Projet d’Analyse syntaxique

Licence d’informatique

—2020-2021—

Table des matières

Introduction :

Le but de ce projet est de créer un analyseur syntaxique pour un pseudo langage dérivé du C, le TPC.
Pour cela nous avons utiliser les outils ***flex*** et ***bison***.
La syntaxe du TPC est définie comme suit :

```
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 '}'
```

. . .

La définition du TPC donne la base du projet **bison**, à cela nous avons dû ajouter les différentes variables obligatoire pour la gestion de l'écriture de la ligne en cas d'erreur.

Le programme **flex** quant a lui a été plus compliquer a réaliser.
La nécessitée de pallier a toutes les éventuels erreurs et a rendre compte de tout les cas possible a induit un grand nombre de test.

Pour tester, voir l'efficacité de notre analyseur syntaxique nous avons implémenter pas moins de 10 tests : 5 correspondant à la syntaxe du langage TPC, et 5 comprenant des erreurs.

Compilation/Utilisation :

Pour compiler et enclencher les tests il suffit de :

soit (dans le dossier courant):

make -C as test

ou

make -C as && ./as/test_bash.sh && cat as/resultat.txt

soit (dans le dossier as):

make test

ou

make && ./test_bash.sh && cat resultat.txt

Pour faire des tests différents du jeu de test déjà présent.
On peut soit ajouter un fichier dans les dossiers ***test_correct/*** *ou* ***test_incorrect/****, soit via l'exécutable* ***as*** *avec* ***./as <*** ***nom_fichier_test.tpc*****.**

Détails des tests incorrects :

test_incorrect1 :

int r1, r2 absence de « ; » ligne 6

test_incorrect2 :

float mult(int a, int b) { type **float** inexistant en TPC ligne 20

test_incorrect3 :

int mult(float a, int b) { type **float** inexistant en TPC ligne 20

test_incorrect4 :

nbé=-1; présence d'un caractère non reconnu pour un nom de variable ligne 3

test_incorrect5 :

if (n =!= 0) opérande non reconnu dans la grammaire TPC ligne 14

```
tests incorrectes :
test_incorrecte2.tpc
 valeur de retour :    1
 retour :
syntax error near line 20, character 1:
float mult(int a, int b) {
^
test_incorrecte3.tpc
 valeur de retour :    1
 retour :
syntax error near line 20, character 10:
int mult(float a, int b) {
         ^
test_incorrecte4.tpc
 valeur de retour :    1
 retour :
syntax error near line 3, character 6:
   nbé=-1;
     ^
test_incorrecte5.tpc
 valeur de retour :    1
 retour :
syntax error near line 14, character 9:
  if (n =!= 0)
        ^
test_incorrecte.tpc
 valeur de retour :    1
 retour :
syntax error near line 7, character 1:
int max;
^
Nombre de tests incorrectes échoues BONNE NOUVELLE:
0,00% des 5 tests
Nombre de tests incorrectes réussis MAUVAISE NOUVELLE:
100,00% des 5 tests
```

Avec ces tests nous avons essayé de pallier à toutes les erreurs pouvant survenir à cause d'une violation de la grammaire TPC.

Le script bash inscrit a la suite des tyest la reussite en poucentage de ces tests.

Ainsi on voit ci dessus que 0,00 % des test corrects ont renvoyé une erreur et que 100,00 % des tests incorrects ont renvoyé une erreur.

Difficultés rencontrées :

Lors du développement du projet nous avons rencontré plusieurs difficultés mais je ne vais en citer que 3 qui sont pour nous les plus importantes.

La première a été *l'incorporation d'un retour d'erreur détaillé avec l'affichage de la flèche* qui indique l'emplacement de l'erreur sur la ligne affichée.
Nous avons eu particulièrement du mal à mettre celle-ci *au début* du dernier lexème lu.

```
            fprintf(stderr, "%s near line %d, character %d:\n%s\n", s, count_line, (count_char - last_token) + 1,
cnt_line);
    for (i = 0; i < count_char - last_token; i++)
        fprintf(stderr, " ");
    fprintf(stderr, "^\n");
    return (0);
```

La deuxième difficulté majeure a été *l'implémentation dans la grammaire des déclarations de structures*, nous avions choisi au *rendu intermédiaire* de forcer la **déclaration** des structures en début de fichier.
Après le premier retour nous avons essayé de rendre les structures déclarables après ou avant la déclaration de variable globale, nous avions trouvé une méthode qui le permettait mais celle-ci étant trop **permissive** (nous pouvions déclarer une structure à l'intérieur d'une autre déclaration de structure).
Nous avons finalement **réussi** en affinant la méthode utilisée précédemment.

La dernière difficulté a été la réalisation d'un script *bash* le plus complet possible, nous ne sommes pas du tout à l'aise avec ce langage ce qui nous a obligé à faire beaucoup de **recherches**.