Le 16/12/2021
L3 Informatique

Semestre 5

Compte Rendu Projet Analyse Syntaxique

Binôme:

RAMAROSON RAKOTOMIHAMINA Johan
MENAA Mathis

Encadrants:
Le corps d'enseignement
d'Analyse Syntaxique

I/Introduction **3**
Etat/Fonctionnalités du projet 3
Exemple d'utilisation 3
Outil de test 4
II/ Choix **4**
III/ Etiquetage **5**
IV/ Difficultés **6**
Conclusion **6**

I/Introduction

Le but de ce projet est d'écrire un analyseur syntaxique en utilisant les outils flex et bison vu ce semestre. Le fichier d'entrée sera en TPC (une variante du C), si il ne contient pas d'erreur lexicale ou syntaxique, notre analyseur devra le traduire en arbre abstrait sur la sortie standard.

1. Etat/Fonctionnalités du projet

La conception de projet a été faite selon les différentes instructions du corps enseignant.
Voici les objectifs et modalités :

- **Option -t, –tree qui affiche l'arbre abstrait sur la sortie standard.**
- **Renvoie 0 sur la sortie standard si aucune erreur dans le fichier en entrée.**
- **Renvoie 1 ainsi que le détail de l'erreur sur la sortie standard (Numéro de la ligne et position dans la ligne).**
- **Option -h, –help qui affiche une description de l'interface utilisateur et termine l'exécution.**

2. Exemple d'utilisation

```
./tpcas [OPTIONS]
```

- Option disponible : -t, --tree / -h --help

```
bin/tpcas < test/good/postif_test3.tpc [OPTION]
```

- Cette commande lance l'analyseur sur le test positif "positif_test3.tpc".
  On peut bien sûr rajouter des options et aussi.

```
bin/tpcas < test/syn-err/[NOM DE TEST] [OPTION]
```

- Cette commande lance l'analyseur sur un test négatif, il suffit de remplacer le premier champ par le nom d'un test et le deuxième champ par une option ci-dessus si vous souhaitez en utiliser une.

3. Outil de test

Pour nous aider dans notre développement, nous avons eu l'idée de développer un fichier **bash** nous permettant d'appliquer nos tests sur tous les fichiers présents dans les dossiers **"good"** et **"syn-err"**. A chaque petit changement dans les fichiers de tests il nous suffit d'appliquer le **bash** pour obtenir le résultat en fonction de chaque fichier.

II/ Choix

L'objectif était dans un premier temps **d'établir un lexeur** reconnaissant tous les lexèmes décrivant le langage source (TPC). **Ajouter les variables** permettant de **compter les caractères ainsi que les lignes** pour plus tard. Une fois notre lexeur terminé, nous devions remplir notre parser et re adapter les fichier tree.c et tree.h vu en TD à notre situation.

Dans l**a conception de notre arbre** pour la règle *"DeclVars"* on a préféré afficher le nœud dans l'arbre même s' il ne contient pas de fils pour simplifier l'implémentation. Lors du rendu intermédiaire, il nous manquait les caractères spéciaux à reconnaître comme **CHARACTER** sinon tout était correct. Pour le rendu final, il nous fallait implémenter **la gestion des switch** en respectant les règles du sujet :

- Après un switch, les parenthèses puis les accolades seront obligatoires.
- il pourra y avoir des **case**, des **default** et des **break** à l'intérieur des accolades reliées directement à un **switch**, mais pas à l'intérieur des autres accolades.
- autoriser qu'il y ait des instructions avant le premier **case**, ou un **default** avant un **case**, ou plusieurs **default** dans le même **switch**.

On a choisi de séparer l'implémentation des switch en 3 règles : ***BeginSwitchExpr, SwitchExpr et EndSwitchExpr***.

```
BeginSwitchExpr:
        BeginSwitchExpr SwitchExpr {if($1){$$ = $1; addSibling($$,$2);}else{$$=$2;}}
    |   %empty                     {$$ = NULL;}
    ;
SwitchExpr:
        CASE Exp ':' EndSwitchExpr {$$ = makeNode(Case); addChild($$, $4);}
    |   DEFAULT ':' EndSwitchExpr {$$ = makeNode(Default); addChild($$, $3);}
    ;
EndSwitchExpr:
        SuiteInstr BREAK ';' {$$ = makeNode(SuiteInstr); addSibling($$, makeNode(Break));}
    |   SuiteInstr           {$$ = $1;}
    ;
```

BeginSwitchExpr : Permet de gérer la possibilité qu'il y ait des instructions avant le premier **case**.

SwitchExpr : Gère les **case** et les **default** dans notre **switch**.

EndSwitchExpr : Gère dans la deuxième partie de *SwitchExpr* en délimitant un **case** ou **default** par une suite d'instruction et un **break** à la fin, ou plusieurs instructions et un **break** à la fin.

III/ Etiquetage

En dernier point, il était nécessaire d'étiqueter nos nœuds de terminaux. Nous en aurons besoin pour le projet de compilation au semestre prochain.

Pour ce faire nous devions récupérer dans notre lexeur les informations lues pour les **TOKENS** : **IDENT, TYPE, EQ, ADDSUB, ORDER, DIVSTAR et NUM**.

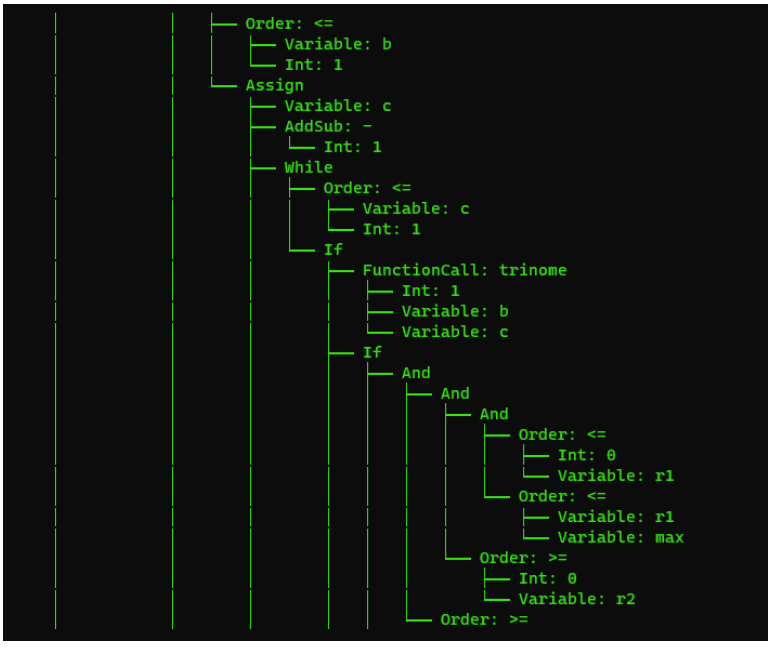
On a ajouté une *Union* dans notre structure *node* et dans notre parseur de la forme :

```
union{
      char ident[64]; //Correspond aux IDENT/ TYPE
      int num; //Correspond aux NUM
      char byte; //Correspond aux DIVESTAR/ ADDSUB
      char comp[3]; //Correspond aux ORDER/ EQ
}u;
```

Dans le parser, on stock les données récupérées dans notre lexeur si on rencontre un token listé.

On a modifié notre fonction d'affichage de l'arbre en ajoutant un *switch* en fonction du type de token lues sa valeur.

Lors de l'affichage on a donc pour nos *identificateurs* **leur nom**, **leur type** ainsi que **leur valeur** affichée plus les **opérations** effectuées sur ces valeurs.



Un exemple d'arbre

IV/ Difficultés

Au niveau des difficultées rencontrées, on a eu quelques soucis au niveau de la conception de l'arbre. Comme on l'a vu que très rapidement en fin d'un TD.

Il nous aura fallu aussi un peu de temps pour comprendre comment correctement étiquetés notre arbre, et sous quel forme les informations d'étiquetages devraient être affichées.

Conclusion

Ce projet nous a permis d'approfondir et pratiquer les connaissances vues tout au long de ce semestre. Que ce soit au niveau de la création d'un lexeur poussé ainsi que d'un parser permettant la création d'arbre abstrait.

Il était nécessaire de faire le meilleurs projet car ce sera la base du projet du prochain semestre en compilation.

Enfin les délais de réalisation du projet sont corrects, nous avons su faire les bon choix pour ne pas perdre de temps et optimiser au maximum l'avancement de notre travail.