

# ICS 142: Compilers and Interpreters

*Prof. Dr. Michael Franz & Dr. Andreas Gal, Fall Quarter 2007*

## Assignment 3

due at the beginning of class on November 15th

### A Compiler for the Programming Language VerySimPL+

Up to this point, we have discussed scanning, parsing, and even evaluation of expressions without any reference to the target architecture. In this assignment, we will now augment the existing scanner and parser with code-generation statements and thereby turn it into a full-fledged *compiler*. To make things a little more interesting, our language has also gained an additional “while” construct (hence the “+” in the name of our language).

As our target architecture, we will use a virtual machine that is almost identical to the “DLX” processor presented by Hennessy and Patterson in their “Computer Architecture” book. This architecture is a classic RISC design; among commercial products, the *MIPS* architecture comes closest to it.

The DLX Processor conveniently has specific instructions for input and output that can be called directly “in-line” by your compiled code. In order to make your compiler simple, you may also use the registers of the processor to simulate an expression stack.

When you have finished implementing the compiler, use it to compile the test programs on the TA’s website. You might also want to create your own test programs or collaborate with other students on developing a suite of such programs for testing specific parts of the compiler (but note that while collaboration on test programs is a good idea, collaboration on the compilers themselves is not and will be considered cheating).

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”.

whileStatement = “while” relation “do” statSequence “od”.

statement = assignment | funcCall | ifStatement | whileStatement .

statSequence = statement { “;” statement }.

varDecl = “var” indent { “,” 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”.*