Teme laborator MMLP

~~~~~~~~~~~~~~~~~~~

Semestrul I (doua teme):

========================

Tema 1: Implementarea unui analizor lexical pentru un limbaj de programare

======= uzual (Pascal, C, etc., la alegerea studentului).

Tema este aceeasi pentru toti.

Se vor respecta urmatoarele indicatii de implementare:

- analizorul lexical va folosi un automat finit determinist, care va avea

cate o stare finala pentru fiecare tip de token;

pentru identificarea tokenilor, el va consuma pe rand caractere din

fisierul sursa si va face tranzitii in automat, pana se blocheaza

(deci nu pana ajunge prima data intr-o stare finala);

daca s-a blocat intr-o stare nefinala, atunci s-a intalnit o eroare

lexicala si analiza se opreste; daca s-a blocat intr-o stare finala,

se genereaza tokenul respectiv (tipul sau este indicat de starea finala

iar valoarea este sirul consumat pentru a ajunge din starea initiala

acolo), dupa care se trece automat in starea initiala (fara a consuma

caractere din sursa) si se reia analiza consumand caractere si facand

tranzitii;

faptul ca nu ne oprim prima data cand ajungem intr-o stare finala este

motivat de dorinta de a identifica de fiecare data cel mai lung token

care se poate forma de la pozitia curenta incolo; intr-adevar, un token

poate fi prefix pentru alt token si daca ne-am opri la prima stare finala

intalnita, atunci la intalnirea lui 1.2, cu 1 s-ar trece de la prima

tranzitie in starea finala corespunzatoare tipului de token "constanta

intreaga", apoi sarind iar in starea initiala si consumand .2 s-ar

ajunge in starea finala pentru "constanta flotanta", generandu-se per

total doi tokeni (o constanta intreaga si o constanta flotanta) si nu

unul singur - constanta flotanta 1.2;

atentie ca incercarea de a identifica tokenii folosind functii simple

predefinite si nu un automat poate conduce la erori; de exemplu daca

folosim doar functia C "strtok" pentru a identifica ca tokeni siruri

de caractere nealbe delimitate de caractere albe, atunci pentru stringul

Pascal "abc def ghi" s-ar genera trei tokeni: "abc, def si ghi" si nu un

singur token - constanta string "abc def ghi", sau s-ar deduce ca def

este un token identificator, ceea ce gresit (nu in orice context in care

apare def este un identificator);

- optional, se poate implementa urmatorul comportament: daca automatul s-a

blocat intr-o stare nefinala si pe traseul parcurs de la starea initiala

pana acolo s-a intalnit macar o stare finala, sa se intoarca pe traseu

pana la ultima stare finala intalnita (introducand inapoi in banda de

intrare caracterele consumate la tranzitii) si sa se genereze tokenul

corespunzator; doar daca pe traseul respectiv nu s-a intalnit nici o

stare finala atunci sa se genereze eroarea;

de exemplu in Pascal la intalnirea lui 1ep (constanta flotanta cu

exponent gresit formata) automatul va consuma 1e indreptandu-se catre

starea finala corespunzatoare constantelor flotante, iar cu p s-ar bloca,

deoarece nu exista constante flotante care sa inceapa cu 1ep si nici

alte tipuri de token (de exemplu nu putem avea un identificator, deoarece

incepe cu o cifra); atunci, in loc sa se genereze o eroare, automatul

va introduce inapoi pe banda de intrare pe e si se va intoarce in starea

finala in care a ajuns prin consumarea lui 1 - starea corespunzatoare

constantelor intregi; dupa ce va emite tokenul constanta intreaga, va

sari in starea initiala si consumand ep va ajunge in starea finala pentru

identificatori; per total se va genera deci tokenul constanta intreaga 1,

apoi tokenul identificator ep;

- conceptul de token va fi implementat unitar, sub forma unui articol

(record, structura) sau obiect;

el va avea cel putin urmatorii membri-data:

\* un membru ce indica tipul tokenului;

\* un membru ce indica valoarea tokenului;

de exemplu in programul Pascal: "var x : integer; begin end." al doilea

token are tipul "identificator" si valoarea "x";

- intr-un token:

\* membrul referitor la tip ve fi intreg (se vor face conventii gen:

0=identificator, 1=constanta intreaga, etc.);

\* membrul referitor la valoare nu va fi string, pentru a nu exista

situatia ca la un moment dat memoria sa fie incarcata cu foarte

multi tokeni avand aceeasi valoare; se va proceda astfel:

analizorul va crea la fiecare noua analiza un tabel de stringuri in

care va stoca valorile distincte ale tuturor tokenilor intalniti la

analiza respectiva; de fiecare data cand se va identifica un nou

token, valoarea acestuia va fi cautata in tabel; daca va fi gasita,

membrul-valoare al tokenului generat va contine pozitia in tabel a

sirului respectiv; daca nu va fi gasita, ea se va adauga la sfarsitul

tabelului iar membrul-valoare al tokenului generat va contine, la fel,

pozitia respectiva;

deci membrul-valoare al tokenului va fi tot un intreg sau un pointer;

- se pot face diverse conventii privind tipurile de tokeni, de exemplu:

\* fiecare operator va fi un tip de token distinct (avand deci o unica

valoare posibila, care nu mai trebuie memorata separat) sau toti

operatorii vor forma un singur tip de token (ale carui valor posibile

sunt operatorii respectivi) - de exemplu "+" sa genereze tokenul cu

tipul "operator" si valoarea "+";

\* fiecare cuvant cheie va fi un tip de token distinct (avand deci o unica

valoare posibila, care nu mai trebuie memorata separat) sau toate

cuvintele cheie vor forma un singur tip de token (ale carui valori

posibile sunt cuvintele respective) - de exemplu "begin" sa genereze

tokenul cu tipul "cuvant cheie" si valoarea "begin"; in ultimul caz,

analizorul va avea in plus un tabel cu toate cuvintele cheie posibile

iar membrul-valoare al tokenului generat va reprezenta pozitia in

acest tabel a valorii respective;

mai mult, pentru recunoasterea tokenilor-cuvinte cheie se poate folosi

starea finala asociata tipului de token "identificator" si o cautare

in tabelul cu cuvinte cheie al analizorului - deci daca automatul se

blocheaza in starea finala asociata tipului de token "identificator",

atunci va cauta sirul consumat in tabelul cu cuvinte cheie si: daca-l

gaseste, va genera tokenul cu tipul "cuvant cheie" si valoarea data

de sirul consumat, iar daca nu-l gaseste, va genera tokenul cu tipul

"identificator" si valoarea data de acelasi sir consumat;

- analizorul lexical insusi va fi implementat unitar sub forma unui articol

sau (preferabil) obiect; acesta va contine toate informatiile care

descriu analizorul lexical respectiv (automatul, tabela de stringuri si

cea de cuvinte cheie, referinta la fisierul sursa curent analizat,

numarul caracterului curent in acest fisier, etc.);

daca analizorul lexical este implementat ca un obiect, acesta va oferi

cel putin o metoda publica pe care o vom numi in continuare "gettoken"

si care la fiecare apelare returneaza un nou token din fisierul analizat;

daca analizorul lexical este implementat ca un articol, se va scrie o

functie "gettoken" care primeste ca parametru analizorul si returneaza

un nou token ca mai sus;

- programul supus analize lexicale va fi scris intr-un fisier de intrare,

iar tokenii rezultati intr-un fisier de iesire;

in programul principal se vor face initializari (de exemplu citirea

numelor celor doua fisiere), apoi intr-un ciclu, la fiecare iteratie, se

va apela o data gettoken apoi tokenul rezultat se va scrie in fisierul

destinatie; atentie ca prelucrarea tokenului rezultat (scrierea lui in

fisierul destinatie) se face in programul principal, nu intr-o metoda a

analizorului lexical - acesta in principiu nu face decat gettoken pentru

a furniza un nou token;

in fisierul destinatie tokenii se vor scrie fiecare pe un rand, indicand

in clar tipul si valoarea (deci prin stringuri, nu prin numere de

ordine); de exemplu pentru programul Pascal "var x : integer; begin end."

am putea obtine:

cuvant cheie - var

identificator - x

...

in acest scop se vor scrie functii/metode prin care sa putem obtine

stringurile din tabelele analizorului corespunzatoare pozitiilor stocate

in tokenii returnati de gettoken;

- analizorul lexical va ignora spatiile albe din fisierul sursa (siruri

maximale de caractere albe si comentarii);

astfel, pentru programul Pascal "var {abc} {def}x : integer; begin end.",

primele doua apeluri ale lui gettoken vor returna tokenii:

cuvant cheie - var

identificator - x

si nu:

cuvant cheie - var

spatiu - {abc} {def}

acest mod de lucru este motivat de faptul ca programul inaintea analizei

este un simplu sir de caractere, in care prezenta spatiilor este necesara

pentru a ajuta analizorul sa identifice tokenii, iar dupa analiza este

un sir de tokeni,in care deci tokenii sunt deja identificati si separati;

pentru identificarea spatiilor automatul poate folosi o stare finala

distincta, dar daca va constata ca s-a blocat in ea sa nu furnizeze un

nou token ci sa sara automat in starea initiala pentru identificarea

tokenului urmator;

- in cazul intalnirii unei erori lexicale in fisierul analizat, gettoken

va returna un token de eroare, al carui membru-tip va indica tipul erorii

si al carui membru-valoare va indica pozitia in fisier unde se afla

caracterul ce a generat eroarea;

in programul principal, la receptionarea unui astfel de token se va scrie

in fisierul destinatie un mesaj adecvat de eroare si pozitia respectiva,

dupa care analiza se va opri.

Tema 2: Un subiect din lista de mai jos, la alegerea studentului.

======= Tema este individuala, in sensul ca studentii aceleiasi semigrupe

trebuie sa aibe subiecte distincte.

4. Analizor sintactic bazat pe metoda generala top-down.

Programul va citi gramatica si un cuvant si va afisa derivarea sa,

iar in cazul unei erori se va opri si va semnala eroarea respectiva.