CMU 17-630 Computer Science for Practicing Engineers Fall 2014

Capstone Final Project

Lattanze/Rosso-Llopart RUDI Interpreter Due: See Calendar

## Overview

An interpreter is an application that accepts input in the form of a text file containing high-level instructions. The program will parse each line of text, make sure that it is syntactically correct, and execute each instruction, and maintain overall program state.

The key difference between a compiler and an interpreter is that a compiler converts high-level languages and converts it into machine code that is directly executable by the machine. An interpreter is a virtual machine that will directly execute each instruction.

Note that the Java compiler (and similar compilers) are not compilers by this definition – this technology represents a "third way". The Java compiler takes high level source code and converts it to an intermediate language that is executed by the Java virtual machine. Still, we often refer to the Java compiler as a compiler.

## Assignment Description

Your team will design and build an interpreter for a rudimentary programming language called RUDI. Your program will accept an input file that contains a program written in RUDI and will execute the program. The RUDI language is a structured language and the syntax is described below:

| **Control Structures** | **Description** |
|---|---|
| program | First executable statement. Indicates the beginning of a main RUDI program. |
| end | Last statement in the program. Ends the program. |
| decs […] | Parameter definition block. This is the second executable statement in a RUDI program. This is where all program parameters must be defined. Brackets define the start and end of parameter definition blocks. Parameters may not be defined outside this block. |
| begin | Defines the start of the program code. |
| if (Boolean) then [...] | If and if-then structures. Note the brackets are required for "if" and "else" blocks even for one statement. |
| if (Boolean) then […] else […] | |
| while (Boolean) […] | While loop. Note the brackets are required for execution blocks. |
| stop | Stops program execution. |
| **Type** | **Description** |
| integer | min integer to max integer |
| float | min float to max float |
| string | strings are comprised of any characters, numbers, and must be enclosed in quotes " " |
| **Arithmetic Operations** | **Description** |
| = | Assignment |
| * | Multiplication |

| | |
|---|---|
| / | Division |
| + | Addition |
| - | Subtraction |
| Note that the order of precedence is *, /, +, -. Parenthesis can be used to force order of operation. | |
| **Comparison** | **Description** |
| :eq: | Equal to |
| :ne: | Not equal to |
| :gt: | Greater than |
| :lt: | Less than |
| :ge: | Greater than or equal to |
| :le: | Less than or equal to |
| **Boolean Operators** | **Description** |
| ^ | and |
| \| | or |
| ~ | not |
| **Comments** | **Description** |
| /* | Beginning of comment… |
| */ | End of comments. All text within comment blocks are ignored by the interpreter. |
| **I/O** | **Description** |
| print "…" | Prints the text that is contained within the quotes to the terminal without a return line-feed |
| print cr | Prints a return-line feed to the terminal |
| print variable | Prints the value of the variable to the terminal |
| input variable | Reads input from keyboard into the variable. Waits until value is typed and return (or enter) is pressed. If the variable is an integer and a floating point number is input, it will be rounded off. |
| **Other Considerations** | |
| • An "&" indicates a line continuation.<br>• Syntax is case insensitive.<br>• Your interpreter will support coercion between integer and floating point types.<br>• Use a text editor to write your RUDI programs (I like TextPad, but you can use what you like).<br>• Implement subroutines… the subroutine structure and syntax are described below: | |
| subroutine name (param1, param2,…) | Subroutines are noted with "subroutine" as the first executable statement, "name" is the name of the subroutine, followed by a start parenthesis and a list of parameters). All parameters are passed by reference. That is, any changes made to the parameters in the subprogram will be "seen" back in the calling program. Subroutines can call subroutines. All of the programs and subroutines will be coded in the same source file. |
| decs […] | Subroutines can have a decs section. The parameters declared in the subroutine are only visible within the subroutine. Once the subroutine ends, locally declared variables are destroyed (e g. memory released). The decs |

| | |
|---|---|
| | syntax within the subroutine is identical to that used in the main program. |
| begin | Defines the start of the subroutine code. |
| return | This is the last statement in a subroutine. Returns control back to the calling program. |

## Example

Here is an example program written in RUDI that computes and prints the Factorial for a number entered by a user.

```
/* Factorial Calculator */
program
decs
     integer fact    /* factorial result              */
     integer x       /* intermediate factorial value  */
     integer n       /* input factorial term          */

begin
     print “Enter the factorial term:”
     input n

     if ( n :eq: 0 )
     [
          fact = 1
     ]
     else
     [    x = 1
          fact = 1
          while (n :le: x)
          [
               x = x + 1
               fact = fact * x
          ]
     ]
     print n
     print “! = “
     print fact
end
```

## Operation and Hints

Your system will run on the command line of your computer. Assume that you write a RUDI program named foobar.rud. To execute your program, you will type: *rudi foobar.rud* at the command line. Your program will execute until completion and control will return to the command line. If your program encounters and error, it will print the line number where the error occurred and the line of text (RUDI code) that it encounter. You will also provide a message describing the nature of the error.

Interpreters are challenging – do not wait to start this project. Design your overall system, then think in terms of partitioning your application into data structures that will allow team members to work concurrently in relative isolation (e g. parser, equation solver, stack,…). Success in this assignment is dependent upon designing and building good data structures and services that all team members can use.

## Deliverables

Your deliverables include:

- Your executable interpreter. We will try to run your program as described above. If we can't run your program as described above, you will be penalized.
- Design document that describes the design of your interpreter. Include a description of the overall system design and a description of the detailed design of the ADTs.
- Your source code. Note that we expect well commented, well-structured code that matches your design description.
- Your RUDI test programs.

NOTE: Please be thorough, but as concise as possible in your discussion. Note that grammar, good style, and format counts. Any reasonable formatting style/structure is acceptable.

Provide your documents, API source code, and executables (java and class files), and data files. Put all of these materials (described above) into a zip file and turn it in electronically (blackboard or email). Use Windows file compression or Winzip to compress your files. Include a "readme" file in your archive that explains the contents of your archive. Name your archive as follows: FP+YourFamilyName. So my assignment archive would be named: FP+Lattanze.

## Grading

This assignment is worth 100 points as follows:

Documentation: 30 points. We will evaluate the overall quality of the design document and the readability/understandability of the source code.

Implementation: 70 points. We will evaluate the based on our RUDI test cases.