Cuprins

1 Ragel si Lemon **2**

1.1 Ragel 2

1.2 Lemon 2

1.3 Instalare 3

1.3.1 Instalare MinGW 3

1.3.2 Instalare Ragel 3

1.3.3 Instalare Lemon 4

1.4 Utilizare 4

1.4.1 Utilizare Ragel 4

1.4.2 Utilizare Lemon 5

1.5 Executare 6

1.6 Bibliografie 1

Documentatie Compilator

Puscasu Constantin-Alexandru
Ilie Florian-Cristian

January 23, 2019

1 Ragel si Lemon

1.1 Ragel

Ragel este un lexer/tokenizer avanand reputatia de a fi simplu si de a produce rezultate consistente, producand un cod rapid si eficient.
In compilatorul nostru, el este folosit pentru a citi cod dintr-un fisier text, cod care mai apoi este impartit in tokeni care vor fi trimisi mai departe parserului. In cazul nostru Lemon.
Cateva caracteristici si diferente a parserului Ragel comparat cu Lex/Flex:

- Foloseste o sintaxa diferita si mai minimalista.
- Comenziile nu se mai impart in 3 sectiuni si se scriu intr-un singur bloc de cod.
- Nu este dependent de limbaj, putand genera cod C, C++ si ASM.
- Putem crea State Machine-uri, generand un format compatibil cu Graphviz pentru a le reprezenta grafic.

1.2 Lemon

Lemon este un LARL parser generator pentru limbajul C. El functioneaza similar lui 'Yacc' si 'Bison' si este folosit pentru generarea arborelui sintactic.
Cateva diferente a parserului Lemon comparat cu Yacc/Bison:

- Este mai nou.
- Foloseste o sintaxa diferita pentru gramatica care este mai putin vulnerabila la erori.
- Este thread-safe si re-entrant.

- Posibiliatatea definirii unui destructor pentru expresiile non-terminale.
- In Yacc/Bison parserul apeleaza generatorul de tokeni. In Lemon, generatorul de tokeni apeleaza parserul.
- Lemon nu foloseste variabile globale in timp ce in Yacc/Bison se folosesc.
- Lemon genereaza 3 fisiere in momentul build-ului:
  - *.c ce contine codul C cu implementarea parserului.
  - *.h ce contine valoarea numerica atribuit fiecarui token/simbol terminal.
  - *.out ce contine un raport cu privire la starile folosite de automat.
- Optional, Lemon, poate genera un fisier cu fiecare pas pe care il realizeaza automatul cu scopul de a urmari si evita eventualele erori. (a se vedea documentatia)

1.3 Instalare

Inainte de a putea folosi Ragel si Lemon sunt necesare cateva instalari si configuratii:

1.3.1 Instalare MinGW

Se va instala MinGW fiind necesara consola **msys** pentru a executa comenziile de instalare si rulare a celor doua programe. Dupa descarcare se va instala tinandu-se cond ca optiunea **msys-base-bin** sa fie bifata.

Link descarcare: `https://osdn.net/projects/mingw/releases/`

1.3.2 Instalare Ragel

Pentru a putea executa comenziile specifice lexerului Ragel va trebuii sa-l instalam folosind consola **msys** precizata in subcapitolul precedent astfel:

Link descarcare: `https://github.com/bnoordhuis/ragel`

Se va naviga in folderul unde este localizat **msys.bat** (de obicei in `C:\MinGW\msys\1.0`) si il vom executa.

Comenzile se vor executa pe rand in consola:

```
1    cd "Locatia folder-ului Ragel descarcat mai sus"
2    ./configure --prefix=PREFIX
3    make
4    make install
```

1.3.3 Instalare Lemon

Pentru a putea executa comenziile specifice parserului Lemon va trebuii sa-l instalam folosind consola **msys** precizata in subcapitolul precedent astfel:

Link descarcare: `https://github.com/sergev/lemon-parser-generator`

Link descarcare surse: `http://www.hwaci.com/sw/lemon/`

Se va naviga in folderul unde este localizat **msys.bat** (de obicei in `C:\MinGW\msys\1.0`) si il vom executa.

Comenzile se vor executa pe rand in consola:

```
cd "Locatia folder-ului Lemon descarcat mai sus"
./configure
make
make install
```

Dupa instalarea Lemon-ului vom crea un executabil din cele doua fisiere sursa ale sale, se va naviga catre folderul celor doua fisiere cu ajutorul consolei **msys** si se va executa comanda:

```
gcc -o lem lemon.c
```

Care va crea un fisier denumit **lem.exe** si va fi necesar in pasul de executare al programului.

1.4 Utilizare

Utilizarea celor doua tool-uri se poate face separat sau impreuna.

1.4.1 Utilizare Ragel

Pentru a realiza tokenizarea symbolurilor noastre vom scrie comenziile specifice Ragel intr-un fisier cu extensia ***.rl**

Sintaxa Ragel din acest fisier arata astfel:

```
%%{
  machine Lexer;
  write data;

  main := |*

  [1-9][0-9]*  =>    { ret = NUMBER; fbreak;};
  [0]          =>    { ret = NUMBER; fbreak;};
  [a-z]        =>    { ret = IDEN; fbreak;};

  '-' =>    { ret = SUB; fbreak;};
  '/' =>    { ret = DIV; fbreak;};
  '=' =>    { ret = ASSIGN; fbreak;};
  '+' =>    { ret = ADD; fbreak;};
  '*' =>    { ret = MUL; fbreak;};

  [ \t\n]+ ;
  *|;
}%%
```

Unde *Lexer* este numele lexerului nostru si *main* este functia noastra principala unde vom scrie denumirea token-ilor si expresiile regulate specifice acestora.

De asemenea variabila *ret* este o variabila declarata mai sus in codul C in care memoram tokenul atribuit simbolului curent.

Vom avea de asemenea doua variabile in care tinem cont de inceputul si respectiv de sfarsitul simbololui. Aceste variabile trebuiesc initializate in momentul in care ii trimitem programului un simbol pentru a fi evaluat astfel:

```
  p = p_;          // p_ fiind inceputul simbolului nostru
  pe = pe_;        // pe_ fiind sfarsitul simbolului nostru
  %% write init;  // comanda necesara pentru initializarea celor doua pozitii
```

Dupa ce am setat inceputul si sfarsitul simbolului in programul nostru, vom semnala tokenizarea astfel:

```
    %% write exec;
```

Si intr-un final vom avea (in cazul nostru) salvat in variabila *ret* tokenul rezultat prin evaluarea simbolului respectiv.

Se va executa tokenizarea pentru fiecare simbol iar fiecare token impreuna cu valoarea acestuia (daca exista) vor fi trimise parserului Lemon.

1.4.2 Utilizare Lemon

Lemon va primi tokenul si valoarea trimisa de catre Ragel si va construi arborele sintactic pe baza acestora. De retinut faptul ca Lemon primeste token-ii, nu invers!

Lemon are numeroase comenzi si functii utile insa ne vom limita doar la cateva in

momentul de fata, documentatia Lemon are lista completa cu acestea.

Dintre acestea vom folosi in momentul de fata trei functii:

```
void *ParseAlloc();      /* folosita pentru alocarea unui nou parser
                         (existand posibilitatea definirii mai multor
                         parsere ce se executa in paralel) */

void Parse(parser, token, value);  /* functia principala ce trimite
                                      token-ul curent pentru a fi evaluat */

void ParseFree();        /* folosita pentru a elibera memoria ocupata de
                        un anumit parser */
```

Pentru a realiza parsarea vom scrie comenziile specifice Lemon intr-un fisier cu extensia spefica analizei sintactice ***.y** cum ar fi:

```
expr(A) ::= NUMBER(B).                { A = B; }
expr(A) ::= expr(B) ADD expr(C).      { A = B + C; }
expr(A) ::= expr(B) SUB expr(C).      { A = B - C; }
expr(A) ::= expr(B) MUL expr(C).      { A = B * C; }
expr(A) ::= expr(B) DIV expr(C).      { A = B / C; }
```

Unde ne vom specifica gramatica noastra. Lemon va executa codul C dintre acolade dupa reducerea expresiei respective.

Important este faptul ca spre deosebire de Yacc/Bison nu avem confuzia numerotarii si avem o declarare mai lizibila folosind denumiri sugestive, cum putea vedea din exemplul urmator:

Yacc/Bison:

```
expr:
    | expr ADD expr        { $$ = $1 + $2; }
    ;
```

Lemon:

```
expr(A) ::= expr(B) ADD expr(C).      { A = B + C; }
```

1.5 Executare

Dupa ce am instalat, setat si scris comenzile prezentate in capitolele precedente vom folosi urmatoarele comenzi pentru a executa tokenizarea in Ragel iar mai apoi parsarea in Lemon.
Mai intai se va intra in consola **msys.bat** si se va naviga pana in folderul unde am salvat fisierele ***rl** si ***y**. (in cazul nostru le vom denumi *lexer.rl* si *parser.y*).

Se vor executa pe rand comenziile:

```
ragel lexer.rl
lem parser.y
gcc -o run lexer.c
```

Ca rezultat al acestor comenzi o sa fie un fisier **run.exe** pe care il putem executa, astfel executandu-se generatorul de tokeni impreuna cu parser-ul creat.

1.6 Bibliografie

- MinGW:

  `https://osdn.net/projects/mingw/releases/`

- Lemon:

  `http://www.hwaci.com/sw/lemon/`

  `https://github.com/sergev/lemon-parser-generator`

- Ragel:

  `http://www.colm.net/open-source/ragel/`

  `https://github.com/bnoordhuis/ragel`