The D Programming Language Specification

D is a small imperative language. It has 32-bit integer constants, variables, and functions, and a minimal set of statements: assignment, if, while, and return. There are no modules, classes, objects, or external variables. There are no header files or libraries, but there are two predefined functions, get and put, which provide the ability to read and write integer values. Syntactically it is much like C. A program consists of a sequence of function definitions.

Grammar

Here is a grammar for D.

program ::= *function-def* | *program function-def*
function-def ::= **`int`** *id* **( ) {** *statements* **}**
| **`int`** *id* **(** *parameters* **) {** *statements* **}**
| **`int`** *id* **( ) {** *declarations statements* **}**
| **`int`** *id* **(** *parameters* **) {** *declarations statements* **}**
parameters ::= *parameter* | *parameters* **,** *parameter*
parameter ::= **`int`** *id*
declarations ::= *declaration* | *declarations declaration*
declaration ::= **`int`** *id* **;**
statements ::= *statement* | *statements statement*
statement ::= *id* **=** *exp* **;** | **`return`** *exp* **;** | **{** *statements* **}**
| **`if`** **(** *bool-exp* **)** *statement* | **`if`** **(** *bool-exp* **)** *statement* **`else`** *statement*
| **`while`** **(** *bool-exp* **)** *statement*
bool-exp ::= *rel-exp* | **!** (*rel-exp*)
rel-exp ::= *exp* **==** *exp* | *exp* **>** *exp*
exp ::= *term* | *exp* **+** *term* | *exp* **-** *term*
term ::= *factor* | *term* ***** *factor*
factor ::= *id* | *int* | **(** *exp* **)** | *id* **( )** | *id* **(** *exps* **)**
exps ::= *exp* | *exps* **,** *exp*

Language Notes

A program consists of one or more function definitions, all of which must have distinct names, and one of which must be named main. A program is executed by evaluating main().

Comments, blanks, and other whitespace are ignored except as needed to separate adjacent syntactic tokens. A comment begins with the token // and continues to the end of the line.

There are two undefined nonterminals in the grammar: *id* and *int*. An integer, *int*, consists of 1 or more digits (0-9) and denotes a decimal integer. An identifier, *id*, must

begin with a letter, and consists of 1 or more letters, digits, and underscores. Upper- and lower-case letters are distinct, thus aa, AA, Aa, and aA are four different identifiers.

The keywords in the grammar (**`int`**, **`if`**, etc.) are reserved and may not be used as identifiers.

All integer values are 32-bit, two's complement numbers.

D includes binary arithmetic operators +, -, and *. There is no division operator or unary + or - operators. The value -n can be computed by evaluating 0-n.

A *bool-exp* is a logical expression, which may only be used as a condition in an if or while statement. Logical expressions do not have integer values and cannot be stored in variables.

In conditional statements, each else is paired with the nearest previous unpaired if.

All local variables must be declared at the beginning of a function, and each declaration introduces a single variable. The local variables and parameters in a function must have distinct names, and their scope extends over the entire function definition.

All functions are integer-valued, including main.

Function execution must terminate by executing a return statement. It is an error to "fall off" the end of the list of statements that make up a function body.

There are two predefined functions that provide integer input and output.

get() yields the next integer value from the standard input. If the next non-whitespace characters in the input do not form an integer constant, execution of get() is not defined.

put(x) yields the value of x and, as a side effect, prints that value on the next line of the standard output. Like all function calls, put(x) is an expression, so it may not be used as a statement. The statement x=put(x); may be used to print the value of a variable.