Kennesaw State University

College of Computing and Software Engineering

Computer Science Department

CS4308 Concepts of Programming Languages, Section 03

Course Project – 3rd Deliverable - Interpreter

Noah Huck | nhuck@students.kennesaw.edu

04 May 2019

Summary

The 3rd deliverable of the course project is an Interpreter for a selected subset of the SCL language. The source files included with this submission comprise a Java implementation of an Interpreter for SCL. The Interpreter itself is mainly implemented in the file called “Parser.java” and the file “Interpreter.java” includes a main method that can be used to demonstrate the function of the program. An included SCL source file called “huck_input.scl” serves as an appropriate input file for the Interpreter. All development was done using NetBeans 8.2, Xcode, and Terminal on MacOS.

Description

When this program is executed, an instance of the Parser class is created and it begins parsing the input file. The parser’s original function involved creating complete statements, and this functionality still exists but can be disabled by setting the Boolean *running* to true. The structure of my original parser involved many methods without return values. Some of the methods, for example the expression parsing method and it’s supporting methods have been changed to have a String return type. The methods that parse for specific commands in the input file also execute those specific commands. Memory in the SCL execution is in the form of an ArrayList which holds Identifier objects. These identifier objects allow values to be stored and associated with an identifier name. Every action that can be called in the SCL subset is emulated by a Java method. To imitate the function of a while loop in the source code, the parser creates a new instance of itself to track back to the beginning of the current while loop so it can be executed again. The program parses the input file until end of file is reached executing as it goes.

The output of the program depends on which options are given at execution time. For option -e, the program executes the SCL input file. For option -p, the program performs the task from deliverable 2, the program parses the input file line by line and generates complete statements. For option -s, the program performs the operation from deliverable 1, the program scans one symbol at a time and returns the location of the symbol and the token value associated with it. Here are sample screenshots of an input file and the corresponding output for execution option:

```
function main is
variables
    define i of type integer
    define max of type integer
    define x of type integer
    define temp of type integer
    define operation of type integer
    define str of type string
begin
    display "This will serve as a test SCL Program"
    input "Please Enter your name: ", str
    set i = 0
    input "Enter a value for integer x: ", x
    display "Your chosen value of x is: ", x

    input "how many times should loop run: ", max

    display "Would you like to add 2 to x, or multiply x by 2 ", max , " times?"
    display "(note: choosing multiplication with a large number of iterations can overflow
the integer data type)"
    input "Enter 0 to use addition, enter non-zero integer to use multiplication: ",
operation

    display "x is: ", x
    while i < max do
        if operation == 0 then
            set temp = x + 2
            display "x is: ", temp
        else
            set temp = x * 2
            display "x is: ", temp
        endif
        set x = temp
        increment i
    endwhile
    display "Final value of x is: ", x
    display "Thank you ", str
endfun main
```

```
Noahs-MBP-3:src noahhuck$ java pkg.Interpreter -e huck_input.scl

Now Executing file huck_input.scl

This will serve as a test SCL Program
Please Enter your name: John Smith

Enter a value for integer x: 43

Your chosen value of x is:  43
how many times should loop run: 13

Would you like to add 2 to x, or multiply x by 2  13  times?
(note: choosing multiplication with a large number of iterations can overflow the integ
er data type)
Enter 0 to use addition, enter non-zero integer to use multiplication: 0

x is:  43
x is:  45
x is:  47
x is:  49
x is:  51
x is:  53
x is:  55
x is:  57
x is:  59
x is:  61
x is:  63
x is:  65
x is:  67
x is:  69
Final value of x is:  69
Thank you  John Smith
```

How to Run

1.) Using the command line, navigate to the directory that contains this report file, the testfile.scl, and the "pkg" directory, using the command: **cd <directory>**
2.) Use the command: **javac ./pkg/*.java** to compile the .java files into .class files
3.) Use the command: **java pkg.Interpreter huck_input.scl** to execute the input file.
4.) If desired options are available to retrieve the Scanner or Parser outputs instead of executing the file. The syntax for this operation is: **java pkg.Interpreter -eps huck_input.scl**
Where "e" is for execute, "p" is for parse", and "s" is for scan

BNF Subset

The following is a list of the grammar statements chosen to make up the subset of SCL used in this scanner:

funct_list → funct_body | funct_list funct_body

funct_body → FUNCTION pother_oper_def

pother_oper_def → pother_oper IS const_var_struct

BEGIN pactions ENDFUN IDENTIFIER

pother_oper → IDENTIFIER DESCRIPTION

const_var_struct → const_dec var_dec struct_dec

const_dec →

| CONSTANTS data_declarations

var_dec → VARIABLES data_declarations

struct_dec →

| STRUCT data_declarations

data_declarations → comp_declare | data_declarations comp_declare

comp_declare → DEFINE data_declaration

data_declaration → IDENTIFIER OF TYPE data_type

data_type → TUNSIGNED | INTEGER | TSTRING

pactions → action_def | pactions action_def

action_def → SET name_ref EQUOP expr

| INPUT name_ref

| DISPLAY pvar_value_list

| INCREMENT name_ref

| DECREMENT name_ref

| IF pcondition THEN pactions opt_else ENDIF

| WHILE pcondition DO pactions ENDWHILE

name_ref → IDENTIFIER

expr → term PLUS term

| term MINUS term

term → punary

| punary STAR punary

| punary DIVOP punary

punary → element

| MINUS element

element → IDENTIFIER

| STRING

pvar_value_list → expr | pvar_value_list COMMA expr

pcondition → expr eq_v expr

eq_v → EQUALS | GREATER THAN | LESS THAN

opt_else →

| ELSE pactions