Oat v. 1 Language Specification

CS153

October 21, 2019

1 Grammar

The following grammar defines the Oat syntax. All binary operations are *left associative* with precedence levels indicated numerically. Higher precedence operators bind tighter than lower precedence ones.

prog	::=		prog
	|	$decl_1 \,..\, decl_n$	
decl	::=		global declarations
	|	*gdecl*	
	|	*fdecl*	
gdecl	::=		global variable declarations
	|	`global` *id* = *gexp* `;`	
fdecl	::=		function declaration
	|	*t id*(*args*) *block*	
args	::=		args
	|	$arg_1, \,..\,, arg_n$	
arg	::=		arg
	|	*t id*	
block	::=		blocks
	|	`{`$stmt_1 \,..\, stmt_n$`}`	
t	::=		types
	|	`int`	
	|	`bool`	
	|	*ref*	
ref	::=		reference types
	|	`string`	
	|	*t*`[]`	

gexp	::=		global initializers
	|	*integer*	64-bit integer literals
	|	*string*	C-style strings
	|	*ref* `null`	
	|	`true`	
	|	`false`	
	|	`new` *t* `[]{`$gexp_1, .., gexp_n$`}`	
stmt	::=		statements
	|	*lhs* `=` *exp* `;`	
	|	*vdecl* `;`	
	|	`return` *exp* `;`	
	|	`return ;`	
	|	*id*`(`$exp_1, .., exp_n$`);`	
	|	*if_stmt*	
	|	`for(`*vdecls* `;` exp_{opt} `;` $stmt_{opt}$`)` *block*	
	|	`while(`*exp*`)` *block*	
if_stmt	::=		if statements
	|	`if(`*exp*`)` *block else_stmt*	
else_stmt	::=		else
	|	ϵ	
	|	`else` *block*	
	|	`else` *if_stmt*	
lhs	::=		lhs expressions
	|	*id*	
	|	$exp1[exp2]$	
vdecls	::=		decl list
	|	$vdecl_1, .., vdecl_n$	
vdecl	::=		local declarations
	|	`var` *id* `=` *exp*	

exp	::=		expressions
	|	*id*	
	|	*integer*	64-bit integer literals
	|	*string*	C-style strings
	|	*ref* `null`	
	|	`true`	
	|	`false`	
	|	exp_1 `[`exp_2`]`	
	|	*id*`(`exp_1`, .. ,`exp_n`)`	
	|	`new` *t* `[]{`exp_1`, .. ,`exp_n`}`	
	|	`new` *t* `[`exp_1`]`	
	|	exp_1 *bop* exp_2	
	|	*uop exp*	
	|	`(`*exp*`)`	
bop	::=		(left associative) binary operations
	|	`*`	precedence 100 (multiplication)
	|	+	precedence 90 (addition)
	|	-	precedence 90 (subtraction)
	|	<<	precedence 80 (shift left)
	|	>>	precedence 80 (shift right logical)
	|	>>>	precedence 80 (shift right arithmetic)
	|	<	precedence 70 (less-than)
	|	<=	precedence 70 (less-than or equal)
	|	>	precedence 70 (greater-than)
	|	>=	precedence 70 (greater-than or equal)
	|	==	precedence 60 (equal)
	|	!=	precedence 60 (not equal)
	|	&	precedence 50 (logical and)
	|	|	precedence 40 (logical or)
	|	[&]	precedence 30 (bit-wise and)
	|	[|]	precedence 20 (bit-wise or)
uop	::=		unary operations
	|	-	
	|	!	
	|	~	