PROLOG PROJECT
198:314 Principles of Programming Languages
Spring 1999
due: March 25, 1999

In this project, you are asked to write a parser and a parse-tree evaluator for a simple pro-
gramming language. The BNF for the language is given below. The token set includes the usual
identifiers and integers, keywords if, then, fi, print, and simple symbols (, ), ;, +, -, *, .

<program> ::= <stmtlist>
<stmtlist> ::= <statement> <rest>
<rest> ::= ; <statement> <rest> | epsilon
<statement> ::= if ( <expr> ) then <stmtlist> fi |
              <var> = <expr> |
              print ( <var> )
<expr> ::= <var> + <int> | <var> - <int> |
           <var> * <int> | <int>
<var> ::= // *** alpha-numerical variable name
<int> ::= // *** integer constant

The scanner and a skeleton parser and evaluator will be provided to you.

ASSIGNMENT

1. Write a parser for the example language.

2. Write an interpreter (parse-tree evaluator) for the example language.

   Grading will be based on code correctness, code efficiency, and code documentation. The exact
   submission procedure will be posted on the news group.

Parser

You should write your parser as follows. Each production in the grammar, for instance the BNF rule
<non-terminal> ::= ..., will be implemented as a Prolog rule and predicate with three arguments.

<non-terminal>( arg1, arg2, arg3 ) :-

  • arg1 - is a token stream in a Prolog list that begins with a prefix derived from <non-terminal>.
  • arg2 - is the generated parse tree from <non-terminal>.
  • arg3 - is the remaining token stream after recognition of the token stream generated by
    <non-terminal>.

For example,

<stmtlist> ::= <statement> <rest>
<statement> ::= print ( <var> )

is implemented by the rules

1
parse-stmtlist(S, statements(Itree, Rtree), R) :-
    parse-statement(S, Itree, M),
    parse-rest(M, Rtree, R).

parse-statement([print, '(' V ',' ')', I R], print(var(V)), R).

The parse tree is represented as a Prolog term. The project directory contains traces for example programs that show you the suggested form of the parse trees. For example, the program A = 1; print(A) has the parse tree

\[ \text{program(stmtlist(assign(var(A), expr(int(1))), rest(print(var(A)), [])))} \]

Interprete

The interpreter (parse-tree evaluator) takes a parse tree as input and “executes” it. The initial call to the evaluator has the form \( \text{eval(ParseTree, D)} \), where \( \text{ParseTree} \) is the computed tree and \( D \) is a symbol table (environment). The table keeps track of the bindings of program variables to integer values. The table is implemented through the \( \text{lookup} \) predicate as discussed in class. Note that you cannot redefine an identifier in our simple language. The output of the evaluator for the parse tree example above is

\[ \text{print} \ A \rightarrow 1 \]

Handout

The directory \( \sim/\text{uli/cs314/project1} \) on the undergraduate UNIX cluster contains the initial code for your interpreter, i.e., a program skeleton for your parser and parse-tree evaluator. In addition, there are four simple sample programs with their expected outputs. These programs will help you to get started on the project. Your final interpreter implementation has to handle all programs in the language!

Good luck.