**TP compilateurs 1 :**

**Prise en main de PLY**

4. Analyseur syntaxique

**Analyse Syntaxique (parser)**

expression -> expression addOP expression

expression -> expression mulOP expression

expression -> nombre

def p\_expression\_op(p):

"""expression : expression ADD\_OP expression

| expression MUL\_OP expression"""

p[0] = operations[p[2]](p[1],p[3])

**Partie opératoire**

p[0] : résultat correspond à la partie gauche de la règle

p[1] : premier élément de la partie de droite (ici expression qui est un nombre)

p[2] : deuxième élément de la partie de droite (ici Opérateur)

p[3] : troisième élément de la partie de droite (ici expression qui est un nombre)

On pourrait écrire

p[0] = p[1]+p[3]

**Exercice 0 : Programme parser1.py**

''' TP1: 4 Analyseur syntaxique

Exercice 0:

expression -> expression ADD\_OP expression

| expression MUL\_OP expression

| NUMBER

'''

import ply.yacc as yacc

from lex2 import tokens

operations = {

'+' : lambda x,y: x+y,

'-' : lambda x,y: x-y,

'\*' : lambda x,y: x\*y,

'/' : lambda x,y: x/y,

}

def p\_expression\_op(p):

"""expression : expression ADD\_OP expression

| expression MUL\_OP expression"""

p[0] = operations[p[2]](p[1],p[3])

def p\_expression\_num(p):

"""expression : NUMBER"""

p[0] = p[1]

def p\_error(p):

print ("Syntax error in line %d" % p.lineno)

yacc.errok()

precedence = (

('left', 'ADD\_OP'),

('left', 'MUL\_OP'),

)

yacc.yacc(outputdir='generated')

if \_\_name\_\_ == "\_\_main\_\_":

import sys

prog = open(sys.argv[1]).read()

result = yacc.parse(prog, debug=1)

print (result)

*Fichier prog0.txt*

10+5

**>python parser1.py prog0.txt**

Generating LALR tables

WARNING: 4 shift/reduce conflicts

**15.0**

Résultat correct : Mais quatre conflits shift/reduce qui correspondent aux 4 opérateurs.

*Fichier prog1.txt*

10\*5+3

**>python parser1.py prog1.txt** (sans mode debug : yacc.parse(prog))

Generating LALR tables

**WARNING: 4 shift/reduce conflicts**

**80.0**

OUPS !!!!

**>python parser1.py prog1.txt** (avec mode debug : yacc.parse(prog, **debug=1**)

Generating LALR tables

**WARNING: 4 shift/reduce conflicts**

PLY: PARSE DEBUG START

State : 0

Stack : . LexToken(NUMBER,10.0,1,0)

Action : **Shift** and goto state 2

State : 2

Stack : NUMBER . LexToken(MUL\_OP,'\*',1,2)

Action : **Reduce rule [expression -> NUMBER]** with [10.0] and goto state 1

Result : <float @ 0x23022d0> (10.0)

State : 1

Stack : expression . LexToken(MUL\_OP,'\*',1,2)

Action : **Shift** and goto state 4

State : 4

Stack : expression MUL\_OP . LexToken(NUMBER,5.0,1,3)

Action : **Shift** and goto state 2

State : 2

Stack : expression MUL\_OP NUMBER . LexToken(ADD\_OP,'+',1,4)

Action : **Reduce rule [expression -> NUMBER]** with [5.0] and goto state 6

Result : <float @ 0x2b9590> (5.0)

State : 6

Stack : expression MUL\_OP expression . LexToken(ADD\_OP,'+',1,4)

Action : **Shift** and goto state 3

State : 3

Stack : expression MUL\_OP expression ADD\_OP . LexToken(NUMBER,3.0,1,5)

Action : **Shift** and goto state 2

State : 2

Stack : expression MUL\_OP expression ADD\_OP NUMBER . $end

Action : **Reduce rule [expression -> NUMBER]** with [3.0] and goto state 5

Result : <float @ 0x2b9e40> (3.0)

State : 5

Stack : expression MUL\_OP expression ADD\_OP expression . $end

Action : **Reduce rule [expression -> expression ADD\_OP expression]** with [5.0,'+', 3.0] and goto state 6

Result : <float @ 0x2b9500> (8.0)

State : 6

Stack : expression MUL\_OP expression . $end

Action : **Reduce rule [expression -> expression MUL\_OP expression]** with [10.0,'\*',8.0] and goto state 1

Result : <float @ 0x2b9590> (80.0)

State : 1

Stack : expression . $end

Done : Returning <float @ 0x2b9590> (80.0)

PLY: PARSE DEBUG END

**80.0**

**Solution : Introduire des priorités**

precedence = (

('left', 'PLUS', 'MINUS'),

('left', 'TIMES', 'DIVIDE'),

)

L’ordre d’apparition dans la liste définit la priorité de bas en haut

left /right définit l’associativité (exécution de droite à gauche ou de gauche à droite)

Fonctionne correctement -> 53

**>python parser1.py prog1.txt** (10\*5+3)

Generating LALR tables

PLY: PARSE DEBUG START

State : 0

Stack : . LexToken(NUMBER,10.0,1,0)

Action : Shift and goto state 2

State : 2

Stack : NUMBER . LexToken(MUL\_OP,'\*',1,2)

Action : Reduce rule [expression -> NUMBER] with [10.0] and goto state 1

Result : <float @ 0x20422d0> (10.0)

State : 1

Stack : expression . LexToken(MUL\_OP,'\*',1,2)

Action : Shift and goto state 4

State : 4

Stack : expression MUL\_OP . LexToken(NUMBER,5.0,1,3)

Action : Shift and goto state 2

State : 2

Stack : expression MUL\_OP NUMBER . LexToken(ADD\_OP,'+',1,4)

Action : Reduce rule [expression -> NUMBER] with [5.0] and goto state 6

Result : <float @ 0x2b9590> (5.0)

State : 6

Stack : expression MUL\_OP expression . LexToken(ADD\_OP,'+',1,4)

Action : Reduce rule [expression -> expression MUL\_OP expression] with [10.0,'\*'

,5.0] and goto state 1

Result : <float @ 0x2b9e40> (50.0)

State : 1

Stack : expression . LexToken(ADD\_OP,'+',1,4)

Action : Shift and goto state 3

State : 3

Stack : expression ADD\_OP . LexToken(NUMBER,3.0,1,5)

Action : Shift and goto state 2

State : 2

Stack : expression ADD\_OP NUMBER . $end

Action : Reduce rule [expression -> NUMBER] with [3.0] and goto state 5

Result : <float @ 0x2b9520> (3.0)

State : 5

Stack : expression ADD\_OP expression . $end

Action : Reduce rule [expression -> expression ADD\_OP expression] with [50.0,'+'

,3.0] and goto state 1

Result : <float @ 0x20422d0> (53.0)

State : 1

Stack : expression . $end

Done : Returning <float @ 0x20422d0> (53.0)

PLY: PARSE DEBUG END

53.0

**Ajout de parenthèses**

expression -> expression addOP expression

expression -> expression mulOP expression

expression -> ( expression )

expression -> -nombre

Modifications :

Côté lexical :

literals = '()'

Coté syntaxique

def p\_expression\_paren(p):

"""expression : '(' expression ')' """

p[0] = p[2]

(3\*((2)+1)) -> 9.0 OK

**Ajout de l’opérateur unaire -**

Différencier l’opérateur « moins » binaire de l’opérateur « moins » unaire. Ce dernier à une plus grande priorité et il son associativité est à droite. Ajouter cet opérateur dans les propriétés

precedence = (

('left', 'PLUS', 'MINUS'),

('left', 'TIMES', 'DIVIDE'),

('right', **'UMINUS'**), # Unary minus operator

)

def p\_minus(p):

"""expression : ADD\_OP expression %prec **UMINUS**"""

p[0] = operations[p[1]](0,p[2])

''' TP1: 4 Analyseur syntaxique

Exercice 0:

expression -> expression ADD\_OP expression

| expression MUL\_OP expression

| NUMBER

Ajout des parenthèses

expression -> ( expression )

Ajout des nombres signés (moins unaire)

expression -> - expression

'''

import ply.yacc as yacc

from lex3 import tokens

operations = {

'+' : lambda x,y: x+y,

'-' : lambda x,y: x-y,

'\*' : lambda x,y: x\*y,

'/' : lambda x,y: x/y,

}

def p\_expression\_op(p):

"""expression : expression ADD\_OP expression

| expression MUL\_OP expression"""

p[0] = operations[p[2]](**p[1]**,p[3])

def p\_expression\_num(p):

"""expression : NUMBER"""

p[0] = p[1]

def p\_expression\_paren(p):

"""expression : '(' expression ')' """

p[0] = p[2]

def p\_minus(p):

"""expression : ADD\_OP expression %prec UMINUS"""

p[0] = operations[p[1]](**0**,p[2])

def p\_error(p):

print ("Syntax error in line %d" % p.lineno)

yacc.errok()

precedence = (

('left', 'ADD\_OP'),

('left', 'MUL\_OP'),

('right', 'UMINUS'),

)

yacc.yacc(outputdir='generated')

if \_\_name\_\_ == "\_\_main\_\_":

import sys

prog = open(sys.argv[1]).read()

result = yacc.parse(prog, debug=1)

print (result)

UMINUS n’est pas un mot réservé, on peut choisir ce qu’on veut !

Fonctionne aussi avec l’opérateur unaire « plus »

**-**5**--**5**++**5**-+**5**+-**5 -> -5 OK

**-**5**--**5**++**5**-+**5**+-+**5 -> -5 OK, mais cette grammaire est peut-être un peu trop tolérante