

I53- Compilation et théorie des langages

Licence 3 - 2020/2021

Calculatrice en Bison

Le programme `bison` est un générateur automatique d'analyseur syntaxique. Pour construire un analyseur syntaxique avec la commande `bison` on édite un fichier de suffixe `.y`, disons fichier `mini.y` incluant la définition d'un analyseur lexical, obligatoirement identifié par `yylex()`, la description des lexèmes (tokens) et des règles. Les actions sémantiques écrites en langage C entre accolades calculent l'attribut du père, représenté par `$$`, en fonction des attributs des fils de gauche à droite représentés par `$1`, `$2`, ...

Un programme `bison` se décompose en 4 parties: prologue, définitions, règles et épilogue. Les zones du code correspondantes sont séparées dans le code source par les balises `%{`, `%}`, `%%`, `%%`.

```
%{
Prologue : declarations pour le compilateur C
%}
Definitions : definition des lexemes
%%
Grammaire : production et regles syntaxiques
%%
Epilogue : corps de la fonction main()
```

1. Recopier le programme `calculatrice.y` suivant:

```
%{
#include <stdio.h>
#include <ctype.h>

int yylex();
int yyerror(char *msg);

int yylval ;
%}
%token NB PLUS FIN
%left PLUS
%start PROG

%%

PROG : EXP FIN { printf("%d", $1 ); return 1;}
EXP  : NB { $$ = $1 }
      | EXP PLUS EXP { $$ = $1 + $3 ;}
      ;
%%

int main( void ) {
    yyparse() ;
}

int yylex( ) { a vous de jouer }
```

```
int yyerror(char *msg) {
    printf("\n%s\n", msg);
}
```

Le main du programme est réduit à sa plus simple expression : un appel de l'analyseur syntaxique `yyparse()` qui utilise implicitement la variable `yylval` et l'analyseur lexical `yylex()`, une fonction qui renvoie la valeur du lexème (token) courant dont l'attribut est transmis par la variable `yylval`. Les erreurs de syntaxes provoquent l'appel de la fonction `yyerror()`. Ci-dessous, un exemple d'analyseur lexical rudimentaire.

```
int yylex( ) {
    int car ;
    car = getchar() ;
    if ( car == EOF ) return 0 ;
    if ( isdigit(car) ) {
        yylval = car - '0';
        return NB;
    }
    switch ( car ) {
        case '+' : return PLUS;
        case '\n': return FIN;
    }
}
```

2. Exécuter les commandes:

```
bison -v calculatrice.y
gcc -Wall calculatrice.tab.c -o calc.exe
./calc.exe
3+2+6
11
```

3. Améliorer l'analyseur lexical pour filtrer les espaces et tabulations.
4. Modifier la grammaire pour gérer les autres opérations : multiplication, division, soustraction.
5. Intégrer une fonction `int myexp(int x, int n)` pour gérer les exponentiations, l'opérateur sera représenté par deux étoiles.
6. Gérez les parenthèses.
7. Modifier `yylex()` pour manipuler des nombres de plusieurs chiffres.
8. Ajoutez un token `MEM` dont les attributs possibles seront A, B, ... dans l'analyseur lexical. Incorporer les règles syntaxiques pour traiter les expressions contenant des variables.
9. Ajouter un lexème `AFFECT`. Incorporer les règles syntaxiques pour reconnaître les affectations `MEM AFFECT EXP`.