Limbaje formale și tehnici de compilare

implementarea analizorului lexical

Implementarea analizorului lexical (ALEX)

Rolul ALEX este să grupeze caracterele de intrare în atomi lexicali (en: tokens). Atomii lexicali sunt unități lexicale indivizibile din punctul de vedere al etapelor ulterioare ale compilatorului (ex: numere, identificatori, cuvinte cheie, …). Totodată vor fi eliminate toate construcțiile care nu mai sunt relevante în etapele ulterioare (comentarii, spații, linii noi). Un atom este compus din:

- **cod** – o constantă ce permite identificarea atomului respectiv. Exemple: INT, ID, STR, ADD. De multe ori când se vorbește despre un atom, de fapt se vorbește despre codul lui.
- **atribute** – informații asociate cum ar fi: caracterele propriu-zise din care este compus un identificator sau un șir de caractere, valoarea numerică pentru constante numerice, linia din fișierul de intrare, …. Dacă unele dintre aceste atribute sunt mutual exclusive (valoarea numerică a unui număr nu se folosește niciodată simultan cu șirul de caractere necesar pentru un identificator), ele se pot grupa într-un *union*, pentru a ocupa mai puțin spațiu în memorie.

În implementarea propusă (**lexer.h**, **lexer.c**) un atom este implementat sub forma unei structuri de date denumită **Token**. Toți atomii extrași din fișierul sursă vor fi păstrați într-o listă simplu înlănțuită. Câmpul **next** indică atomul următor din listă.

Implementarea funcției tokenize

Funcția **tokenize** primește ca parametru un pointer la codul sursă și returnează lista cu toți atomii extrași. Parametrul **pch** (*pointer at current char*) este folosit ca iterator în textul de intrare. Tot algoritmul este conținut într-o buclă din care se iese atunci când se întâlnește terminatorul de șir ('*0*'). Cu această ocazie se adaugă în lista de atomi atomul final END cu rol de terminator și se returnează lista de atomi.

În interiorul buclei se folosește un **switch** care separă atomii după primul caracter. Dacă un atom poate începe cu mai multe caractere (ex: ID, INT), acesta se va testa pe ramura de **default** a **switch**-ului, astfel încât să se poată testa toate posibilitățile pentru primul caracter (ex: folosind **isdigit** pentru a se testa dacă este început de INT sau DOUBLE).

Tot pe ramura de **default** se tratează și cazul în care nu există niciun atom care să înceapă cu caracterul curent. În această situație se generează o eroare. Se pot implementa asemenea teste și în alte situații în care un caracter este necesar pentru atomul respectiv, dar acel caracter lipsește (ex: apostroful de final după o constantă caracter).

După ce s-a început procesarea unui atom (după selecția acestuia conform primului caracter), se consumă caracterele de intrare până la sfârșitul atomului respectiv. În final se folosește funcția **addTk** care adaugă un nou atom cu codul dat în lista de atomi. Totodată **addTk** folosește variabila globală **line** pentru a seta automat în atom linia din fișierul de intrare la care a fost găsit acesta. Variabila **line** este actualizată în **case**-urile pentru '*r*' și '*n*', care sunt concepute în așa fel încât să poată procesa diverși terminatori de linie pentru sistemele de operare mai răspândite. Funcția **addTk** returnează un pointer la atomul adăugat, astfel încât, dacă este necesar, la acesta se mai pot seta și alte câmpuri (ex: câmpul **text** dacă a fost procesat un ID).

Dacă mai mulți atomi încep cu același prefix (ex: {ASSIGN, EQUAL} sau {INT, DOUBLE}), atunci implementarea trebuie să distingă între atomii respectivi. Aceasta se poate realiza parcurgând prima oară prefixul comun, iar apoi testând ce caractere urmează. În funcție de acestea se decide ce atom a fost întâlnit și se continuă cu consumarea restului de caractere ale atomului, dacă este nevoie.

Cuvintele cheie (en: keywords) sunt de fapt tot identificatori (ID), dar ele au un rol specific în limbajul de programare implementat. Din acest motiv, pentru ele se vor folosi coduri de atomi distincte. De exemplu, dacă se întâlnesc caracterele '*while*', se va adăuga atomul WHILE, nu ID. Pentru aceasta, după ce s-a întâlnit un ID, prima oară se verifică dacă acesta este un cuvânt cheie. Dacă da, se adaugă cuvântul cheie respectiv. Altfel, dacă nu este cuvânt cheie, se adaugă ca ID. Pentru ID se vor memora și caracterele constituente, folosind câmpul **text** din atom.

Dacă se întâlnesc constante numerice (ex: 108 sau 3.14), acestea vor trebui convertite la valori numerice, iar valorile depuse în câmpurile corespunzătoare din atomul lexical. De exemplu, valoarea numerică pentru constante DOUBLE va fi depusă în câmpul **d**.

Pentru extragerea unui subșir dintr-un șir de caractere dat se folosește funcția **extract**. Aceasta primește ca parametri doi pointeri care reprezintă începutul subșirului de extras și adresa primului caracter de după subșir. Funcția alocă dinamic memorie pentru subșirul extras, copiază caracterele acestuia din șirul inițial și returnează un pointer la memoria alocată.

Atenție: a se face distincția dintre atomii constante numerice (CHAR, DOUBLE, INT) și atomii care desemnează cuvintele cheie corespunzătoare tipurilor de date respective (TYPE_CHAR, TYPE_DOUBLE, TYPE_INT). De exemplu, **108** este o constantă întreagă (INT), iar **int** este un cuvânt cheie care desemnează un tip de date (TYPE_INT).

Observație: dacă compilatorul se implementează în limbaje de programare care nu permit pointeri (ex: Java, Python etc.), toți pointerii (**pch**, parametrii lui **extract** etc.) se pot înlocui cu indecși în șiruri de caractere. De exemplu, funcția **tokenize** va primi ca parametru un șir de caractere (**String src**), iar în interiorul ei se va folosi o variabilă (**int idx**=0;) cu care se va indexa **src**. În această situație vor trebui modificate testele de sfârșit de șir, astfel încât să nu se depășească conținutul acestuia la niciuna dintre operațiile de indexare.