Laboratorul 2.
Analiză lexicală

Detalii tehnice

Scheletul de laborator este furnizat sub forma unor **proiecte IntelliJ**, **Eclipse** și `Makefile`, cu clasa principală `Test`. Dacă totuși doriți să rulați din **linia de comandă**, utilizând facilitatea predefinită de testare a ANTLR, puteți urma pașii de mai jos.

Configurări:

```
$ export CLASSPATH=".:<path-către-antlr>/antlr-4.8-complete.jar:$CLASSPATH"
$ alias antlr4='java -jar <path-către-antlr>/antlr-4.8-complete.jar'
$ alias grun='java org.antlr.v4.gui.TestRig'
```

Pentru a testa codul:

```
# Generam codul Java pentru analizorul lexical
$ antlr4 CPLangLexer.g4

# Obținem bytecode-ul Java
$ javac CPLangLexer.java

# Rulăm analizorul lexical pe fișierul "program.txt" și
# direcționăm output-ul atât catre fișierul temp.txt
# cât și la consolă
$ grun CPLangLexer tokens -tokens program.txt | tee temp.txt

# Comparăm output-ul dat de program cu referința
$ diff temp.txt reference.txt

# Rulăm analizorul lexical pe input-ul de la consolă.
# Pentru a trimite un EOF: Ctrl+D.
$ grun CPLangLexer tokens -tokens -stdin
```

Exercițiul 0

În cadrul laboratorului, vom utiliza limbajul CPLang. Începeți prin a parcurge manualul acestuia: `https://curs.upb.ro/mod/resource/view.php?id=42324`.

De asemenea, familiarizați-vă cu scheletul de laborator, implementat la cursul anterior, și rulați clasa `Test` pe fișierul `program.txt`. Parcurgeți și comentariile din fișierul `CPLangLexer.g4`.

Obiectivul laboratorului este ca analizorul lexical implementat de voi să recunoască în final toții *token*-ii din fișierul `manual.txt`, care conține secvența de cod din manualul CPLang. Un exemplu de ieșire posibilă a analizorului pentru acest fișier se găsește în `reference.txt`.

Exercițiul 1

Implementați analizorul lexical astfel încât să poată clasifica identificatori (nume de variabile și de funcții), numere întregi, operatori (aritmetici, de comparație, de atribuire), simbolul „`;`” și cuvintele cheie (`if`, `then`, `else`, `fi`, `true`, `false`).

Observații:

- Analizorul lexical va considera toți literalii întregi ca fiind numere pozitive. Semnul „minus” va fi tratat doar ca operator aritmetic.
- Identificatorii încep cu litera mică, și pot conține apoi litere mici, litere mari și *underscore*.

Exercițiul 2

Extindeți analizorul astfel încât să trateze corect numele de tipuri de date, parantezele, acoladele și comentariile pe un singur rând.

Observații:

- Numele de tipuri de date încep mereu cu o literă mare. Verificarea validității tipurilor se va face în pasul de analiză semantică.
- Comentariile vor fi ignorate (acțiunea `skip`).

Exercitiul 3

Implementați tratarea literalilor cu virgula mobilă și a comentariilor bloc imbricate împerecheate.

Observații:

- Comentariile care nu au simbol-pereche (ex: "`/* /* */`") sunt considerate erori lexicale. Prin urmare veți afișa un mesaj la `stderr` dacă întâlniți o astfel de situație. Utilizați pentru aceasta acțiuni introduse direct în specificața lexicală (vezi ca exemplu regula `STRING`) și *token*-ul predefinit `EOF`.
- Valoarea `Float .25` este echivalentă la nivel semantic cu `0.25`. Valoarea `3.` este echivalentă cu `3.0`. Valoarea "`.`" nu reprezintă un literal `Float` valid. Pentru recunoașterea literalilor de forma `.25` va trebui să extindeți regula existentă din schelet.