Oat v. 1 Language Specification

CIS341 – Steve Zdancewic

March 6, 2025

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
	|	$decl_1$ .. $decl_i$	
decl	::=		global declarations
	|	*gdecl*	
	|	*fdecl*	
gdecl	::=		global variable declarations
	|	`global` *id* = *gexp* ;	
arg	::=		arg
	|	*t id*	
args	::=		args
	|	arg_1, .. , arg_n	
fdecl	::=		function declaration
	|	*retty id* `(` *args* `)` *block*	
block	::=		blocks
	|	`{`$stmt_1$ .. $stmt_n$`}`	
t	::=		types
	|	`int`	
	|	`bool`	
	|	*ref*	
ref	::=		reference types
	|	`string`	
	|	*t* `[]`	
	|	`(`t_0, .. , t_n`)` -> *retty*	function pointer

retty	::=		return types
	|	`void`	
	|	*t*	
bop	::=		(left associative) binary operations
	|	`*`	multiplication (precedence 100)
	|	`+`	addition (precedence 90)
	|	`-`	subtraction (precedence 90)
	|	`<<`	shift left (precedence 80)
	|	`>>`	shift right logical (precedence 80)
	|	`>>>`	shift right arithmetic (precedence 80)
	|	`<`	less-than (precedence 70)
	|	`<=`	less-than or equal (precedence 70)
	|	`>`	greater-than (precedence 70)
	|	`>=`	greater-than or equal (precedence 70)
	|	`==`	equal (precedence 60)
	|	`!=`	not equal (precedence 60)
	|	`&`	logical and (precedence 50)
	|	`|`	logical or (precedence 40)
	|	`[&]`	bit-wise and (precedence 30)
	|	`[|]`	bit-wise or (precedence 20)
uop	::=		unary operations
	|	`-`	
	|	`!`	
	|	`~`	
gexp	::=		global initializers
	|	*integer*	64-bit integer literals
	|	*string*	C-style strings
	|	*ref* `null`	
	|	`true`	
	|	`false`	
	|	`new` t`[]{`$gexp_1, .., gexp_n$`}`	
lhs	::=		left-hand-sides for assignment
	|	*id*	
	|	exp_1`[`exp_2`]`	

exp	::=		expressions
	|	*integer*	64-bit integer literals
	|	*string*	C-style strings
	|	*ref* `null`	
	|	`true`	
	|	`false`	
	|	*lhs*	left-hand-side as an expression
	|	*id*`(`exp_1`, .. ,`exp_n`)`	function call
	|	`new` *t* `[]{`exp_1`, .. ,`exp_n`}`	Explicitly initialized array
	|	`new int [`exp_1`]`	Default-initialize int array
	|	`new bool [`exp_1`]`	Default-initialize bool array
	|	exp_1 *bop* exp_2	
	|	*uop exp*	
	|	`(`*exp*`)`	
vdecl	::=		local declarations
	|	`var` *id* `=` *exp*	
vdecls	::=		decl list
	|	$vdecl_1$`, .. ,`$vdecl_n$	
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*	