DReaM

Group 20

April 29, 2022

Introduction

The design of DReaM is a mix of to the database procedural languages like Transact SQL (T-SQL) and Procedural Languages (PL) and main stream programming languages like c++ and java. The main rationally for choosing the mix is to test out the applicability of database procedural languages style coding along with the main stream programming style.

Java is used to tokenize DReaM program at the set off and prolog is used for the rest of the task, i.e., creating and analyzing the parse tree, and generating the semantics.

- *Lexer*: The first component in DReaM language is lexer. It is responsible for taking the input code from the user and producing tokens. The lexer is implemented in java/prolog. The generated tokens will be passed ahead to a parser that will further analyze and verify the tokens.

- *Parser*: The parser is responsible for checking the syntax of the stream of tokens. It is also responsible for providing structural representation of the same. We made use of Definite Clause Grammar for parsing and for generating concrete syntax tree. The parser checks for function, commands, conditional and loops. Finally, the parser supports String, Number and Boolean datatypes.

- *Interpreter*: It is the last component of the language. The interpreter will first take the parse tree, then evaluate it and finally generate the output. We will make use of Java for interpreter.

Design

DReaM programming language is case insensitive and will have the following basic language constructs.

- Datatype
  DReaM supports String, Number and Boolean datatypes. String variables store any literal that conforms to string and defined in double quotes. A boolean variables hold true/false values. Number variables contain integer, float, and double values.

- Variable
  Variables for DReaM can be any name that constructed with the English alphabets (a-z). The structure of declaring variable is <*variable_name*> <*data_type*> and can be declared anywhere in the program.
  Examples:

```
x string, y number, z boolean;
a string;
b number;
c boolean;
```

- Block
  Blocks start with start-stop statements instead of the usual bracket symbols used in many popular languages. A block can be nested several times.
  Example:

```
start
      greeting string;
      greeting = “Hello, world!”;
      print greeting;
stop
```

- Function

  A function contains a set of statement/expressions that perform some logical task in the program. Functions start with the key word function followed by a one or more arguments and *should* return a single value. A function can be used as a program by itself when a single value is needed to return. If multiple values are needed to be returned, then block is a good options. Note that, functions can not be nested and can't call another function.

  Example:

```
function (<variable_name> <data_type>)
start
      …
      return <identifier>;
stop
```

- Key Words

  Key words are predefined strings that are used in our programming language and these cannot be used in some contexts like function names, declaring variables or classes. These key words can be used in the sentences. DReaM keywords include.

print/println	stop
number	start
string	else
boolean	while
for	range
function	in
if	then

Operators

DReaM supports these operators over number and boolean datatypes:

- Mathematical Operators

  Numeric datatype and uses PEMDAS (Parentheses, Exponents[1], Multiplication/ Division, Addition/Subtraction) precedence rules for operations.
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division[2] (/)
  - Modules (@)
  - Incrementer(++)
  - Decrementer(--)

- String Operators

  A string is any number or alphabet that is bounded by double quotes. A concatenation operator can be used to join strings together.
  - Concatenation (#)

1 exponents are to be handled by functions

2 integer division use modules to find the fractions

- Relational Operators

  Relational operators are used to compare numeric expressions and implicitly return a boolean value.
  - Equal (==)
  - Greater than (>)
  - Greater than and equal to (>=)
  - Less than (<)
  - Less than and equal to (=<)
  - Not equal to (!=)

- Boolean Operators

  Boolean operators are applied in boolean expressions and implicitly return a boolean value.
  - And (&&)
  - Or (||)
  - Not (!)
  - Xor (^)

Assignment

The assignment operator in DReaM used to assign values from the left side to the right side of the operator. The following example shows instantiating the number variable to 5.

```
start
      x number;
      x = 5;
stop
```

Loop

DReaM supports three kinds of loop structures:

- `for` loop

  The for loop uses an already instantiated variable to iterate over a block of statements. Example:

```
start
      x number;
      x = 5;
      for (x; x < 10; x = x + 1)
      start
        print x;
      stop;
stop
```

- `while` loop

  The while loop first checks a condition and loops over until the condition fails.
  Example:

```
start
      x number;
      x = 5;
      while (x < 10)
            start
                  x = x + 1;
                  print x;
            stop;
stop
```

- short `for` loop
  Similar to the for loop, short for loop do not require variable initialization and loops over in a range of numbers, inclusive.
  Example:

```
start
      x number;
      x = 5;
      for x in range(5..10)
      start
            print x;
      stop;
stop
```

Conditional Statements

DReaM supports two kinds of conditional constructs.

- If-then-else condition
  The if-then-else evaluate the if expression which returns a boolean value and executes the value in the then condition if true, otherwise the else statements. Note that, if statement can also be used without else statement.
  Example:

```
start
      x number;
      x = 5;
      if (x < 6)
            start
                  x = 1 + x;
            stop;
      else
            start
                  x = 6;
            stop;
stop
```

- Ternary operator
  A ternary operator is a short hand representation of the above if-then-else representations that become handy when the statements in the then and else statement are single expressions.
  Example:

```
start
      x number;
      r string;
      x = 5;
      r = x < 6 ? “smaller than six” : “greater than six”;
stop
```

Print

The print statement can be used to print a literal values of string and number as well as variable values. Example:

```
start
      x, y,sum number;
      sum = x + y;
      print “The sum is ”;
      print sum;
stop
```

Comment

DreaM language supports a multi-line comment with a present symbol, `%  …  %`. The language considers anything between the symbols as a comment and can be placed anywhere in the code.

§