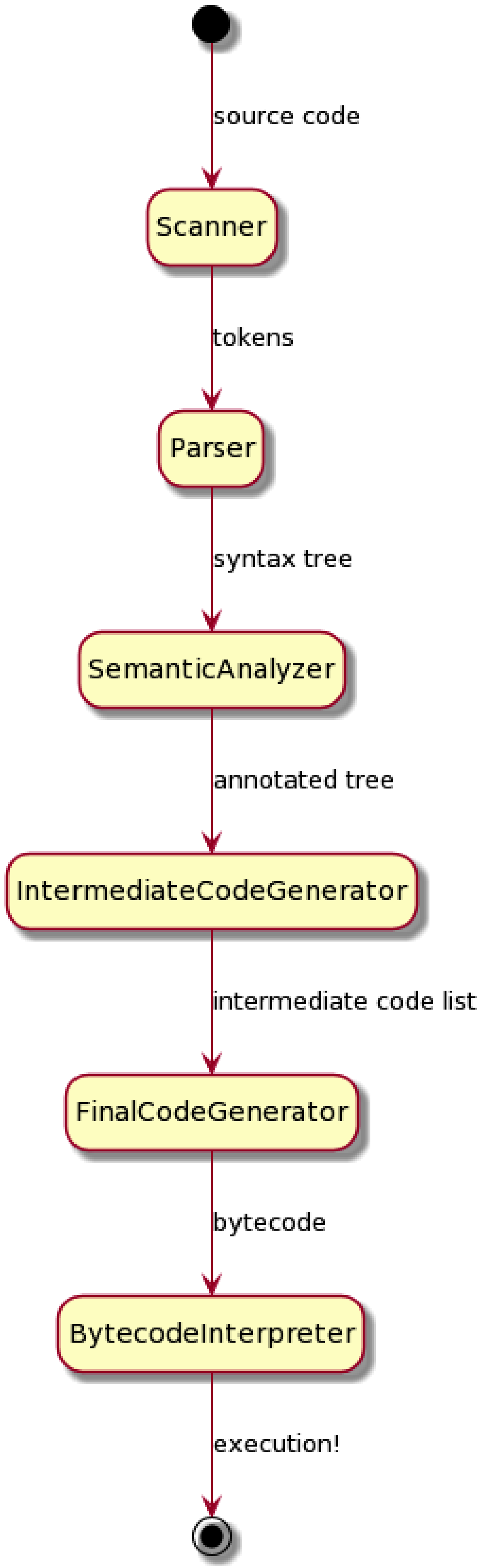
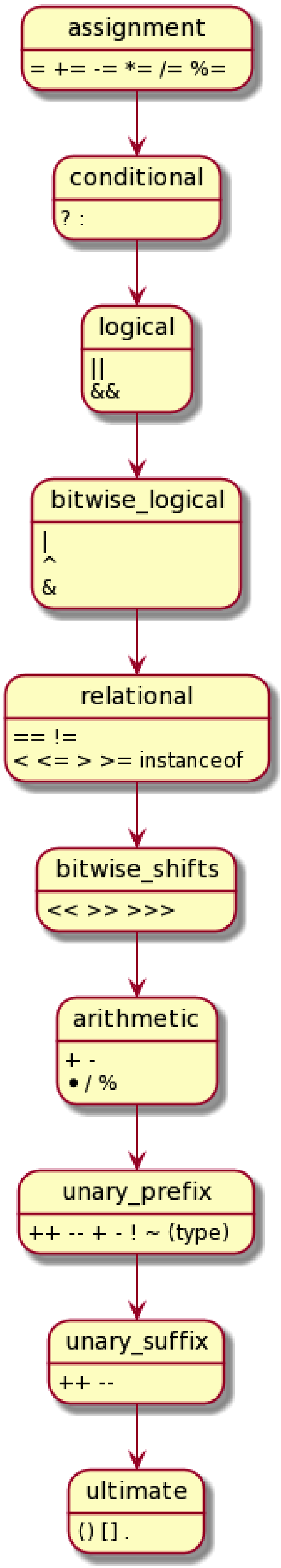
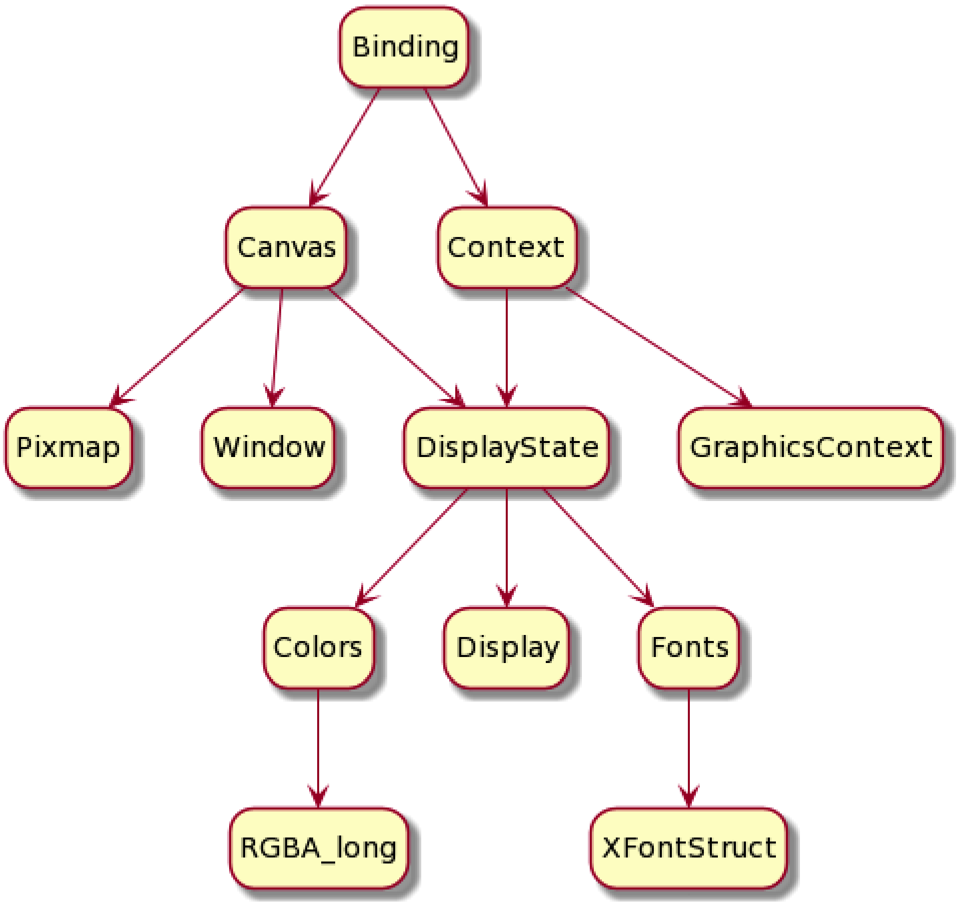
Chapter 1: Why Build Another Programming Language?

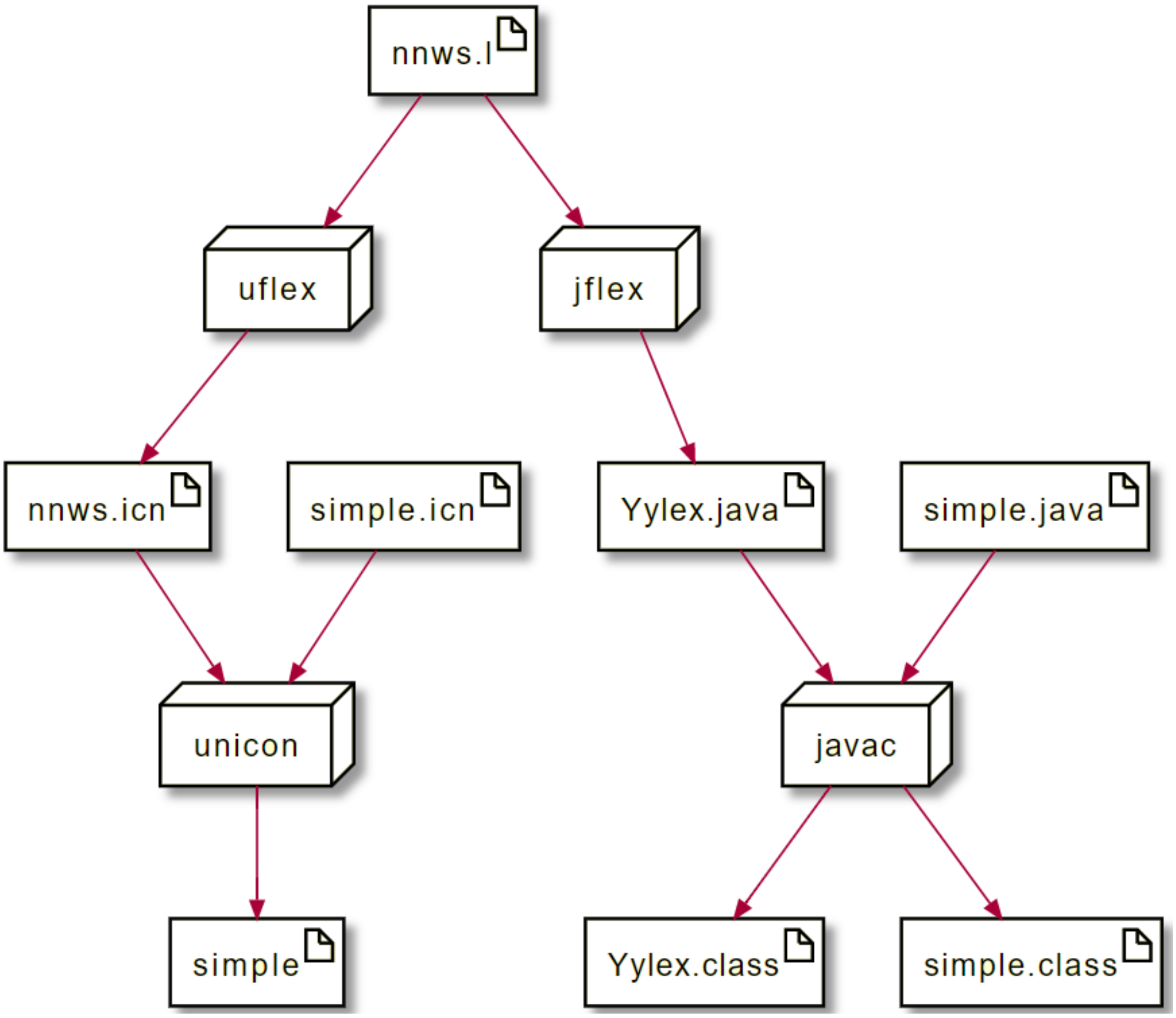


Chapter 2: Programming Language Design

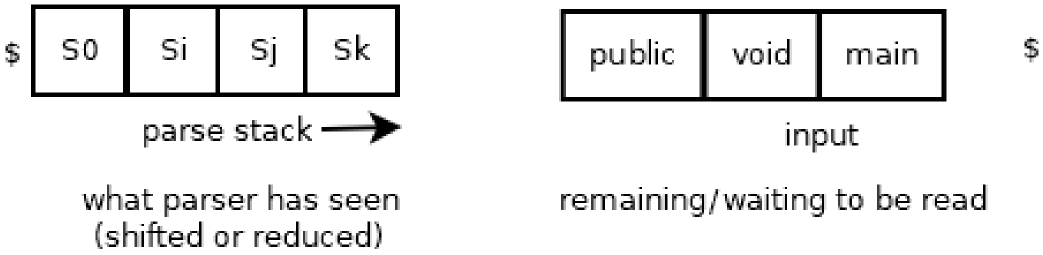



Binding
Canvas
Context
Pixmap
Window
DisplayState
GraphicsContext
Colors
Display
Fonts
RGBA_long
XFontStruct

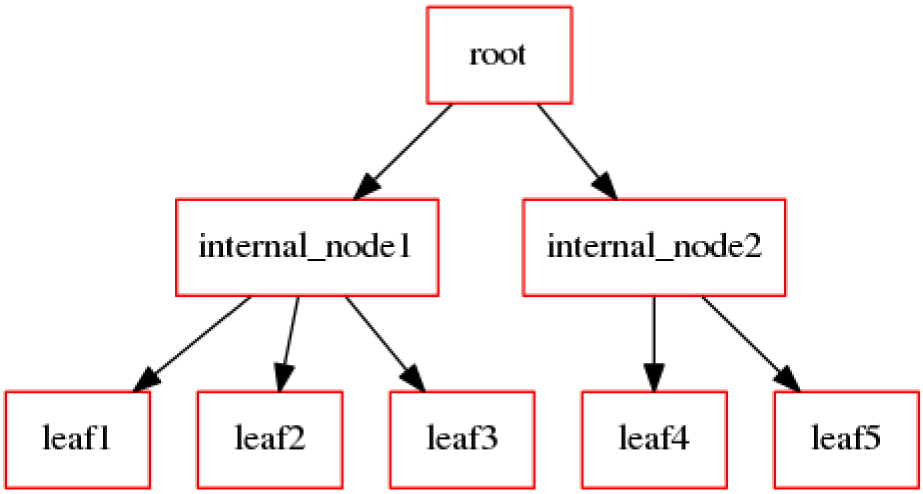
Chapter 3: Scanning Source Code

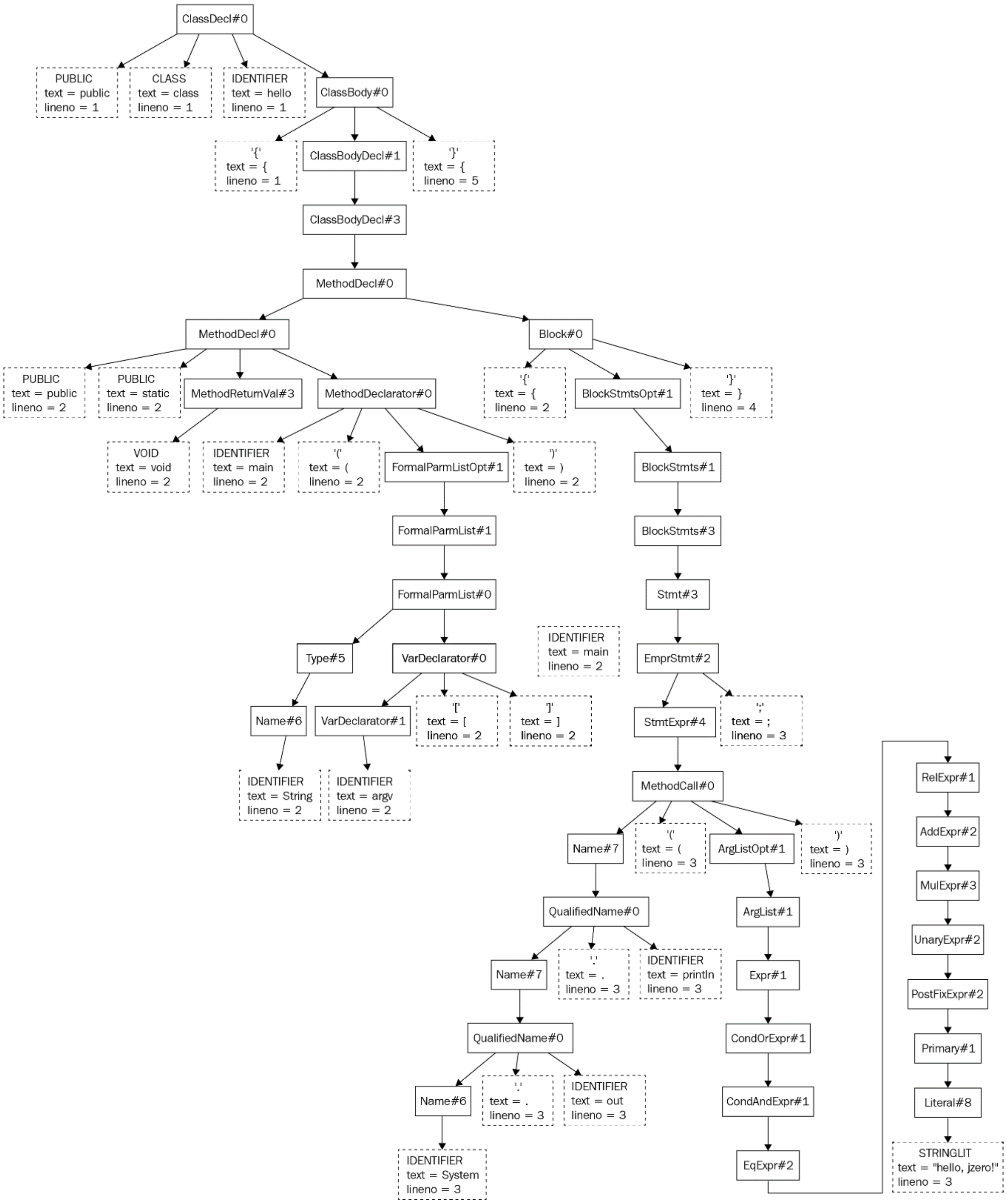


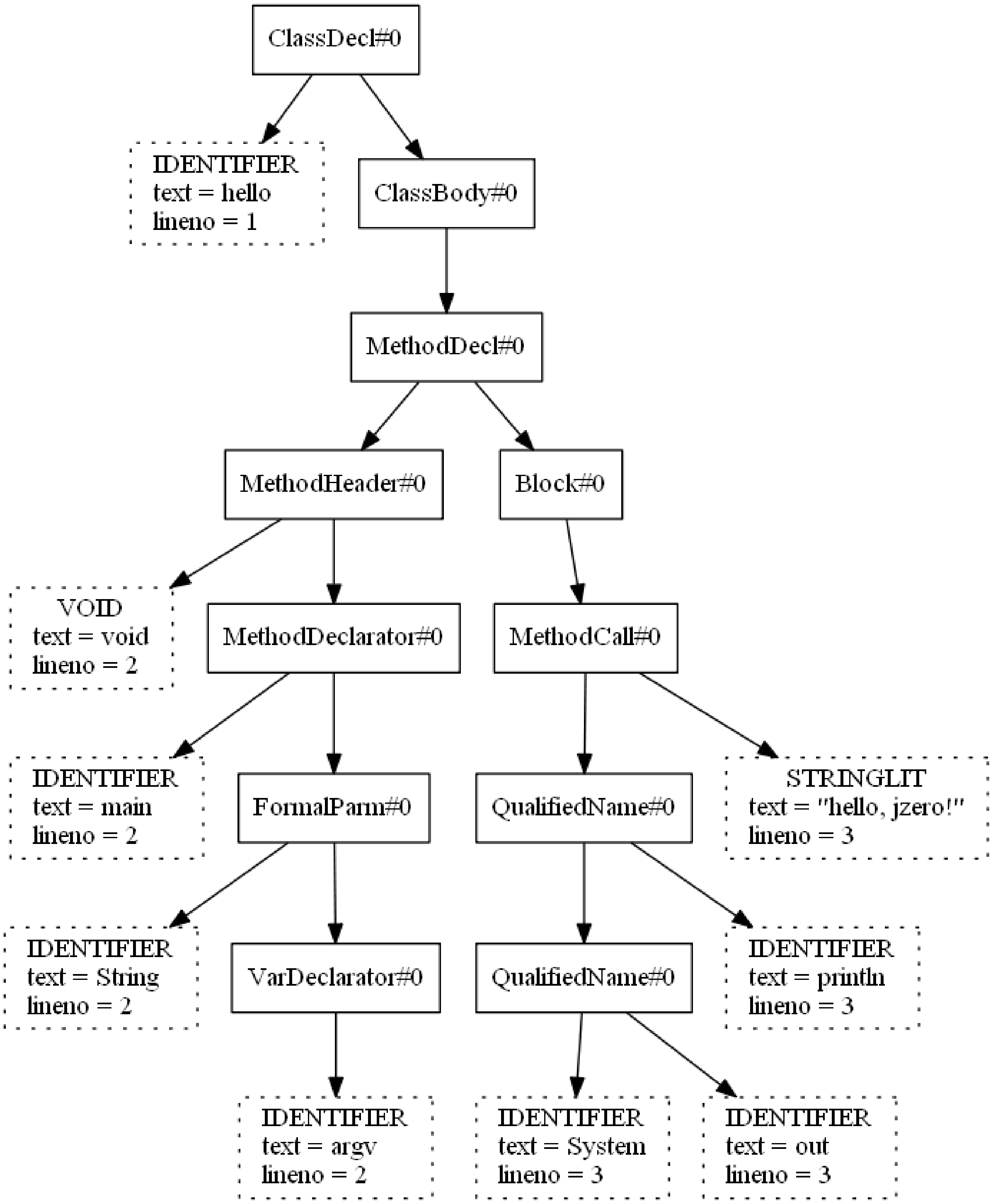
Chapter 4: Parsing

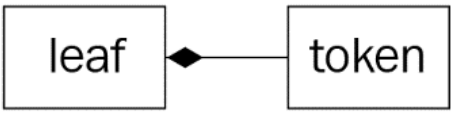


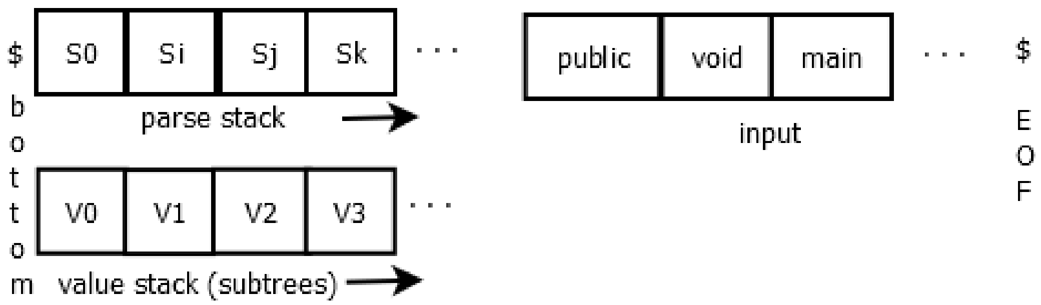
Chapter 5: Syntax Trees

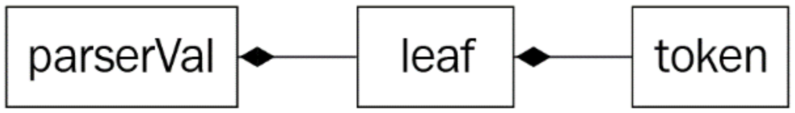


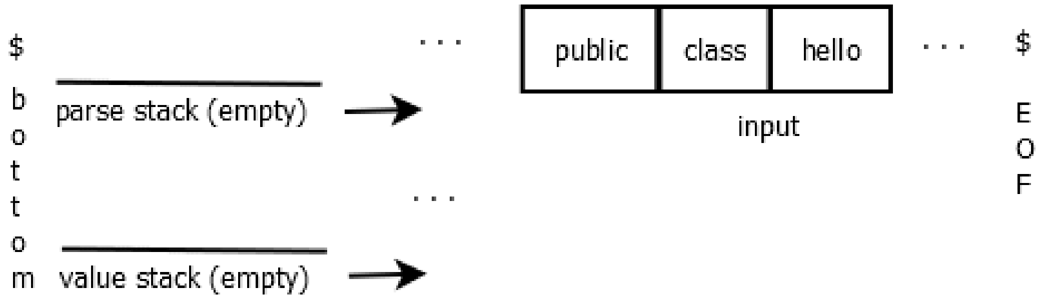

ClassDecl#0
PUBLIC text = public lineno = 1
CLASS text = class lineno = 1
IDENTIFIER text = hello lineno = 1
ClassBody#0
'{' text = { lineno = 1
ClassBodyDecl#1
'}' text = { lineno = 5
ClassBodyDecl#3
MethodDecl#0
MethodDecl#0
Block#0
PUBLIC text = public lineno = 2
PUBLIC text = static lineno = 2
MethodReturnVal#3
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

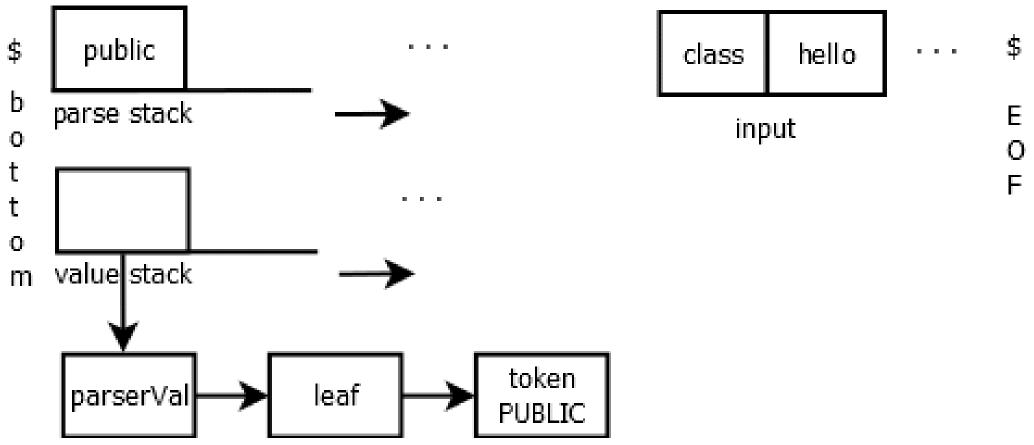

ClassDecl#0
IDENTIFIER
text = hello
lineno = 1
ClassBody#0
MethodDecl#0
MethodHeader#0
Block#0
VOID
text = void
lineno = 2
MethodDeclarator#0
MethodCall#0
IDENTIFIER
text = main
lineno = 2
FormalParm#0
QualifiedName#0
STRINGLIT
text = "hello, jzero!"
lineno = 3
IDENTIFIER
text = String
lineno = 2
VarDeclarator#0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

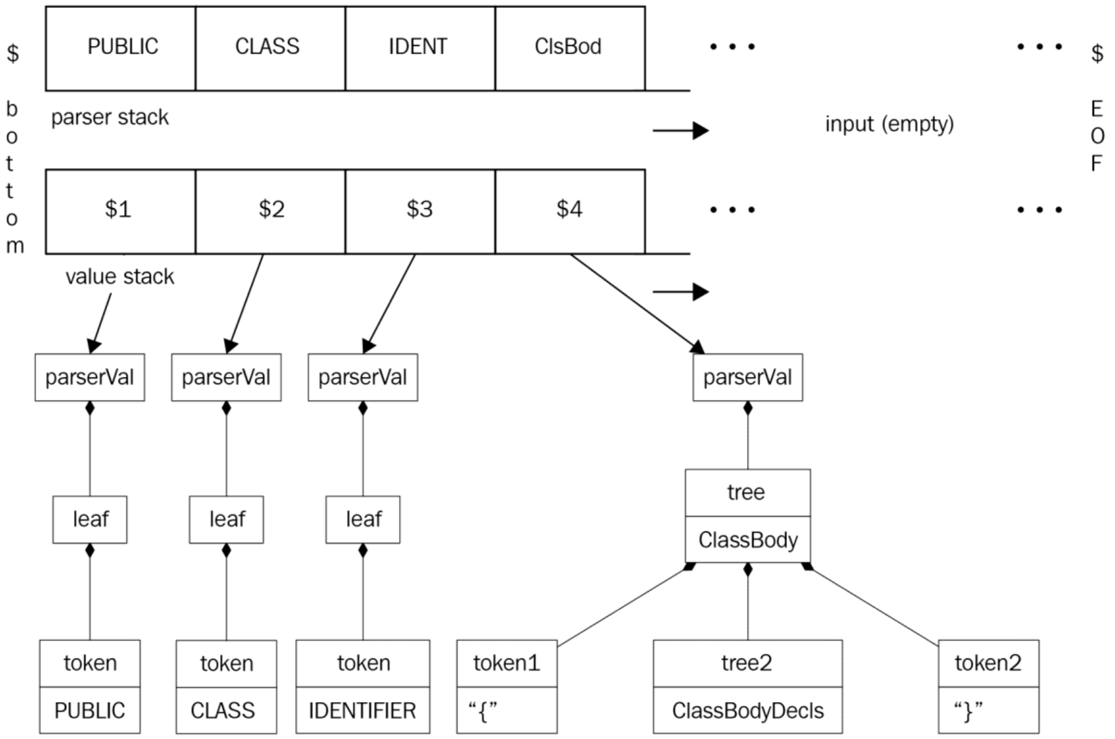

leaf
token

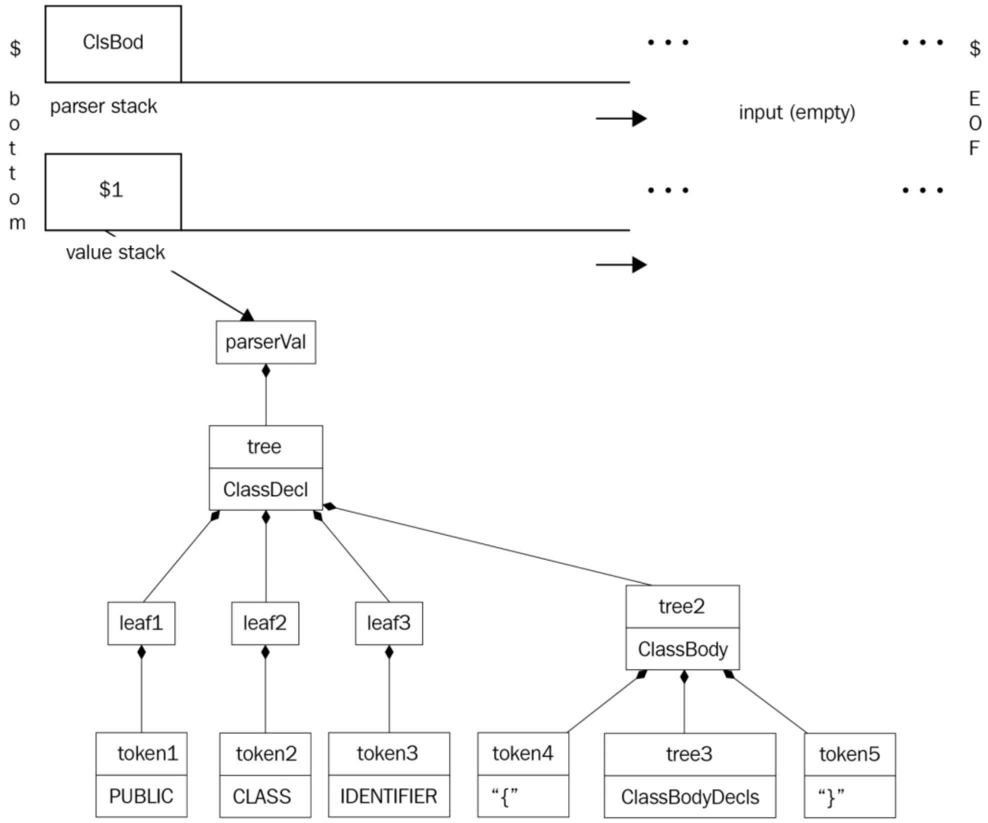

$
bottom
S0
Si
Sj
Sk
. . .
parse stack
V0
V1
V2
V3
. . .
value stack (subtrees)
public
void
main
. . .
input
$
EOF

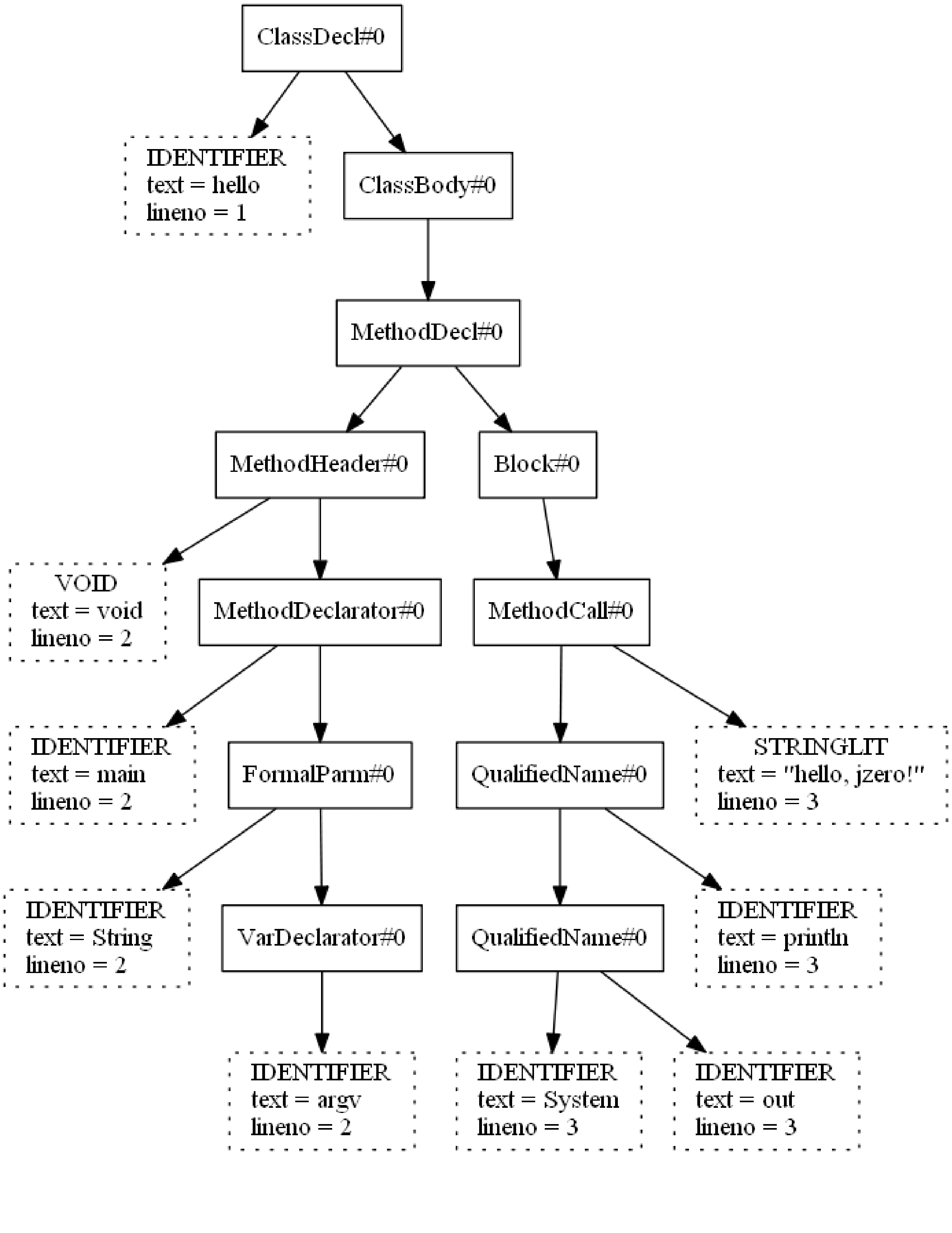

parserVal
leaf
token


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

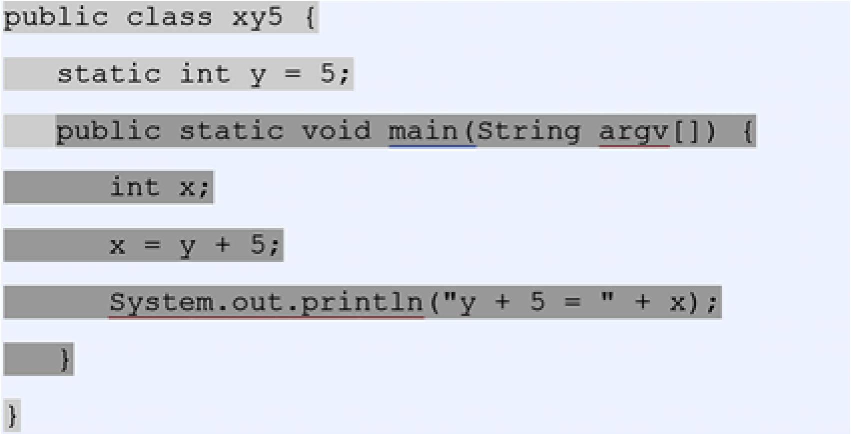

$
bottom
public
parse stack
. . .
value stack
. . .
parserVal
leaf
token
PUBLIC
class
hello
. . .
input
$
EOF


$
PUBLIC
CLASS
IDENT
ClsBod
• • •
• • •
$
bottom
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
"}"

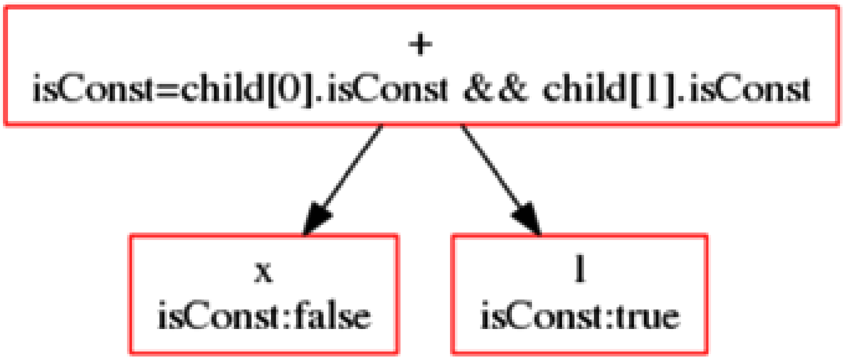

$
ClsBod
• • •
• • •
$
bottom
parser stack
input (empty)
EOF
$1
value stack
parserVal
tree
ClassDecl
leaf1
leaf2
leaf3
tree2
ClassBody
token1
PUBLIC
token2
CLASS
token3
IDENTIFIER
token4
"{"
tree3
ClassBodyDecls
token5
"}"

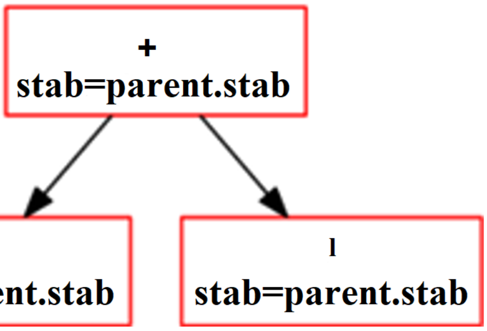

ClassDecl#0
IDENTIFIER
text = hello
lineno = 1
ClassBody#0
MethodDecl#0
MethodHeader#0
Block#0
VOID
text = void
lineno = 2
MethodDeclarator#0
MethodCall#0
IDENTIFIER
text = main
lineno = 2
FormalParm#0
QualifiedName#0
STRINGLIT
text = "hello, jzero!"
lineno = 3
IDENTIFIER
text = String
lineno = 2
VarDeclarator#0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

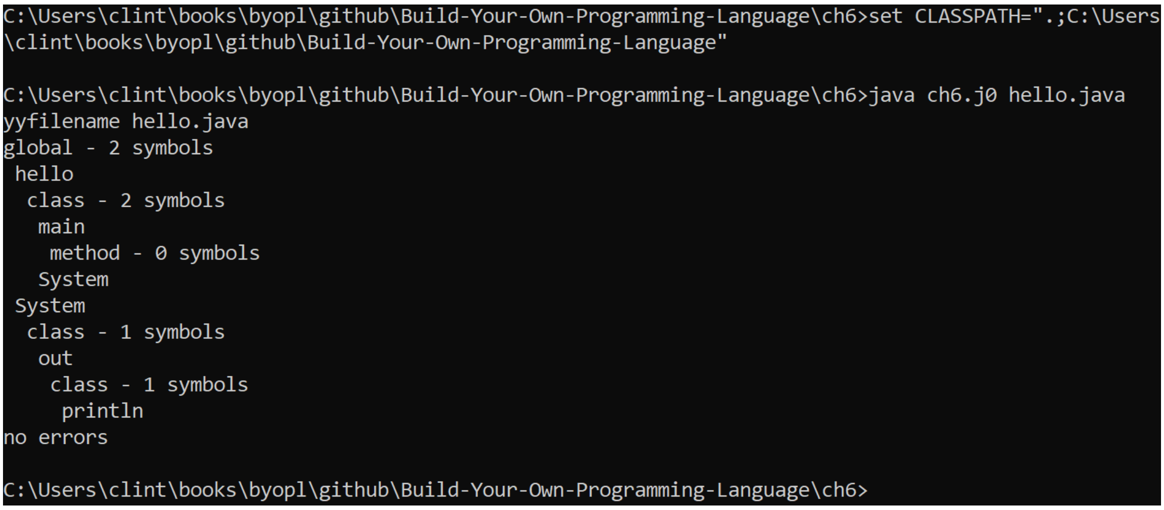
Chapter 6: Symbol Tables



```
public class xy5 {
   static int y = 5;
   public static void main(String argv[]) {
      int x;
      x = y + 5;
      System.out.println("y + 5 = " + x);
   }
}
```

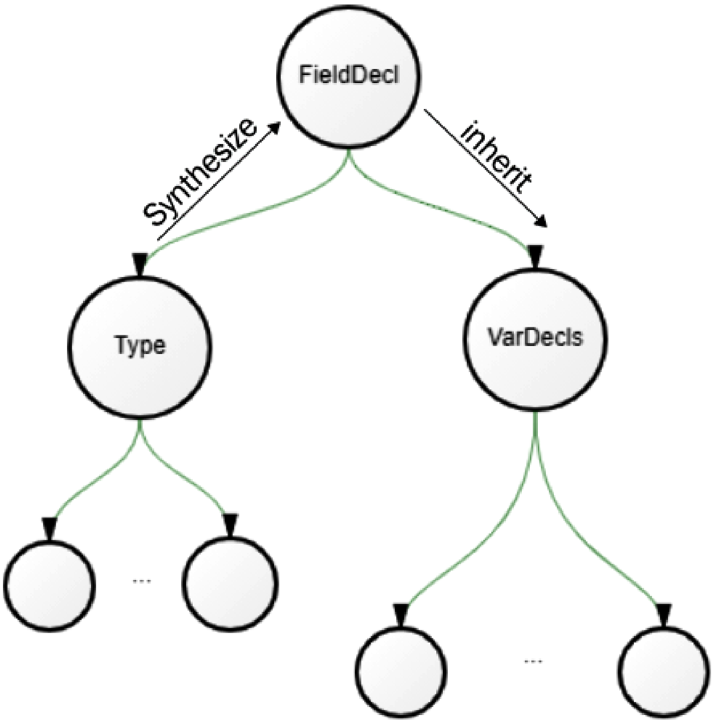


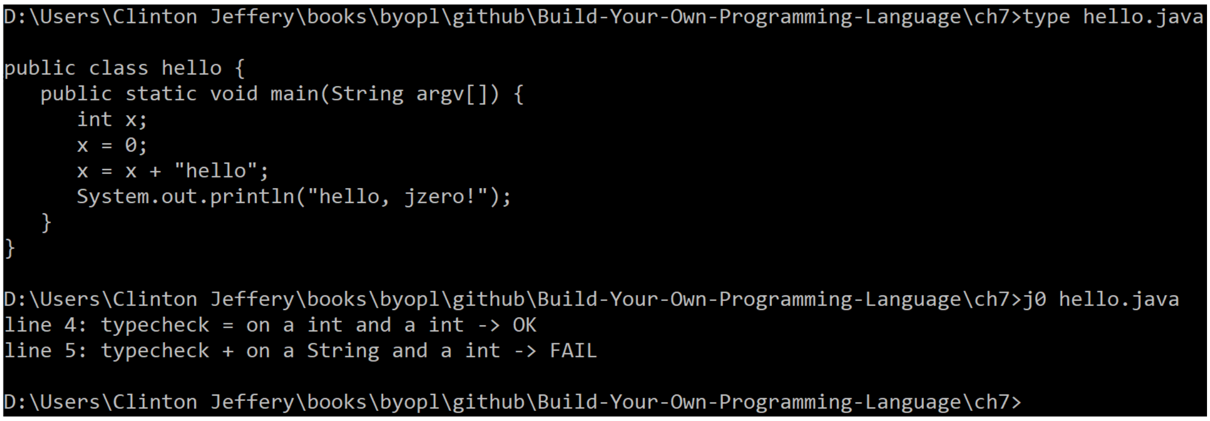




```
C:\Users\clint\books\byopl\github\Build-Your-Own-Programming-Language\ch6>set CLASSPATH=".;C:\Users
\clint\books\byopl\github\Build-Your-Own-Programming-Language"

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>
```

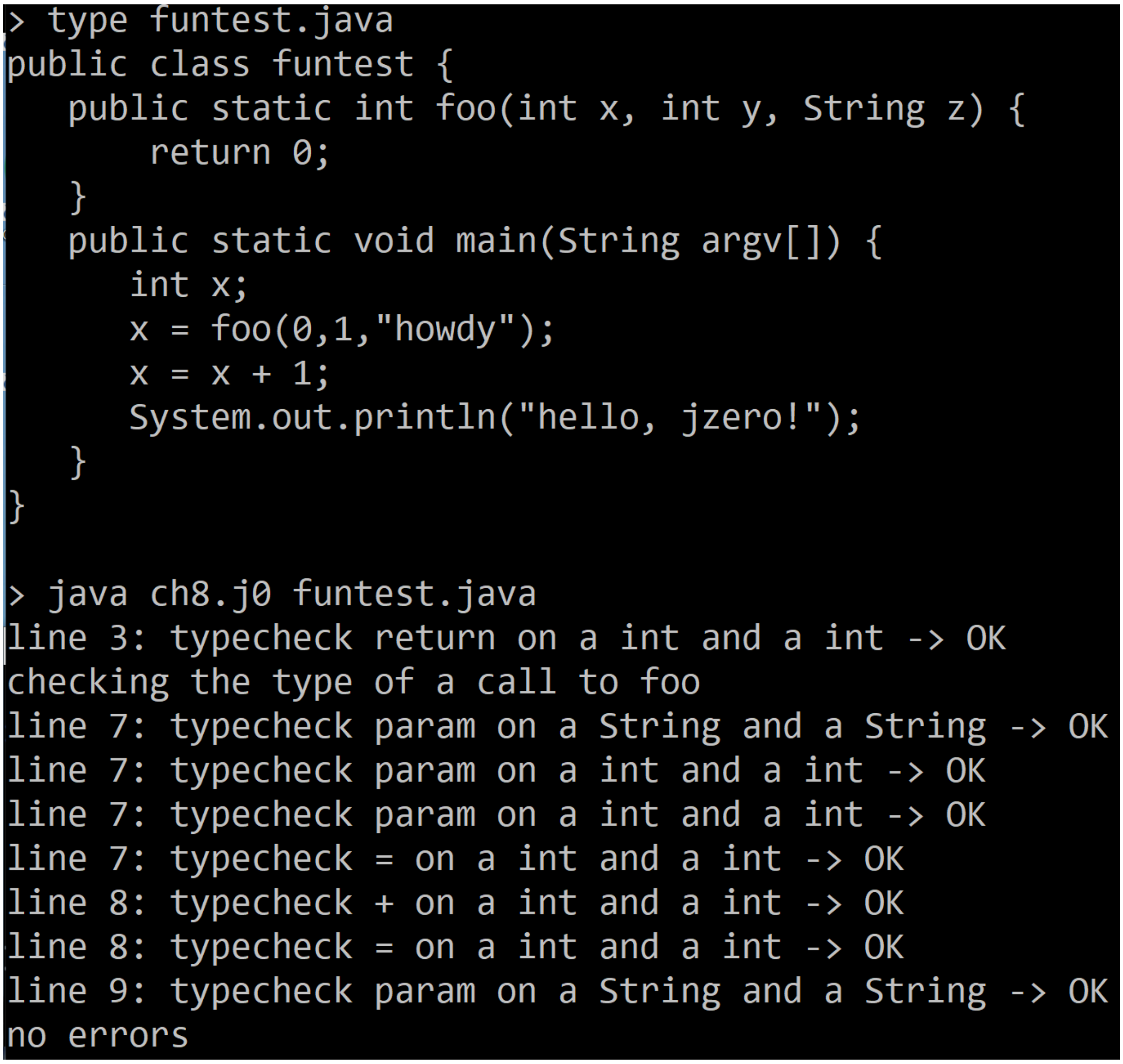
Chapter 7: Checking Base Types





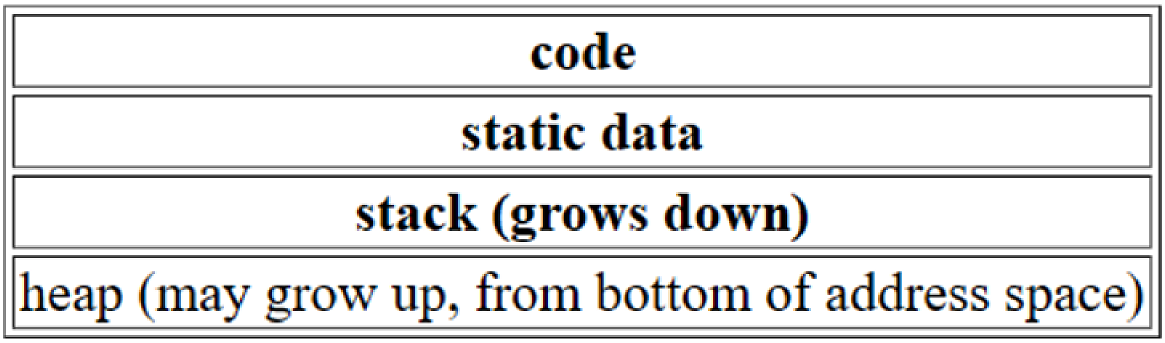
```
D:\Users\Clinton Jeffery\books\byopl\github\Build-Your-Own-Programming-Language\ch7>type hello.java

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
checking the type of a call to foo
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



code

static data

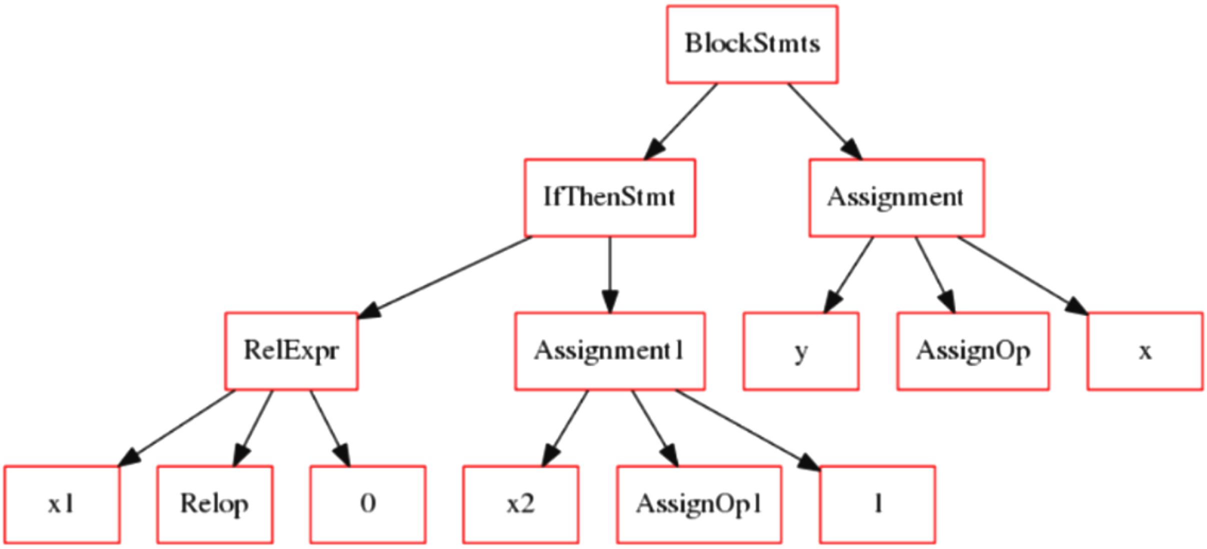
stack (grows down)

heap (may grow up, from bottom of address space)

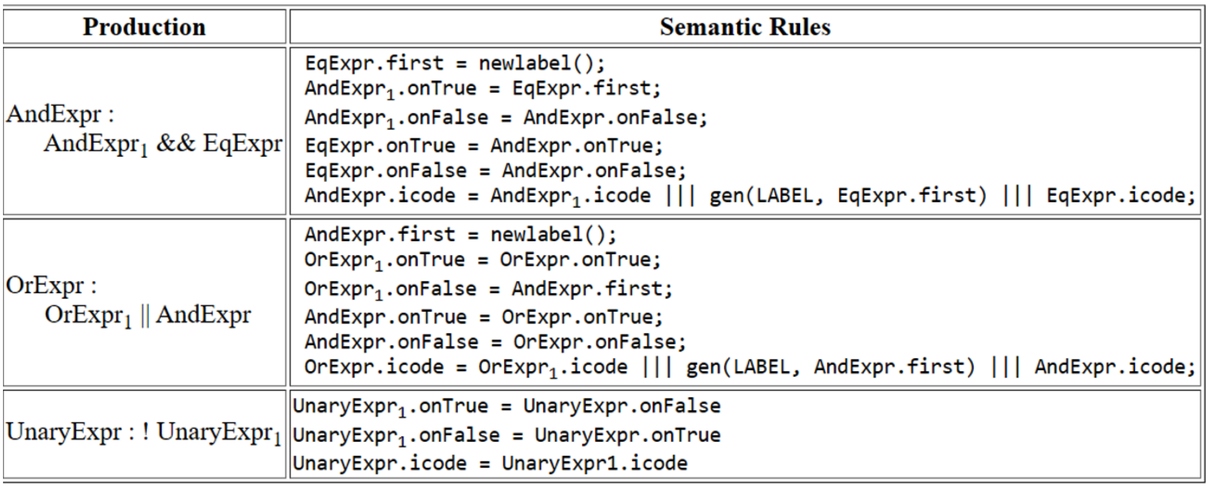
Opcode	C equivalent	Description
ADD,SUB,MUL,DIV	x=y op z	Store result of binary operation on y and z to x
NEG	x = -y	Store result of unary operation on y to x
ASN	x = y	Store y to x
ADDR	x = &y	Store address of y to x
LCON	x = *y	Store contents pointed to by y to x
SCON	*x = y	Store y to location pointed 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

<table>
<tr><th>Production</th><th>Semantic Rules</th></tr>
<tr><td>Assignment : IDENT '=' AddExpr</td><td>Assignment.addr = IDENT.addr
Assignment.icode = AddExpr.icode |||
gen(ASN, IDENT.addr, AddExpr.addr)</td></tr>
<tr><td>AddExpr : AddExpr$_1$ '+' MulExpr</td><td>AddExpr.addr = newtemp()
AddExpr.icode = AddExpr$_1$.icode ||| MulExpr.icode |||
gen(ADD,AddExpr.addr,AddExpr$_1$.addr,MulExpr.addr)</td></tr>
<tr><td>AddExpr : AddExpr$_1$ '-' MulExpr</td><td>AddExpr.addr = newtemp()
AddExpr.icode = AddExpr$_1$.icode ||| MulExpr.icode |||
gen(SUB,AddExpr.addr,AddExpr$_1$.addr,MulExpr.addr)</td></tr>
<tr><td>MulExpr : MulExpr$_1$ '*' UnaryExpr</td><td>MulExpr.addr = newtemp()
MulExpr.icode = MulExpr$_1$.icode ||| UnaryExpr.icode |||
gen(MUL,MulExpr.addr,MulExpr$_1$.addr,UnaryExpr.addr)</td></tr>
<tr><td>MulExpr : MulExpr$_1$ '/' UnaryExpr</td><td>MulExpr.addr = newtemp()
MulExpr.icode = MulExpr$_1$.icode ||| UnaryExpr.icode |||
gen(DIV,MulExpr.addr,MulExpr$_1$.addr,UnaryExpr.addr)</td></tr>
<tr><td>UnaryExpr : '-' UnaryExpr$_1$</td><td>UnaryExpr.addr = newtemp()
UnaryExpr.icode = UnaryExpr$_1$.icode |||
gen(NEG,UnaryExpr.addr,UnaryExpr$_1$.addr)</td></tr>
<tr><td>UnaryExpr : '(' AddExpr ')'</td><td>UnaryExpr.addr = AddExpr.addr
UnaryExpr.icode = AddExpr.icode</td></tr>
<tr><td>UnaryExpr : IDENT</td><td>UnaryExpr.addr = IDENT.addr
UnaryExpr.icode = emptylist()</td></tr>
</table>



<table>
<tr><th>Production</th><th>Semantic Rules</th></tr>
<tr><td>IfThenStmt :
if '(' Expr ')' Stmt</td><td>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</td></tr>
<tr><td>IfThenElseStmt :
if '(' Expr ')' Stmt$_1$ else Stmt$_2$</td><td>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</td></tr>
</table>

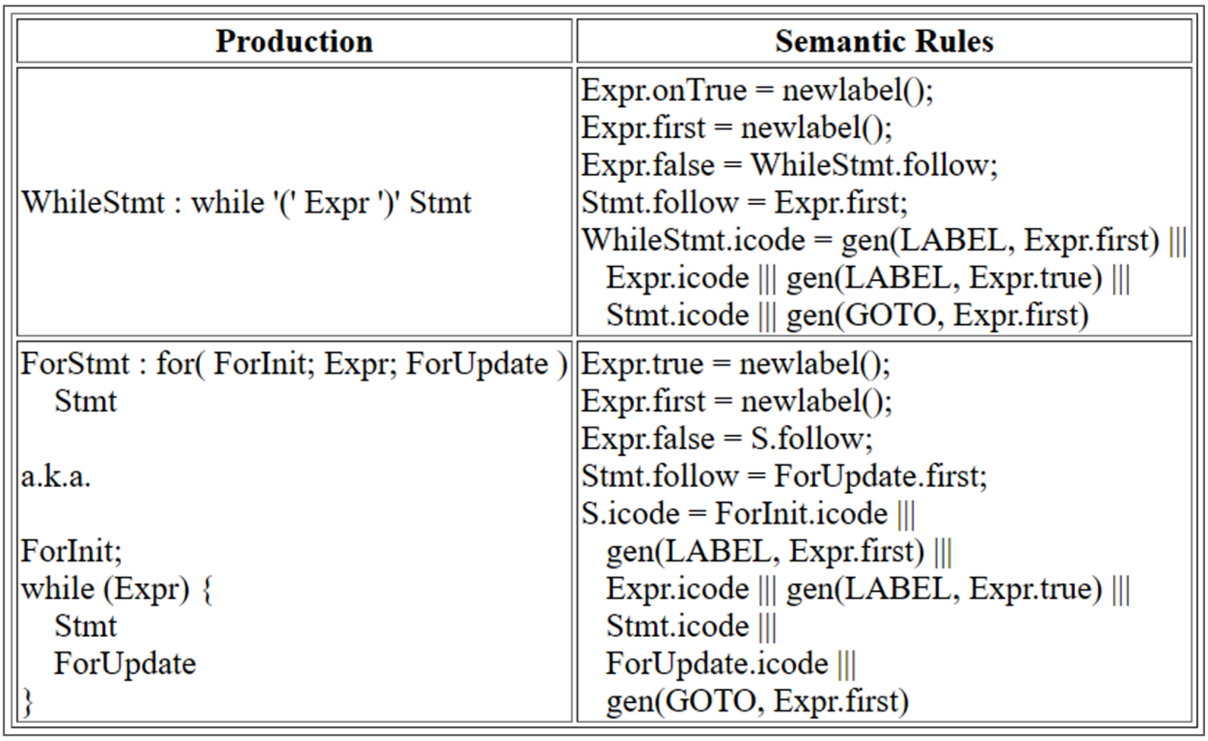


Production	Semantic Rules

AndExpr :
AndExpr$_1$ && EqExpr
	EqExpr.first = newlabel();
	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 = newlabel();
	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

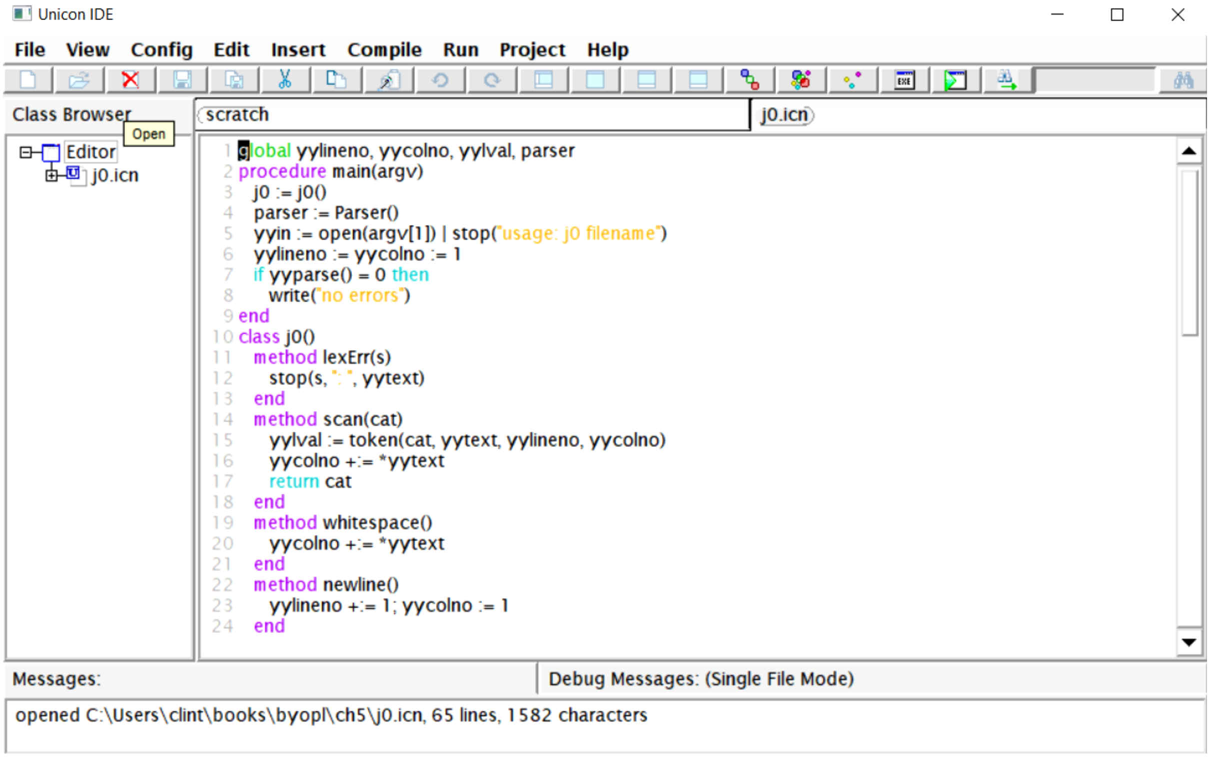


Production	Semantic Rules

WhileStmt : while '(' Expr ')' Stmt	Expr.onTrue = newlabel();
	Expr.first = newlabel();
	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)

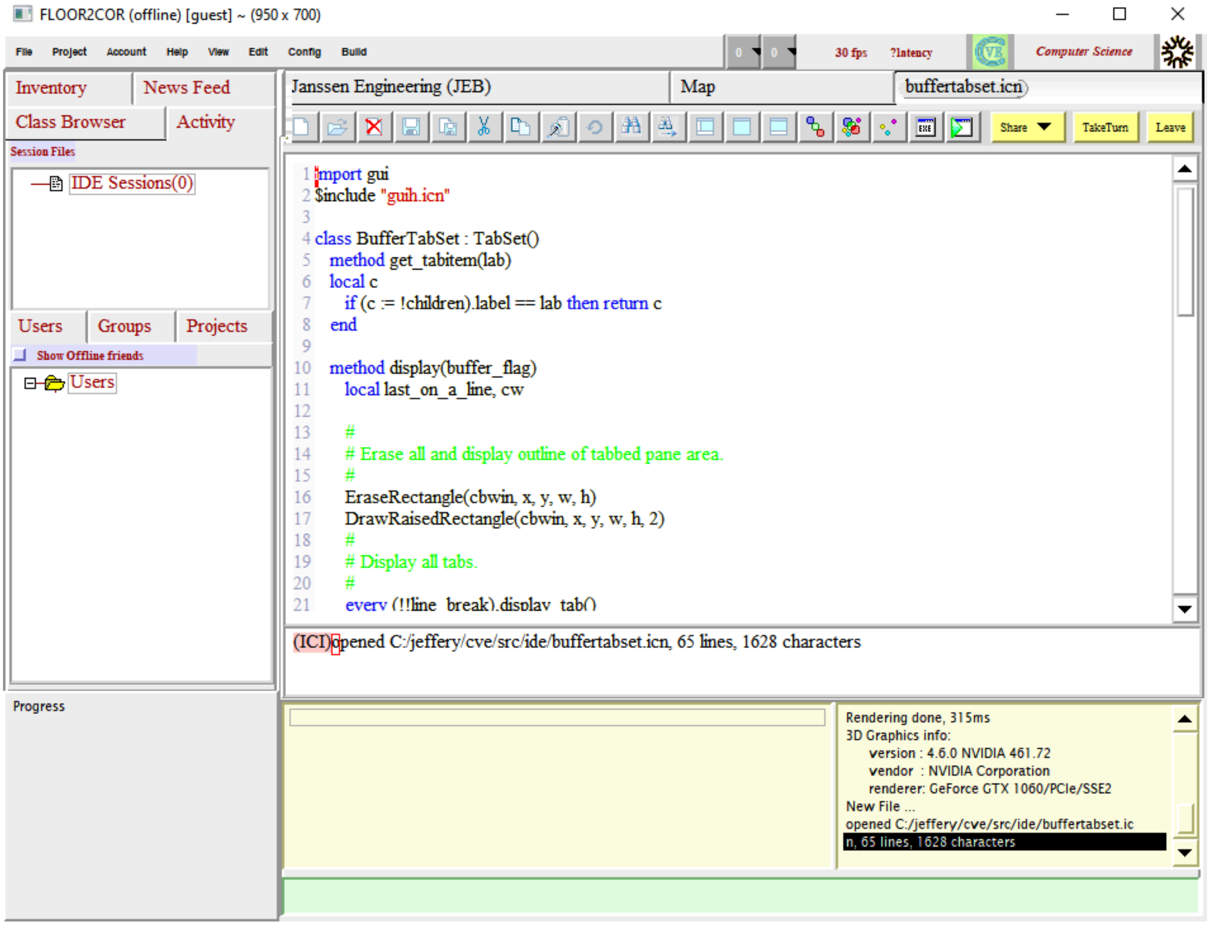
ForStmt : for(ForInit; Expr; ForUpdate)
Stmt
 	
a.k.a.
 	
ForInit;
while (Expr) {
Stmt
ForUpdate
}
	Expr.true = newlabel();
	Expr.first = newlabel();
	Expr.false = S.follow;
	Stmt.follow = ForUpdate.first;
	S.icode = ForInit.icode |||
	gen(LABEL, Expr.first) |||
	Expr.icode ||| gen(LABEL, Expr.true)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

Chapter 10: Syntax Coloring in an IDE


Unicon IDE
File View Config Edit Insert Compile Run Project Help
Class Browser
Open
Editor
j0.icn
scratch
j0.icn
Messages:
Debug Messages: (Single File Mode)

```
global yylineno, yycolno, yylval, parser
procedure main(argv)
  j0 := j0()
  parser := Parser()
  yyin := open(argv[1]) | stop("usage: j0 filename")
  yylineno := yycolno := 1
  if yyparse() = 0 then
    write("no errors")
end
class j0()
  method lexErr(s)
    stop(s, ": ", yytext)
  end
  method scan(cat)
    yylval := token(cat, yytext, yylineno, yycolno)
    yycolno +:= *yytext
    return cat
  end
  method whitespace()
    yycolno +:= *yytext
  end
  method newline()
    yylineno +:= 1; yycolno := 1
  end
```

```
opened C:\Users\clint\books\byopl\ch5\j0.icn, 65 lines, 1582 characters
```

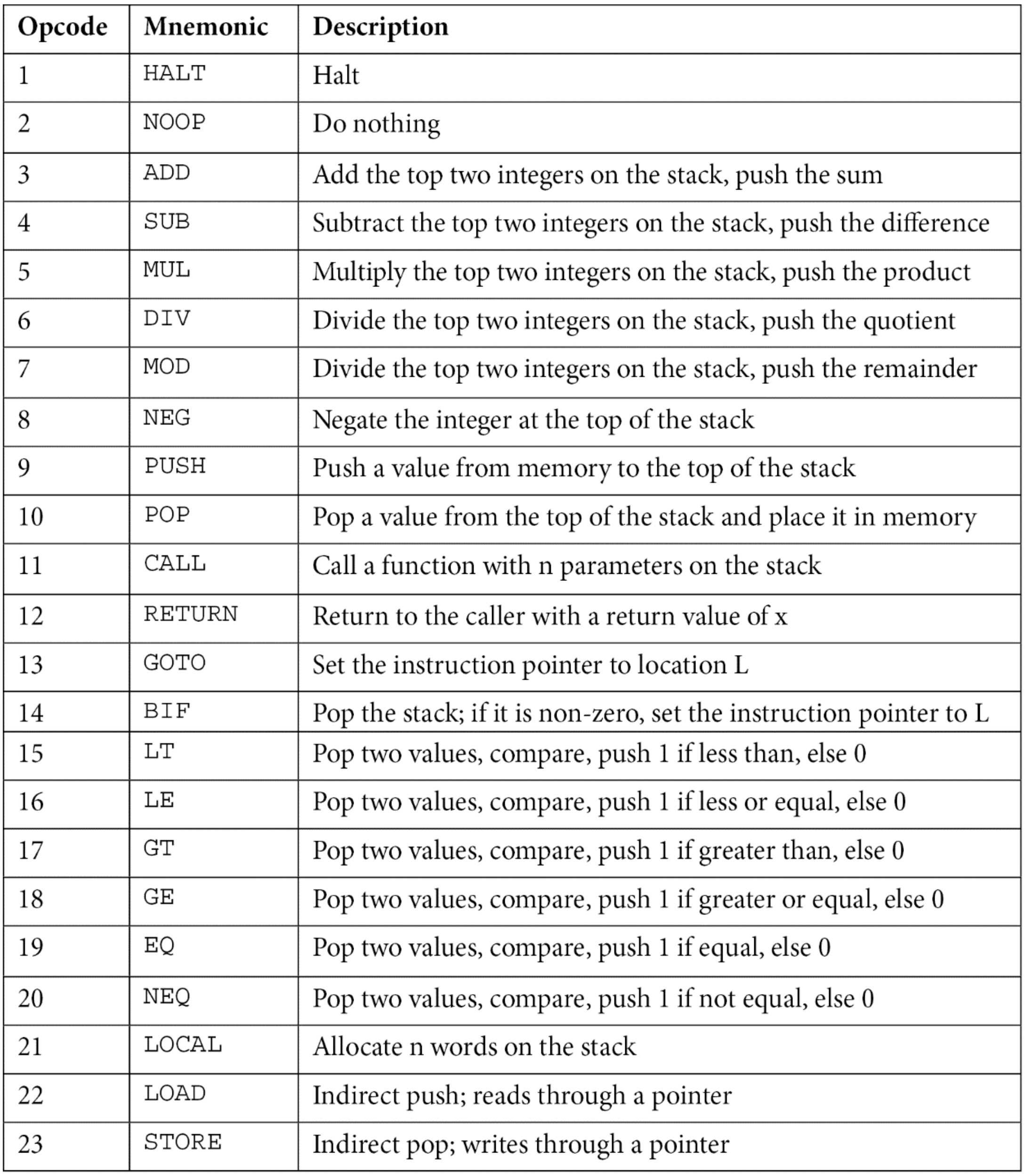


```
import gui
$include "guih.icn"

class BufferTabSet : TabSet()
  method get_tabitem(lab)
  local c
    if (c := !children).label == lab then return c
  end

  method display(buffer_flag)
    local last_on_a_line, cw

    #
    # Erase all and display outline of tabbed pane area.
    #
    EraseRectangle(cbwin, x, y, w, h)
    DrawRaisedRectangle(cbwin, x, y, w, h, 2)
    #
    # Display all tabs.
    #
    every (!!line_break).display_tab()
```

```
(ICI)opened C:/jeffery/cve/src/ide/buffertabset.icn, 65 lines, 1628 characters
```

```
Rendering done, 315ms
3D Graphics info:
    version : 4.6.0 NVIDIA 461.72
    vendor  : NVIDIA Corporation
    renderer: GeForce GTX 1060/PCIe/SSE2
New File ...
opened C:/jeffery/cve/src/ide/buffertabset.ic
n, 65 lines, 1628 characters
```

Chapter 11: Bytecode Interpreters

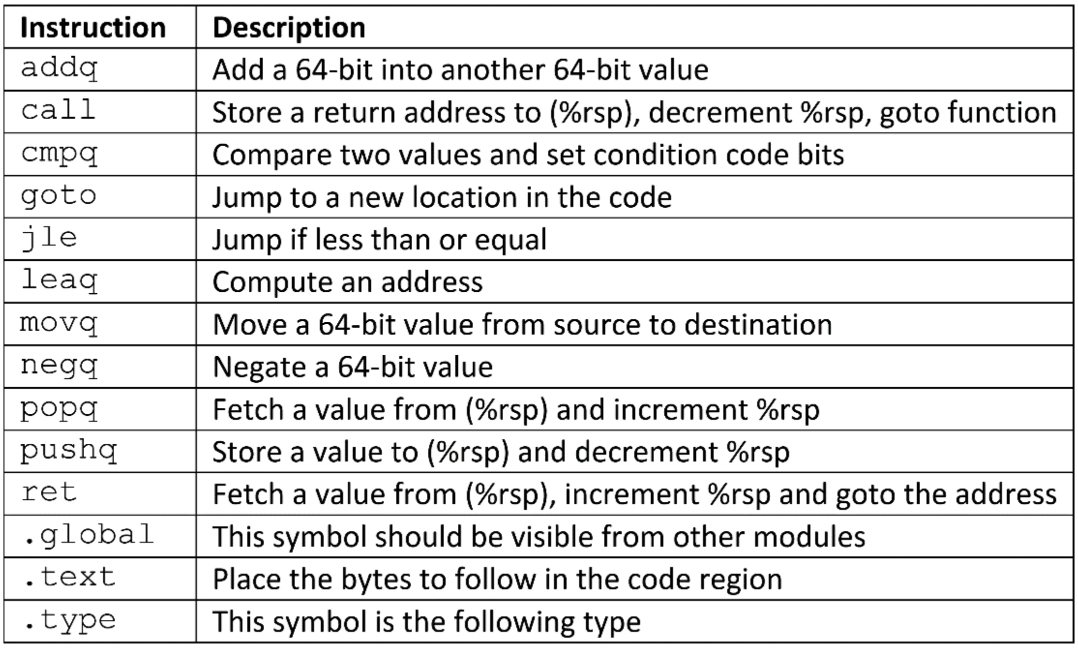


Opcode	Mnemonic	Description
1	`HALT`	Halt
2	`NOOP`	Do nothing
3	`ADD`	Add the top two integers on the stack, push the sum
4	`SUB`	Subtract the top two integers on the stack, push the difference
5	`MUL`	Multiply the top two integers on the stack, push the product
6	`DIV`	Divide the top two integers on 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

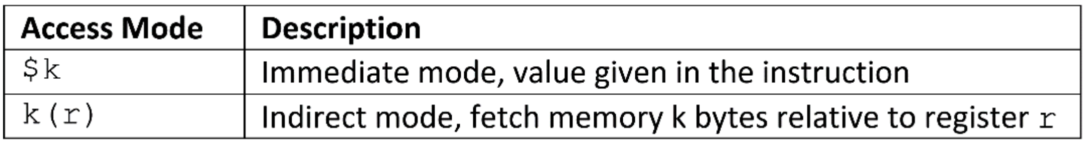
Chapter 12: Generating Bytecode

No images…

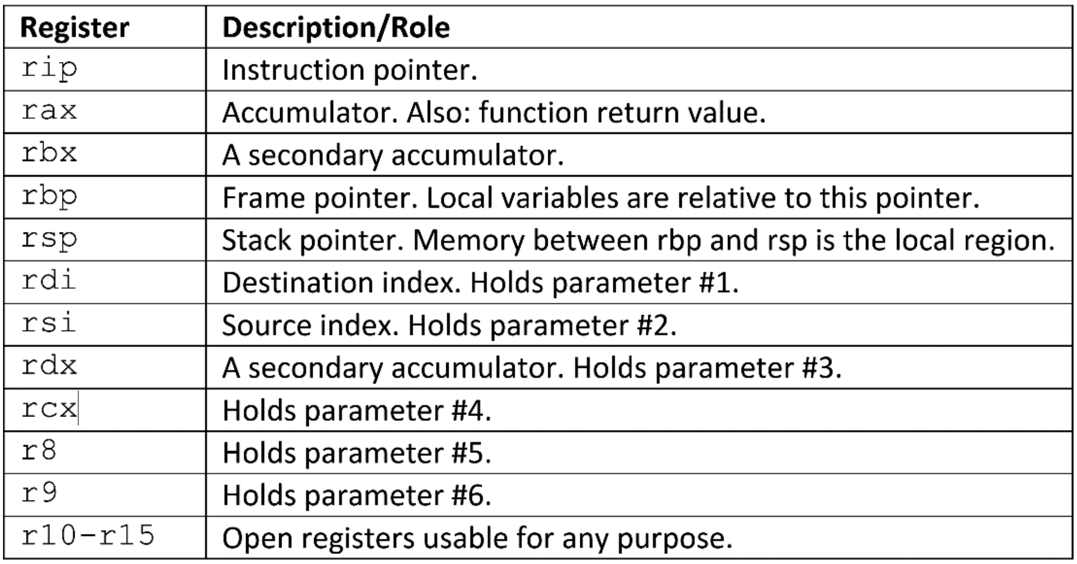
Chapter 13: Native Code Generation



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

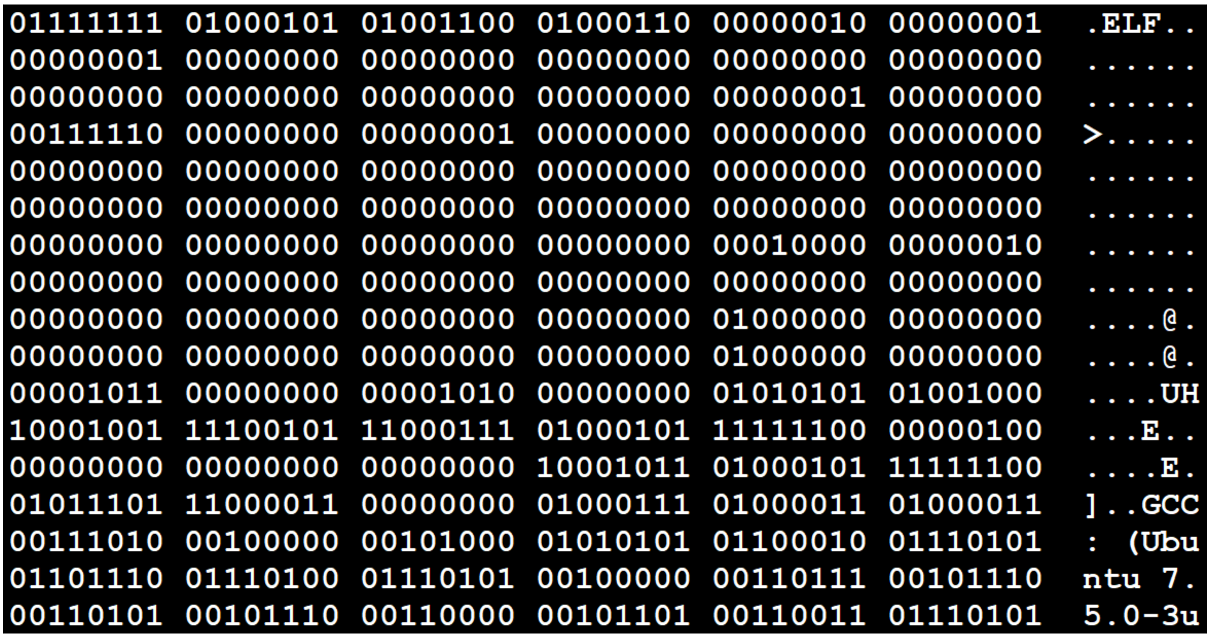


Access Mode	Description
`$k`	Immediate mode, value given in the instruction
`k(r)`	Indirect mode, fetch memory k bytes relative to register `r`



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.

⋮	⋮ **earlier activation record** ⋮
⋮	⋮ **earlier activation record** ⋮
return value parameter ⋮ parameter previous frame pointer (FP) saved registers ⋮ %rbp → saved PC local ⋮ local temporaries %rsp → ⋮	**current activation record**
"top" of stack ⋮ **grows down by subtracting from %rsp**	**calls create new activation records here**

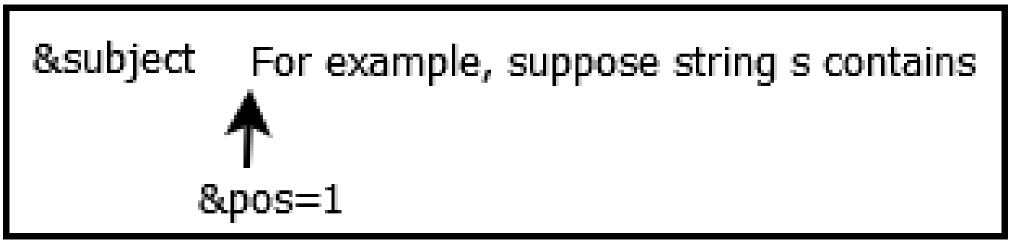


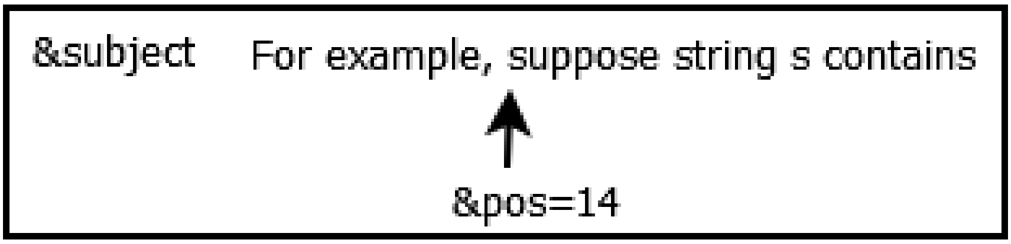
```
01111111 01000101 01001100 01000110 00000010 00000001  .ELF..
00000001 00000000 00000000 00000000 00000000 00000000  ......
00000000 00000000 00000000 00000000 00000001 00000000  ......
00111110 00000000 00000001 00000000 00000000 00000000  >.....
00000000 00000000 00000000 00000000 00000000 00000000  ......
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
```

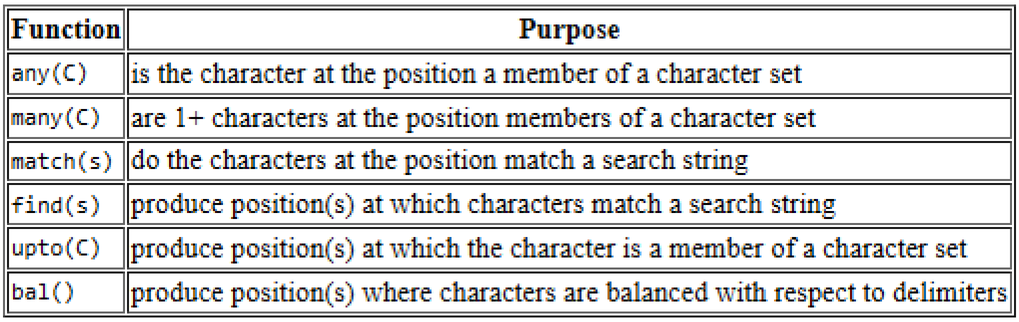
Chapter 14: Implementing Operators and Built-In Functions

No images…

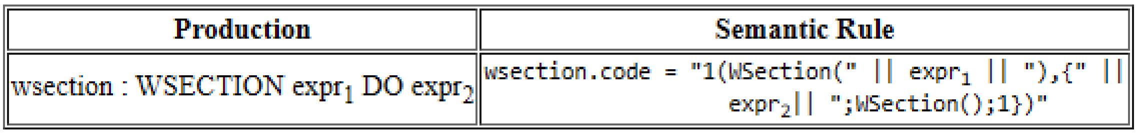
Chapter 15: Domain Control Structures





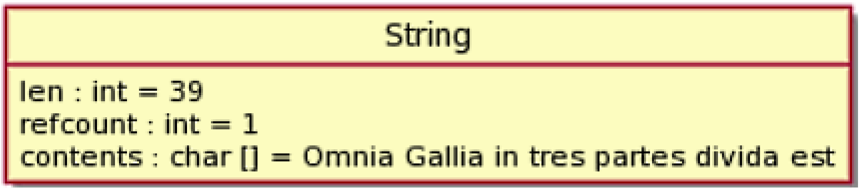


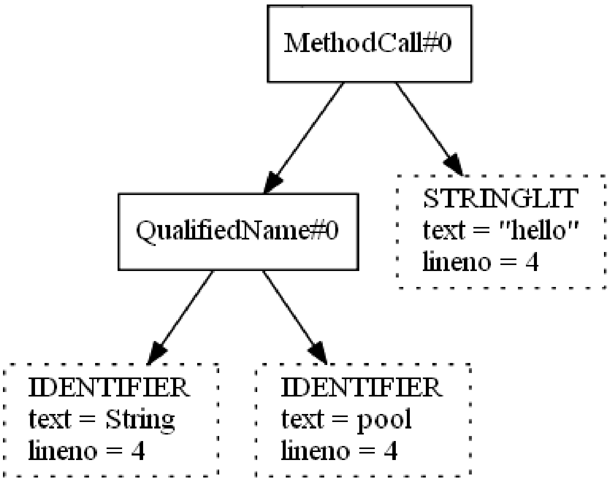
Function	Purpose
`any(C)`	is the character at the position a member of a character set
`many(C)`	are 1+ characters at the position members of a character set
`match(s)`	do the characters at the position match a search string
`find(s)`	produce position(s) at which characters match a search string
`upto(C)`	produce position(s) at which the character is a member of a character set
`bal()`	produce position(s) where characters are balanced with respect to delimiters

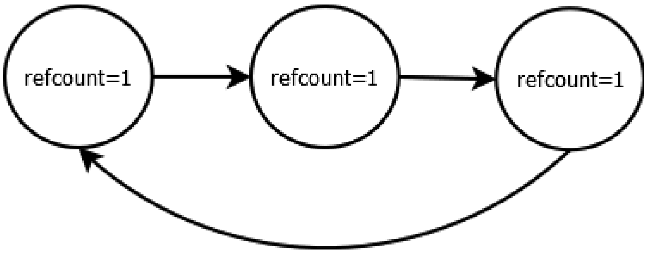


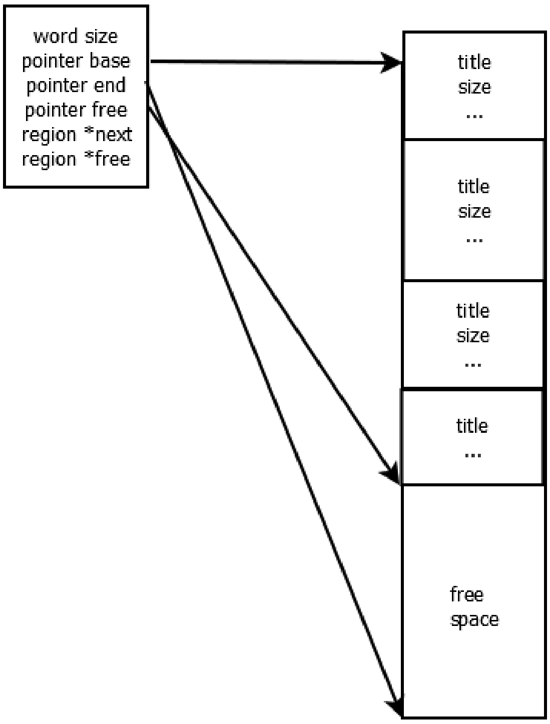
Production	Semantic Rule
wsection : WSECTION $expr_1$ DO $expr_2$	`wsection.code = "1(WSection(" || expr`$_1$` || "),{" || expr`$_2$`|| ";WSection();1})"`

Chapter 16: Garbage Collection





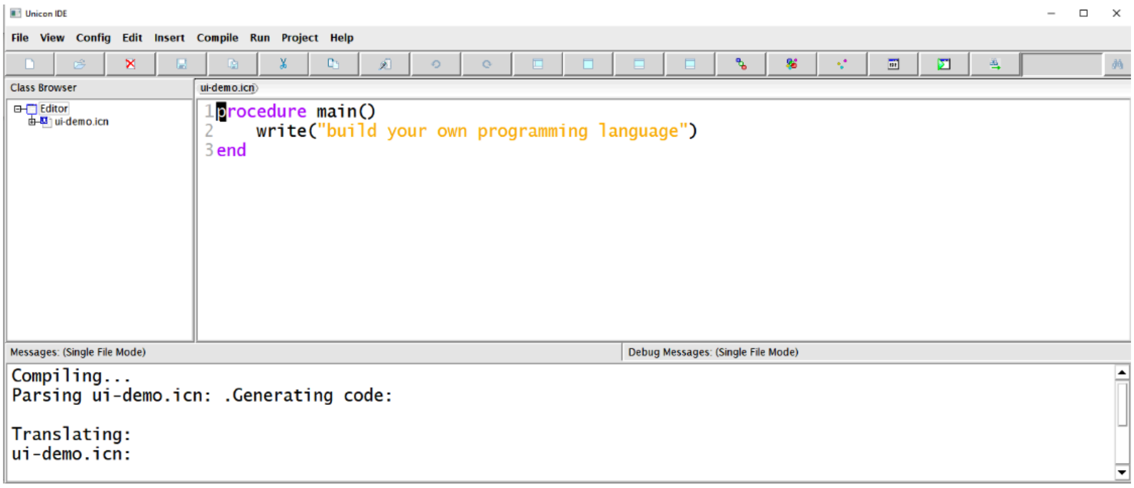




Chapter 17: Final Thoughts

No images…

Appendix: Unicon Essentials



```
procedure main()
    write("build your own programming language")
end
```

```
Compiling...
Parsing ui-demo.icn: .Generating code:

Translating:
ui-demo.icn:
```

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