruction nulle est notée '**;**'.

4 Grammaire du langage TPC

Prog : DeclVars DeclFoncts ;
DeclVars : DeclVars **TYPE** Declarateurs **';'**
| ;
Declarateurs : Declarateurs **','** **IDENT**
| **IDENT** ;
DeclFoncts : DeclFoncts DeclFonct
| DeclFonct ;
DeclFonct : EnTeteFonct Corps ;
EnTeteFonct : **TYPE IDENT '('** Parametres **')'**
| **VOID IDENT '('** Parametres **')'** ;
Parametres : **VOID**
| ListTypVar ;
ListTypVar : ListTypVar **','** **TYPE IDENT**
| **TYPE IDENT** ;
Corps : **'{'** DeclVars SuiteInstr **'}'** ;
SuiteInstr : SuiteInstr Instr
| ;
Instr : LValue **'='** Exp **';'**
| **READE '('** **IDENT** **')' ';'**
| **READC '('** **IDENT** **')' ';'**
| **PRINT '('** Exp **')' ';'**
| **IF '('** Exp **')'** Instr
| **IF '('** Exp **')'** Instr **ELSE** Instr
| **WHILE '('** Exp **')'** Instr
| **IDENT '('** Arguments **')' ';'**
| **RETURN** Exp **';'**
| **RETURN ';'**
| **'{'** SuiteInstr **'}'**
| **';'** ;
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   ;
F               : ADDSUB F
                | '!' F
                | '(' Exp ')'
                | NUM
                | CHARACTER
                | LValue
                | IDENT '(' Arguments ')'   ;
LValue          : IDENT   ;
Arguments       : ListExp
                |  ;
ListExp         : ListExp ',' Exp
                | Exp   ;
```

5 Travail demandé

Analyseur syntaxique Écrivez un analyseur syntaxique de ce langage en utilisant `flex` pour l'analyse lexicale et `bison` pour l'analyse syntaxique. Les messages d'erreur doivent donner le numéro de ligne et le numéro du caractère dans la ligne, puis reproduire la ligne et indiquer le caractère par une flèche verticale.

Extension du langage Modifiez la grammaire de façon à autoriser les types structures. Il doit pouvoir y avoir des types structures, déclarés globalement et avant les fonctions. La déclaration doit être conforme au langage C et chaque structure doit avoir au moins un champ. Une fois un type structure déclaré, on doit pouvoir utiliser la syntaxe `struct my_structure` (le mot-clé `struct` suivi de l'identificateur) pour déclarer des variables globales et locales, des paramètres, et des types de retour des fonctions :

```
struct aircraft my_plane, my_shuttle, my_satellite ;
```

On ne demande ni les `typedef`, ni les structures imbriquées (une structure dont un champ est de type structure), ni les types structures déclarés localement dans une fonction.

Vous pouvez faire des modifications supplémentaires à la grammaire, mais elles ne peuvent affecter le langage engendré que si cela enrichit le langage TPC.

Le répertoire que vous déposerez doit être organisé correctement : un répertoire pour les sources, un autre pour la documentation, un autre pour les tests... Le répertoire pour les sources doit contenir un Makefile nommé `Makefile` ou `makefile`. L'analyseur créé avec le Makefile doit être nommé `as`. La commande suivante doit exécuter votre analyseur :

./as < *prog***.tpc**

Le programme devra renvoyer 0 si et seulement si *prog***.tpc** est correct. **Cette consigne est importante, parce que les correcteurs de votre projet font des tests automatiques, du coup un projet qui fait une bonne analyse syntaxique mais renvoie une mauvaise valeur est pénalisé.**

Tests Écrire deux jeux de tests, un pour les programmes TPC corrects et un autre pour les programmes incorrects, et un script de déploiement des tests, qui produit un rapport unique donnant les résultats de tous les tests. (Nous utilisons nos propres jeu d'essais pour les corrections.)

Documentation Vous décrirez dans votre documentation vos choix et les difficultés que vous avez rencontrées.

Déposez votre projet sur la plateforme elearning dans la zone prévue à cet effet, sous la forme d'une archive tar compressée de nom "ProjetASL3_NOM1_NOM2.tar.gz", qui, au désarchivage, crée un répertoire "ProjetASL3_NOM1_NOM2" contenant le projet.