Compilation : analyse syntaxique avec lex et yacc

Pierre Wodey
ISIMA/LIMOS
Campus des Cézeaux
BP 10125
63173 Aubière Cedex

5 novembre 2010

Table des matières

Chapitre 1

Introduction

Nous traitons ici le cas du langage des expressions.

Le document est décomposé en deux grande parties, la première traite de l'analyse syntaxique seule et la seconde présente la méthode de description d'actions sémantiques en cours d'analyse syntaxique.

Nous rappelons ici le contenu du fichier d'analyse lexxicale.

1.1 Le fichier lex

1.1.1 Structure globale

Le fichier lex, analex.l, comporte 5 parties.La première permet de fixer un certain nombre d'otpions.

"analex.l" 2a ≡

```
⟨les options 2b⟩
%{
⟨code C pour les déclarations 3a⟩
%}
⟨les motifs nommés 3c⟩
%%
⟨le traitement associé aux motifs 4a⟩
%%
⟨code C supplémentaire 4b⟩
♢
```

1.1.2 Les options

Nous choissons l'option "yylineno" qui permet d'avoir dans la variable "yylineno" le numéro de la ligne courante, utile pour les messages d'erreurs.

⟨les options 2b⟩ ≡

```
%option yylineno
♢
```

Macro référencée dans fragment 2a.

1.1.3 Le code C en début de fichier

1.1.4 Structure du code C

Nous incluons le quelques fichiers standard, le fichier y.tab.h généré par yacc, le fichier files.h (pour les traces), et le fichier table.h pour la table des symboles.

⟨code C pour les déclarations 3a⟩ ≡

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "y.tab.h"

#include "files.h"
#include "table.h"

#undef DEBUG_L
#define DEBUG_L


/* ------------------------------------------------------------------- */
/* Definition des prototypes de quelques methodes utiles.              */
⟨les prototypes des fonctions 3b⟩
♢
```

Macro référencée dans fragment 2a.

⟨les prototypes des fonctions 3b⟩ ≡

```
extern void yyerror(char *); /* Definie dans anasyn.y */
static int ChercheIdent(char *);
♢
```

Macro référencée dans fragment 3a.

1.1.5 Les motifs

⟨les motifs nommés 3c⟩ ≡

```
/* ------------------------------------------------------------------- */
/*  Description des caracteres constituant une expression               */
delim         [ \t\n]
blanc         {delim}+
lettre        [a-zA-Z]
ident         [a-zA-Z][a-zA-Z0-9_]*
entier [0-9]+
autre         .
♢
```

Macro référencée dans fragment 2a.

1.1.6 Les motifs et les traitements

⟨le traitement associé aux motifs 4a⟩ ≡

```
{blanc}       ;
{entier}      { fprintf(lexi, "entier lu %s", yytext);
                yylval.val = atol(yytext);
                return ENTIER;
                                }
{ident}       { int j;
                                j=ChercheIdent(yytext);
                                yylval.no_ident=j;
                                fprintf(lexi,"AL ident  : %i %s\n",j,tab_symb[j]);

                return IDENT; /* IDENT defini comme token ds anasyn.y*/
              }
"**"          { return POWER;}
"+"           |
"*"           |
"("           |
"-"           |
"/"           |
")"           {
                fprintf(lexi, "Symbole lu : %c \n", yytext[0]);
                return yytext[0]; }
{autre}       {
                char msg[40];

                sprintf(msg, "Caractere illegal %c (ASCII = %d)", yytext[0], (int)yytext[0
                yyerror(msg);
                exit(1);
              }
```

◊

Macro référencée dans fragment 2a.

1.1.7 Code C en fin de fichier

⟨code C supplémentaire 4b⟩ ≡

```
static int ChercheIdent(char *mot) {
   int i=0;

   while (tab_symb[i]&&strcasecmp(tab_symb[i],mot)) i++;

   if (!tab_symb[i])
      tab_symb[i]=strdup(mot);
   return i;
}
```

◊

Macro référencée dans fragment 2a.

Chapitre 2

Analyse syntaxique

Nous décrivons dans ce chapitre la structure du fichier yacc pour faire uniquement l'analyse syntaxique, aucune action sémantique n'est réalisée.

2.1 La structure globale du fichier yacc

Le fichier s'appelle anasyn.y

"anasyn.y" 5 ≡
```
%{
⟨code C en début de fichier yacc 6a⟩
%}
⟨le type de yylval 6b⟩

⟨l'axiome 6c⟩

⟨les lexèmes non caractères 6d⟩

%%
⟨les règles syntaxiques 7a⟩
%%

⟨code C en fin de fichier yacc 7b⟩
```
◊

2.2 Code C en début de fichier

Comme pour le cas du fichier lex, il y a l'inclusion d'un certain nombre de fichiers, la déclaration de macros, de fonctions et de variables.

⟨code C en début de fichier yacc 6a⟩ ≡

```
#include <stdio.h>

#include "files.h"
#include "table.h"

/* Pour l'affichage du parcours des regles lors de l'analyse
   syntaxique                                                */
#undef DEBUG_S
#define DEBUG_S

/* prototype de la fonction d'erreur */
void yyerror(char *);

extern int yylineno;
```

◊

Macro référencée dans fragment 5.

2.3 Type de yylval

On fait l'union pour les différentes types de valeurs retournée par les lexèmes pour lesquels il faut une valeur.

⟨le type de yylval 6b⟩ ≡

```
%union {
   int   no_ident;
   int   val;
}
```

◊

Macro référencée dans fragment 5.

2.4 L'axiome

Très simple, commande %start.

⟨l'axiome 6c⟩ ≡

```
%start E
```

◊

Macro référencée dans fragment 5.

2.5 Les lexèmes autres que les caractères

Ces lexèmes sont déefinis à l'aide de la commande %token. Dans le cas où une valeur est associéee au lexème, il faut donner le nom du champ de l'union de yylval. Pour chaque token, il y aura une constante dans le fichier y.tab.h, cette valeur est celle que doit renvoyer l'analyse lexicale.

⟨les lexèmes non caractères 6d⟩ ≡

```
%token POWER
%token <no_ident> IDENT
%token <val> ENTIER
```

◊

Macro référencée dans fragment 5.

2.6 La grammaire

⟨les règles syntaxiques 7a⟩ ≡

```
E : E '+' T | E '-' T | T ;

T : T '*' F | T '/' F | F;

F : EX | EX POWER EX

EX : ENTIER | IDENT | '(' E ')';
```

◊

Macro référencée dans fragment 5.

2.7 Code C en fin de fichier

⟨code C en fin de fichier yacc 7b⟩ ≡

```
/* Affiche le message d'erreur msg                                    */
void yyerror(char *msg)
{
 if(!msg || *msg=='\0')
  fprintf(stderr, "Erreur ligne %d\n", yylineno);
 else
  fprintf(stderr, "Erreur ligne %d : %s\n", yylineno, msg);
}
```

◊

Macro référencée dans fragment 5.

Chapitre 3

Actions sémantiques

Pour décrire les actions sémantique nous considérons la construction de l'arbre abstrait pour les expressions, nous supprimons la notion d'exposant (POWER).

Les actions sont décrites dans les règles, c'est du code C qui est placé entre accolades. Le code st exécuté lors de la réduction de la règle.

Il faut donc réaliser plusieurs choses :

- typer les symboles non-terminaux,
- affecter symbole lors de la réduction, le nom terminal réduit est identifier par le symbole `$$`,
- la valeur est calculée à partir des valeurs des symboles de la partie droite de la règle identifiées par `$i`, où i est la position du symbole dans la chaîne,
- la valeur peut aussi prendre en compte des variables globales.

3.1 Le nouveau fichier yacc

"anasyn-sem.y" 8 ≡

```
%{
⟨code C modifié en début de fichier yacc 9a⟩
%}
⟨le type de yylval sem 9b⟩

⟨l'axiome sem 9c⟩

⟨les lexèmes non caractères sem 9d⟩

⟨le type des non-terminaux 10a⟩
%%
⟨les règles syntaxiques sem 10b⟩
%%

⟨code C en fin de fichier yacc sem 12⟩
```

◊

3.2 Code C en début de fichier

On déclare juste la variable globale racine qui contiendra l'arbre. Son type est type_expr, définissant le type des noeuds et feuilles de l'arbre.

⟨code C modifié en début de fichier yacc 9a⟩ ≡

```
#include <stdio.h>

#include "files.h"
#include "table.h"

/* Pour l'affichage du parcours des regles lors de l'analyse
   syntaxique                                                */
#undef DEBUG_S
#define DEBUG_S

/* prototype de la fonction d'erreur */
void yyerror(char *);

extern int yylineno;

type_expr *racine;

◊
```

Macro référencée dans fragment 8.

3.3 Type de yylval

On fait l'union pour les différentes types de valeurs retournée par les lexèmes pour lesquels il faut une valeur. Il faut aussi mettre dans l'union le type des différents non-terminaux typés, à savoir le type des de l'arbre des expressions.

⟨le type de yylval sem 9b⟩ ≡

```
%union {
   int   no_ident;
   int   val;
   /* valeur pour les non terminaux */
   struct _type_expr    *expr;}
◊
```

Macro référencée dans fragment 8.

3.4 L'axiome

Très simple, commande %start.

⟨l'axiome sem 9c⟩ ≡

```
%start E
◊
```

Macro référencée dans fragment 8.

3.5 Les lexèmes autres que les caractères

⟨les lexèmes non caractères sem 9d⟩ ≡

```
%token <no_ident> IDENT
%token <val> ENTIER
◊
```

Macro référencée dans fragment 8.

3.6 Les non terminaux

⟨le type des non-terminaux 10a⟩ ≡

```
%type <expr> E
%type <expr> T
%type <expr> F
```

◊

Macro référencée dans fragment 8.

3.7 La grammaire avec les actions

⟨les règles syntaxiques sem 10b⟩ ≡

⟨expressions 10c⟩

⟨termes 11a⟩

⟨facteurs 11b⟩

◊

Macro référencée dans fragment 8.

3.7.1 Expressions

Ceci est à la fois l'axiome et un non-terminal apparaissant en partie droite de règle. Il faut donc affecter à la fois `$$` et la variable globale racine.

⟨expressions 10c⟩ ≡

```
E : E '+' T
     {
            fprintf(synt, "reduction r1\n");
                fprintf(synt, "   creation noeud d'addition\n");
                racine = creer_expr(A_EXP_OP_BIN, '+', $1, $3, -1);
                $$ = racine;
         }
    |E '-' T
     {
            fprintf(synt, "reduction r2\n");
                fprintf(synt, "   creation noeud de soustraction\n");
                racine = creer_expr(A_EXP_OP_BIN, '-', $1, $3, -1);
                $$ = racine;
      }

       | T
            {
            fprintf(synt, "reduction r3\n");
                fprintf(synt, "   transfert du terme vers l'expression\n");
                racine = $1;
                $$ = racine;
      }
;
```

◊

Macro référencée dans fragment 10b.

3.7.2 Termes

⟨termes 11a⟩ ≡

```
T : T '*' F
     {
            fprintf(synt, "reduction r4\n");
                fprintf(synt, "   creation noeud de multiplication\n");
                $$ = creer_expr(A_EXP_OP_BIN, '*', $1, $3, -1);
         }
    |T '/' F
         {
            fprintf(synt, "reduction r5\n");
                fprintf(synt, "   creation noeud de division\n");
                $$ = creer_expr(A_EXP_OP_BIN, '/', $1, $3, -1);
         }
    | F
     {
            fprintf(synt, "reduction r6\n");
                fprintf(synt, "   transfert du facteur vers le terme\n");
                $$ = $1;
         }
    ;
```

◊

Macro référencée dans fragment 10b.

3.7.3 Facteurs

⟨facteurs 11b⟩ ≡

```
F : ENTIER
         {
            fprintf(synt, "reduction r7\n");
                fprintf(synt, "   creation du noeud pour la valeur %d\n", $1);
                $$ = creer_expr(A_EXP_VAL_NUM, ' ', NULL,NULL,$1);
         }
  | IDENT
         {
            fprintf(synt, "reduction r8\n");
                fprintf(synt, "   creation du noeud pour l'ident nr %d, de nom : %s\n",
                        $1, tab_symb[$1]);
                $$ = creer_expr(A_EXP_ID, ' ', NULL,NULL,$1);
         }
| '(' E ')'
    {
            fprintf(synt, "reduction r9\n");
                fprintf(synt, "   transfert l'arbre expression sur facteur, sans parenthe
                $$ = $2;
     }
         ;

```

◊

Macro référencée dans fragment 10b.

3.8 Code C en fin de fichier sem

⟨code C en fin de fichier yacc sem 12⟩ ≡

```
/* Affiche le message d'erreur msg                                    */
void yyerror(char *msg)
{
 if(!msg || *msg=='\0')
  fprintf(stderr, "Erreur ligne %d\n", yylineno);
 else
  fprintf(stderr, "Erreur ligne %d : %s\n", yylineno, msg);
}
```

◊

Macro référencée dans fragment 8.