Limbaje Formale si Translatoare
Laborator 1

Anca Mărginean, Camelia Pintea, Vanessa Mercea, Sonia Cibu, Cenan Călin

Februarie 2021

1 Scopul lucrarii

1. Studiul generatorului de analizoare lexicale LEX
2. Efectuarea unor exercitii practice de utilizare a acestuia

2 Notiuni teoretice

Analiza lexicala converteste un program sursa dintr-un sir de caractere intr-o secventa de simboluri semnificative din punct de vedere semantic. Aceste simboluri, care sunt iesirea analizorului lexical, constituie un limbaj intermediar. Analizorul lexical are functii precum contorizare linii, de generare liste, eliminare spatii sau comentarii.

Gramatica limbajului sursa, G, poate fi partitionata in subgramatici G0, G1, G2, ..., Gn.

- Gramaticile G1, G2, ..., Gn descriu structura simbolurilor fundamentale: constantele, identificatorii si delimitatorii (cuvinte cheie, caractere speciale sau combinatii de caractere speciale)
- Gramatica G0 descrie structura limbajului in functie de simbolurile fundamentale.

Structura si reprezentarea simbolurilor fundamentale poate fi modificata fara a afecta puterea limbajului. Interactiunea dintre analizorul lexical si analizorul sintactic se face prin intermediul unui fisier de "lexeme". Lexemul este entitatea extrasa de analizorul lexical, furnizata analizorului sintactic.

2.1 LEX

LEX-ul e un generator de programe pentru procesarea lexicala a unor fluxuri de intrare. El accepta o specificatie de nivel inalt, orientata pe problema potrivirii sirurilor de caractere, si produce un program intr-un limbaj de programare universal.

Sursa de intrare pentru LEX e un fisier ce contine expresii regulate si fragmente de program asociate acestora. La iesire rezulta o functie numita yylex(), care poate citi un flux de intrare, partitionand-ul in siruri de caractere ce se potrivesc cu expresiile regulate date. Pe masura ce fiecare expresie apare in intrarea programului scris de LEX, este executat fragmentul de cod corespunzator.

Functia yylex() este implementarea unui automat finit determinist. LEX-ul poate fi folosit singur pentru transformari simple, sau impreuna cu alte instrumente pentru analiza si statistica la nivel lexical. Functia yylex() poate fi vazuta ca un preprocesor lexical ce recunoaste simboluri terminale,

sau tokeni. Ea poate fi combinata cu un generator de analizoare sintactice (parsers), cum e de exemplu YACC-ul. YACC genereaza o functie, yyparse(), ce atribuie pieselor lexicale o structura, conform unei gramatici independente de context. Procesul analizei sintactice, sau procesul de parsare, este urmatorul:

Intrare → yylex() → yyparse() → Intrare analizata sintactic.

Automatul generat este interpretat, nu compilat, pentru a salva spatiu. Rezultatul este un analizor rapid. Mai concret, timpul cerut de un program generat de LEX pentru a recunoaste si a partitiona un flux de intrare e proportional cu lungimea intrarii.

Numarul de reguli LEX, sau complexitatea lor nu sunt importante in determinarea vitezei, cat timp regulile ce includ 'context inainte' nu cer o cantitate semnificativa de rescanare. Dimensiunea automatului, sau a programului generat de LEX, creste cu numarul si complexitatea regulilor.

Fragmentele de cod utilizator sunt structurate ca si clauze case intr-un switch. Utilizatorul e liber sa insereze declaratii si instructiuni aditionale in rutina ce contine actiunile, sau sa adauge subrutine in afara acestor tuple de tip rutina-actiune.

2.2 Sursa LEX

Formatul general al sursei LEX este:

```
{ definitii }
%%
{ reguli }
%%
{ subrutine utilizator }
```

Definitiile si subrutinele utilizator pot lipsi. Al doilea %% e optional, dar primul e obligatoriu pentru a marca inceputul regulilor. Cea mai scurta sursa pentru LEX arata astfel:

```
%% (nici o definitie, nici o regula).
```

Aceasta sursa e translatata intr-un program care copiaza intrarea la iesire. Regulile reprezinta deciziile de control ale utilizatorului. Ele formeaza o tabela in care coloana stanga contine expresii regulate, iar coloana dreapta contine actiuni, fragmente de program ce trebuie executate cand expresiile sunt recunoscute. Regula de mai jos cauta sirul integer in fluxul de intrare si tipareste "found keyword INT" oricand acesta e gasit. sfarsitul expresiei e indicat de primul "blanc" sau "tab".

```
integer  printf("found keyword INT");
```

2.3 Expresii regulate LEX

2.3.1 Caractere text si caractere operator

O expresie regulata specifica un set de siruri de potrivit. Ea contine:

- caractere text ce corespund caracterelor din sirurile de cautat (litere si cifre)
- caractere operator

  ```
  " \ [ ] ^ - ? . * + | ( ) $ / { } % < >
  ```

- caractere ce specifica repetitii, alegeri si alte proprietati

Cand se doreste utilizarea lor ca si caractere text, operatorii trebuie "protejati". Operatorul " se poate folosi pentru a trata orice caractere continute intre o pereche de ghilimele ca si text. Un caracter operator mai poate fi protejat precedandu-l de un caracter backslash ca in:

```
xyz\+\+, care este echivalent cu xyz"++" si cu "xyz++".
```

In mod normal, "blanc"-urile si "tab"-urile incheie o regula. Exemple:

```
\n \t \b
```

Pentru a introduce caracterul \, se foloseste \\.

2.3.2 Clase de caractere

Perechea de operatori [ ], defineste clase de caractere. De exemplu [abc] se potriveste cu un caracter care poate fi a, b sau c. Intre paranteze drepte semnificatiile celor mai multi operatori sunt ignorate. Exista insa operatorii speciali: \, - si ^.

- Caracterul - indica domenii. De exemplu : [a-z0-9<>] se potriveste cu orice caracter din multimea continand literele mici , cifrele si parantezele unghiulare. Domeniile pot fi date in orice ordine. Daca se doreste includerea caracterului - intr-o clasa de caractere, el trebuie sa fie primul sau ultimul: [-+0-9] se potriveste cu toate cifrele si cele doua semne + -.
- Operatorul ^trebuie sa apara primul dupa [. El indica faptul ca sirul rezultat trebuie sa fie complementat fata de setul de caractere al calculatorului. Astfel:
  - Regula [^abc] se potriveste cu toate caracterele cu exceptia lui a, b si c.
  - Regula [^a-zA-Z] reprezinta orice caracter care nu e o litera .
- Caracterul \furnizeaza "escape"-urile uzuale intre clasele de caractere dintre paranteze drepte. Secventele escape in octal sunt legale, dar neportabile:
  - Regula [\40-\176] se potriveste cu toate caracterele din setul ASCII, de la octal 40 (blanc) pana la octal 176 (tilda).

2.3.3 Caractere arbitrare

Caracterul operator . se potriveste cu orice caracter exceptand "newline".

2.3.4 Expresii optionale

Operatorul ? indica optionalitatea unui element. Astfel: ab?c se potriveste cu ac sau cu abc ('b' fiind optional).

2.3.5 Expresii repetate

Repetitiile sunt indicate de operatorii * si +. De exemplu:

- a* reprezinta zero sau oricate aparitii ale lui a
- a+ reprezinta una sau oricate aparitii ale lui a
- Regula [a-z]+ reprezinta orice sir de litere mici (mai putin sirul vid)
- Regula [A-Za-z][A-Za-z0-9]* reprezinta toate sirurile alfanumerice cu un caracter alfabetic in fata.

2.3.6 Alternare si grupare

Operatorul | indica alternare: (ab|cd) se potriveste fie cu ab, fie cu cd. De notat ca parantezele sunt folosite pentru grupare, cu toate ca ele nu sunt necesare la nivelul exterior unde ab|cd este suficient.

Parantezele pot fi folosite pentru expresii mai complicate, ca de exemplu (ab|cd+)?(ef)* , care se potriveste cu siruri ca abefef, efefef,cdef, cddd, dar nu si cu abc, abcd sau abcdef.

2.3.7 Dependente de context

LEX-ul recunoaste si o anumita cantitate de context inconjurator.

- Daca primul caracter dintr-o expresie este ^, ea va fi recunoscuta numai la inceputul unei linii.
- Daca ultimul caracter este $, ea va fi potrivita numai cand este urmata de un "newline".
- Un operator special, /, indica context "urma" (ce urmeaza in continuare). Astfel, ab/cd se potriveste cu sirul ab numai daca e urmat de cd, iar ab$ e identic cu ab/\n.
- Contextul stanga e tratat in LEX prin conditii de start. Daca o regula trebuie sa fie executata numai cand automatul LEX e in conditia de start x, regula trebuie prefixata de $< x >$, folosind operatorul <>. Daca luam "a fi la inceputul liniei" ca fiind conditia de start ONE, atunci operatorul ^ar fi echivalent cu $< ONE >$.

2.3.8 Repetitii si definitii

Operatorii { } specifica repetitii sau expandari de definitii. De exemplu: digit , cauta un sir predefinit numit digit si il expandeaza in punctul respectiv, iar a{1,5} cauta una pana la cinci aparitii ale lui a. Un % initial e necesar pentru a separa segmentele sursa LEX.

2.4 Actiuni Lex

La identificarea unei expresii regulate in intrare programul generat de LEX executa actiunea corespunzatoare. Exista o actiune implicita, si anume copierea intrarii la iesire, actiune ce e efectuata si asupra tuturor sirurilor ce nu sunt potrivite cu nici o expresie. Astfel, daca se doreste sa nu scape nimic nefiltrat in iesire, trebuie sa se furnizeze reguli care sa recunoasca totul.

- Deoarece se poate considera ca actiunile sunt ceea ce se face in loc de a copia intrarea la iesire, in general, o regula care doar copiaza poate fi omisa.
- O regula des intalnita este [\t\n], care elimina spatiile, tab-urile si newline-urile.
- Caracterul | indica faptul ca actiunea pentru regula respectiva e definita la urmatoarea regula.

Exemplul precedent, [\t\n], poate fi scris si astfel, rezultatul fiind acelasi:

```
" "     |
"\t"    |
"\n"    ;
```

In actiuni mai complexe, se doreste adesea sa se stie sirul curent potrivit cu o expresie. LEX-ul lasa acest text intr-un tablou extern de caractere numit yytext. Astfel, regula de mai jos va tipari sirul din yytext:

```
[a-z]+   printf("\%s",yytext);
```

Aceasta actiune poate fi scrisa ca ECHO:

```
[a-z]+    ECHO;
```

Uneori e convenabil sa se cunoasca sfarsitul sirului gasit. LEX-ul furnizeaza un contor, yyleng, al numarului de caractere potrivite. Pentru a contoriza atat numarul de cuvinte, cit si numarul de caractere din cuvintele de la intrare, se poate scrie regula:

```
[a-zA-Z]+ {words++; chars+=yyleng;}
```

Regula aduna in chars numarul de caractere din cuvintele recunoscute, numarate in words. Ultimul caracter din sirul potrivit poate fi accesat prin yytext[yyleng-1].

Ocazional, o actiune LEX poate decide ca o regula nu a recunoscut sirul de caractere corect. Pentru aceasta situatie sunt furnizate doua rutine, yyless si yymore, discutate mai in detaliu in Laboratorul 2.

LEX-ul permite accesul la rutinele de I/O ce le foloseste. Ele sint:

- input() - returneaza urmatorul caracter din intrare
- output(c) - scrie caracterul c la iesire
- unput(c) - pune caracterul c inapoi in fluxul de intrare spre a fi citit mai tirziu de input()

Implicit, aceste rutine sunt furnizate ca macrodefinitii, dar utilizatorul le poate inlocui cu versiuni proprii. Aceste rutine definesc relatia dintre fisierele externe si caracterele interne, si trebuie sa fie pastrate toate sau modificate toate, pentru consistenta.

Setul de caractere folosit trebuie sa fie consistent in toate rutinele. Un zero citit la intrare trebuie sa insemne "End_of_file", iar relatia dintre unput() si input() trebuie pastrata, altfel "look-ahead"-ul LEX-ului nu va functiona. Acesta e necesar pentru a potrivi o expresie care e prefixul altei expresii. Nu are loc mereu, insa orice regula terminata in +,*,? sau $, sau continand /, implica look ahead.

O rutina LEX pe care utilizatorul o poate redefini e yywrap(), apelata cand se atinge un "end_of_file". Daca yywrap() returneaza 1 (implicit), LEX-ul continua cu terminarea normala, la sfarsitul intrarii. Uneori, e convenabil sa se preia intrare in continuare, de la o noua sursa. In acest caz, utilizatorul trebuie sa furnizeze un yywrap() care furnizeaza noua intrare, returneaza 0 si informeaza LEX-ul sa continue procesarea. Aceasta rutina e un loc convenabil de a tipari tabele la sfarsitul programului.

2.4.1 Reguli ambigue

LEX-ul poate manipula si specificatii ambigue. Cind mai mult de o expresie se poate potrivi cu intrarea curenta, LEX-ul decide dupa urmatoarele criterii:

1. alege cea mai lunga potrivire,
2. intre regulile care se potrivesc, si au acelasi numar de caractere, alege regula ce apare prima in tabela de reguli.

Sa presupunem ca avem regulile

```
%%
integer       actiune pentru cuvint cheie...;
[a-z]+        actiune pentru identificator...;
```

Daca intrarea e *integers*, ea e tratata ca un identificator, deoarece $[a-z]+$ se potriveste cu 8 caractere iar integer doar cu 7. Daca intrarea e *integer*, ambele reguli se potrivesc cu 7 caractere, si intrarea e tratata drept cuvint cheie, conform regulii ce apare prima.

Observatie Principiul preferintei pentru cea mai lunga potrivire face regulile ce contin operatorii . si $*$ relativ periculoase. De exemplu: $'.*'$ pare o buna maniera de a specifica un sir de caractere intre apostroafe, insa practic e o invitatie pentru program sa citeasca mult inainte, cautind un apostrof indepartat. Daca la intrare avem:

$$'first' \textit{ quoted string here, } 'second' \textit{ here}$$

, expresia de mai sus se va potrivi cu *'first' quoted string here, 'second'*, ceea ce probabil ca nu e ceea ce s-a dorit. O regula mai buna e: $'[\hat{}'\backslash n]*'$, care pentru intrarea de mai sus se va opri dupa $'first'$.

Consecintele unor erori ca acestea sint restrinse de faptul ca operatorul . (punct) nu se va potrivi cu "newline". Astfel, expresii ca .* se opresc pe linia curenta. Nu incercati sa "invingeti" aceasta cu expresii ca [.]+ sau echivalente, deoarece programul generat de LEX va incerca sa citeasca intregul fisier de intrare si vor rezulta depasiri de buffer intern.

2.5 Utilizare

Exista urmatoarele etape in realizarea unui program executabil pornind de la o sursa LEX:

1. Optional: Instalarea LEX

   ```
   (cu comanda: sudo apt install flex)
   ```

2. Translatarea sursei de catre LEX intr-un program generat in limbajul gazda. Daca nu este specificat un nume pentru fisierul de iesire, se va genera predefinit un fisier cu numele lex.yy.c.

   ```
   (cu comanda: lex [-o nume_fis.c] nume_fis_sursa.l);
   ```

3. Optional: Vizualizarea tuturor comenzilor LEX.

   ```
   (cu comanda: lex -h);
   ```

4. Compilarea programului generat, de obicei impreuna cu o biblioteca de subrutine LEX.

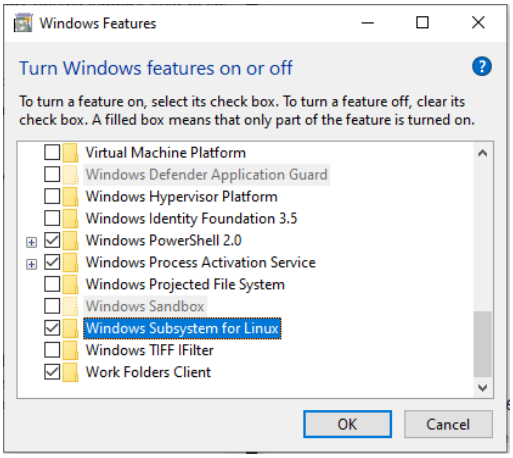
   ```
   (cu comanda: gcc nume_fis.c -ll);
   ```

3 Desfasurarea Lucrarii

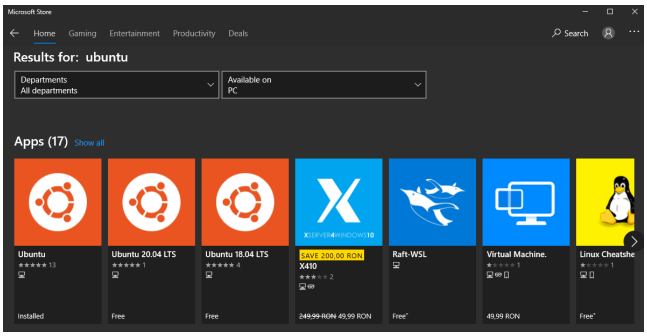
3.1 Pregatirea Mediului

Pentru translatarea surselor lex si compilarea programelor generate, se recomanda utilizarea sistemului de operare Linux. Astfel, se propune fie rularea surselor pe Linux, fie utilizarea unei masini virtuale pentru Linux, fie utilizarea subsistemului Linux pentru Windows. Pentru cea din urma pasii sunt descrisi mai departe.

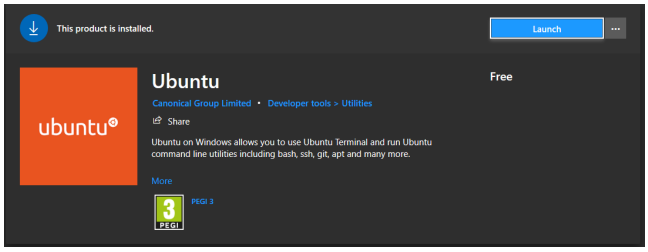
1. **Preconditii: Veti avea nevoie de Windows 10, x86.**
2. **Activarea subsistemului Windows pentru Linux**
   - Accesati Control Panel → Programs → Turn Windows Features on or off
   - Dati drepturi de administrator actiunii (daca e cazul)
   - In lista de programe afisata bifati Windows Subsystem for Linux si apasati OK



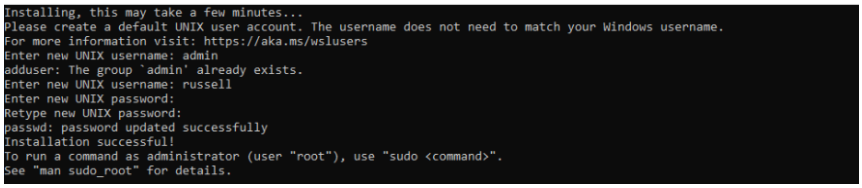
   - Restartati calculatorul la finalul instalarii
3. **Instalarea unei distributii Linux**
   - Deschideti aplicatia Microsoft Store si selectati Search in coltul dreapta sus al aplicatiei, scrieti Ubuntu, iar dintre aplicatiile indicate, selectati Ubuntu



- Odata ajunsi la pagina aplicatiei Ubuntu apasati Get
- Dupa ce instalarea e completa, apasati Launch. Va aparea o consola, si vor mai dura cateva minute pana la finalizarea instalarii. Cand Ubuntu e gata de folosire, va trebui sa va creati un cont.

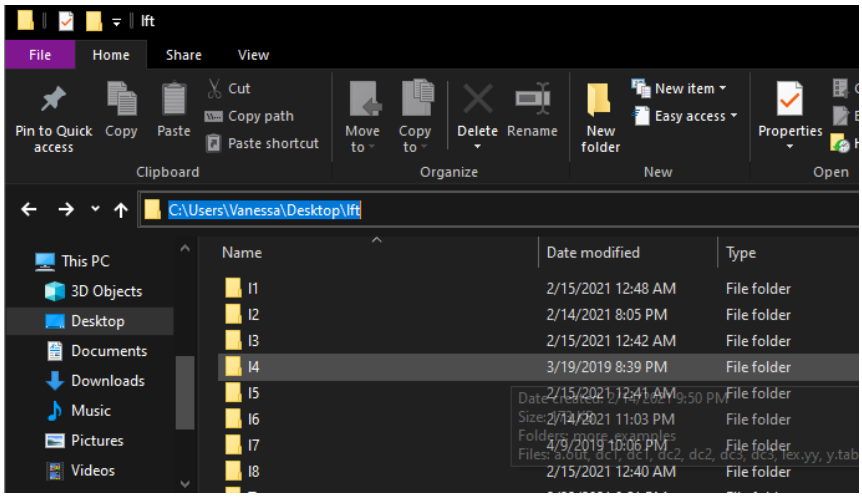


- Introduceti username-ul, parola, si confirmarea parolei. Utilizatorul nou creat va fi administratorul subsistemului Linux si va fi logat automat la fiecare deschidere a terminalului Ubuntu, sau sudo. Acum veti putea folosi Linux pe masinile Windows fara necesitatea utilizarii unei masini virtuale

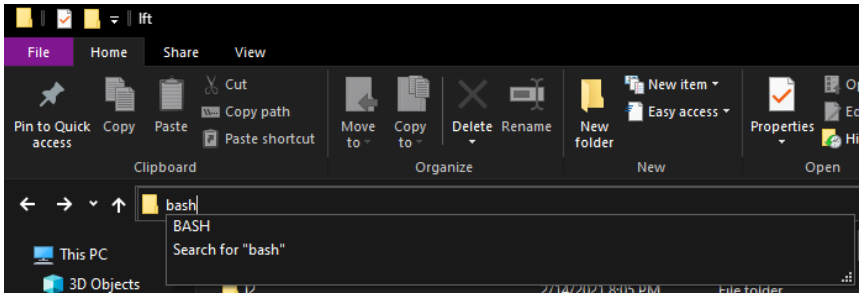


4. **Deschidere Terminal**

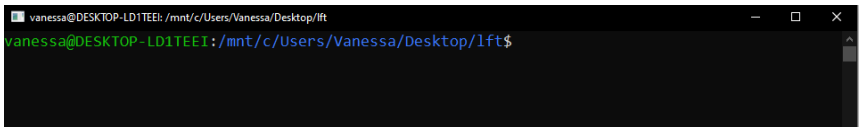
- Pentru deschiderea terminalului Linux in calea dorita unde se afla fisierele pe disk, deschideti aceasta locatie folosind File Explorer
- Selectati calea curenta in File Explorer



- Inlocuiti calea curenta cu comanda **bash** si apasati tasta enter



- Acum puteti utiliza terminalul Linux



3.2 Generator de Analizoare Lexicale Lex

1. Se vor executa practic exemplele prezentate in sectiunea 1.2 a lucrarii, si se va analiza codul generat de LEX.

2. Se vor testa urmatoarele exemple:

   (a) Eliminarea spatiilor si "tab"-urilor de la sfarsitul liniilor unui text de intrare si inlocuirea celorlalte spatii si "tab"-uri cu un singur spatiu

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

   (b) Identificarea in sirul de intrare a numerelor intregi si reale in conventia FORTRAN:

```
D [0-9]
E [DEde][-+]?{D}+
%%
{D}+                                printf("integer");
{D}+"."{D}*({E})?                   |
{D}*"."{D}+({E})?                   |
{D}+{E}                             printf("real");
%%
```

   (c) Realizarea unei histograme dupa lungimea cuvintelor din textul de intrare, unde un cuvint e definit ca un sir de litere:

```
int lung[99];
 %%
 [a-z]+      lung[yyleng]++;
 .           |
 \n          ;
 %%
 yywrap()
 {
     int i;
         printf("Lungime Nr. cuvinte\n");
         for(i=0; i<99; i++)
             if(lung[i] > 0)
                 printf("%6d  %7d\n", i, lung[i]);
             return(1);
  }
```

4 Intrebari si Dezvoltari

1. Sa se propuna si alte exemple sau imbunatatiri la cele prezentate.

2. Implementati un exemplu care returneaza numarul de caractere dintr-un fisier (exceptand caracterul newline).

3. Implementati un exemplu care numara comentariile marcate cu // la inceput de linie dupa oricat de multe caractere spatiu sau taburi.

4. Implementati un exemplu care sa copieze intrarea la iesire adunand 7 la numerele cuprinse in intervalul [7-12], si inmultind cu 10 numerele negative mai mici decat -100.

5. Implementatin un exemplu care identifica stringuri marcate cu ". Un string poate contine si ghilimele daca sunt precedate de \. De exemplu, daca la intrare avem *aaa "string cu escaped \" "bb*, stringul identificat e "*string cu escaped \" "*