TD/ TP : Analyse sémantique

< Calculatrice >

Soit la phrase suivante qui représente une opération arithmétique : A-B*(C+D)/C

Nous voulons générer un analyseur sémantique de cette expression

A. Analyse lexicale et syntaxique

En utilisant les code source suivants générer l'analyseur syntaxique.

```
/* fichier lexique.l */
%{
#include "calcY.h"
%}
%%
[0-9]+ {return NOMBRE;}
"+" {return PLUS;}
"-" {return MOIN;}
"*" {return MULT;}
"/" {return DIVS;}
"(" {return PAR_O;}
")" {return PAR_F;}
[\t];
\n return 0;
. return yytext[0];
%%
int yywrap(void){
     return 0;
}
```

```
/* fichier syntaxe.y */
%{
int yyerror();
int yylex();
#include <stdio.h>
#include <stdlib.h>
%}

%token NOMBRE PLUS MOIN MULT DIVS PAR_O PAR_F
%start expression
%%

expression: expression PLUS terme
| expression MOIN terme
| terme
;
terme: terme MULT facteur
| terme DIVS facteur
| facteur
;
facteur: PAR_O expression PAR_F
| MOIN facteur
| NOMBRE
;
%%
int main(void){
if (yyparse()==0)
     printf("expression correcte");
}

int yyerror(){
     fprintf(stderr,"erreur de syntaxe\n");
     return 1;
}
```

- Tester l'analyseur générer

B. Analyse sémantique

Dans cette étape nous ajouter du sens aux expressions. Pour cela nous allons modifier le fichier syntaxe.y comme suit :

```
/* fichier syntaxe.y version 2*/
%{
int yyerror();
int yylex();
#include <stdio.h>
#include <stdlib.h>
%}

%token NOMBRE PLUS MOIN MULT DIVS PAR_O PAR_F
%start calcul
%%
calcul : expression {printf("%d\n",$1);}
expression: expression PLUS terme {$$=$1 +
$3;}
| expression MOIN terme {$$=$1 - $3;}
| terme {$$=$1 ;}
;
terme: terme MULT facteur {$$=$1 * $3;}
| terme DIVS facteur {$$=$1 / $3;}
| facteur {$$=$1;}
;
facteur: PAR_O expression PAR_F {$$= $2;}
| MOIN facteur {$$= - $2;}
| NOMBRE {$$= $1;}
;
%%
int main(void){
yyparse();
}

int yyerror(){
     fprintf(stderr,"erreur de syntaxe\n");
     return 1;
}
```

- Modifier l'expression régulière du nombre entier comme suit pour récupérer le résultat du calcul

```
[0-9]+ {yylval = atoi(yytext); return NOMBRE;}
```

- Générer et tester la calculatrice avec ces deux operations :
  - 1+2*4
  - 4/3

Que remarquez-vous ?

- Jusqu'à ici notre calculatrice prend en charge que les résultats en entier (pas de nombre réel). Pour résoudre ce problème nous allons redéfinir dans ER les réels

```
([0-9]+)|([0-9]+"."[0-9]+) {yylval.fval = atof(yytext); return NOMBRE;}
```

- Dans le fichier syntaxe nous allons définir un nouveau type comme suit :

```
%union{double fval; int ival;}
```

- Maintenant nous allons affecter le type fval au non terminal NOMBRE et aux terminaux expression, facteur et terme

```
%token<fval> NOMBRE

%type<fval> expression

%type<fval> terme

%type<fval> facteur
```

- La dernière étape consiste à modifier le type de retour de calcul en float

```
calcul : expression {printf("%f\n",$1);}
```

- Générer et tester la calculatrice 😊