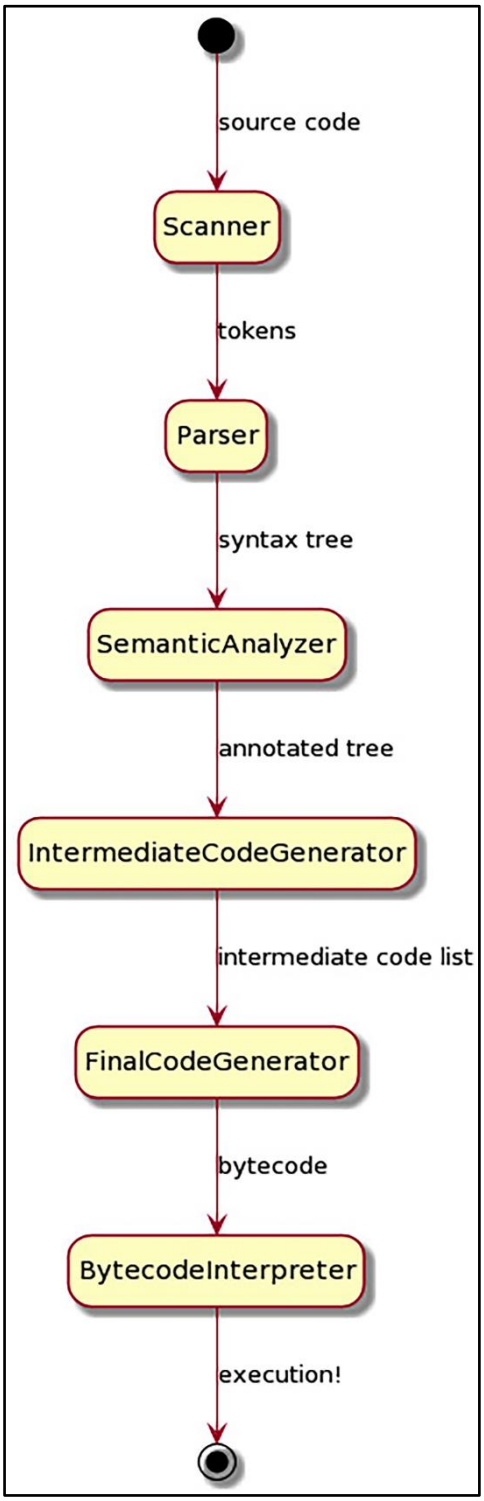
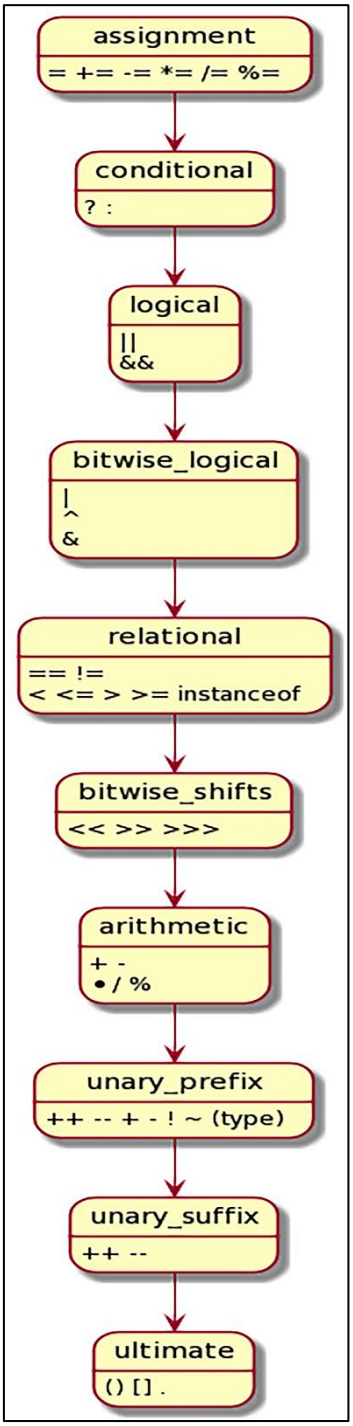
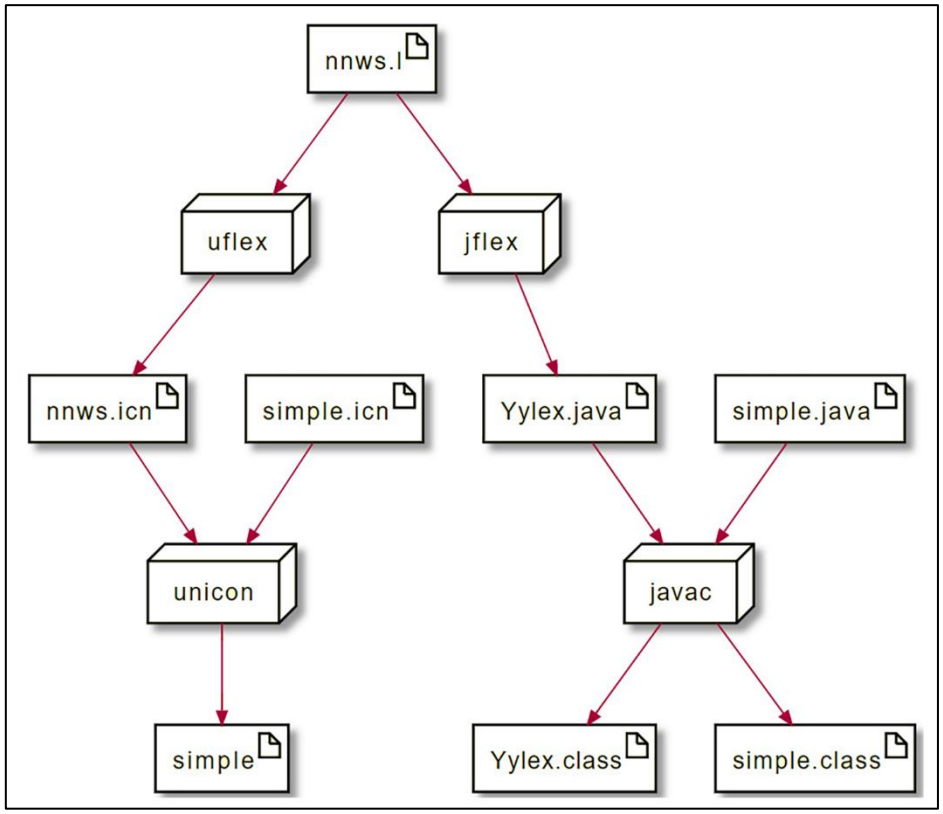
Chapter 1: Why Build Another Programming Language?



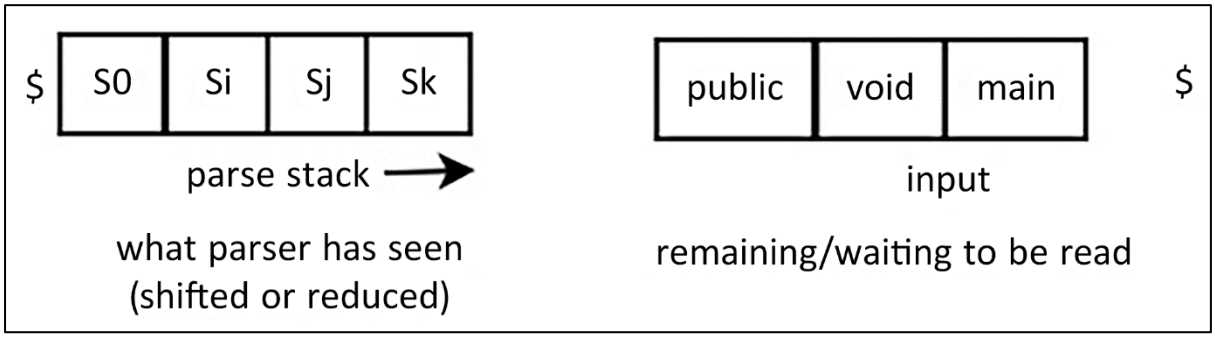
Chapter 2: Programming Language Design



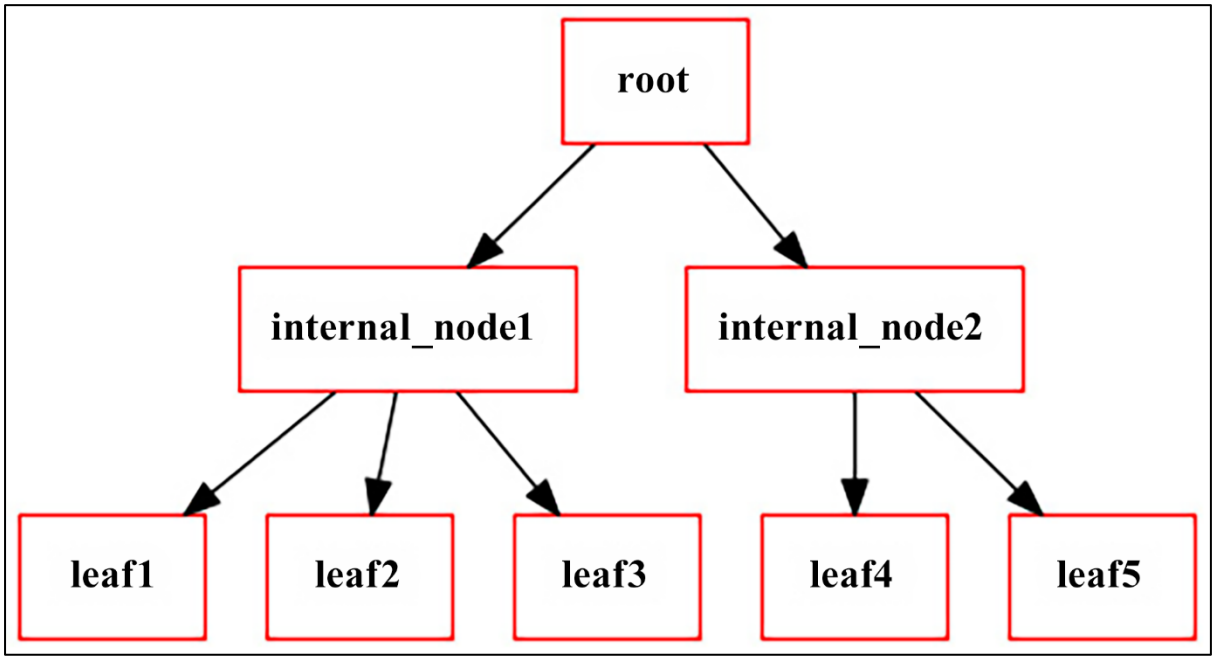
Chapter 3: Scanning Source Code

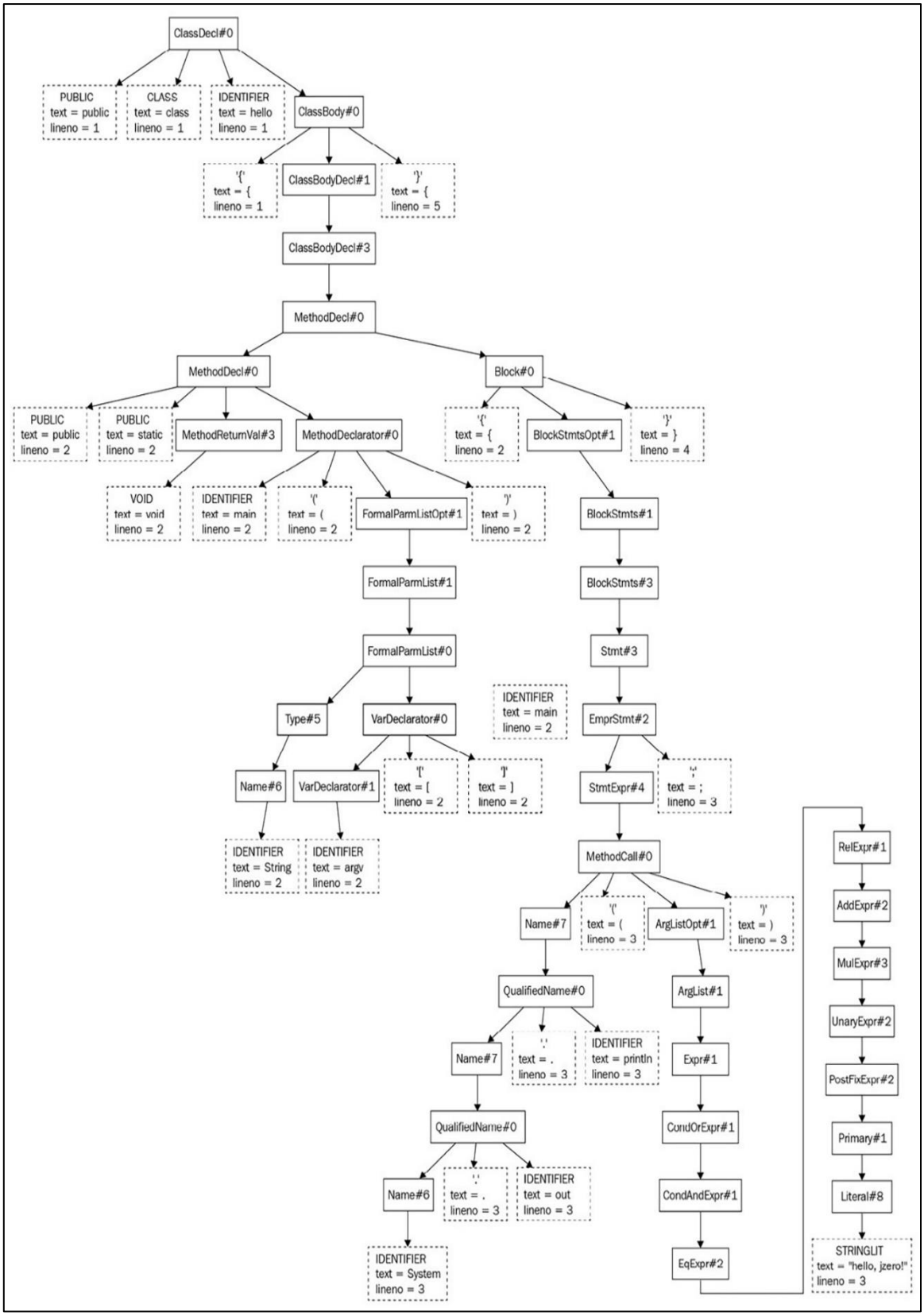


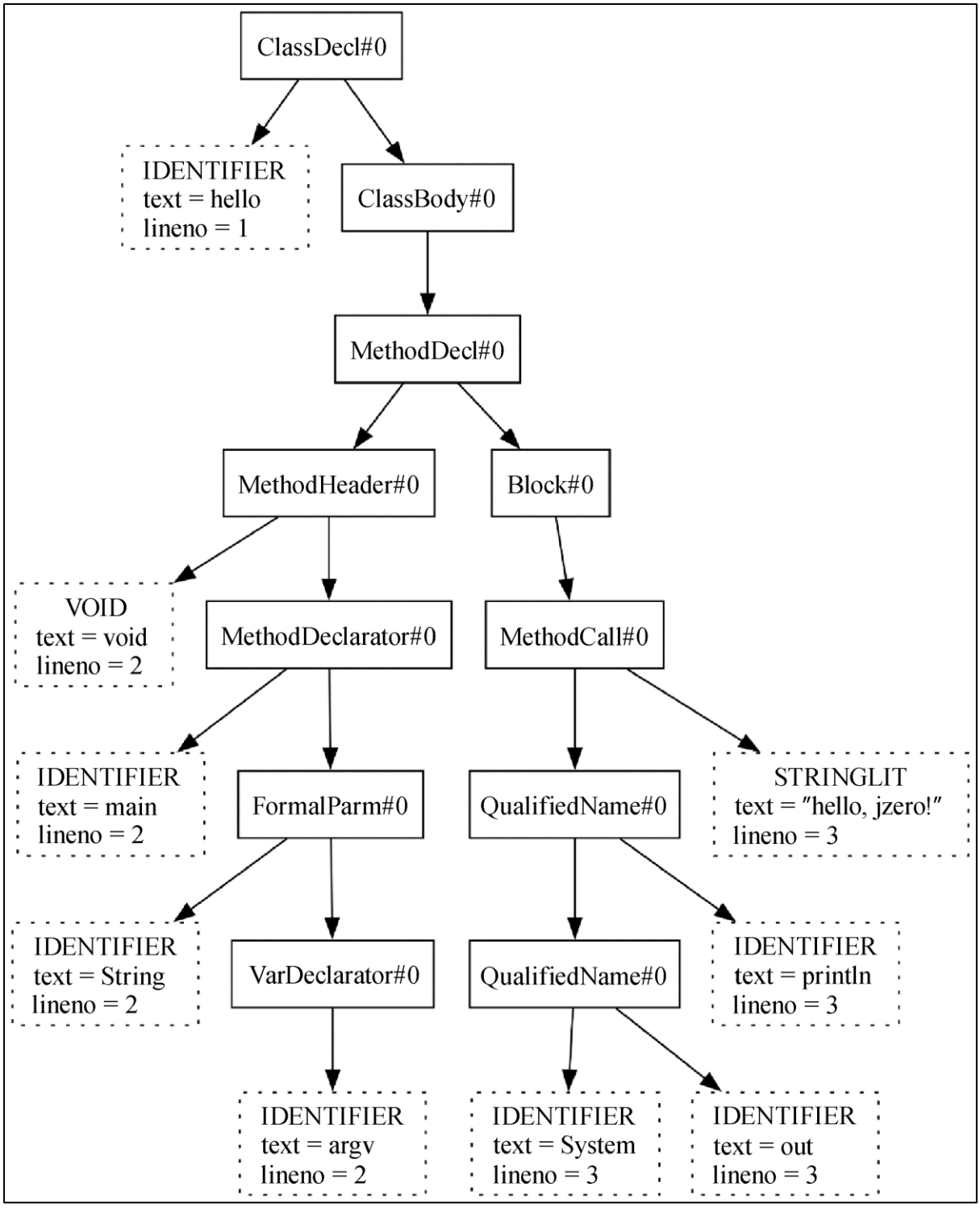
Chapter 4: Parsing

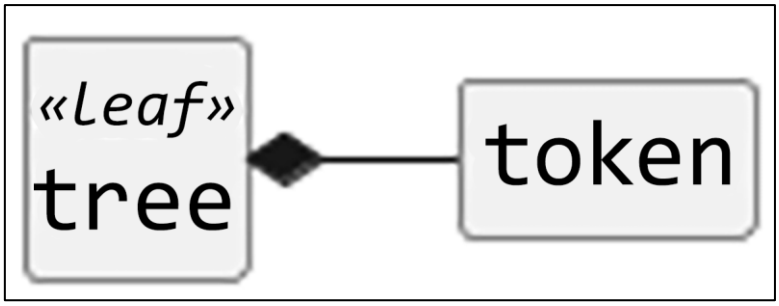


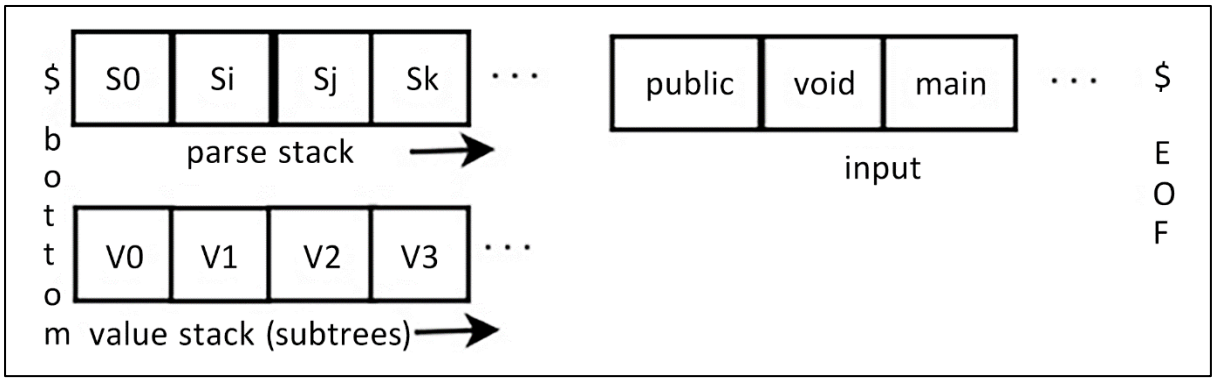
Chapter 5: Syntax Trees

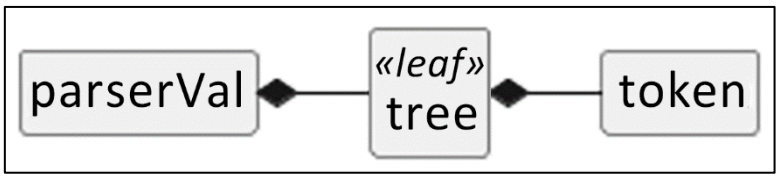


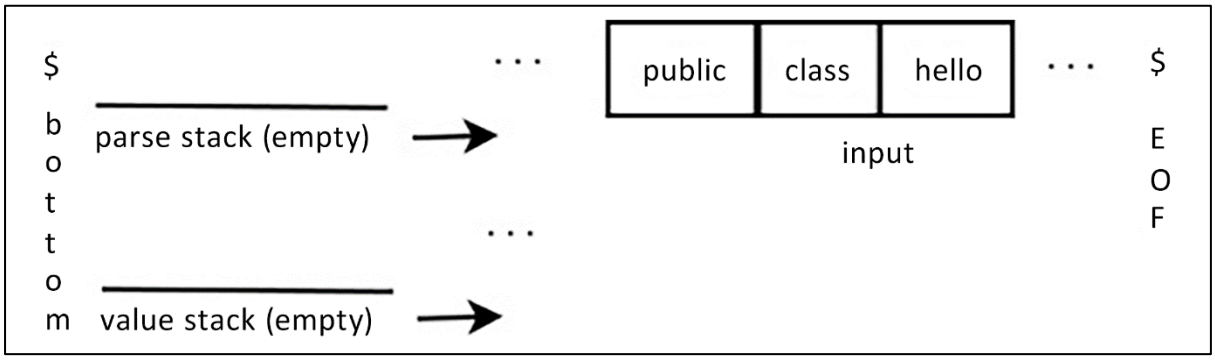

ClassDecl#0
PUBLIC
text = public
lineno = 1
CLASS
text = class
lineno = 1
IDENTIFIER
text = hello
lineno = 1
ClassBody#0
'{'
text = {
lineno = 1
ClassBodyDecl#1
'}'
text = {
lineno = 5
ClassBodyDecl#3
MethodDecl#0
MethodDecl#0
Block#0
PUBLIC
text = public
lineno = 2
PUBLIC
text = static
lineno = 2
MethodReturnVal#3
MethodDeclarator#0
'{'
text = {
lineno = 2
BlockStmtsOpt#1
'}'
text = }
lineno = 4
VOID
text = void
lineno = 2
IDENTIFIER
text = main
lineno = 2
'('
text = (
lineno = 2
FormalParmListOpt#1
')'
text =)
lineno = 2
BlockStmts#1
FormalParmList#1
BlockStmts#3
FormalParmList#0
Stmt#3
IDENTIFIER
text = main
lineno = 2
Type#5
VarDeclarator#0
EmprStmt#2
Name#6
VarDeclarator#1
'['
text = [
lineno = 2
']'
text =]
lineno = 2
StmtExpr#4
';'
text = ;
lineno = 3
IDENTIFIER
text = String
lineno = 2
IDENTIFIER
text = argv
lineno = 2
MethodCall#0
Name#7
'('
text = (
lineno = 3
ArgListOpt#1
')'
text =)
lineno = 3
QualifiedName#0
ArgList#1
Name#7
'.'
text = .
lineno = 3
IDENTIFIER
text = println
lineno = 3
Expr#1
QualifiedName#0
CondOrExpr#1
Name#6
'.'
text = .
lineno = 3
IDENTIFIER
text = out
lineno = 3
CondAndExpr#1
IDENTIFIER
text = System
lineno = 3
EqExpr#2
RelExpr#1
AddExpr#2
MulExpr#3
UnaryExpr#2
PostFixExpr#2
Primary#1
Literal#8
STRINGLIT
text = "hello, jzero!"
lineno = 3

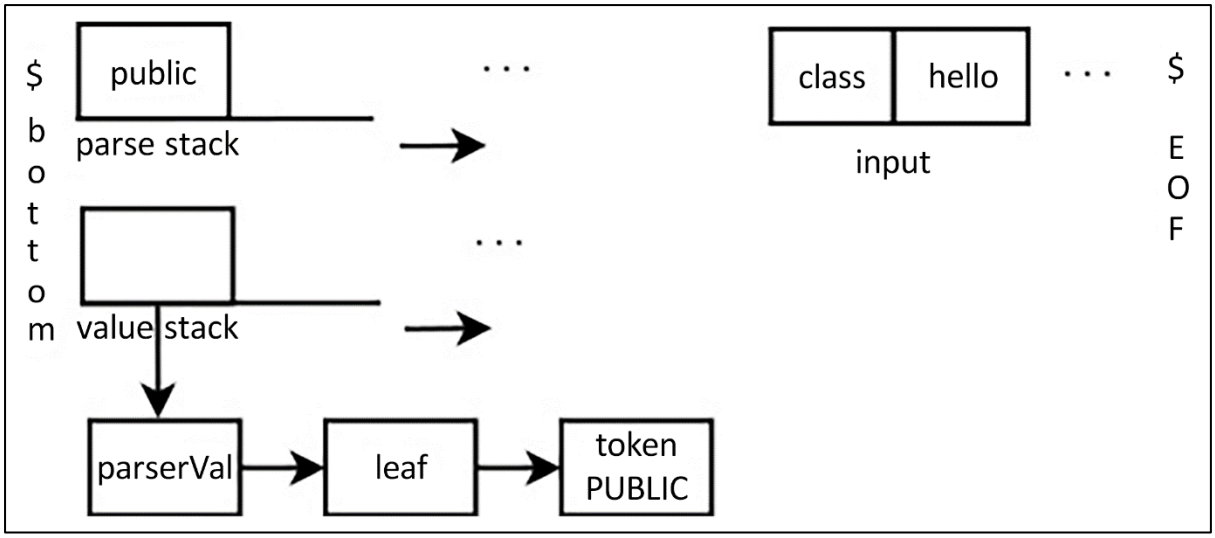

ClassDecl#0
IDENTIFIER
text = hello
lineno = 1
ClassBody#0
MethodDecl#0
MethodHeader#0
Block#0
VOID
text = void
lineno = 2
MethodDeclarator#0
MethodCall#0
IDENTIFIER
text = main
lineno = 2
FormalParm#0
QualifiedName#0
STRINGLIT
text = "hello, jzero!"
lineno = 3
IDENTIFIER
text = String
lineno = 2
VarDeclarator#0
QualifiedName#0
IDENTIFIER
text = println
lineno = 3
IDENTIFIER
text = argv
lineno = 2
IDENTIFIER
text = System
lineno = 3
IDENTIFIER
text = out
lineno = 3

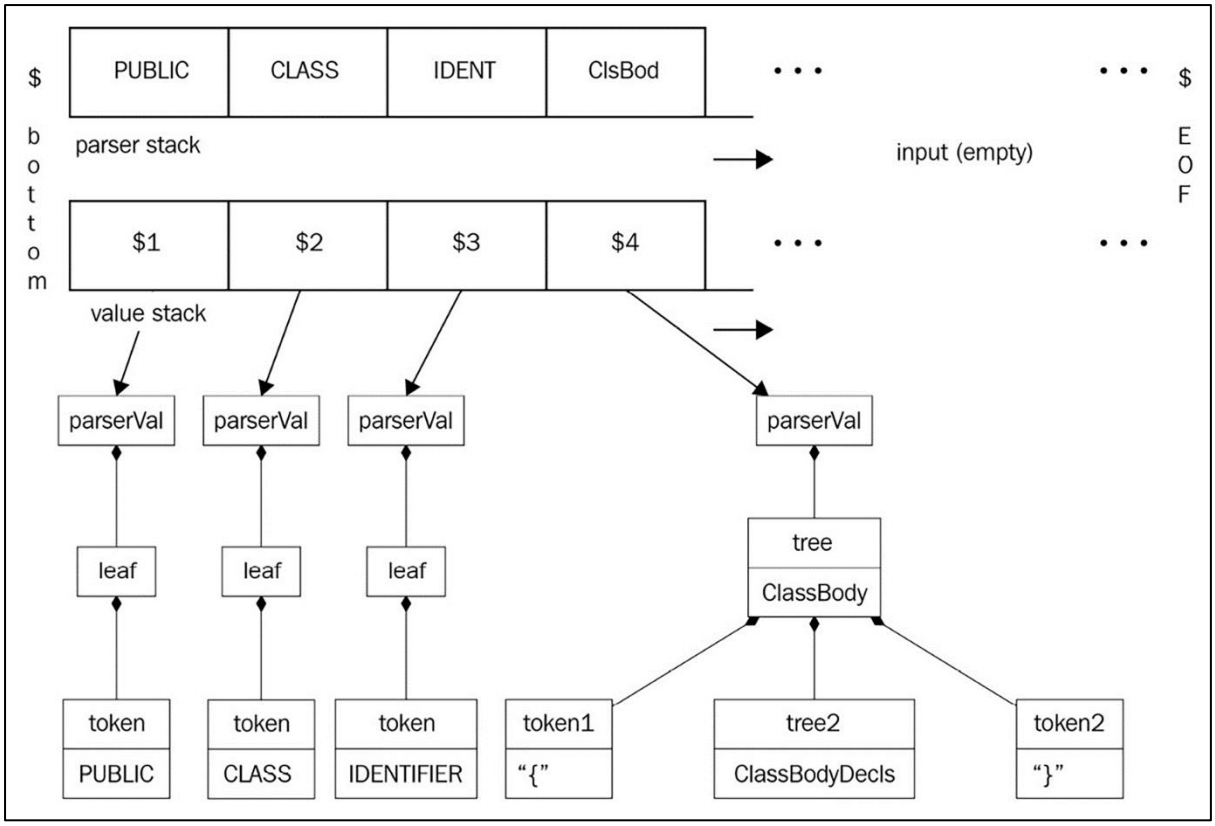

«Leaf»
tree
token

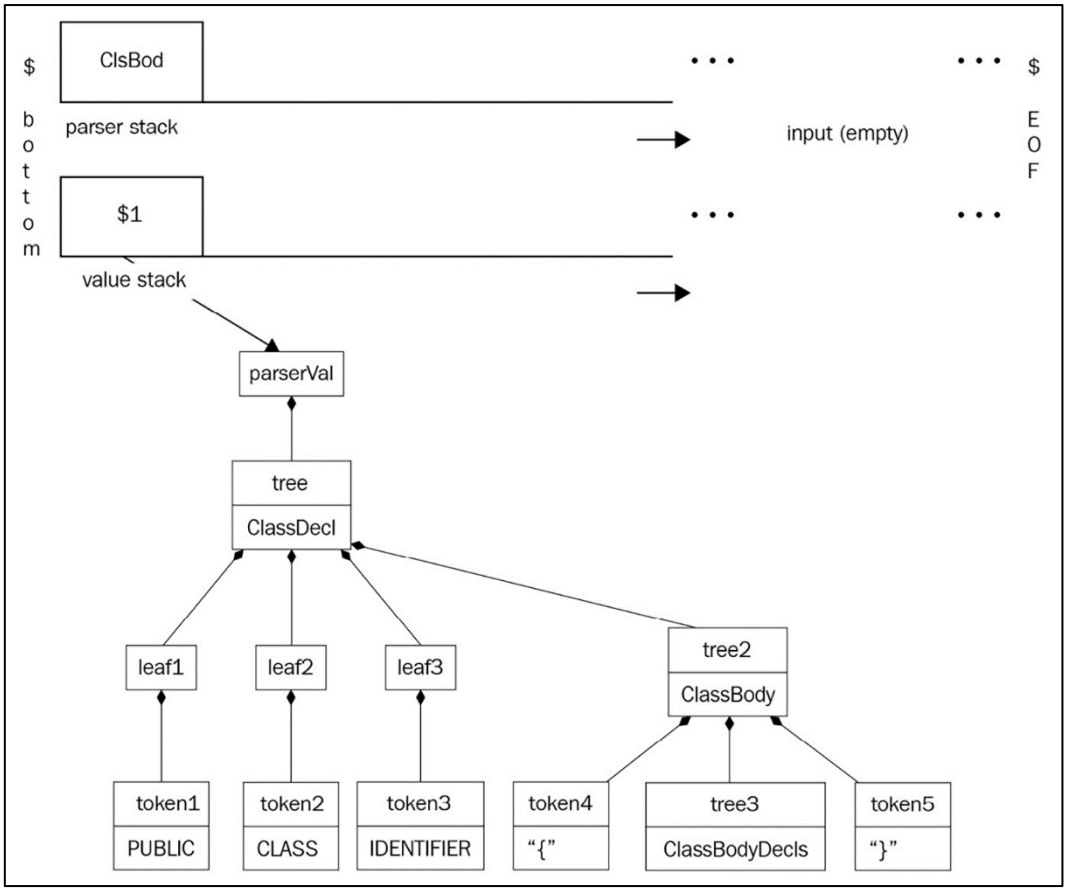

$
S0
Si
Sj
Sk
parse stack
bottom
V0
V1
V2
V3
value stack (subtrees)
public
void
main
input
$
EOF

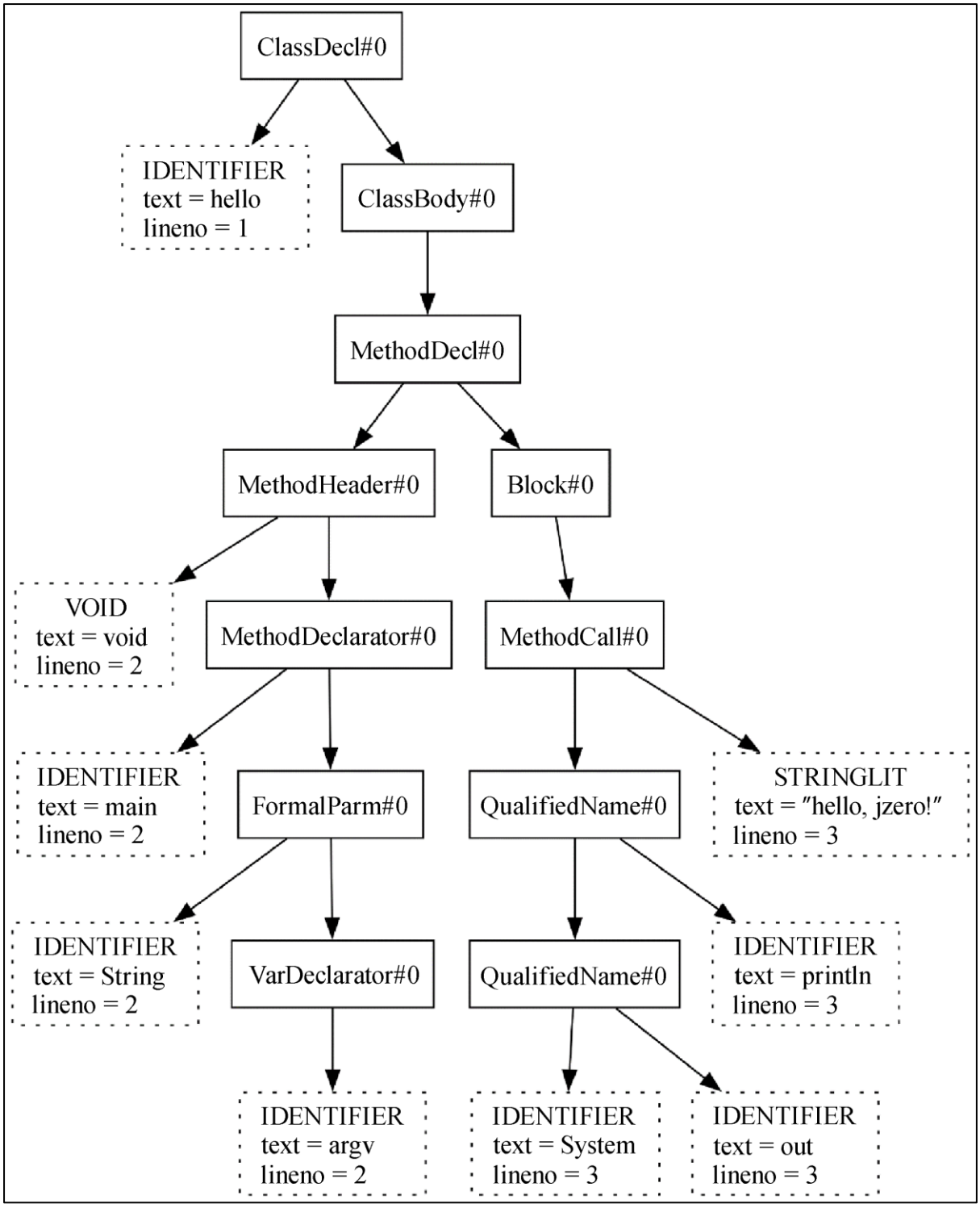

parserVal
«leaf»
tree
token


$
parse stack (empty)
bottom
value stack (empty)
public
class
hello
input
$
EOF

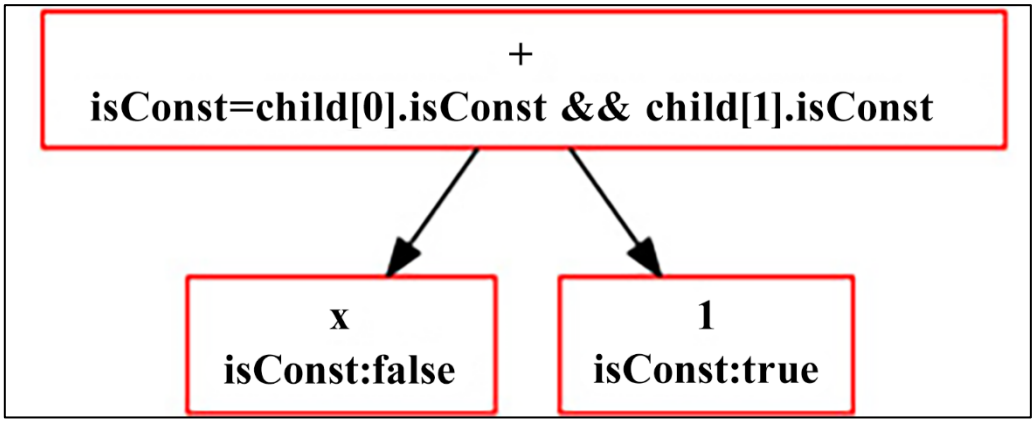

$
bottom
public
parse stack
class
hello
input
$
EOF
value stack
parserVal
leaf
token
PUBLIC

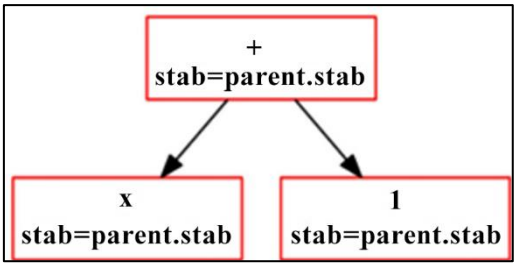

$
bottom
PUBLIC
CLASS
IDENT
ClsBod
parser stack
input (empty)
$
EOF
$1
$2
$3
$4
value stack
parserVal
parserVal
parserVal
parserVal
leaf
leaf
leaf
tree
ClassBody
token
PUBLIC
token
CLASS
token
IDENTIFIER
token1
"{"
tree2
ClassBodyDecls
token2
"}"


$
bottom
ClsBod
parser stack
$1
value stack
input (empty)
$
EOF
parserVal
tree
ClassDecl
leaf1
leaf2
leaf3
tree2
ClassBody
token1
PUBLIC
token2
CLASS
token3
IDENTIFIER
token4
"{"
tree3
ClassBodyDecls
token5
"}"


ClassDecl#0
IDENTIFIER
text = hello
lineno = 1
ClassBody#0
MethodDecl#0
MethodHeader#0
Block#0
VOID
text = void
lineno = 2
MethodDeclarator#0
MethodCall#0
IDENTIFIER
text = main
lineno = 2
FormalParm#0
QualifiedName#0
STRINGLIT
text = "hello, jzero!"
lineno = 3
IDENTIFIER
text = String
lineno = 2
VarDeclarator#0
QualifiedName#0
IDENTIFIER
text = println
lineno = 3
IDENTIFIER
text = argv
lineno = 2
IDENTIFIER
text = System
lineno = 3
IDENTIFIER
text = out
lineno = 3

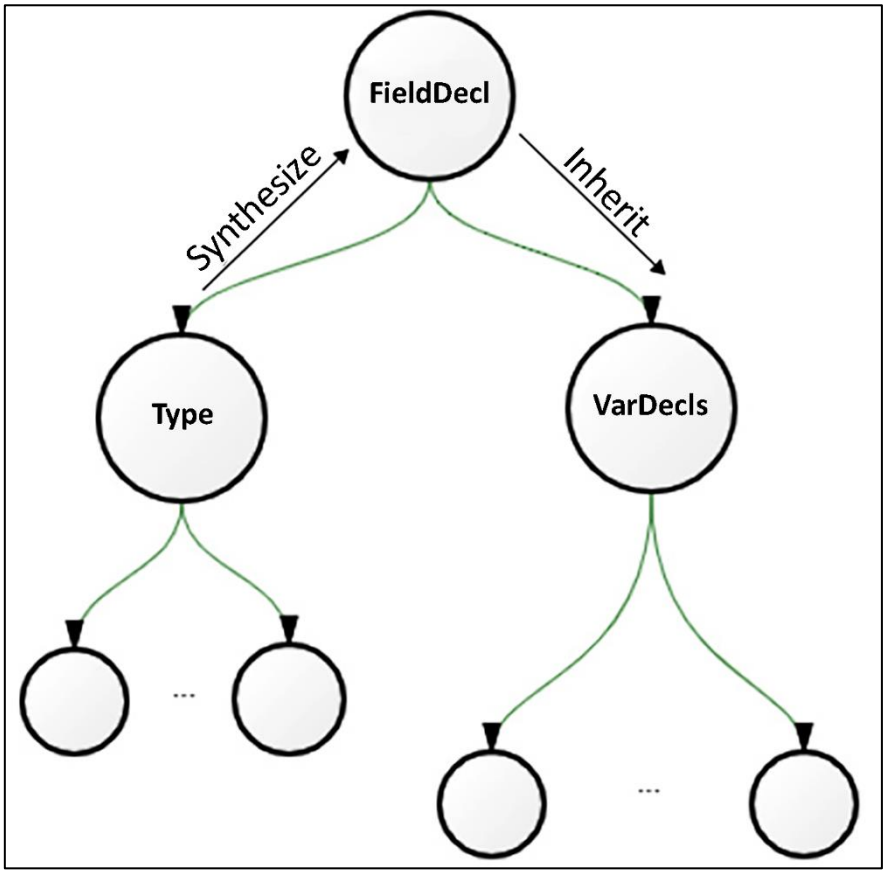
Chapter 6: Symbol Tables






```
C:\Users\clint\books\byopl\github\Build-Your-Own-Programming-Language\ch6>set CLASSPATH=".;C:\Users
\clint\books\byopl\github\Build-Your-Own-Programming-Language"

C:\Users\clint\books\byopl\github\Build-Your-Own-Programming-Language\ch6>java ch6.j0 hello.java
yyfilename hello.java
global - 2 symbols
 hello
  class - 2 symbols
   main
    method - 0 symbols
   System
 System
  class - 1 symbols
   out
    class - 1 symbols
     println
no errors

C:\Users\clint\books\byopl\github\Build-Your-Own-Programming-Language\ch6>
```

Chapter 7: Checking Base Types

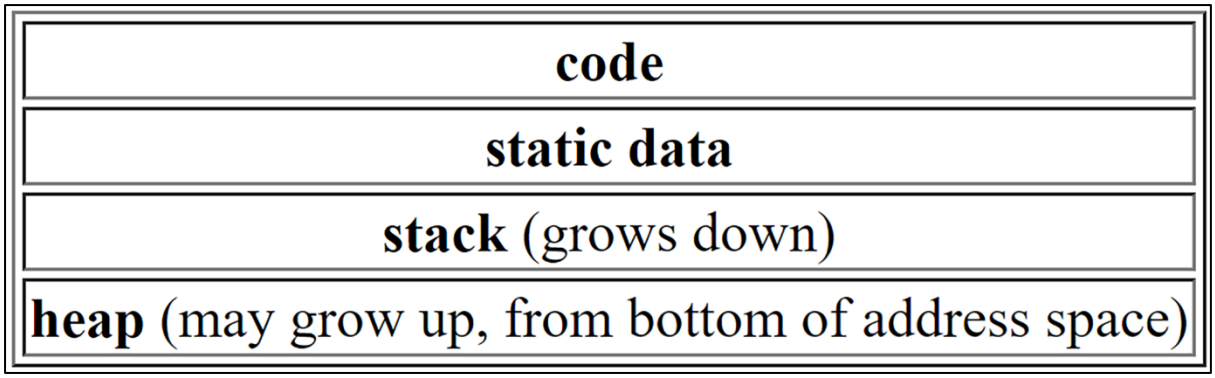



```
D:\Users\Clinton Jeffery\books\byopl\github\Build-Your-Own-Programming-Language\ch7>type hello.java

public class hello {
   public static void main(String argv[]) {
      int x;
      x = 0;
      x = x + "hello";
      System.out.println("hello, jzero!");
   }
}

D:\Users\Clinton Jeffery\books\byopl\github\Build-Your-Own-Programming-Language\ch7>j0 hello.java
line 4: typecheck = on a int and a int -> OK
line 5: typecheck + on a String and a int -> FAIL

D:\Users\Clinton Jeffery\books\byopl\github\Build-Your-Own-Programming-Language\ch7>
```

Chapter 8: Checking Types on Arrays, Method Calls, and Structure Accesses

```
>type funtest.java
public class funtest {
   public static int foo(int x, int y, String z) {
       return 0;
   }
   public static void main(String argv[]) {
      int x;
      x = foo(0,1,"howdy");
      x = x + 1;
      System.out.println("hello, jzero!");
   }
}

>java ch8.j0 funtest.java
line 3: typecheck return on a int and a int -> OK
line 7: typecheck param on a String and a String -> OK
line 7: typecheck param on a int and a int -> OK
line 7: typecheck param on a int and a int -> OK
line 7: typecheck = on a int and a int -> OK
line 8: typecheck + on a int and a int -> OK
line 8: typecheck = on a int and a int -> OK
line 9: typecheck param on a String and a String -> OK
no errors
```

Chapter 9: Intermediate Code Generation

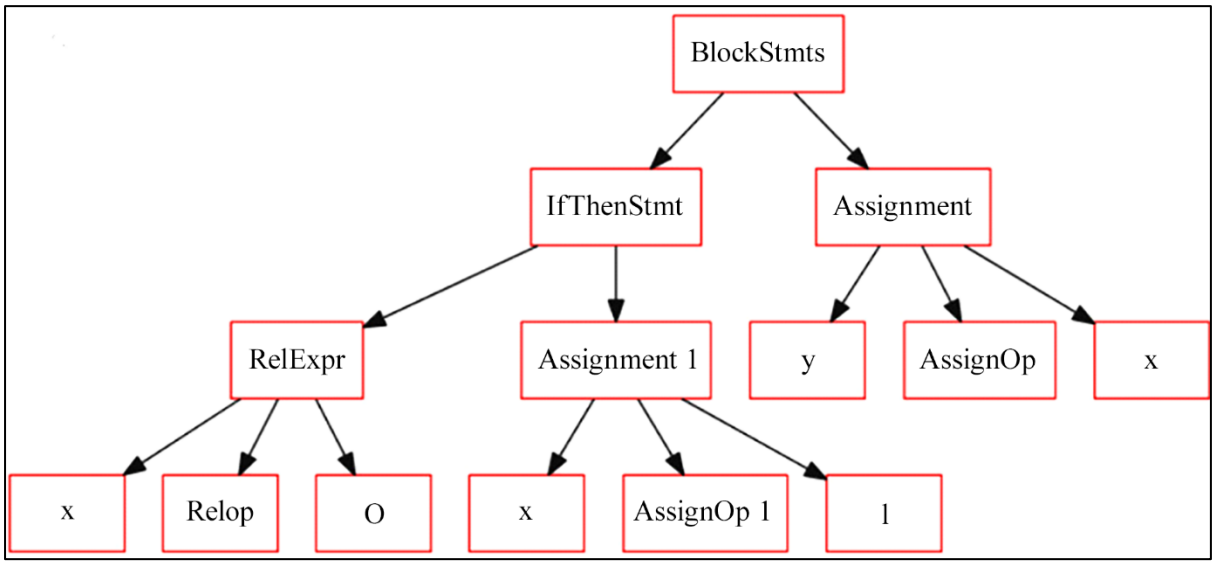


Opcode	C equivalent	Description
ADD,SUB,MUL,DIV	x=y op z	Store result of binary operation on y and z to x
NEG	x = -y	Store result of unary operation on y to x
ASN	x = y	Store y to x
ADDR	x = &y	Store address of y to x
LCON	x = *y	Store contents pointed to by y to x
SCON	*x = y	Store y to location pointed to by x
GOTO	goto L	Unconditional jump to L
BLT,BLE,BGT,BGE	if(x rop y)goto L	Test relation and conditionally jump to L
BIF	if (x) goto L	Conditionally jump to L if x != 0
BNIF	if (!x) goto L	Conditionally jump to L if x == 0
PARM		Store x as a parameter (push onto call stack)
CALL	x=p(...)	Call procedure p with n words of parameters
RET	return x	Return from function with result x

Declaration	Description
glob x,n	Declare a global variable named x that refers to offset n in the global region
proc x,n1,n2	Declare a procedure x with n1 words of parameters and n2 words of locals
loc x,n	Declare a local variable named x that refers to offset n in the local region
lab Ln	Declare a label Ln that will be a name for an instruction in the code region
end	Declare the end of the current procedure

B__.jpg

Production	Semantic Rules
Assignment : IDENT '=' AddExpr	Assignment.addr = IDENT.addr Assignment.icode = AddExpr.icode ||| gen(ASN, IDENT.addr, AddExpr.addr)
AddExpr : AddExpr_1 '+' MulExpr	AddExpr.addr = genlocal() AddExpr.icode = AddExpr_1.icode ||| MulExpr.icode ||| gen(ADD,AddExpr.addr,AddExpr_1.addr,MulExpr.addr)
AddExpr : AddExpr_1 '-' MulExpr	AddExpr.addr = genlocal() AddExpr.icode = AddExpr_1.icode ||| MulExpr.icode ||| gen(SUB,AddExpr.addr,AddExpr_1.addr,MulExpr.addr)
MulExpr : MulExpr_1 '*' UnaryExpr	MulExpr.addr = genlocal() MulExpr.icode = MulExpr_1.icode ||| UnaryExpr.icode ||| gen(MUL,MulExpr.addr,MulExpr_1.addr,UnaryExpr.addr)
MulExpr : MulExpr_1 '/' UnaryExpr	MulExpr.addr = genlocal() MulExpr.icode = MulExpr_1.icode ||| UnaryExpr.icode ||| gen(DIV,MulExpr.addr,MulExpr_1.addr,UnaryExpr.addr)
UnaryExpr : '-' UnaryExpr_1	UnaryExpr.addr = genlocal() UnaryExpr.icode = UnaryExpr_1.icode ||| gen(NEG,UnaryExpr.addr,UnaryExpr_1.addr)
UnaryExpr : '(' AddExpr ')'	UnaryExpr.addr = AddExpr.addr UnaryExpr.icode = AddExpr.icode
UnaryExpr : IDENT	UnaryExpr.addr = IDENT.addr UnaryExpr.icode = emptylist()



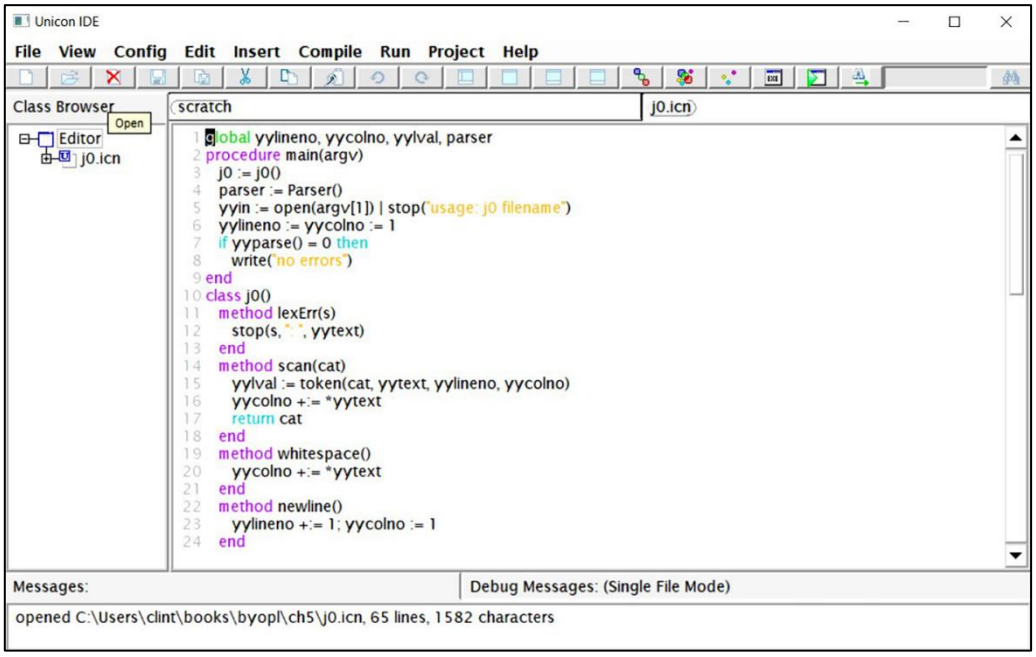
Production	Semantic Rules
IfThenStmt : if '(' Expr ')' Stmt	Expr.onTrue = Stmt.first Expr.onFalse = IfThenStmt.follow Stmt.follow = IfThenStmt.follow IfThenStmt.icode = (Expr.icode != null) ? Expr.icode : gen(BIF, Expr.onFalse, Expr.addr, con:0) IfThenStmt.icode |||:= gen(LABEL, Expr.onTrue) ||| Stmt.icode
IfThenElseStmt : if '(' Expr ')' $Stmt_1$ else $Stmt_2$	Expr.onTrue = $Stmt_1$.first Expr.onFalse = $Stmt_2$.first $Stmt_1$.follow = IfThenElseStmt.follow; $Stmt_2$.follow = IfThenElseStmt.follow; IfThenElseStmt.icode = (Expr.icode != null) ? Expr.icode : gen(BIF, Expr.onFalse, Expr.addr, con:0) IfThenElseStmt.icode |||:= gen(LABEL, Expr.onTrue) ||| $Stmt_1$.icode ||| gen(GOTO, IfThenElseStmt.follow) ||| gen(LABEL, Expr.onFalse) ||| $Stmt_2$.icode

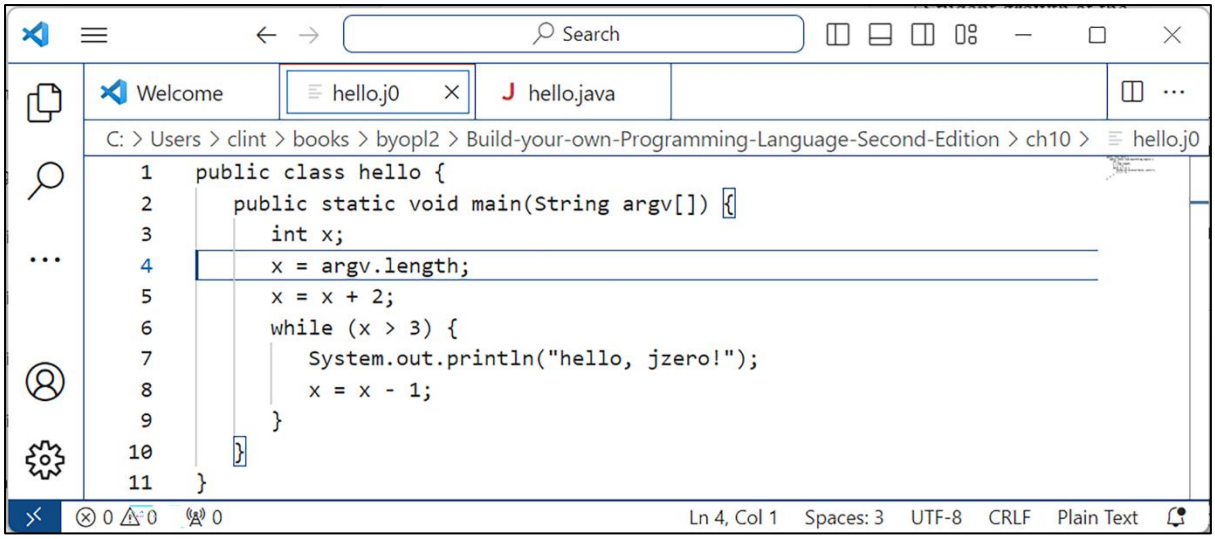
Production	Semantic Rules
AndExpr : $AndExpr_1$ && EqExpr	EqExpr.first = genlabel(); $AndExpr_1$.onTrue = EqExpr.first; $AndExpr_1$.onFalse = AndExpr.onFalse; EqExpr.onTrue = AndExpr.onTrue; EqExpr.onFalse = AndExpr.onFalse; AndExpr.icode = $AndExpr_1$.icode ||| gen(LABEL, EqExpr.first) ||| EqExpr.icode;
OrExpr : $OrExpr_1$ || AndExpr	AndExpr.first = genlabel(); $OrExpr_1$.onTrue = OrExpr.onTrue; $OrExpr_1$.onFalse = AndExpr.first; AndExpr.onTrue = OrExpr.onTrue; AndExpr.onFalse = OrExpr.onFalse; OrExpr.icode = $OrExpr_1$.icode ||| gen(LABEL, AndExpr.first) ||| AndExpr.icode;
UnaryExpr : ! $UnaryExpr_1$	$UnaryExpr_1$.onTrue = UnaryExpr.onFalse $UnaryExpr_1$.onFalse = UnaryExpr.onTrue UnaryExpr.icode = UnaryExpr1.icode

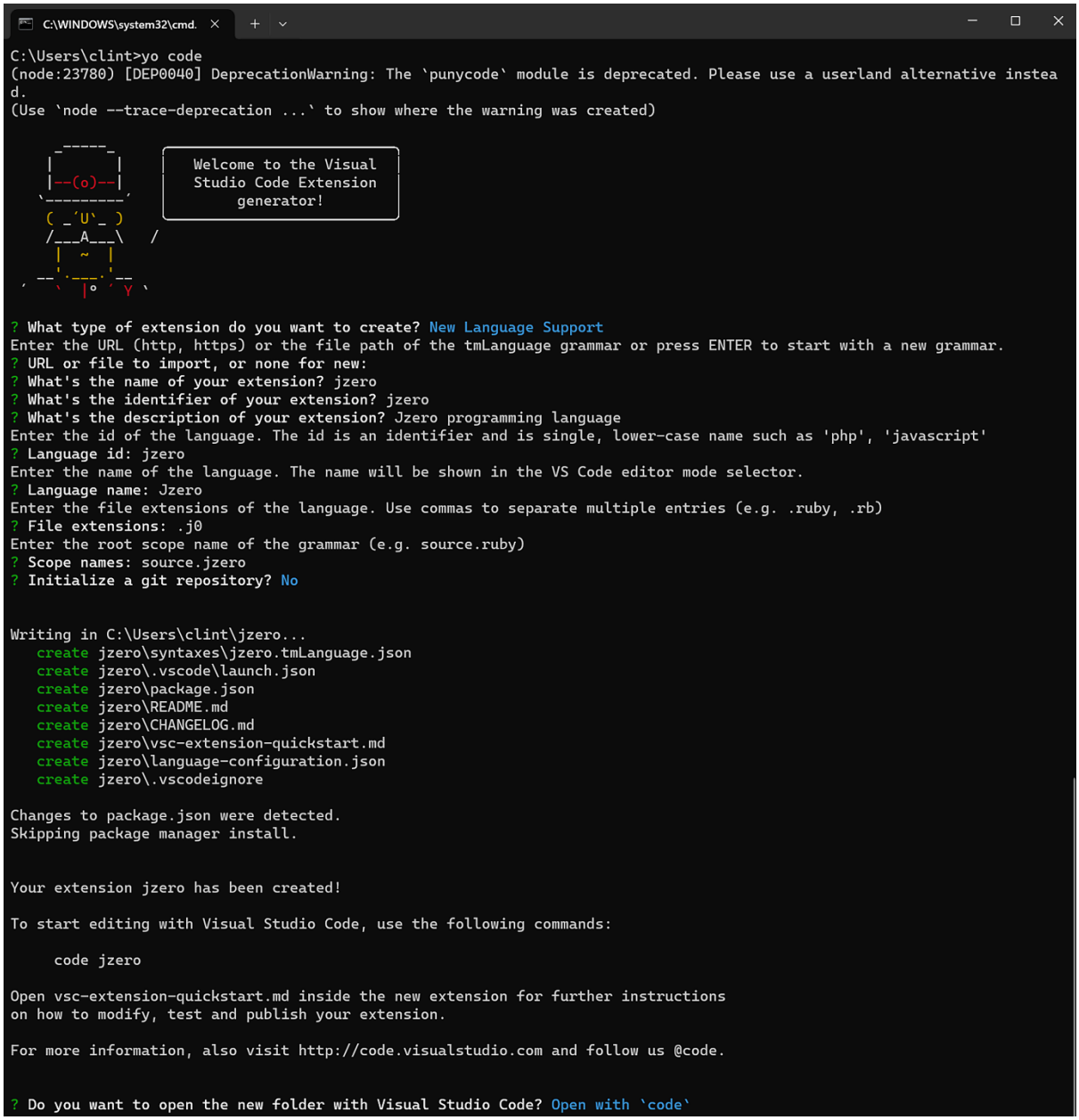
Production	Semantic Rules
WhileStmt : while '(' Expr ')' Stmt	Expr.onTrue = genlabel(); Expr.first = genlabel(); Expr.false = WhileStmt.follow; Stmt.follow = Expr.first; WhileStmt.icode = gen(LABEL, Expr.first) ||| Expr.icode ||| gen(LABEL, Expr.true) ||| Stmt.icode ||| gen(GOTO, Expr.first)

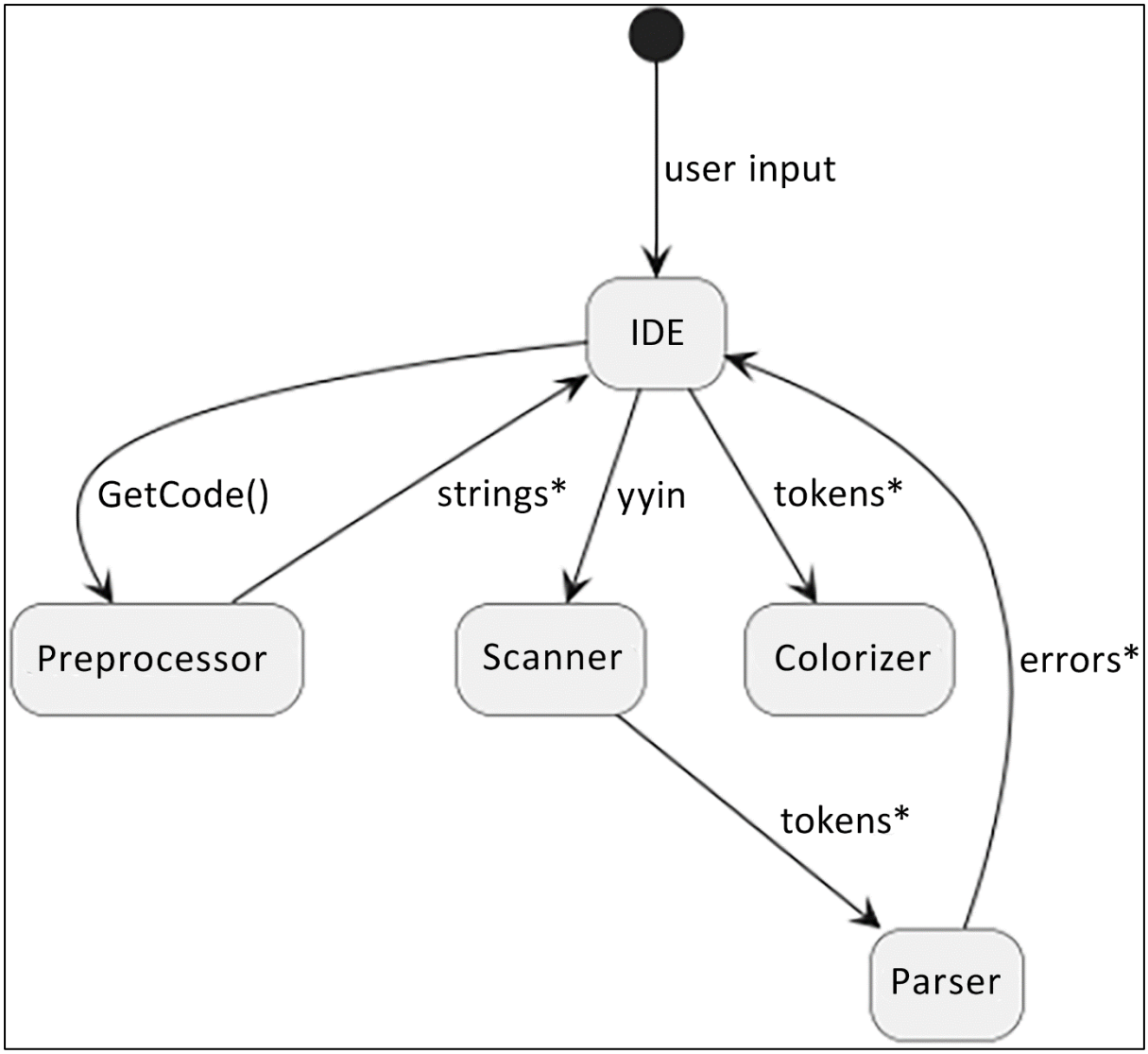
ForStmt : for(ForInit; Expr; ForUpdate)
Stmt
 	
a.k.a.
 	
ForInit;
while (Expr) {
Stmt
ForUpdate
}
	Expr.true = genlabel();
	Expr.first = genlabel();
	Expr.false = S.follow;
	Stmt.follow = ForUpdate.first;
	S.icode = ForInit.icode |||
	gen(LABEL, Expr.first) |||
	Expr.icode ||| gen(LABEL, Expr.true) |||
	Stmt.icode |||
	ForUpdate.icode |||
	gen(GOTO, Expr.first)

Chapter 10: Syntax Coloring in an IDE

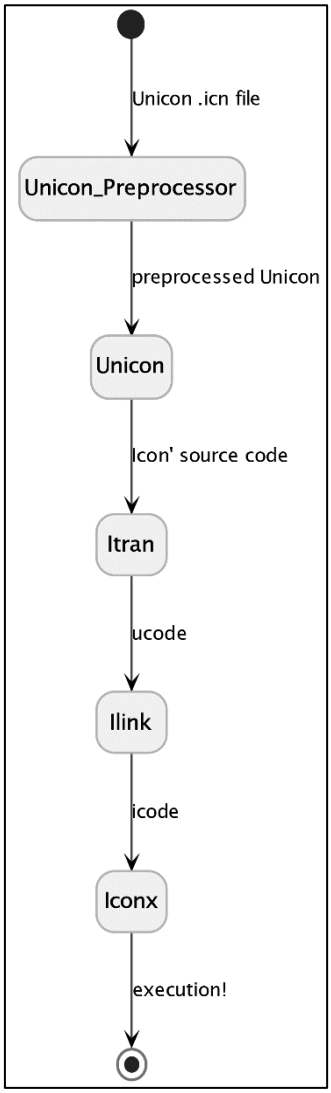



Search
Welcome
hello.j0
hello.java
C: > Users > clint > books > byopl2 > Build-your-own-Programming-Language-Second-Edition > ch10 > hello.j0
1 public class hello {
2 public static void main(String argv[]) {
3 int x;
4 x = argv.length;
5 x = x + 2;
6 while (x > 3) {
7 System.out.println("hello, jzero!");
8 x = x - 1;
9 }
10 }
11 }
Ln 4, Col 1
Spaces: 3
UTF-8
CRLF
Plain Text


C:\WINDOWS\system32\cmd.
C:\Users\clint>yo code
(node:23780) [DEP0040] DeprecationWarning: The `punycode` module is deprecated. Please use a userland alternative instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
Welcome to the Visual Studio Code Extension generator!
? What type of extension do you want to create? New Language Support
Enter the URL (http, https) or the file path of the tmLanguage grammar or press ENTER to start with a new grammar.
? URL or file to import, or none for new:
? What's the name of your extension? jzero
? What's the identifier of your extension? jzero
? What's the description of your extension? Jzero programming language
Enter the id of the language. The id is an identifier and is single, lower-case name such as 'php', 'javascript'
? Language id: jzero
Enter the name of the language. The name will be shown in the VS Code editor mode selector.
? Language name: Jzero
Enter the file extensions of the language. Use commas to separate multiple entries (e.g. .ruby, .rb)
? File extensions: .j0
Enter the root scope name of the grammar (e.g. source.ruby)
? Scope names: source.jzero
? Initialize a git repository? No
Writing in C:\Users\clint\jzero...
create jzero\syntaxes\jzero.tmLanguage.json
create jzero\.vscode\launch.json
create jzero\package.json
create jzero\README.md
create jzero\CHANGELOG.md
create jzero\vsc-extension-quickstart.md
create jzero\language-configuration.json
create jzero\.vscodeignore
Changes to package.json were detected.
Skipping package manager install.
Your extension jzero has been created!
To start editing with Visual Studio Code, use the following commands:
code jzero
Open vsc-extension-quickstart.md inside the new extension for further instructions
on how to modify, test and publish your extension.
For more information, also visit http://code.visualstudio.com and follow us @code.
? Do you want to open the new folder with Visual Studio Code? Open with `code`


user input
IDE
GetCode()
strings*
yyin
tokens*
Preprocessor
Scanner
Colorizer
errors*
tokens*
Parser

Chapter 11: Preprocessors and Transpilers



Production	Semantic Rules
Assignment : IDENT '=' AddExpr	`Assignment.icncode := [ IDENT.text || " := " ] ||| AddExpr.icncode`
AddExpr : AddExpr_1 '+' MulExpr	`AddExpr.icncode := AddExpr`$_1$`.icncode ||| ["+"] ||| MulExpr.icncode`
AddExpr : AddExpr_1 '-' MulExpr	`AddExpr.icncode := AddExpr`$_1$`.icncode ||| ["-"] ||| MulExpr.icncode`
MulExpr : MulExpr_1 '*' UnaryExpr	`MulExpr.icncode := MulExpr`$_1$`.icncode ||| ["*"] ||| UnaryExpr.icncode`
MulExpr : MulExpr_1 '/' UnaryExpr	`MulExpr.icncode := MulExpr`$_1$`.icncode ||| ["/"] ||| UnaryExpr.icncode`
UnaryExpr : '-' UnaryExpr_1	`UnaryExpr.icncode := ["-"] ||| UnaryExpr`$_1$`.icncode`
UnaryExpr : '(' AddExpr ')'	`UnaryExpr.icncode := ["("] ||| AddExpr.icncode ||| [")"]`
UnaryExpr : IDENT	`UnaryExpr.icncode := [IDENT.text]`

Production	Semantic Rules
MethodCall : Name (ArgList)	`MethodCall.icncode := Name.icncode ||| ["("] ||| ArgList.icncode ||| [")"]`
MethodCall : Primary . IDENT (ArgList)	`MethodCall.icncode := Name.icncode ||| ["." || IDENT.text || "("] ||| ArgList.icncode ||| [")"]`

Production	Semantic Rules
IfThenStmt : if '(' Expr ')' Stmt	`IfThenStmt.icncode := [ "if " ] ||| Expr.icncode ||| [" then "] ||| Stmt.icncode`
IfThenElseStmt : if '(' Expr ')' $Stmt_1$ else $Stmt_2$	`IfThenElseStmt.icncode := [ "if " ] ||| Expr.icncode ||| [" then "] ||| Stmt1.icncode ||| [" else "] ||| Stmt2.icncode`
WhileStmt : while '(' Expr ')' Stmt	`WhileStmt.icncode := [ "while " ] ||| Expr.icncode ||| [" do "] ||| Stmt.icncode`

Production	Semantic Rules
RelExpr : $RelExpr_1$ RelOp AddExpr	`RelExpr.icncode := RelExpr1.icncode ||| [RelOp.text] ||| AddExpr.icncode`
AndExpr : $AndExpr_1$ && EqExpr	`AndExpr.icncode := AndExpr1.icncode ||| ["&"] ||| EqExpr.icncode`
OrExpr : $OrExpr_1$ || AndExpr	`OrExpr.icncode := OrExpr1.icncode ||| ["|"] ||| AndExpr.icncode`
UnaryExpr : ! $UnaryExpr_1$	`UnaryExpr.icncode := ["~"] ||| UnaryExpr1.icncode`

Production	Semantic Rules
ClassDecl : public class name ClassBody	`ClassDecl.icncode := ["package " || name.text] ||| ClassBody.staticcode ||| ["class " || name.text || "()"] ||| ClassBody.icncode` `ClassDecl.helper := ["procedure main(argv)", "  " || name.text || "__main ! argv", "end"]`

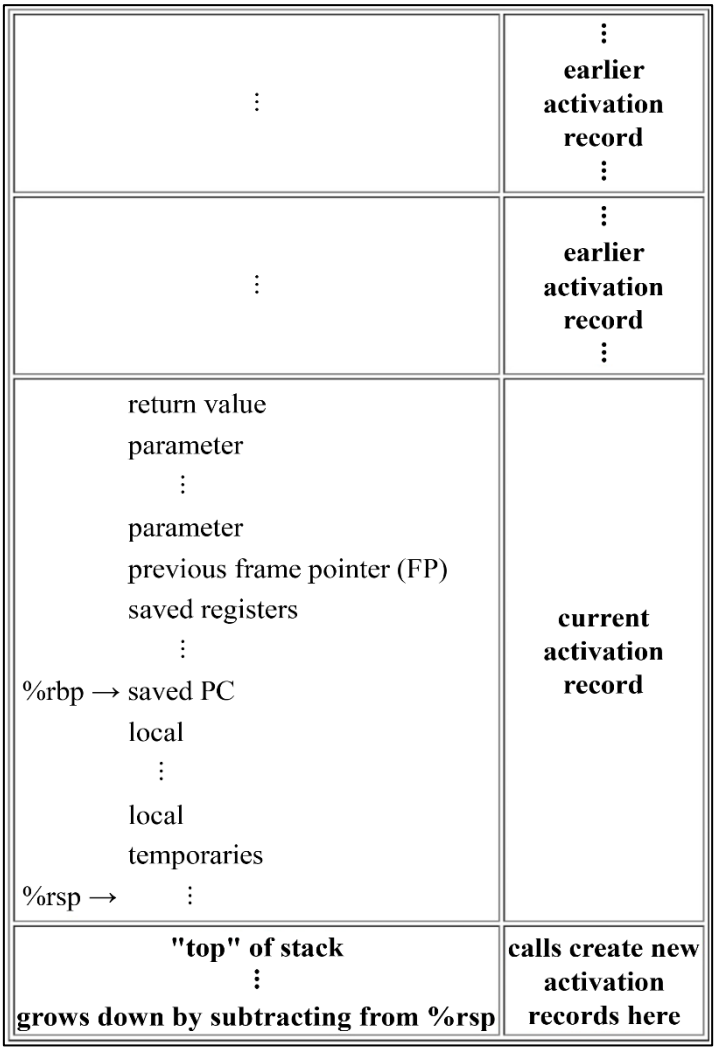
Chapter 12: Bytecode Interpreters

0x00 J	0x01 z	0x02 e	0x03 r	0x04 o	0x05 !	0x06 !	0x07 "\x00"
0x08 1	0x09 .	0x0a 0	0x0b "\x00"	0x0c "\x00"	0x0d "\x00"	0x0e "\x00"	0x0f "\x00"
0x10 "\x00"	0x11 "\x00"	0x12 "\x00"	0x13 "\x00"	0x14 "\x00"	0x15 "\x00"	0x16 "\x00"	0x17 "\x03"
0x18 "\x01"	0x19 "\x00"	0x1a "\x00"	0x1b "\x00"	0x1c "\x00"	0x1d "\x00"	0x1e "\x00"	0x1f "\x00"

Opcode	Mnemonic	Description
1	HALT	Halt
2	NOOP	Do nothing
3	ADD	Add the top two integers on the stack, push the sum
4	SUB	Subtract the top two integers on the stack, push the difference
5	MUL	Multiply the top two integers on the stack, push the product
6	DIV	Divide the top two integers on the stack, push the quotient
7	MOD	Divide the top two integers on the stack, push the remainder
8	NEG	Negate the integer at the top of the stack
9	PUSH	Push a value from memory to the top of the stack
10	POP	Pop a value from the top of the stack and place it in memory
11	CALL	Call a function with n parameters on the stack
12	RETURN	Return to the caller with a return value of x
13	GOTO	Set the instruction pointer to location L
14	BIF	Pop the stack; if it is non-zero, set the instruction pointer to L
15	LT	Pop two values, compare, push 1 if less than, else 0
16	LE	Pop two values, compare, push 1 if less or equal, else 0
17	GT	Pop two values, compare, push 1 if greater than, else 0
18	GE	Pop two values, compare, push 1 if greater or equal, else 0
19	EQ	Pop two values, compare, push 1 if equal, else 0
20	NEQ	Pop two values, compare, push 1 if not equal, else 0
21	LOCAL	Allocate n words on the stack
22	LOAD	Indirect push; reads through a pointer
23	STORE	Indirect pop; writes through a pointer

Table ..png

Chapter 14: Native Code Generation




```
01111111 01000101 01001100 01000110 00000010 00000001  .ELF..
00000001 00000000 00000000 00000000 00000000 00000000  ......
00000000 00000000 00000000 00000000 00000001 00000000  ......
00111110 00000000 00000001 00000000 00000000 00000000  >.....
00000000 00000000 00000000 00000000 00000000 00000000  ......
00000000 00000000 00000000 00000000 00000000 00000000  ......
00000000 00000000 00000000 00000000 00010000 00000010  ......
00000000 00000000 00000000 00000000 00000000 00000000  ......
00000000 00000000 00000000 00000000 01000000 00000000  ....@.
00000000 00000000 00000000 00000000 01000000 00000000  ....@.
00001011 00000000 00001010 00000000 01010101 01001000  ....UH
10001001 11100101 11000111 01000101 11111100 00000100  ...E..
00000000 00000000 00000000 10001011 01000101 11111100  ....E.
01011101 11000011 00000000 01000111 01000011 01000011  ]..GCC
00111010 00100000 00101000 01010101 01100010 01110101  : (Ubu
01101110 01110100 01110101 00100000 00110111 00101110  ntu 7.
00110101 00101110 00110000 00101101 00110011 01110101  5.0-3u
```

Instruction	Description
`addq`	Add a 64-bit into another 64-bit value
`call`	Store a return address to (%rsp), decrement %rsp, goto function
`cmpq`	Compare two values and set condition code bits
`goto`	Jump to a new location in the code
`jle`	Jump if less than or equal
`leaq`	Compute an address
`movq`	Move a 64-bit value from source to destination
`negq`	Negate a 64-bit value
`popq`	Fetch a value from (%rsp) and increment %rsp
`pushq`	Store a value to (%rsp) and decrement %rsp
`ret`	Fetch a value from (%rsp), increment %rsp and goto the address
`.global`	This symbol should be visible from other modules
`.text`	Place the bytes to follow in the code region
`.type`	This symbol is the following type

Table ..png

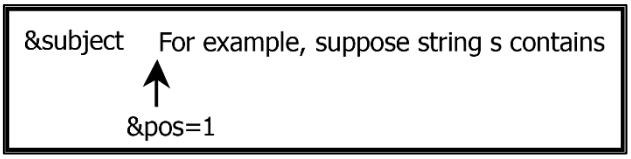
Access Mode	Description
`$k`	Immediate mode, value given in the instruction
`k(r)`	Indirect mode, fetch memory k bytes relative to register `r`

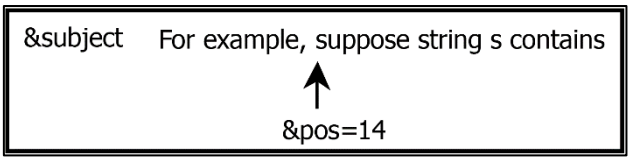
Table ..png

Register	Description/Role
`rip`	Instruction pointer.
`rax`	Accumulator. Also: function return value.
`rbx`	A secondary accumulator.
`rbp`	Frame pointer. Local variables are relative to this pointer.
`rsp`	Stack pointer. Memory between rbp and rsp is the local region.
`rdi`	Destination index. Holds parameter #1.
`rsi`	Source index. Holds parameter #2.
`rdx`	A secondary accumulator. Holds parameter #3.
`rcx`	Holds parameter #4.
`r8`	Holds parameter #5.
`r9`	Holds parameter #6.
`r10-r15`	Open registers usable for any purpose.

Table ..png

Chapter 16: Domain Control Structures

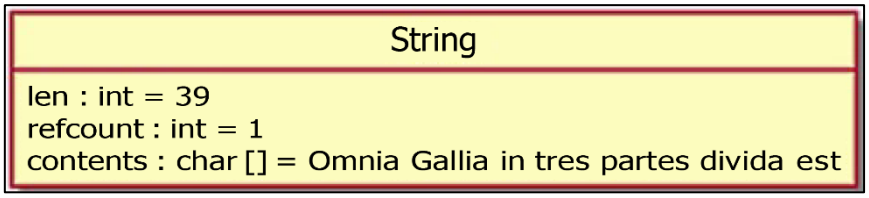


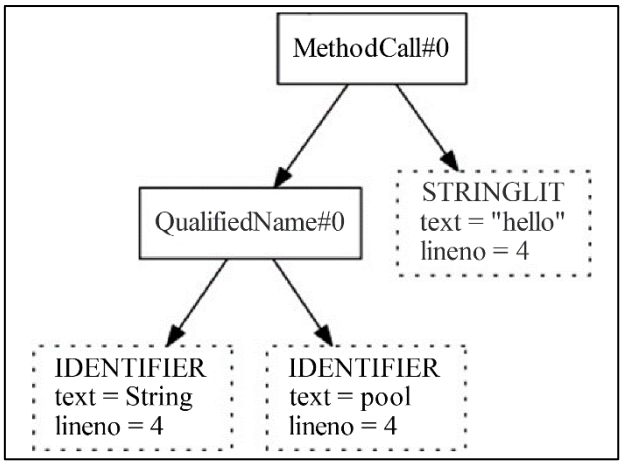


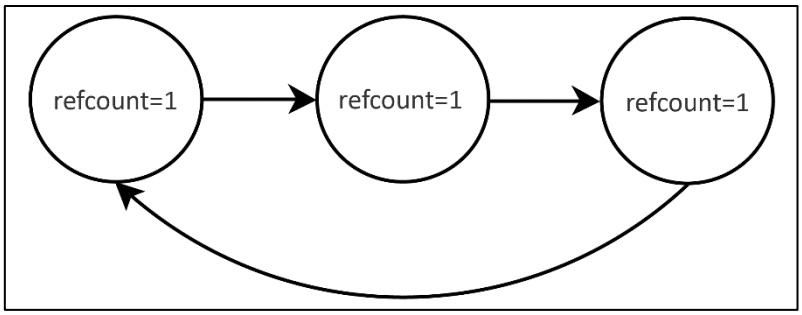
Function	Purpose
`any(C)`	Is the character at the current position a member of character set `C`?
`many(C)`	Are 1+ characters at the current position members of character set `C`?
`match(s)`	Do the characters at the current position match a search string `s`?
`find(s)`	At what positions do the characters match a search string `s`?
`upto(C)`	At what positions are characters found that are members of `C`?
`bal()`	At what positions are characters balanced with respect to delimiters?

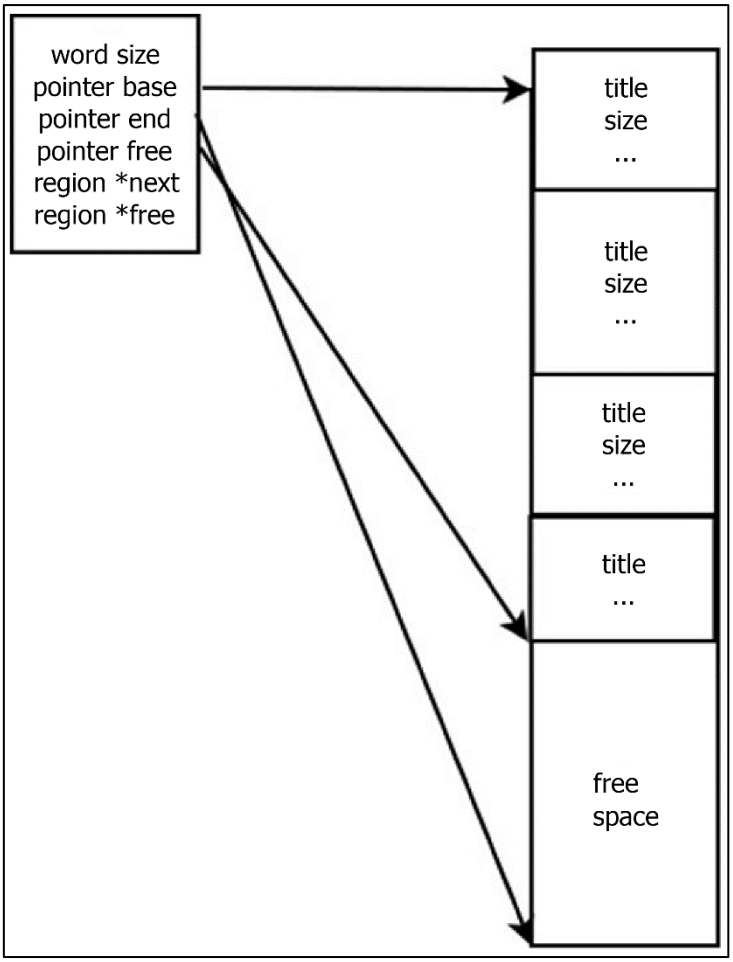
Production	Semantic Rule
wsection : WSECTION $expr_1$ DO $expr_2$	`wsection.code = "1(WSection(" || expr1 || "),{" || expr2|| ";WSection();1})"`

Chapter 17: Garbage Collection

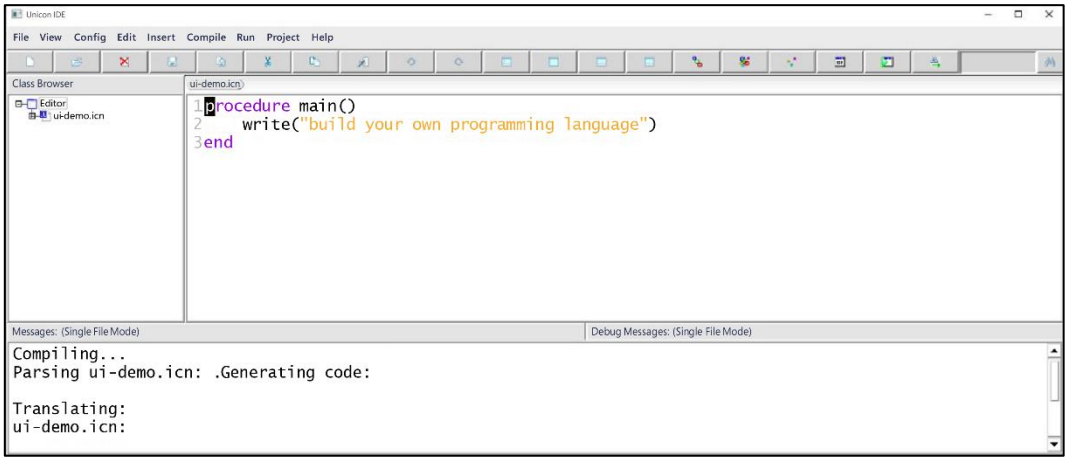







word size
pointer base
pointer end
pointer free
region *next
region *free
title
size
...
title
size
...
title
size
...
title
...
free
space

Appendix: Unicon Essentials



Code	Character	Code	Character	Code	Character	Code	Character
\b	backspace	\d	delete	\e	escape	\f	form feed
\l	line feed	\n	newline	\r	carriage return	\t	tab
\v	vertical tab	\’	quote	\"	double quote	\\	backslash
ooo	octal	\x*hh*	hexadecimal	\^*x*	Control-*x*		

Environment variable	Description
BLKSIZE	Bytes in the block heap
IPATH	List of directories to search for linking
LPATH	List of directories to search for includes
MSTKSIZE	Bytes on the main stack
STKSIZE	Bytes on co-expression stacks
STRSIZE	Bytes in the string heap
TRACE	Initial value of &trace

Defined macro	Meaning	Defined macro	Meaning
_CO_EXPRESSIONS	Synchronous threads	_MESSAGING	HTTP, SMTP, etc.
_CONSOLE_WINDOW	Emulated terminal	_MS_WINDOWS	Microsoft Windows
_DBM	DBM	_MULTITASKING	load(), etc.
_DYNAMIC_LOADING	Code can be loaded	_POSIX	POSIX
_EVENT_MONITOR	Code is instrumented	_PIPES	unidirectional pipes
_GRAPHICS	Graphics	_SYSTEM_FUNCTION	system()
_KEYBOARD_FUNCTIONS	kbhit(), getc(), etc.	_UNIX	UNIX, Linux, ...
_LARGE_INTEGERS	Arbitrary precision	_WIN32	Win32 graphics
_MACINTOSH	Macintosh	_X_WINDOW_SYSTEM	X Windows graphics

Mode letter(s)	Description	Mode letter(s)	Description
a	add/append	nl	listen on a TCP port
b	open for both reading and writing	nu	connect to a UDP port
c	make a new file	m	connect to messaging server
d	GDBM database	o	ODBC (SQL) connection
g	2D graphics window	p	execute a command line and pipe it
gl	3D graphics window	r	read
n	TCP client	t	translate newlines
na	accept TCP connection	u	use a binary untranslated mode
nau	accept UDP datagrams	w	write