

Oat v. 1 Language Specification

CIS341 – Steve Zdancewic

March 19, 2020

1 Grammar

The following grammar defines the Oat syntax. In the grammar, *id* denotes an identifier, *n* denotes a non-negative integer, and *s* denotes a string literal. All binary operations are *left associative* with precedence levels indicated numerically. Higher precedence operators bind tighter than lower precedence ones.

<i>prog</i>	::=	<i>prog</i>
		<i>decl</i> ₁ .. <i>decl</i> _{<i>i</i>}
<i>decl</i>	::=	global declarations
		<i>gdecl</i>
		<i>fdecl</i>
<i>gdecl</i>	::=	global variable declarations
		global <i>id</i> = <i>gexp</i> ;
<i>fdecl</i>	::=	function declaration
		<i>t id(args) block</i>
<i>args</i>	::=	args
		<i>arg</i> ₁ , .., <i>arg</i> _{<i>i</i>}
<i>arg</i>	::=	arg
		<i>t id</i>
<i>block</i>	::=	blocks
		{ <i>stmt</i> ₁ .. <i>stmt</i> _{<i>i</i>} }
<i>t</i>	::=	types
		int
		bool
		<i>ref</i>
<i>ref</i>	::=	reference types
		string
		<i>t</i> []

| *fty*

<i>gexp</i>	::= <ul style="list-style-type: none"> <i>n</i> <i>s</i> tnull true false new <i>t</i>[] {<i>gexp</i>₁, .., <i>gexp</i>_{<i>i</i>}} 	global initializers
<i>stmt</i>	::= <ul style="list-style-type: none"> <i>lhs</i> = <i>exp</i>; <i>vdecl</i>; return <i>exp</i>; return ; <i>id</i>(<i>exp</i>₁, .., <i>exp</i>_{<i>i</i>}); <i>if_stmt</i> for(<i>vdecls</i>; <i>exp_opt</i>; <i>stmt_opt</i>) <i>block</i> while(<i>exp</i>) <i>block</i> 	statements
<i>if_stmt</i>	::= <ul style="list-style-type: none"> if(<i>exp</i>) <i>block</i> <i>else_stmt</i> 	if statements
<i>else_stmt</i>	::= <ul style="list-style-type: none"> ϵ else <i>block</i> else <i>if_stmt</i> 	else
<i>lhs</i>	::= <ul style="list-style-type: none"> <i>id</i> <i>exp</i>₁[<i>exp</i>₂] 	lhs expressions
<i>vdecls</i>	::= <ul style="list-style-type: none"> <i>vdecl</i>₁, .., <i>vdecl</i>_{<i>i</i>} 	decl list
<i>vdecl</i>	::= <ul style="list-style-type: none"> var <i>id</i> = <i>exp</i> 	local declarations

<i>exp</i>	::=	expressions
	<i>id</i>	
	<i>n</i>	
	<i>s</i>	
	<i>t null</i>	
	<i>true</i>	
	<i>false</i>	
	<i>exp</i> ₁ [<i>exp</i> ₂]	
	<i>id</i> (<i>exp</i> ₁ , .., <i>exp</i> _{<i>i</i>})	
	<i>new t</i> [] { <i>exp</i> ₁ , .., <i>exp</i> _{<i>i</i>} }	
	<i>new t</i> [<i>exp</i> ₁]	
	<i>exp</i> ₁ <i>bop exp</i> ₂	
	<i>uop exp</i>	
	(<i>exp</i>)	
<i>bop</i>	::=	(left associative) binary operations
	*	precedence 100
	+	precedence 90
	-	precedence 90
	<<	precedence 80
	>>	precedence 80
	>>>	precedence 80
	<	precedence 70
	<=	precedence 70
	>	precedence 70
	>=	precedence 70
	==	precedence 60
	!=	precedence 60
	&	precedence 50
		precedence 40
	[&]	precedence 30
	[]	precedence 20
<i>uop</i>	::=	unary operations
	-	
	!	
	~	