Democritus Language Reference Manual

Amy Xu
xx2152

Emily Pakulski
enp2111

Amarto Rajaram
aar2160

Kyle Lee
kpl2111

March 3, 2016

Contents

Chapter 1

Introduction

Democritus is a programming language with a static type system and native support for concurrent programming via its `atomic` keyword, with facilities for both imperative and functional programming. Democritus is compiled to the LLVM (Low Level Virtual Machine) intermediate form, which can then be optimized to machine-specific assembly code. Democritus' syntax draws inspiration from contemporary languages, aspiring to emulate Go and Python in terms of focusing on excelling in use cases familiar to the modern software engineer[1] as well as emphasizing readability and having "one – and preferably only one – obvious way to do it"[2].

[1]https://golang.org/doc/faq#Origins
[2]http://c2.com/cgi/wiki?PythonPhilosophy

Chapter 2

Data types

2.1 Primitive Types

int

A standard 32-bit two's-complement signed integer. It can take any value in the inclusive range (-2147483648, 2147483647).

float

A 64-bit floating precision number, represnted in the IEEE 754 format.

char

An 8-bit ASCII character.

boolean

A boolean may take a true or false value.

pointer

An 8-bit pointer holds the value to a location in memory; they operate similarly to those found in C.

2.2 Complex Types

List (array)

A list object can be accessed by array notation, such as `list1[0]`.

struct

A struct is a simple user-defined data structure that holds various primitives, similar to the ones found in C.

Chapter 3

Lexical Conventions

In this section, we will cover the standard lexical conventions for Democritus. Similarly to most other languages, Democritus is whitespace independent. The parser will discard whitespace characters such as ‘ ’, ‘\t’, and ‘\n’.

3.1 Identifiers

Identifiers for Democritus will be defined in the same way as they are in most other languages; any sequence of letters and numbers without whitespace between them that are not keywords will be parsed as an identifier. Note that, as in other languages, identifiers cannot begin with a number. Somewhat different, however, is the order of variable declarations; in Democritus, declarations are made following the *varname vartype* structure.

```
2wrongID int;    /* Not a valid identifier */
mySecond float; /* Valid */
my_Second char; /* Valid */
```

3.2 Keywords

The list of reserved keywords used in Democritus are as follows:

```
if
else
elif
for
return
int
float
char
boolean
function
void
list
struct
string
true
false
```

```
break
continue
atomic
```

These words have been reserved by the compiler and hold special meaning within the language. Though most are self-explanatory, we will delve into their usage later on.

3.3 Punctuation

;

Similarly to C, the semicolon ‘;’ is required to terminate any statement in Democritus.

{ and }

For the sake of keeping the language whitespace independent, curly braces are used to delineate between separate and nested blocks. These braces are required even for single-statement conditional and iteration loops.

(and)

Expressions may be encapsulated within parentheses to guarantee order of operations.

Comments

For now, comments are initiated with `/*` and closed with `*/`.

Chapter 4

Expressions and Operators

4.1 Assignment

Assignment is done with `=`. As mentioned above, variables and declared with the `varname vartype` syntax. Variables can be assigned to a single value or to the result of an expression.

```
x float = 4.0;
y int = 5/2 + 1; /* y = 3 */
```

4.2 Arithmetic Operations

Democritus supports all the arithmetic operations standard to most general-purpose languages like C and Java. Note that casting is not built into the language; this functionality will instead be implemented through the standard library.

Addition and Subtraction

Addition works with the `+` character, behaving as expected.

```
x int = 4;
y int = 2;
x = x+y;  /* x = 6  */
y = y-x   /* y = -4 */
```

Multiplication

```
x int = 4;
y int = 2;
x = x*y+y; /* x = 10 */
```

Multiplication follows the same rules as well.

Chapter 5

Statements

Chapter 6

Functions

Chapter 7

Concurrency