TL 16.0 Specification

Introduction

The TL programming language is designed by previous instructor of CS5363, Jeff Von Ronne. It is "Toy Language" derived from a simplified subset of Pascal. It is designed to provide you the experience writing a compiler without getting burdened by all of the complexities and details of a complete, standard programming language.

Outline

Example TL Program

```
program
  var N as int ;
  var SQRT as int ;
begin
  N := readInt ;
  SQRT := 0 ;

  % go until SQRT exceeds the square root of N
  while SQRT * SQRT <= N do
    SQRT := SQRT + 1 ;
  end ;

  SQRT := SQRT - 1 ; % subtract one SQRT is <= sqrt(N)

  writeInt SQRT ;

end
```

Language Overview

Lexical Features

The TL language, is lexically simple. All identifiers start with a letter or underscore, and may be followed by numbers and letters. All keywords are start with lower-case letters or are a symbol. The symbols "(", ")", ":=", ";", "*", "div", "mod", "+", "-", "=", "!=", "<", "<=", ">", ">=" are used.

Data Types

Core TL supports 32-bit integers ("int"), booleans ("bool"). Variables are always declared to be of a particular type.

Operators

TL has several infix binary operators that work on either integer operands. The multiplication "*", division "div", modulus "mod", addition "+", and subtraction "-" produce integer results. The comparison operators (i.e., equals "=", not equal "!=", less than "<", less-than or equal-to "<=", greater than ">", and greater-than or equal-to ">=") all produce boolean results.

Control Structures

TL is a structured programming language. The only control structures supported are "if" and "while" statements. Both take a boolean expression that guards the body of the control structure. In the case of an "if" statement, the statements after the "then" are executed if the expression is true, and the statements after the "else" (if there is one) are executed if the expression is false. In the case of the "while" statement, the loop is exited if the expression false; otherwise if the expression is true, the body will be executed, and then the expression will be re-evaluated.

Assignment

Assignments are a kind of statement rather than a kind of operator. The ":=" keyword is used to separate the left hand side (which is the variable being assigned to) from the right hand side, which is an expression that must be of the same type as the left hand side.

Built-in Procedures

Core TL does not support user-defined functions or procedures, but it does support one built-in procedures "writeInt" that outputs an integer and a new-line to the console (respectively), and one user-defined function, "readInt" that reads an integer from the console. The syntax for these is hard-coded into TL's BNF grammar.

You are encouraged to post additional TL programs (or not-quite TL13 programs) to the Piazza forum.

Syntax

Comments

The first occurrence of the character "%" on a line denotes the start of a comment that extends to the end of that line. For purposes of determining the lexical elements of the source file, the entire comment will be treated as if it were whitespace.

Lexical Elements

All TL lexical elements (a.k.a. tokens) can be separated by spaces, returns, and new lines, and should match one of the categories below.

If the definition of a lexical element is in quotes, then it is meant to match exactly, the contained string. Otherwise, it is a regular expression. Square brackets in regular expressions are used as an abbreviation for matching ranges of letters. For example, [0-9] matches any digit, and [a-zA-Z] matches any English letter in capital or lower case.

Numbers, Literals, and Identifiers:

- num = [1-9][0-9]*|0

  The regular expression allows all natural numbers, but since we are using 32-bit integers. Only 0 through 2147483647 are valid integer constants. Integer constants outside of that range should be flagged as illegal.

- boollit = false|true
- ident = [a-z_A-Z][a-zA-Z0-9]*

Symbols and Operators:

- LP = "("
- RP = ")"
- ASGN = ":="
- SC = ";"
- MULTIPLICATIVE = "*" | "div" | "mod"
- ADDITIVE = "+" | "-"
- COMPARE = "=" | "!=" | "<" | ">" | "<=" | ">="

Keywords:

- IF = "if"
- THEN = "then"
- ELSE = "else"
- BEGIN = "begin"
- END = "end"
- WHILE = "while"
- DO = "do"
- PROGRAM = "program"
- VAR = "var"
- AS = "as"
- INT = "int"

- BOOL = "bool"

Built-in Procedures:

- WRITEINT = "writeInt"
- READINT = "readInt"

BNF Grammar

```
<program> ::= PROGRAM <declarations> BEGIN <statementSequence>
END


<declarations> ::= VAR ident AS <type> SC <declarations>

               | ε


<type> ::= INT | BOOL


<statementSequence> ::= <statement> SC <statementSequence>

                    | ε


<statement> ::= <assignment>

            | <ifStatement>

            | <whileStatement>

            | <writeInt>


<assignment> ::= ident ASGN <expression>

             | ident ASGN READINT
```

```
<ifStatement> ::= IF <expression> THEN <statementSequence> <els
eClause> END


<elseClause> ::= ELSE <statementSequence>

             | ε


<whileStatement> ::= WHILE <expression> DO <statementSequence>
END


<writeInt> ::= WRITEINT <expression>


<expression> ::= <simpleExpression>

             | <simpleExpression> COMPARE <expression>


<simpleExpression> ::= <term> ADDITIVE <simpleExpression>

                   | <term>


<term> ::= <factor> MULTIPLICATIVE <term>

       | <factor>


<factor> ::= ident

         | num

         | boollit

         | LP <simpleExpression> RP
```

Operator Precedence and Associativity

The order of precedence among the operators is:

1. The MULTIPLICATIVE operators.
2. The ADDITIVE operators.
3. The COMPARE operators.

All binary operators are left-associative.

Informal Type Rules

1. The operands of all MULTIPLICATIVE, ADDITIVE, and COMPARE operators must be integers
2. The MULTIPLICATIVE and ADDITIVE operators create an integer result.
3. The COMPARE operators create boolean results.
4. All variables must be declared with a particular type.
5. Each variables may only be declared once.
6. The left-hand of assignment must be a variable, and the right-hand side must be an expression of the variable's type.
7. When used as a value, a variable's type is its declared type.
8. Only integer variables may be assigned the result of readInt.
9. writeInt's expression must evaluate to an integer.
10. The expression guarding if-statements and while-loops must be boolean.
11. The literals "false" and "true" are boolean.
12. The literal numbers are integers.

Informal Semantics

- Only those variables which have been declared can be assigned to or used. Variables must be assigned before being used.

- All binary operators operate on signed integer operands:

  - "x * y" results in the product of x and y.

  - "x div y" which results in the integer quotient of x divided by y.

    The behavior is not defined if y is 0 or if x is the smallest 32-bit negative integer and y is -1.

    - "x mod y" is the results in the remainder of x divided by y when x is non-negative and y is positive. Otherwise, the result is undefined.
    - "x + y" results in the sum of x and y.
    - "x - y" is the difference of y subtracted from x.
    - "x = y" is true if x and y are the same, otherwise it is false.
    - "x != y" is false if x and y are the same, otherwise it is true.
    - "x < y" is true if x is less than y, otherwise it is false.
    - "x > y" is true if x is greater than y, otherwise it is false.
    - "x <= y" is true if x is less than or equal to y, otherwise it is false.
    - "x >= y" is true if x is greater than or equal to y, otherwise it is false.
    - Computations on TL integers should be done using a 32-bit 2's complement representation. Overflowing computations should simply "wrap around" that is the result of all integer operations should be the integer that is not less than -2^{31}−231 , not more than $2^{31}-1$231−1 , and congruent modulus 2^{32}232 with the result of performing the normal mathematical operation on actual mathematical integers ($\mathbb{Z}$Z).
      [This is what MIPS does, so implementing this shouldn't require any extra work on your part.]

- "if" statements evaluate their expression, if the expression is true, then the "then-statements" are executed, if it is fales, the "else-statements" are executed.
- "while" statements first evaluates its expression. If it is false, execution continues after the end of the "while" statement. If it is true, the statements in the body of the "while" loop are executed. After they finish executing, the expression is re-evaluated. As long as the expression is true, the process repeats itself, alternatively evaluating the expression and executing the statements in the body. Once the expression is false, execution continues after the end of the "while" loop.

- "writeInt" evaluates its expression and outputs the result to the console and causes the cursor to move to the beginning of the next line.
- "readInt" reads an integer from the console and updates an integer variable to hold that value.