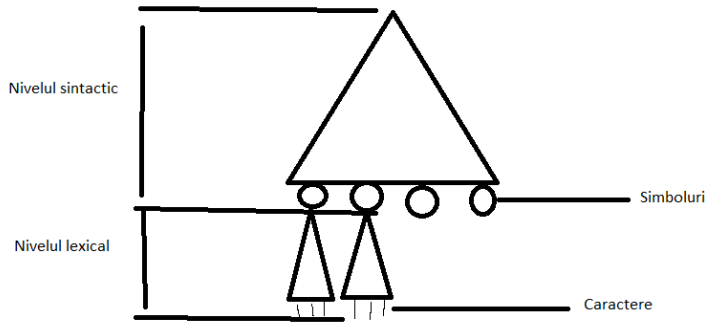
Recapitulare

Thursday, March 26, 2020 11:54 AM

LEX = analizor lexical la nivel de caractere
YACC = parser sintactic la nivel de simboluri (=tokeni)

Simbolurile (tokenii) pot fi:
- Identificatori (variabile, nume de functii)
- Constante
- Cuvinte cheie
- Separatori
- Operatori
- Semne de punctuatie

La fiecare pas in parcurgerea codului, Yacc-ul apeleaza Lex-ul.



-ly genereaza un main care contine functia yyparse(); - apeleaza yacc-ul
-ll genereaza un main care contine functia yylex(); - apeleaza lex-ul

./C_PARSER <input.c >output

dc1.l, dc1.y - efectueaza operatii aritmetice
dc2.l, dc2.y - efectueaza operatii aritmetice, dar interpreteaza si parantezele

Operatiile de SHIFT, REDUCE
Propagarea $ (dolarilor)

Exemplu de set de productii scris teoretic:

```
E   ->   E + E
       | E * E
       | i
```

Scris practic:

```
Expr :   expr '+' expr
       | expr '*' expr
       | INTEGER
       ;
```

Structura unui fisier LEX:

{definitions}
%%
{rules}
%%
{user subroutines}

Exemplu de fisier Lex:

```
%%
[ \t]+$   ;
[ \t]+    printf(" ");

//reducem spatiile mari din input
// lex exemplu.l
// gcc -o EXEMPLU lex.yy.c -ll
// ./EXEMPLU
Reducem      spatiile      din   input.
> Reducem spatiile din input.

+   ->   "unul sau mai multe"
*   ->   "zero sau mai multe"
$   ->   "la final de rand"
Doar  ";" -> nu facem nimic, nici o actiune
```

Input: 1+2
SHIFT = citire de token (pune un simbol pe stiva)
REDUCE = reduce o expresie (partea dreapta a unui set de productii se reduce la partea stanga E+E->E sau i->E)

	STACK	INPUT	VALUE STACK
	empty	8+5*7	empty
Dupa shift	i	+5*7	8
Dupa reduce	E	+5*7	8
Dupa shift	E+	5*7	8_
Dupa shift	E+i	*7	8_5
Dupa reduce	E+E	*7	8_5
Conflict de shift - reduce, facem shift	E+E*	7	8_5_
Dupa shift	E+E*i	empty	8_5_7
Dupa reduce	E+E*E	empty	8_5_7
Dupa reduce	E+E	empty	8_35
Dupa reduce	E	empty	43

Value stack e corespondentul Stack-ului.
(valori) (E, i)

Prioritatea intr-un conflict de shift-reduce e data de precedentele setate de noi (prioritatea creste de sus in jos):

%left '+' '-' -> cel mai putin prioritara
%left '*' '\' -> cea mai prioritara

1+2+3+4 are precedenta definita ca %left '+' ceea ce inseamna ca intr-o expresie cu mai multe operatii de aceeasi importanta, expresia se evalueaza din stanga spre dreapta
Daca era %right '+', expresia 1+2+3+4 se evalua din stanga spre dreapta

Ridicarea la putere are precedenta %right '^', deci o expresie de genul 2^3^4 se evalueaza din dreapta spre stanga

$$ = f($1, $2, $3, …., $n); **Propagarea dolarilor:**

După reduce	E	Empty	43

dreapta spre stanga

$$ = f($1, $2, $3, …., $n);
$$ - output

Propagarea dolarilor:

8+5*7