TD N° 2

Analyse syntaxique

1 Analyseur syntaxique descendant sans analyse lexicale

On s'intéresse à analyser un langage artificiel avec des expressions et des instructions conditionnelles données par la grammaire suivante :

S → "print" NUM
| "if" E "then" S "else" S
E → NUM "=" NUM

Les terminaux NUM sont des nombres.
S et E sont les non terminaux.
On suppose qu'on a la définition des terminaux comme suit :

```
#define TOKTYPE enum token
TOKTYPE {IF, THEN, ELSE,  PRINT, NUM, EQ};
TOKTYPE uc;
```

1. Quelle est la procédure principale de l'analyseur syntaxique descendant pour cette grammaire ?
2. Créez un fichier analyse_desc.c qui contient
   - la définition des terminaux
   - la fonction main :

```
int main () {
    uc = getUnite();
    reconnaitre_S();
    return 0;
}
```

   où getUnite() est la fonction d'analyse lexicale. Vous la programmerez plus tard dans la question 3.2. L'unité lexicale retournée par cette fonction est stockée dans la variable **uc.**

3. Pour l'instant, on souhaite se focaliser sur l'analyse syntaxique; on cherche donc à éviter d'écrire l'analyseur lexical. C'est pour cela qu'on va supposer que les unités ont déjà été stockées dans un tableau tabunites, ainsi la fonction getUnite n'aura qu'a lire depuis ce tableau. A titre d'exemple, pour analyser la phrase "1 = 2", vous définissez le tableau tabunites comme suit :

```
TOKTYPE tabunites [] = {NUM, EQ, NUM};
```

   3.1 Quelles seraient les définitions de tabunites pour les phrases suivantes ?
   - if 1 = 2 then print 3 else print 4
   - if 1 = 2 then print else print 4
   - if 1 = 2 then if 2 = 2 then print 3 else print 5 else print 4

   3.2 Implémenter getUnite() en lisant les unités à partir du tableau tabunites.

4. Testez votre analyseur syntaxique avec chacune des phrases de la question 3.1 et vérifiez la conformité des résultats (pensez à bien modifier la définition de tabunites à chaque fois).

2 Analyse lexicale avec Lex / analyse syntaxique descendante

On va maintenant se focaliser sur l'analyse lexicale. Pour cela nous allons remplacer la lecture du tableau des unites par des appels à Lex.

- Lex lit un fichier pointé par **yyin**
- Lex fournit les unités par des appels consécutifs à **yylex**.

Voici les modifications nécessaires à apporter à votre programme pour intégrer l'analyseur lexicale :

1. Rajoutez dans le programme de l'analyseur syntaxique une ligne

```
extern FILE *yyin;
```

   Dans la fonction main il ne faut pas oublier d'ouvrir un fichier et l'associer à l'entrée yyin comme suit :

```
yyin = fopen("essai.txt", "r");
```

2. Remplacez dans l'analyseur syntaxique le code de l'analyse lexicale par des appels à yylex() au lieu de getUnite(). Par exemple au lieu de uc = getUnite() on écrira :

```
uc = yylex();
```

3. Pour finir, il suffit de créer un fichier Lex analyse_desc.l et de compléter la section des règles pour reconnaître toutes les unités du langage. Voici la squelette à compléter en ajoutant les définitions/actions des autres unités :

```
%option noyywrap
%{
#include <stdio.h>
%}
%%
"if"    return IF;
.       printf("Erreur : Caractère non reconnu ! %s\n", yytext);
%%
```

4. Testez votre programme avec les exemples de la question 3 de l'exercice précédent

3 Analyse syntaxique avec Lex/Yacc

Testez l'analyseur syntaxique défini par les programmes suivants. Que fait cet analyseur ?

arith.l

```
%{
#include "syntaxe.tab.h"
extern char nom[]; /* chaine de caracteres partagee avec l'analyseur syntaxique */
%}
chiffre [0-9]
```

```
lettre [A-Za-z]
%%
[" "\t\n]                       { }
{chiffre}+                      { yylval = atoi(yytext); return nombre; }
{lettre}({lettre}|{chiffre})*   { strcpy(nom, yytext); return identif; }
.                               { return yytext[0]; }
%%
int yywrap(void) {
        return 1;
}
```

arith.y

```
%{
char nom[256]; /* chaîne de caractères partagée avec l'analyseur lexical */
%}
%token nombre identificateur
%%
expression : expression '+' terme { printf(" +"); }
           | terme
           ;
terme : terme '*' facteur { printf(" *"); }
      | facteur
      ;
facteur : nombre { printf(" %d", yylval); }
        | identificateur { printf(" %s", nom); }
        | '(' expression ')'
        ;
%%
void yyerror(char *s) {
        printf("<<< \n%s", s);
}
main() {
        if (yyparse() == 0)
        printf(" Expression correcte\n");
}
```