Step 4

node \* parse\_statement(lexan & LEX)

{ LEX.nextlex();

if (LEX.kind == LX\_RW\_print)

{ node \* r = new node(N\_printstmt);

r->subtree.push\_back(parse\_expression(LEX));

return r; }

else if (LEX.kind == LX\_RW\_when)

{ node \* r = new node(N\_whenstmt);

r->subtree.push\_back(parse\_expression(LEX));

r->subtree.push\_back(parse\_statement(LEX));

return r; }

else if (LEX.kind == LX\_RW\_var)

{ node \* r = new node(N\_vardecl);

LEX.nextlex();

if (LEX.kind != LX\_variable)

LEX.error("in assignment, expecting variable, found " + LEX.form);

r->syminfo = LEX.syminfo;

LEX.nextlex();

if (LEX.kind != LX\_OP\_assign)

LEX.error("in assignment, expecting =, found " + LEX.form);

r->subtree.push\_back(parse\_expression(LEX));

return r; }

else if (LEX.kind == LX\_opencurly)

{ node \* r = new node(N\_sequence);

while (true)

{ r->subtree.push\_back(parse\_statement(LEX));

LEX.nextlex();

if (LEX.kind != LX\_semicolon)

break; }

if (LEX.kind != LX\_closecurly)

LEX.error("expecting close curly, found " + LEX.form);

return r; }

else

LEX.error("expecting statement, found " + LEX.form); }

$ lang4

{ var x = (1+2); var y = (x-1); when (y>x) print y; print (x\*3) }

sequence

vardecl syminfo = x @ 0x804e080

binaryexp intvalue=3

integer intvalue=1

integer intvalue=2

vardecl syminfo = y @ 0x804e0d0

binaryexp intvalue=4

variable syminfo = x @ 0x804e080

integer intvalue=1

whenstmt

binaryexp intvalue=11

variable syminfo = y @ 0x804e0d0

variable syminfo = x @ 0x804e080

printstmt

variable syminfo = y @ 0x804e0d0

printstmt

binaryexp intvalue=5

variable syminfo = x @ 0x804e080

integer intvalue=3