

ICS 142: Compilers and Interpreters

Prof. Dr. Michael Franz, Fall Quarter 2006

Assignment 2

due at the beginning of class on November 1st

An Interpreter for VerySimPL

The goal of this assignment is to build an interpreter for the programming language *VerySimPL*. You will do this in two steps, using the Scanner you already constructed in Assignment 1. First, you will write a Parser. Second, you will augment the Parser to turn it into an Interpreter.

If you didn't manage to finish Assignment 1, the TA will give you a working Scanner in compiled form (after October 24th). If you did finish Assignment 1, we encourage you to use your own Scanner for this Assignment.

In order to make input and output possible in the interpreter, the language *VerySimPL* has three *predefined identifiers*. They are introduced overleaf. These are not reserved words (terminals of the grammar, like “if” or “else”) but merely identifiers that “are already there when the program starts”. Predefined identifiers can be “shadowed” by program declarations; i.e., if you declare a variable of the same name, it takes precedence over the predefined identifier. Almost all programming languages have predefined identifiers in addition to reserved words.

Your interpreter takes as its input a file name and a sequence of numbers. The file name designates the name of the source program to be interpreted. During interpretation, whenever a call to *InputNum* is encountered, the next number from the command line is used as the user's input. Calls to *OutputNum* and *OutputNewLine* result in writing to the standard output.

The TAs will post a test harness and further instructions on their web page. Please follow these instructions carefully to avoid deduction of points from your solution.

Note: To obtain credit, **your solution must be submitted electronically by the due date**, using the instructions on the TA's web page.

EBNF for VerySimPL

letter = “a” | “b” | ... | “z”.

digit = “0” | “1” | ... | “9”.

relOp = “==” | “!=” | “<” | “<=” | “>” | “>=”.

ident = letter {letter | digit}.

number = digit {digit}.

factor = ident | number | “(“ expression “)” | funcCall .

term = factor { (“*” | “/”) factor }.

expression = term { (“+” | “-”) term }.

relation = expression relOp expression .

assignment = “let” ident “<-” expression.

funcCall = “call” ident [“(“ [expression { “,” expression }] “)”].

ifStatement = “if” relation “then” statSequence [“else” statSequence] “fi”.

statement = assignment | funcCall | ifStatement .

statSequence = statement { “;” statement }.

varDecl = “var” ident { “,” ident } “;” .

computation = “main” [varDecl] “{” statSequence “}” “.” .

Predefined Function

InputNum() read a number from the standard input

Predefined Procedure

OutputNum(x) write a number to the standard output

OutputNewLine() write a carriage return to the standard output

The start symbol of this grammar is “computation”.