Secondo Laboratorio di Linguaggi e Computabilità

Esercizio svolto – calcolatrice (file **calc.l** e **calc.y** reperibili nella cartella **materialeLaboratorio2)**
Modificare la grammatica definita in calc.l e calc.y in modo che accetti anche le operazioni modulo (%) e logaritmo in base 10 (log)
N.B. per rendere operativo l'esercizio, assicurarsi che il lexer NON sia autonomo, cioè che abbia la dichiarazione %byaccj e non %standalone

Svolgimento

a. Operazione modulo

Per prima cosa è necessario identificare il token che rappresenta l'operazione modulo, cioè il simbolo '%'.
Una volta identificato il simbolo da gestire (%) è necessario:

- definire una produzione che permetta al parser di riconoscere l'operazione modulo, modificando (nel file .y) la produzione 'exp' aggiungendo una parte destra che gestisca quest'operazione:

```
exp:     NUM                { $$ = $1; }
       | exp '+' exp        { $$ = $1 + $3; }
       | exp '-' exp        { $$ = $1 - $3; }
       | exp '*' exp        { $$ = $1 * $3; }
       | exp '/' exp        { $$ = $1 / $3; }
       | '-' exp  %prec NEG { $$ = -$2; }
       | exp '^' exp        { $$ = Math.pow($1, $3); }
       | '(' exp ')'        { $$ = $2; }
       ;
```

➔

```
exp:     NUM                { $$ = $1; }
       | exp '+' exp        { $$ = $1 + $3; }
       | exp '-' exp        { $$ = $1 - $3; }
       | exp '*' exp        { $$ = $1 * $3; }
       | exp '/' exp        { $$ = $1 / $3; }
       | exp '%' exp        { $$ = $1 % $3; }
       | '-' exp  %prec NEG { $$ = -$2; }
       | exp '^' exp        { $$ = Math.pow($1, $3); }
       | '(' exp ')'        { $$ = $2; }
       ;
```

- modificare la definizione di precedenza tra gli operatori "%left '*' '/'" in "%left '*' '/' '%' ", nella parte delle definizioni del file .y
- definire una regola lessicale che permetta al lexer di riconoscere il carattere '%' ed identificarlo come token (nel file .l); in particolare, modificare la regola sotto il commento 'operators', aggiungendo un pattern che riconosca il carattere '%':

```
/* operators */
"+" |
"-" |
"*" |
"/" |
"^" |
"(" |
")"     { return (int) yycharat(0); }
```

➔

```
/* operators */
"+" |
"-" |
"*" |
"/" |
"^" |
"%" |
"(" |
")"     { return (int) yycharat(0); }
```

b. Logaritmo in base 10 (log)

Per prima cosa è necessario identificare il token che rappresenta l'operazione logaritmo in base 10, che chiameremo LOG10, e dichiararlo nella sezione delle dichiarazioni del file .y con il comando "%token LOG10"; inoltre si può assegnare al token una associatività a sinistra con il comando "%left LOG10".
Una volta identificato il simbolo da gestire, è necessario:

- definire una produzione che permetta al parser di riconoscere l'operazione logaritmo (file .y); in particolare, bisogna modificare la produzione 'exp', aggiungendo una parte destra che gestisca l'operazione modulo:

  LOG10 exp { $$ = Math.log10($2); }

- definire una regola lessicale che permetta al lexer di riconoscere i caratteri che identificarlo il token LOG10 (file .l); in particolare aggiungere nelle definizioni, una macro che permetta di riconoscere la sequenza di caratteri "log10" oppure "LOG10":

  LOG10 = ("log10" | "LOG10")

- aggiungere una regola lessicale che informi il parser che è stato riconosciuto il token LOG10:

  {LOG10} { return Parser.LOG10; }

Esercizi da svolgere

Modificare calc.* in modo che

a. Sia trattato anche il caso del logaritmo naturale (ln)

b. Nelle produzioni sia restituita (in $$) una stringa in cui l'espressione NON sia calcolata ma sia riscritta in modo "ben impaginato", ovvero siano eliminati e/o aggiunti gli spazi necessari tra numeri e simboli. Ad esempio,

  13+45 /28
  dovrà essere restituita nella forma
  13 + 45 / 28
  Suggerimento. utilizzare sval anziché dval.

c. Dopo aver svolto i punti a. e b., fare in modo che la stampa delle espressioni ricevute in input per gli operatori binari sia in modalità "postfissa". Ad esempio per l'operazione somma, originariamente trattata in calc.y con la regola grammaticale e corrispondente azione

  | exp '+' exp { $$ = $1 + $3; }
  poi modificata per produrre una stringa da stampare e non un calcolo interno al Parser in:
  | exp '+' exp { $$ = $1 + " + " + $3; }
  deve essere modificata in modo che la stringa prodotta sia con il simbolo "+" in coda, ovvero:
  Esempio 1: input: 12+43 output: 11 43 +
  Esempio 2: (12+36)*25 output: (12 36 +) 25 *

Secondo Laboratorio di Linguaggi e Computabilità

ESERCIZIO DA SVOLGERE E CONSEGNARE - infDyck.l e infDyck.y (reperibili su sito e.learning)
Guardare nel dettaglio il codice e, utilizzando JFlex e BYACC/J, provare il funzionamento degli esempi **infDyck.l** e i**nfDyck.y** che implementano un analizzatore sintattico e parser che accettano righe con parentesi bilanciate, alternate a lettere minuscole.
In particolare, notare che il lexer ritorna in "yylval" la stringa su cui avviene il match e il token corrispondente con return.
Notare inoltre la corrispondenza tra i token: definiti in .y e usati in .l

Esercizio: modificare "infDyck.l" e "infDyck.y" in modo che

- siano gestite le parentesi quadre in modo analogo alle parantesi tonde
- il testo libero (corrispondente al token SKIP nella grammatica) possa essere presente in tre forme:
  - con sole lettere minuscole (caso già presente), oppure
  - con due lettere maiuscole come inizio, seguite da minuscole, oppure
  - con sequenza iniziale di uno o più caratteri ':', seguiti da lettere minuscole
  - con cifre appartenenti al proprio numero di matricola
- se i caratteri ':' sono in numero pari, vengano stampate solo le lettere minuscole senza i ':' e se invece i caratteri ':' sono in numero dispari, vengano stampate le lettere minuscole precedute da un solo ':'
- infine: se il testo libero inizia con due maiuscole, stampare anche un messaggio "Err:" quando queste sono diverse tra loro; indipendentemente dal fatto che le due lettere maiuscole siano uguali o diverse, stampare le sole lettere minuscole che le seguono

Seguono alcuni esempi di pattern e relativo comportamento richiesto:

- in "(abc())" ho il testo libero "abc": lo stampo come "abc" (caso già implementato)
- se ho testo libero "::::abc": stampo "abc"
- se ho testo libero ":::abc": stampo ":abc"
- se ho testo libero "ABxyz": stampo "Err:" e "xyz"
- se ho testo libero "AAxyz": stampo “xyz"
- se ho testo libero "AA817101": stampo “817101” (assumendo che la mia matricola sia 817101)

Consegna: consegnare in un unico file "cognomeMATR.zip" i due file con nome "esercizioInfDyck.l" e "esercizioInfDyck.y"