Recitation 5: The Parser

COP3402 FALL 2015 – ARYA POURTABATABAIE FROM EURIPIDES MONTAGNE, FALL 2014

The Parsing Problem

Take a **string of symbols** in a language (tokens) and a **grammar** for that language to construct the **parse tree** or report that the sentence is syntactically **incorrect**.

Two ways to do this:

- Top-Down (recursive descending parser).
- Bottom-Up. (We don't focus on this).

The top-down approach uses recursive procedures to model the parse tree.

Beginning with the start symbol, for every non-terminal (syntactic class) a procedure which parses that syntactic class is created.

PL/0 Grammar

```
<blook>
                             ::= <const-decl> <var-decl> <proc-decl> <statement>
                             ::= const <const-assignment-list> ; | e
    <const-decl>
    <const-assignment-list> ::= <ident> = <number>
                              <const-assignment-list> , <ident> = <number>
    <var-decl>
                             ::= var <ident-list> ; | e
    <ident-list>
                             ::= <ident> | <ident-list> , <ident>
                             ::= <proc-decl> procedure <ident> ; <block> ; | e
    c-decl>
                             ::= <ident> := <expression>
                                                            I call <ident>
    <statement>
                                                            | if <condition> then <statement>
                              | begin <statement-list> end
                              | while <condition> do <statement> | e
    <statement-list>
                             ::= <statement> | <statement-list> ; <statement>
    <condition>
                             ::= odd <expression> | <expression> <relation> <expression>
                             ::= = | <> | < | > | <= | >=
    <relation>
                             ::= <term> | <adding-operator> <term>
    <expression>
                              <expression> <adding-operator> <term>
    <adding-operator>
                              ::= + | -
                             ::= <factor> | <term> <multiplying-operator> <factor>
    <term>
                             ::= * | /
    <multiplying-operator>
                             ::= <ident> | <number> | (<expression> )
    <factor>
```

This is a slightly different version of the PL-0 grammar than the one you have in your notes.

- It's intended to be clearer for purposes of going through these slides
- We use <tokens> and literals instead of tokens and "literals" for the same reason
- The equivalence should be obvious enough

PL/0 Grammar

Terminals

const, var, procedure, call, begin, end, if, then, while, do, odd

```
<> < > <= >= + - * / =
, ; e
```

Non-Terminals

We must implement a procedure for each one of this non-terminals.

In this parser we use:

TOKEN –a global variable that stores the current token to analyze.

GET_TOKEN() – a procedure that takes the next token in the string and stores it in TOKEN.

ENTER(*type, name, params*) – a procedure that stores a new symbol into the Symbol Table.

ERROR() – a procedure that stops parsing, and shows an error message.

program> Procedure

```
procedure PROGRAM;
begin
GET_TOKEN();
BLOCK();
if TOKEN <> "." then ERROR (No Period at end of file)
end;
```

<blook> Procedure

<bloom> ::= <const-decl> <statement>

```
procedure BLOCK;
begin
  if TOKEN = "const" then CONST-DECL();
  if TOKEN = "var" then VAR-DECL();
  if TOKEN = "procedure" then PROC-DECL();
  STATEMENT;
end;
```

<const-decl> Procedure

```
<const-decl> ::= const <const-assignment-list> ; | e
 <const-assignment-list> ::= <ident> = <number>
                            <const-assignment-list> , <ident> = <number>
procedure CONST-DECL;
begin
 repeat
          GET TOKEN;
          if TOKEN <> IDENT then ERROR (missing identifier);
          GET TOKEN;
          if TOKEN <> "=" then ERROR (identifier should be followed by =);
          GET TOKEN;
          if TOKEN <> NUMBER then ERROR (= should be followed by number);
          ENTER(constant, ident, number);
          GET TOKEN;
until TOKEN <> ",";
 if TOKEN <> ";" then ERROR (declaration must end with ;);
GET TOKEN;
end;
```

<var-decl> Procedure

```
<var-decl> ::= var <ident-list> ; | e
<ident-list> ::= <ident> | <ident-list> , <ident>
procedure VAR-DECL;
begin
repeat
       GET_TOKEN;
       if TOKEN <> IDENT then ERROR (missing identifier);
       GET TOKEN;
       ENTER(variable, ident, level);
until TOKEN <> ",";
if TOKEN <> ";" then ERROR (declaration must end with ;);
GET_TOKEN;
end;
```

proc-decl> Procedure

```
coroc-decl> ::= coroc-decl> procedure <ident> ; <block> ; | e
procedure PROC-DECL;
begin
while TOKEN = "procedure" do begin
GET TOKEN;
 if TOKEN <> IDENT then ERROR (missing procedure declaration);
 ENTER(procedure, ident);
 GET TOKEN;
 if TOKEN <> ";" then ERROR (procedure declaration must end with ;);
 GET TOKEN;
 BLOCK(level+1);
 if TOKEN <> ";" then ERROR (no; at the end of block);
GET TOKEN;
end;
end;
```

```
begin <statement-list> end | if <condition> then <statement>
               while <condition> do <statement> | e
        <statement-list> ::= <statement> | <statement-list> ; <statement>
       procedure STATEMENT;
       begin
        if TOKEN = IDENT then begin
                GET TOKEN();
                If TOKEN <> ":=" then ERROR (:= missing in statement);
                GET_TOKEN();
                EXPRESSION();
        end
        else if TOKEN = "call" then begin
                GET TOKEN();
                if TOKEN <> IDENT then ERROR (missing identifier);
                GET_TOKEN();
        end
```

```
begin <statement-list> end | if <condition> then <statement>
             while <condition> do <statement> | e
       <statement-list> ::= <statement> | <statement-list> ; <statement>
       procedure STATEMENT;
       else if TOKEN = "begin" then begin
              GET TOKEN();
              STATEMENT();
              while TOKEN = ";" do begin
                      GET_TOKEN();
                      STATEMENT();
              end;
              if TOKEN <> "end" then ERROR (begin must be closed with end);
              GET_TOKEN();
       end;
       . . .
```

```
begin <statement-list> end | if <condition> then <statement>
              while <condition> do <statement> | e
       <statement-list> ::= <statement> | <statement-list> ; <statement>
       procedure STATEMENT;
       else if TOKEN = "if" then begin
               GET TOKEN();
               CONDITION();
               if TOKEN <> "then" then ERROR (if condition must be followed by then);
               GET TOKEN();
               STATEMENT();
       end;
```

```
begin <statement-list> end | if <condition> then <statement>
              while <condition> do <statement> | e
       <statement-list> ::= <statement> | <statement-list> ; <statement>
       procedure STATEMENT;
       else if TOKEN = "while" then begin
              GET_TOKEN();
              CONDITION();
              if TOKEN <> "do" then ERROR (while condition must be followed by do);
              GET TOKEN();
              STATEMENT();
       end
       end;
```

<condition> Procedure

<condition> ::= odd <expression> | <expression> <relation> <expression>

<expression> Procedure

```
<expression> ::= <term> | <adding-operator> <term>
      <expression> <adding-operator> <term>
procedure EXPRESSION;
begin
if TOKEN = ADDING OPERATOR then GET TOKEN();
TERM();
while TOKEN = ADDING OPERATOR do begin
      GET TOKEN();
      TERM();
end
end;
```

<term> Procedure

```
<term> ::= <factor> | <term> <multiplying-operator> <factor>
procedure TERM;
begin
FACTOR();
while TOKEN = MULTIPLYING_OPERATOR do begin
      GET_TOKEN();
      FACTOR();
end
end;
```

<factor> Procedure

```
<factor> ::= <ident> | <number> | ( <expression> )
procedure FACTOR;
begin
if TOKEN = IDENTIFIER then
          GET_TOKEN();
 else if TOKEN = NUMBER then
          GET_TOKEN();
 else if TOKEN = "(" then begin
          GET_TOKEN();
          EXPRESSION();
          if TOKEN <> ")" then ERROR( left ( has not been closed );
          GET_TOKEN();
 end
 else ERROR (identifier, ( or number expected);
end;
```

TOKEN=

```
const m = 8;
var a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
Symbol Table
                                program()
procedure PROGRAM;
begin
→ GET TOKEN();
     BLOCK();
     if TOKEN <> "." then ERROR (No Period at
end of file)
end;
```

```
m = 8;
var a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= const Symbol Table program()
```

```
procedure PROGRAM;
begin
    GET_TOKEN();
    BLOCK();
    if TOKEN <> "." then ERROR (No Period at end of file)
end;
```

```
m = 8;
var a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= const Symbol Table program() block()
```

```
procedure BLOCK;
begin

if TOKEN = "const" then CONST-DECL();
if TOKEN = "var" then VAR-DECL();
if TOKEN = "procedure" then PROC-DECL();
STATEMENT;
end;
```

```
m = 8;
var a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
            y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
procedure CONST-DECL;
begin
     repeat
     → GET TOKEN();
          if TOKEN <> IDENT then ERROR ();
          GET TOKEN();
          if TOKEN <> "=" then ERROR ();
          GET TOKEN();
          if TOKEN <> NUMBER then ERROR);
          ENTER(constant, ident, number);
          GET TOKEN();
     until TOKEN <> ",";
     if TOKEN <> ";" then ERROR ();
     GET TOKEN;
end;
```

```
= 8;
var a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= m Symbol Table program()
block()
const-decl()
```

```
procedure CONST-DECL;
begin
     repeat
          GET TOKEN();
          if TOKEN <> IDENT then ERROR ();
        →GET TOKEN();
          if TOKEN <> "=" then ERROR ();
          GET TOKEN();
          if TOKEN <> NUMBER then ERROR);
          ENTER(constant, ident, number);
          GET TOKEN();
     until TOKEN <> ",";
     if TOKEN <> ";" then ERROR ();
     GET TOKEN;
end;
```

```
8;
var a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
            y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
procedure CONST-DECL;
begin
     repeat
          GET TOKEN();
          if TOKEN <> IDENT then ERROR ();
          GET TOKEN();
          if TOKEN <> "=" then ERROR ();
     \longrightarrow GET TOKEN();
          if TOKEN <> NUMBER then ERROR);
          ENTER(constant, ident, number);
          GET TOKEN();
     until TOKEN <> ",";
     if TOKEN <> ";" then ERROR ();
     GET TOKEN;
end;
```

```
var a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
            y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= 8 Symbol Table program() block() const-decl()
```

```
procedure CONST-DECL;
begin
     repeat
          GET TOKEN();
          if TOKEN <> IDENT then ERROR ();
          GET TOKEN();
          if TOKEN <> "=" then ERROR ();
          GET TOKEN();
          if TOKEN <> NUMBER then ERROR);
         >ENTER(constant, ident, number);
          GET_TOKEN();
     until TOKEN <> ",";
     if TOKEN <> ";" then ERROR ();
     GET TOKEN;
end;
```

```
var a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
            y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

TOKEN= 8 Symbol Table program()

m=8;

Recursion stack

block()

const-decl()

```
procedure CONST-DECL;
begin
     repeat
          GET TOKEN();
          if TOKEN <> IDENT then ERROR ();
          GET TOKEN();
          if TOKEN <> "=" then ERROR ();
          GET TOKEN();
          if TOKEN <> NUMBER then ERROR);
          ENTER(constant, ident, number);
        →GET TOKEN();
     until TOKEN <> ",";
     if TOKEN <> ";" then ERROR ();
     GET TOKEN;
end;
```

```
var a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
            y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN=; Symbol Table program()
m=8; block()
const-decl()
```

```
procedure CONST-DECL;
begin
      repeat
           GET TOKEN();
           if TOKEN <> IDENT then ERROR ();
           GET TOKEN();
           if TOKEN <> "=" then ERROR ();
           GET TOKEN();
           if TOKEN <> NUMBER then ERROR);
           ENTER(constant, ident, number);
           GET TOKEN();
      until TOKEN <> ",";
      if TOKEN <> ";" then ERROR ();
→ GET TOKEN();
end;
```

```
a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
            y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= var

Symbol Table
program()
block()
const-decl()
```

```
procedure CONST-DECL;
  begin
       repeat
            GET TOKEN();
            if TOKEN <> IDENT then ERROR ();
            GET TOKEN();
            if TOKEN <> "=" then ERROR ();
            GET TOKEN();
            if TOKEN <> NUMBER then ERROR);
            ENTER(constant, ident, number);
            GET TOKEN();
       until TOKEN <> ",";
       if TOKEN <> ";" then ERROR ();
       GET TOKEN();
→ end;
```

```
TOKEN= var

Symbol Table
program()
block()
```

```
a, b, c;
                                                        m=8;
procedure ratio;
var x, y;
begin
x = a; y = b;
                               procedure BLOCK;
 if b > a then begin
                               begin
          x = b;
                                   if TOKEN = "const" then CONST-DECL();
          y = a;
                            if TOKEN = "var" then VAR-DECL();
 end
 c = x / y;
                                   if TOKEN = "procedure" then PROC-DECL();
end;
                                   STATEMENT;
begin
                               end;
 a = m;
 b = 4;
 call ratio;
end.
```

```
a, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= var

Symbol Table

m=8;

program()
block()
var-decl()
```

```
, b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= a Symbol Table program()
m=8;
block()
var-decl()
```

```
b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN=, Symbol Table program()
m=8;
block()
var-decl()
```

```
b, c;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN=, Symbol Table program()
m=8; a; block()
var-decl()
```

```
, C;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= b

Symbol Table

m=8; a;

program()
block()
var-decl()
```

```
С;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN=, Symbol Table program()
m=8; a; block()
var-decl()
```

```
procedure VAR-DECL;
begin
    repeat
        GET_TOKEN();
        if TOKEN <> IDENT then ERROR ();
        GET_TOKEN();
        ** ENTER(variable, ident, level);
        until TOKEN <> ",";
        if TOKEN <> ";" then ERROR ();
        GET_TOKEN();
end;
```

```
С;
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN=,

Symbol Table

program()
block()
var-decl()
```

```
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= c Symbol Table program()

m=8; a; b;

block()
var-decl()
```

```
TOKEN=;

Symbol Table

m=8; a; b;

recursion stack

program()
block()
var-decl()
```

```
procedure ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
procedure VAR-DECL;
begin
    repeat
        GET_TOKEN();
        if TOKEN <> IDENT then ERROR ();
        GET_TOKEN();
        PENTER(variable, ident, level);
        until TOKEN <> ",";
        if TOKEN <> ";" then ERROR ();
        GET_TOKEN();
end;
```

```
procedure ratio;
var x, y;
begin
x = a; y = b;
                                procedure VAR-DECL;
 if b > a then begin
                                begin
           x = b;
                                     repeat
           y = a;
                                           GET TOKEN();
 end
                                           if TOKEN <> IDENT then ERROR ();
 c = x / y;
                                           GET TOKEN();
end;
                                           ENTER(variable, ident, level);
begin
                                     until TOKEN <> ",";
 a = m;
                                     if TOKEN <> ";" then ERROR ();
 b = 4;
                                  → GET_TOKEN();
 call ratio;
                                end;
end.
```

```
TOKEN=;

Symbol Table

m=8; a; b; c;

program()
block()
var-decl()
```

Recursion stack

program()

var-decl()

block()

Symbol Table

```
m=8; a; b; c;
          ratio;
var x, y;
begin
x = a; y = b;
                                procedure VAR-DECL;
 if b > a then begin
                                begin
          x = b;
                                     repeat
          y = a;
                                           GET TOKEN();
 end
                                           if TOKEN <> IDENT then ERROR ();
 c = x / y;
                                           GET TOKEN();
end;
                                           ENTER(variable, ident, level);
begin
                                     until TOKEN <> ",";
 a = m;
                                     if TOKEN <> ";" then ERROR ();
b = 4;
                                     GET_TOKEN();
call ratio;
                              →end;
end.
```

TOKEN= procedure

Recursion stack

program()

block()

```
ratio;
var x, y;
begin
x = a; y = b;
                                procedure BLOCK;
 if b > a then begin
                                begin
          x = b;
                                    if TOKEN = "const" then CONST-DECL();
          y = a;
                                    if TOKEN = "var" then VAR-DECL();
 end
 c = x / y;
                                   if TOKEN = "procedure" then PROC-DECL();
end;
                                    STATEMENT;
begin
                                end;
 a = m;
 b = 4;
 call ratio;
end.
```

TOKEN= procedure

Symbol Table

m=8; a; b; c;

```
ratio;
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= procedure

Symbol Table

m=8; a; b; c;

proc-decl()
```

```
procedure PROC-DECL;
begin
while TOKEN = "procedure" do begin

    GET_TOKEN();
    if TOKEN <> IDENT then ERROR ();
    ENTER(procedure, ident);
    GET_TOKEN();
    if TOKEN <> ";" then ERROR ();
    GET_TOKEN();
    BLOCK(level+1);
    if TOKEN <> ";" then ERROR ();
    GET_TOKEN();
end;
end;
```

Recursion stack

```
m=8; a; b; c;
var x, y;
begin
x = a; y = b;
                               procedure PROC-DECL;
 if b > a then begin
                               begin
          x = b;
                                while TOKEN = "procedure" do begin
          y = a;
                                    GET TOKEN();
 end
                                    if TOKEN <> IDENT then ERROR ();
 c = x / y;
                                 ENTER(procedure, ident);
end;
                                    GET TOKEN();
begin
                                    if TOKEN <> ";" then ERROR ();
 a = m;
                                    GET TOKEN();
 b = 4;
                                    BLOCK(level+1);
 call ratio;
                                    if TOKEN <> ";" then ERROR ();
end.
                                    GET_TOKEN();
                                end;
```

end;

TOKEN= ratio

```
Symbol Table program()
m=8; a; b; c; proc-decl()
```

```
var x, y;
begin
x = a; y = b;
 if b > a then begin
           x = b;
            y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
TOKEN= ratio

Symbol Table

m=8; a; b; c; ratio;

program()

block()

proc-decl()
```

```
procedure PROC-DECL;
begin
    while TOKEN = "procedure" do begin
        GET_TOKEN();
    if TOKEN <> IDENT then ERROR ();
        ENTER(procedure, ident);

    GET_TOKEN();
    if TOKEN <> ";" then ERROR ();
        GET_TOKEN();
        BLOCK(level+1);
        if TOKEN <> ";" then ERROR ();
        GET_TOKEN();
    end;
end;
```

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;

program()

block()

proc-decl()
```

```
procedure PROC-DECL;
begin
while TOKEN = "procedure" do begin
     GET TOKEN();
     if TOKEN <> IDENT then ERROR ();
     ENTER(procedure, ident);
     GET TOKEN();
     if TOKEN <> ";" then ERROR ();
  \rightarrow GET TOKEN();
     BLOCK(level+1);
     if TOKEN <> ";" then ERROR ();
     GET_TOKEN();
end;
end;
```

```
TOKEN= var

Symbol Table

m=8; a; b; c; ratio;

proc-decl()
```

```
x, y;

begin

x = a; y = b;

if b > a then begin

x = b;

y = a;

end

c = x / y;

end;

begin

a = m;

b = 4;

call ratio;

end.
```

```
procedure PROC-DECL;
begin
while TOKEN = "procedure" do begin
     GET TOKEN();
     if TOKEN <> IDENT then ERROR ();
     ENTER(procedure, ident);
     GET TOKEN();
     if TOKEN <> ";" then ERROR ();
     GET TOKEN();
  → BLOCK(level+1);
     if TOKEN <> ";" then ERROR ();
     GET_TOKEN();
end;
end;
```

```
TOKEN= var

Symbol Table

m=8; a; b; c; ratio;

program()

block(1)

proc-decl(1)

block(2)
```

```
x, y;

begin

x = a; y = b;

if b > a then begin

x = b;

y = a;

end

c = x / y;

end;

begin

a = m;

b = 4;

call ratio;

end.
```

```
procedure BLOCK;
begin

if TOKEN = "const" then CONST-DECL();
   if TOKEN = "var" then VAR-DECL();
   if TOKEN = "procedure" then PROC-DECL();
   STATEMENT;
end;
```

TOKEN= var Symbol Table m=8; a; b; c; ratio; program() block(1) proc-decl(1) block(2)

```
x, y;

begin

x = a; y = b;

if b > a then begin

x = b;

y = a;

end

c = x / y;

end;

begin

a = m;

b = 4;

call ratio;

end.
```

```
procedure BLOCK;
begin
    if TOKEN = "const" then CONST-DECL();
    if TOKEN = "var" then VAR-DECL();
    if TOKEN = "procedure" then PROC-DECL();
    STATEMENT;
end;
```

TOKEN= var

```
х, у;
begin
x = a; y = b;
if b > a then begin
                                  begin
           x = b;
           y = a;
 end
c = x / y;
end;
begin
 a = m;
b = 4;
call ratio;
                                  end;
end.
```

```
Symbol Table

m=8; a; b; c; ratio;

proc-decl(1)
block(2)
var-decl(2)
```

TOKEN= x

```
Recursion stack
Symbol Table
```

program()

proc-decl(1)

var-decl(2)

block(1)

block(2)

```
m=8; a; b; c; ratio;
procedure VAR-DECL;
begin
     repeat
          GET TOKEN;
          if TOKEN <> IDENT then ERROR ();
     GET TOKEN;
          ENTER(variable, ident, level);
     until TOKEN <> ",";
     if TOKEN <> ";" then ERROR ();
     GET_TOKEN;
end;
```

```
, y;
begin
x = a; y = b;
if b > a then begin
           x = b;
           y = a;
 end
c = x / y;
end;
begin
 a = m;
b = 4;
call ratio;
```

end.

```
TOKEN=,

Symbol Table

m=8; a; b; c; ratio;

program()

block(1)

proc-decl(1)

block(2)

var-decl(2)
```

```
y;

begin

x = a; y = b;

if b > a then begin

x = b;

y = a;

end

c = x / y;

end;

begin

a = m;

b = 4;

call ratio;

end.
```

```
procedure VAR-DECL;
begin
    repeat
        GET_TOKEN;
        if TOKEN <> IDENT then ERROR ();
        GET_TOKEN;
        ENTER(variable, ident, level);
    until TOKEN <> ",";
    if TOKEN <> ";" then ERROR ();
        GET_TOKEN;
end;
```

```
y;
begin
    x = a; y = b;
    if b > a then begin
        x = b;
        y = a;
end
    c = x / y;
end;
begin
    a = m;
    b = 4;
call ratio;
end.
```

```
TOKEN=,

Symbol Table

m=8; a; b; c; ratio;
x;

procedure VAR-DECL:

procedure Symbol Table

program()
block(1)
proc-decl(1)
block(2)
var-decl(2)
```

```
m=8; a; b; c; ratio;
                                                        х;
begin
x = a; y = b;
                               procedure VAR-DECL;
if b > a then begin
                               begin
          x = b;
                                    repeat
          y = a;
                                          GET TOKEN;
 end
                                          if TOKEN <> IDENT then ERROR ();
c = x / y;
                                     GET TOKEN;
end;
                                          ENTER(variable, ident, level);
begin
                                    until TOKEN <> ",";
 a = m;
                                    if TOKEN <> ";" then ERROR ();
b = 4;
                                    GET_TOKEN;
call ratio;
                               end;
end.
```

TOKEN= y

Recursion stack

Symbol Table

```
program()
block(1)
proc-decl(1)
block(2)
var-decl(2)
```

Recursion stack

```
Symbol Table
TOKEN=;
                                             program()
                                             block(1)
                        m=8; a; b; c; ratio;
                                             proc-decl(1)
                        х;
                                             block(2)
                                             var-decl(2)
```

begin

```
x = a; y = b;
if b > a then begin
           x = b;
           y = a;
 end
c = x / y;
end;
begin
 a = m;
b = 4;
call ratio;
end.
```

```
procedure VAR-DECL;
begin
     repeat
          GET TOKEN;
          if TOKEN <> IDENT then ERROR ();
          GET_TOKEN;
        ENTER(variable, ident, level);
     until TOKEN <> ",";
     if TOKEN <> ";" then ERROR ();
     GET_TOKEN;
end;
```

Recursion stack

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

block(1)
proc-decl(1)
block(2)
var-decl(2)
```

call ratio;

end.

begin

```
procedure VAR-DECL;
begin
    repeat
        GET_TOKEN;
        if TOKEN <> IDENT then ERROR ();
        GET_TOKEN;
        ENTER(variable, ident, level);
        until TOKEN <> ",";
        if TOKEN <> ";" then ERROR ();

        GET_TOKEN;
end;
```

```
TOKEN= begin Symbol Table program()

m=8; a; b; c; ratio;
x; y;

proc-decl(1)
proc-decl(2)
```

```
procedure VAR-DECL;
begin

repeat

GET_TOKEN;

if TOKEN <> IDENT then ERROR ();

GET_TOKEN;

ENTER(variable, ident, level);

until TOKEN <> ",";

if TOKEN <> ";" then ERROR ();

GET_TOKEN;

→ end;
```

```
TOKEN= begin

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
```

```
TOKEN= begin

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
```

```
procedure BLOCK;
begin
    if TOKEN = "const" then CONST-DECL();
    if TOKEN = "var" then VAR-DECL();
    if TOKEN = "procedure" then PROC-DECL();

> STATEMENT;
end;
```

```
TOKEN= begin

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
statement(2)
```

TOKEN= x Symbol Table m=8; a; b; c; ratio; x; y; program() block(1) proc-decl(1) block(2)

```
= a; y = b;

if b > a then begin

x = b;
y = a;

end
c = x / y;

end;
begin
a = m;
b = 4;
call ratio;
end.
```

```
procedure STATEMENT;
...

else if TOKEN = "begin" then begin
    GET TOKEN();

>STATEMENT();

while TOKEN = ";" do begin
    GET_TOKEN();
    STATEMENT();

end;

if TOKEN <> "end" then ERROR ();

GET_TOKEN();

end;
```

statement(2)

TOKEN= x Symbol Table m=8; a; b; c; ratio; x; y;

```
= a; y = b;

if b > a then begin

x = b;
y = a;

end
c = x / y;

end;
begin
a = m;
b = 4;
call ratio;
end.
```

program() block(1) proc-decl(1) block(2) statement(2)

Recursion stack

```
block(2)
statement(2)
statement(2)

begin

if TOKEN = IDENT then begin

GET_TOKEN();

If TOKEN <> ":=" then ERROR ();

GET_TOKEN();

EXPRESSION();
```

end

TOKEN= =

```
Symbol Table
```

```
m=8; a; b; c; ratio; x; y;
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
```

```
a; y = b;

if b > a then begin

x = b;
y = a;

end
c = x / y;

end;
begin
a = m;
b = 4;
call ratio;
end.
```

```
procedure STATEMENT;
begin

if TOKEN = IDENT then begin

GET_TOKEN();

If TOKEN <> ":=" then ERROR ();

GET_TOKEN();

EXPRESSION();

end
...
```

; y = b;

end

end;

begin

end.

a = m; b = 4;

call ratio;

c = x / y;

if b > a then begin

x = b;

y = a;

TOKEN= a Symbol Table m=8; a; b; c; ratio; x; y;

```
procedure STATEMENT;
begin

if TOKEN = IDENT then begin

GET_TOKEN();

If TOKEN <> ":=" then ERROR ();

GET_TOKEN();

—> EXPRESSION();

end
...
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
```

Recursion stack Symbol Table TOKEN= a program() block(1) m=8; a; b; c; ratio; proc-decl(1) x; y; block(2) statement(2) statement(2) expression(2) procedure EXPRESSION; begin if TOKEN = ADDING_OPERATOR then GET_TOKEN(); \rightarrow TERM(); while TOKEN = ADDING OPERATOR do begin GET_TOKEN(); TERM(); end end;

```
; y = b;

if b > a then begin

x = b;

y = a;

end

c = x / y;

end;

begin

a = m;

b = 4;

call ratio;

end.
```

TOKEN= a Symbol Table m=8; a; b; c; ratio; x; y; proc-decl(1) block(2) statement(2) statement(2) expression(2) term(2)

while TOKEN = MULTIPLYING_OPERATOR do begin

GET TOKEN();

FACTOR();

end

end;

```
TOKEN= a
                         Symbol Table
                         m=8; a; b; c; ratio;
                         x; y;
```

```
; y = b;
 if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
procedure FACTOR;
begin
     if TOKEN = IDENTIFIER then
     \longrightarrow GET_TOKEN();
     else if TOKEN = NUMBER then
           GET_TOKEN();
     else if TOKEN = "(" then begin
           GET_TOKEN();
           EXPRESSION();
           if TOKEN <> ")" then ERROR );
           GET TOKEN();
     end
     else ERROR ();
end:
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
expression(2)
term(2)
factor(2)
```

TOKEN=; Symbol Table m=8; a; b; c; ratio; x; y;

```
y = b;
 if b > a then begin
           x = b;
           y = a;
 end
c = x / y;
end;
begin
 a = m;
b = 4;
call ratio;
end.
```

```
procedure FACTOR;
 begin
       if TOKEN = IDENTIFIER then
            GET_TOKEN();
       else if TOKEN = NUMBER then
            GET_TOKEN();
       else if TOKEN = "(" then begin
            GET_TOKEN();
            EXPRESSION();
            if TOKEN <> ")" then ERROR );
            GET TOKEN();
       end
       else ERROR ();
→end:
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
expression(2)
term(2)
factor(2)
```

```
Symbol Table
TOKEN=;
                                              program()
                                              block(1)
                         m=8; a; b; c; ratio;
                                              proc-decl(1)
                         x; y;
                                              block(2)
                                              statement(2)
                                              statement(2)
                                              expression(2)
 procedure TERM;
                                              term(2)
 begin
      FACTOR();
       while TOKEN = MULTIPLYING_OPERATOR do begin
            GET TOKEN();
           FACTOR();
       end
\rightarrowend;
```

y = b;

end

end;

begin

a = m;

b = 4;

end.

call ratio;

c = x / y;

if b > a then begin

x = b;

y = a;

```
Symbol Table
TOKEN=;
                                            program()
                                            block(1)
                        m=8; a; b; c; ratio;
                                            proc-decl(1)
                        x; y;
                                            block(2)
                                            statement(2)
                                            statement(2)
                                            expression(2)
  procedure EXPRESSION;
 begin
      if TOKEN = ADDING_OPERATOR then GET_TOKEN();
      TERM();
      while TOKEN = ADDING OPERATOR do begin
           GET_TOKEN();
           TERM();
      end
→end;
```

y = b;

end

end;

end.

begin

a = m;b = 4;

call ratio;

c = x / y;

if b > a then begin

x = b;

y = a;

TOKEN=; Symbol Table m=8; a; b; c; ratio;

x; y;

```
procedure STATEMENT;
begin

if TOKEN = IDENT then begin

GET_TOKEN();

If TOKEN <> ":=" then ERROR ();

GET_TOKEN();

EXPRESSION();

end
...
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
```

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
statement(2)
```

```
y = b;

if b > a then begin

x = b;
y = a;

end
c = x / y;

end;
begin
a = m;
b = 4;
call ratio;
end.
```

TOKEN=; Symbol Table m=8; a; b; c; ratio; x; y; program() block(1) proc-decl(1) block(2) statement(2)

Recursion stack

...

Symbol Table TOKEN= y program() block(1) m=8; a; b; c; ratio; proc-decl(1) x; y; block(2)

```
= b;
 if b > a then begin
           x = b;
           y = a;
 end
c = x / y;
end;
begin
 a = m;
b = 4;
call ratio;
end.
```

```
procedure STATEMENT;
     else if TOKEN = "begin" then begin
          GET TOKEN();
          STATEMENT();
          while TOKEN = ";" do begin
               GET_TOKEN();
          → STATEMENT();
          end;
          if TOKEN <> "end" then ERROR ();
          GET_TOKEN();
     end;
```

Recursion stack

statement(2)

TOKEN= y Symbol Table m=8; a; b; c; r

```
m=8; a; b; c; ratio; proc-decl(1) block(2) statement(2) statement(2) in
```

Recursion stack

program()

```
= b;

if b > a then begin

x = b;
y = a;

end
c = x / y;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

TOKEN==

```
Recursion stack
```

```
Symbol Table
m=8; a; b; c; ratio;
x; y;
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
```

```
b;

if b > a then begin

x = b;
y = a;

end
c = x / y;

end;
begin
a = m;
b = 4;
call ratio;
end.
```

```
procedure STATEMENT;
begin

if TOKEN = IDENT then begin

GET_TOKEN();

If TOKEN <> ":=" then ERROR ();

GET_TOKEN();

EXPRESSION();

end
...
```

TOKEN= b

```
Symbol Table
```

```
m=8; a; b; c; ratio; x; y;
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
```

```
procedure STATEMENT;
begin

if TOKEN = IDENT then begin
    GET_TOKEN();
    If TOKEN <> ":=" then ERROR ();
    GET_TOKEN();
    EXPRESSION();
  end
...
```

if b > a then begin

end

end;

begin

end.

a = m; b = 4;

call ratio;

c = x / y;

x = b;

y = a;

```
Symbol Table
TOKEN= b
                                            program()
                                            block(1)
                        m=8; a; b; c; ratio;
                                            proc-decl(1)
                        x; y;
                                            block(2)
                                            statement(2)
                                            statement(2)
                                            expression(2)
  procedure EXPRESSION;
 begin
      if TOKEN = ADDING_OPERATOR then GET_TOKEN();
    →TERM();
      while TOKEN = ADDING OPERATOR do begin
           GET_TOKEN();
           TERM();
      end
 end;
```

```
Symbol Table
TOKEN= b
                                              program()
                                              block(1)
                         m=8; a; b; c; ratio;
                                              proc-decl(1)
                         x; y;
                                              block(2)
                                              statement(2)
                                              statement(2)
                                              expression(2)
 procedure TERM;
                                              term(2)
 begin
    \rightarrow FACTOR();
      while TOKEN = MULTIPLYING_OPERATOR do begin
           GET TOKEN();
```

FACTOR();

end

end;

TOKEN= b Symbol Table m=8; a; b; c; ratio; x; y;

```
if b > a then begin
           x = b;
           y = a;
 end
c = x / y;
end;
begin
 a = m;
 b = 4;
call ratio;
end.
```

```
procedure FACTOR;
begin
     if TOKEN = IDENTIFIER then
     \longrightarrow GET_TOKEN();
     else if TOKEN = NUMBER then
           GET_TOKEN();
     else if TOKEN = "(" then begin
           GET_TOKEN();
           EXPRESSION();
           if TOKEN <> ")" then ERROR );
           GET TOKEN();
     end
     else ERROR ();
end:
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
expression(2)
term(2)
factor(2)
```

TOKEN=; Symbol Table m=8; a; b; c; ratio; x; y;

```
procedure FACTOR;
 begin
       if TOKEN = IDENTIFIER then
            GET_TOKEN();
       else if TOKEN = NUMBER then
            GET_TOKEN();
       else if TOKEN = "(" then begin
            GET_TOKEN();
            EXPRESSION();
            if TOKEN <> ")" then ERROR );
            GET TOKEN();
       end
       else ERROR ();
→end:
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
expression(2)
term(2)
factor(2)
```

```
Recursion stack
                        Symbol Table
TOKEN=;
                                            program()
                                            block(1)
                        m=8; a; b; c; ratio;
                                            proc-decl(1)
                        x; y;
                                            block(2)
                                            statement(2)
                                            statement(2)
                                            expression(2)
 procedure TERM;
                                            term(2)
 begin
      FACTOR();
```

```
procedure TERM;
begin

FACTOR();
while TOKEN = MULTIPLYING_OPERATOR do begin
GET_TOKEN();
FACTOR();
end

end;
```

```
Recursion stack
                        Symbol Table
TOKEN=;
                                            program()
                                            block(1)
                        m=8; a; b; c; ratio;
                                            proc-decl(1)
                        x; y;
                                            block(2)
                                            statement(2)
                                            statement(2)
                                            expression(2)
  procedure EXPRESSION;
 begin
      if TOKEN = ADDING_OPERATOR then GET_TOKEN();
      TERM();
      while TOKEN = ADDING OPERATOR do begin
```

GET_TOKEN();

TERM();

end

→end;

```
Recursion stack
```

statement(2)

```
Symbol Table
TOKEN=;
                                             program()
                                             block(1)
                        m=8; a; b; c; ratio;
                                             proc-decl(1)
                        x; y;
                                             block(2)
                                             statement(2)
```

```
if b > a then begin
           x = b;
           y = a;
 end
 c = x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
procedure STATEMENT;
begin
     if TOKEN = IDENT then begin
          GET TOKEN();
          If TOKEN <> ":=" then ERROR ();
          GET_TOKEN();
          EXPRESSION();
    >end
```

Recursion stack

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
statement(2)
```

...

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

block(1)
proc-decl(1)
block(2)
statement(2)
```

```
procedure STATEMENT;
...

else if TOKEN = "begin" then begin
    GET TOKEN();
    STATEMENT();
    while TOKEN = ";" do begin
    GET_TOKEN();
    STATEMENT();
    end;
    if TOKEN <> "end" then ERROR ();
    GET_TOKEN();
    end;
```

```
TOKEN= if

Symbol Table

m=8; a; b; c; ratio;
x; y;

block(1)
proc-decl(1)
block(2)
statement(2)
```

```
b > a then begin

x = b;

y = a;

end

c = x / y;

end;

begin

a = m;

b = 4;

call ratio;

end.
```

```
procedure STATEMENT;
...

else if TOKEN = "begin" then begin
    GET TOKEN();
    STATEMENT();
    while TOKEN = ";" do begin
        GET_TOKEN();
    ——> STATEMENT();
    end;
    if TOKEN <> "end" then ERROR ();
    GET_TOKEN();
    end;
```

TOKEN= if S

Symbol Table m=8; a; b; c; ratio; x; y;

```
Recursion stack
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
```

```
b > a then begin

x = b;

y = a;

end

c = x / y;

end;

begin

a = m;

b = 4;

call ratio;

end.
```

> a then begin

end

end;

begin

a = m;

b = 4;

end.

call ratio;

c = x / y;

x = b;

y = a;

TOKEN= b Symbol Table m=8; a; b; c; ratio; x; y;

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
```

TOKEN= b

```
Symbol Table
```

```
m=8; a; b; c; ratio; x; y;
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
condition(2)
```

```
> a then begin

x = b;

y = a;

end

c = x / y;

end;

begin

a = m;

b = 4;

call ratio;

end.
```

```
procedure CONDITION;
begin

if TOKEN = "odd" then begin
    GET_TOKEN();
    EXPRESSION();
else begin
    EXPRESSION();
    if TOKEN <> RELATION then ERROR ();
    GET_TOKEN();
    EXPRESSION();
    end
end;
```

TOKEN= b

```
Symbol Table
```

```
m=8; a; b; c; ratio; x; y;
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
condition(2)
```

```
> a then begin

x = b;

y = a;

end

c = x / y;

end;

begin

a = m;

b = 4;

call ratio;

end.
```

```
procedure CONDITION;
begin

if TOKEN = "odd" then begin
    GET_TOKEN();
    EXPRESSION();
else begin

EXPRESSION();
if TOKEN <> RELATION then ERROR ();
    GET_TOKEN();
    EXPRESSION();
end
end;
```

ample TOKEN=>

end;

Recursion stack

program()

block(1)

```
proc-decl(1)
                       x; y;
                                            block(2)
                                            statement(2)
                                            statement(2)
procedure CONDITION;
                                            condition(2)
begin
     if TOKEN = "odd" then begin
          GET TOKEN();
          EXPRESSION();
     else begin
          EXPRESSION();
          if TOKEN <> RELATION then ERROR ();
       → GET_TOKEN();
          EXPRESSION();
     end
```

Symbol Table

m=8; a; b; c; ratio;

```
a then begin

x = b;

y = a;

end

c = x / y;

end;

begin

a = m;

b = 4;

call ratio;

end.
```

TOKEN= a

end;

Recursion stack

program()

block(1)

```
proc-decl(1)
                       x; y;
                                            block(2)
                                            statement(2)
                                            statement(2)
procedure CONDITION;
                                            condition(2)
begin
     if TOKEN = "odd" then begin
          GET TOKEN();
          EXPRESSION();
     else begin
          EXPRESSION();
          if TOKEN <> RELATION then ERROR ();
          GET TOKEN();
     EXPRESSION();
     end
```

Symbol Table

m=8; a; b; c; ratio;

then begin x = b; y = a; end c = x / y; end; begin a = m; b = 4; call ratio;

end.

x = b;

y = a;

end

end;

begin

a = m;

b = 4;

end.

call ratio;

c = x / y;

```
Symbol Table
               TOKEN= then
                                                             program()
                                                             block(1)
                                        m=8; a; b; c; ratio;
                                                             proc-decl(1)
                                        x; y;
                                                             block(2)
                                                             statement(2)
                                                             statement(2)
                 procedure CONDITION;
begin
                                                             condition(2)
                begin
                      if TOKEN = "odd" then begin
                           GET TOKEN();
                           EXPRESSION();
                      else begin
                           EXPRESSION();
                           if TOKEN <> RELATION then ERROR ();
                           GET_TOKEN();
                           EXPRESSION();
                      end
               →end;
```

```
TOKEN= then

Symbol Table

m=8; a; b; c; ratio;
x; y;

block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
```

```
procedure STATEMENT;
             begin
          x = b;
                                    else if TOKEN = "if" then begin
          y = a;
                                          GET TOKEN();
 end
                                          CONDITION();
 c = x / y;
                                          if TOKEN <> "then" then ERROR ();
end;
                                        →GET_TOKEN();
begin
                                          STATEMENT();
 a = m;
                                    end;
 b = 4;
 call ratio;
end.
```

```
TOKEN= begin
                        Symbol Table
                                             program()
                                             block(1)
                        m=8; a; b; c; ratio;
                                             proc-decl(1)
                        x; y;
                                             block(2)
                                             statement(2)
                                             statement(2)
  procedure STATEMENT;
      else if TOKEN = "if" then begin
           GET TOKEN();
           CONDITION();
           if TOKEN <> "then" then ERROR ();
           GET_TOKEN();
      → STATEMENT();
      end;
```

```
x = b;
y = a;
end
c = x / y;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

x = b;

y = a;

end

end;

begin

a = m;

b = 4;

end.

call ratio;

c = x / y;

m=8; a; b; c; ratio; x; y; procedure STATEMENT; else if TOKEN = "begin" then begin →GET TOKEN(); STATEMENT(); while TOKEN = ";" do begin GET_TOKEN(); STATEMENT(); end; if TOKEN <> "end" then ERROR (); GET_TOKEN(); end;

TOKEN= begin

Symbol Table

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
statement(2)
```

= b;

y = a;

end

end;

begin

a = m;

b = 4;

end.

call ratio;

c = x / y;

m=8; a; b; c; ratio; x; y; procedure STATEMENT; else if TOKEN = "begin" then begin GET TOKEN(); →STATEMENT(); while TOKEN = ";" do begin GET_TOKEN(); STATEMENT(); end; if TOKEN <> "end" then ERROR (); GET_TOKEN(); end;

TOKEN= x

Symbol Table

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
statement(2)
```

y = a;

end

end;

begin

a = m;

b = 4;

end.

call ratio;

c = x / y;

Symbol Table TOKEN=; m=8; a; b; c; ratio; x; y; procedure STATEMENT; else if TOKEN = "begin" then begin GET TOKEN(); STATEMENT(); while TOKEN = ";" do begin \longrightarrow GET_TOKEN(); STATEMENT(); end; if TOKEN <> "end" then ERROR (); GET_TOKEN(); end;

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
statement(2)
```

= a;

end

end;

begin

a = m;

b = 4;

end.

call ratio;

c = x / y;

Symbol Table TOKEN= y m=8; a; b; c; ratio; x; y; procedure STATEMENT; else if TOKEN = "begin" then begin GET TOKEN(); STATEMENT(); while TOKEN = ";" do begin GET_TOKEN(); → STATEMENT(); end; if TOKEN <> "end" then ERROR (); GET_TOKEN(); end;

Recursion stack

program()

proc-decl(1)

statement(2)

statement(2)

statement(2)

block(1)

block(2)

end

end;

begin

a = m;

b = 4;

end.

call ratio;

c = x / y;

Symbol Table TOKEN=; m=8; a; b; c; ratio; x; y; procedure STATEMENT; else if TOKEN = "begin" then begin GET TOKEN(); STATEMENT(); while TOKEN = ";" do begin GET_TOKEN(); STATEMENT(); →end; if TOKEN <> "end" then ERROR (); GET_TOKEN(); end;

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
statement(2)
```

end

end;

begin

a = m;

b = 4;

end.

call ratio;

c = x / y;

Recursion stack

program()

proc-decl(1)

statement(2)

statement(2)

statement(2)

block(1)

block(2)

```
Symbol Table
TOKEN=;
                          m=8; a; b; c; ratio;
                          x; y;
 procedure STATEMENT;
       else if TOKEN = "begin" then begin
            GET TOKEN();
            STATEMENT();
            while TOKEN = ";" do begin
            \longrightarrow GET_TOKEN();
                 STATEMENT();
            end;
            if TOKEN <> "end" then ERROR ();
            GET_TOKEN();
       end;
```

c = x / y;

end;

begin

a = m;

b = 4;

end.

call ratio;

N= end Symbol Table Program()

block(1)

block(2)

proc-decl(1)

statement(2)

statement(2)

statement(2)

```
Symbol Table
TOKEN= end
                         m=8; a; b; c; ratio;
                         x; y;
 procedure STATEMENT;
      else if TOKEN = "begin" then begin
           GET TOKEN();
           STATEMENT();
           while TOKEN = ";" do begin
                GET_TOKEN();
           → STATEMENT();
           end;
           if TOKEN <> "end" then ERROR ();
           GET_TOKEN();
      end;
```

...

c = x / y;

end;

begin

a = m;

b = 4;

end.

call ratio;

Symbol Table TOKEN= end m=8; a; b; c; ratio; x; y; procedure STATEMENT; else if TOKEN = "begin" then begin GET TOKEN(); STATEMENT(); while TOKEN = ";" do begin GET_TOKEN(); STATEMENT(); end; if TOKEN <> "end" then ERROR (); →GET_TOKEN(); end;

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
statement(2)
```

c = x / y;

end;

begin

a = m;

b = 4;

end.

call ratio;

Symbol Table TOKEN=; m=8; a; b; c; ratio; x; y; procedure STATEMENT; else if TOKEN = "begin" then begin GET TOKEN(); STATEMENT(); while TOKEN = ";" do begin GET_TOKEN(); STATEMENT(); end; if TOKEN <> "end" then ERROR (); GET_TOKEN(); end;

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
statement(2)
```

```
Symbol Table
TOKEN=;
                                              program()
                                              block(1)
                         m=8; a; b; c; ratio;
                                              proc-decl(1)
                         x; y;
                                              block(2)
                                              statement(2)
                                              statement(2)
  procedure STATEMENT;
      else if TOKEN = "if" then begin
           GET_TOKEN();
           CONDITION();
           if TOKEN <> "then" then ERROR ();
           GET_TOKEN();
           STATEMENT();
     →end;
```

```
c = x / y;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
statement(2)
```

```
c = x / y;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
statement(2)
```

```
c = x / y;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

```
TOKEN= c

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
statement(2)
```

```
= x / y;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

Recursion stack Symbol Table TOKEN= c program() block(1) m=8; a; b; c; ratio; proc-decl(1) x; y; block(2) statement(2) statement(2) procedure STATEMENT; begin if TOKEN = IDENT then begin

```
= x / y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

→ GET_TOKEN(); If TOKEN <> ":=" then ERROR); GET_TOKEN(); EXPRESSION(); end

TOKEN= = Symbol Table program() m=8; a; b; c; ratio; x; y; procedure STATEMENT; begin Recursion stack program() block(1) proc-decl(1) block(2) statement(2) statement(2)

```
x / y;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

```
procedure STATEMENT;
begin
    if TOKEN = IDENT then begin
        GET_TOKEN();
        If TOKEN <> ":=" then ERROR);
        GET_TOKEN();
        EXPRESSION();
    end
```

Symbol Table TOKEN= x block(1) m=8; a; b; c; ratio; x; y; block(2) procedure STATEMENT; begin if TOKEN = IDENT then begin GET_TOKEN(); If TOKEN <> ":=" then ERROR);

GET_TOKEN();

EXPRESSION();

end

```
/ y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
program()
proc-decl(1)
statement(2)
statement(2)
```

```
Symbol Table
TOKEN= x
                                              program()
                                              block(1)
                         m=8; a; b; c; ratio;
                                              proc-decl(1)
                         x; y;
                                              block(2)
                                              statement(2)
                                              statement(2)
                                              expression(2)
 procedure EXPRESSION;
 begin
      if TOKEN = ADDING_OPERATOR then GET_TOKEN();
\longrightarrow TERM();
      while TOKEN = ADDING_OPERATOR do begin
           GET_TOKEN();
           TERM();
      end
 end;
```

```
/ y;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

```
Symbol Table
TOKEN= x
                                             program()
                                             block(1)
                        m=8; a; b; c; ratio;
                                             proc-decl(1)
                        x; y;
                                             block(2)
                                             statement(2)
                                             statement(2)
                                             expression(2)
 procedure TERM;
                                             term(2)
 begin
```

```
/ y;
end;
begin
 a = m;
 b = 4;
 call ratio;
end.
```

```
→ FACTOR();
     while TOKEN = MULTIPLYING_OPERATOR do begin
         GET_TOKEN();
         FACTOR();
     end
end;
```

```
/ y;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

```
Symbol Table
TOKEN= x
                         m=8; a; b; c; ratio;
                         x; y;
 procedure FACTOR;
 begin
      if TOKEN = IDENTIFIER then
     → GET TOKEN();
      else if TOKEN = NUMBER then
           GET_TOKEN();
      else if TOKEN = "(" then begin
           GET_TOKEN();
           EXPRESSION();
           if TOKEN <> ")" then ERROR ();
           GET_TOKEN();
      end
      else ERROR ();
 end;
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
expression(2)
term(2)
factor(2)
```

у;

end;

begin

a = m;

b = 4;

end.

call ratio;

```
Symbol Table
TOKEN= /
                         m=8; a; b; c; ratio;
                         x; y;
 procedure FACTOR;
 begin
      if TOKEN = IDENTIFIER then
            GET TOKEN();
       else if TOKEN = NUMBER then
           GET_TOKEN();
       else if TOKEN = "(" then begin
            GET_TOKEN();
            EXPRESSION();
            if TOKEN <> ")" then ERROR ();
           GET_TOKEN();
      end
      else ERROR ();
→end;
```

```
program()
block(1)
proc-decl(1)
block(2)
statement(2)
statement(2)
expression(2)
term(2)
factor(2)
```

```
TOKEN=/
                        Symbol Table
                                            program()
                                            block(1)
                        m=8; a; b; c; ratio;
                                            proc-decl(1)
                        x; y;
                                            block(2)
                                            statement(2)
                                            statement(2)
                                            expression(2)
 procedure TERM;
 begin
                                            term(2)
      FACTOR();
      while TOKEN = MULTIPLYING_OPERATOR do begin
         GET_TOKEN();
           FACTOR();
      end
 end;
```

```
y;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

```
TOKEN= y
                         Symbol Table
                                               program()
                                               block(1)
                         m=8; a; b; c; ratio;
                                               proc-decl(1)
                         x; y;
                                               block(2)
                                               statement(2)
                                               statement(2)
                                               expression(2)
 procedure TERM;
 begin
                                              term(2)
      FACTOR();
      while TOKEN = MULTIPLYING_OPERATOR do begin
           GET_TOKEN();
     \longrightarrow FACTOR();
      end
 end;
```

```
;
end;
begin
a = m;
b = 4;
call ratio;
end.
```

end;

begin

a = m;

b = 4;

end.

call ratio;

```
TOKEN=;
                         Symbol Table
                                             program()
                                             block(1)
                         m=8; a; b; c; ratio;
                                             proc-decl(1)
                         x; y;
                                             block(2)
                                             statement(2)
                                             statement(2)
                                             expression(2)
  procedure TERM;
  begin
                                             term(2)
       FACTOR();
       while TOKEN = MULTIPLYING_OPERATOR do begin
            GET_TOKEN();
           FACTOR();
       end
→ end;
```

end;

begin

a = m;

b = 4;

end.

call ratio;

```
Symbol Table
TOKEN=;
                                            program()
                                            block(1)
                        m=8; a; b; c; ratio;
                                            proc-decl(1)
                        x; y;
                                            block(2)
                                            statement(2)
                                            statement(2)
                                            expression(2)
 procedure EXPRESSION;
 begin
      if TOKEN = ADDING_OPERATOR then GET_TOKEN();
      TERM();
      while TOKEN = ADDING_OPERATOR do begin
           GET_TOKEN();
           TERM();
      end
→end;
```

end;

end.

begin

a = m;b = 4;

call ratio;

Recursion stack EN= ; Symbol Table program()

block(1)

block(2)

proc-decl(1)

statement(2)

statement(2)

```
TOKEN=;
                          Symbol Table
                          m=8; a; b; c; ratio;
                          x; y;
  procedure STATEMENT;
 begin
      if TOKEN = IDENT then begin
            GET_TOKEN();
            If TOKEN <> ":=" then ERROR);
            GET_TOKEN();
            EXPRESSION();

ightarrow end
```

end;

begin

a = m;

b = 4;

end.

call ratio;

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
statement(2)
```

end;

begin

a = m;

b = 4;

end.

Recursion stack

```
Symbol Table
TOKEN=;
                                             program()
                                             block(1)
                        m=8; a; b; c; ratio;
                                             proc-decl(1)
                        x; y;
                                             block(2)
                                             statement(2)
```

```
STATEMENT();
                                        end;
call ratio;
                                        GET_TOKEN();
```

procedure STATEMENT; else if TOKEN = "begin" then begin GET TOKEN(); while TOKEN = ";" do begin → GET_TOKEN(); STATEMENT(); if TOKEN <> "end" then ERROR (); end;

```
TOKEN= end

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
statement(2)
```

```
;
begin
a = m;
b = 4;
call ratio;
end.
```

Recursion stack

```
TOKEN= end

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
statement(2)
```

```
;
begin
a = m;
b = 4;
call ratio;
end.
```

• • •

begin

a = m;

b = 4;

end.

call ratio;

Recursion stack

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

block(1)
proc-decl(1)
block(2)
statement(2)
```

```
else if TOKEN = "begin" then begin

GET TOKEN();

STATEMENT();

while TOKEN = ";" do begin

GET_TOKEN();

STATEMENT();

end;

if TOKEN <> "end" then ERROR ();

GET_TOKEN();

——> end;
```

procedure STATEMENT;

Recursion stack

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
proc-decl(1)
block(2)
```

```
procedure BLOCK;
begin
    if TOKEN = "const" then CONST-DECL();
    if TOKEN = "var" then VAR-DECL();
    if TOKEN = "procedure" then PROC-DECL();
        STATEMENT();
        end;
```

begin

```
a = m;
b = 4;
call ratio;
end.
```

Recursion stack

```
Symbol Table
TOKEN=;
                                             program()
                                             block(1)
                        m=8; a; b; c; ratio;
                                             proc-decl(1)
                        x; y;
```

```
a = m;
b = 4;
call ratio;
```

begin

end.

```
procedure PROC-DECL;
begin
 while TOKEN = "procedure" do begin
      GET TOKEN();
      if TOKEN <> IDENT then ERROR ();
      ENTER(procedure, ident);
      GET TOKEN();
      if TOKEN <> ";" then ERROR ();
      GET_TOKEN();
      BLOCK(level+1);
      if TOKEN <> ";" then ERROR ();
→ GET_TOKEN();
 end;
 end;
```

Recursion stack

TOKEN= begin	Symbol Table	program()
	m=8; a; b; c; ratio; x; y;	block(1) proc-decl(1)
	, , ,	

```
begin
  while TOKEN = "procedure" do begin
    GET_TOKEN();
  if TOKEN <> IDENT then ERROR ();
    ENTER(procedure, ident);
    GET_TOKEN();
  if TOKEN <> ";" then ERROR ();
    GET_TOKEN();
    BLOCK(level+1);
  if TOKEN <> ";" then ERROR ();
    GET_TOKEN();
  end;
  end;
```

procedure PROC-DECL;

a = m;
b = 4;
call ratio;
end.

```
TOKEN= begin

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
```

```
procedure BLOCK;
begin
    if TOKEN = "const" then CONST-DECL();
    if TOKEN = "var" then VAR-DECL();
    if TOKEN = "procedure" then PROC-DECL();

>> STATEMENT;
end;
```

```
a = m;
b = 4;
call ratio;
end.
```

Recursion stack

TOKEN= begin	Symbol Table	program()
	m=8; a; b; c; ratio;	block(1) statement(1)
	x; y;	

```
procedure STATEMENT;
...
else if TOKEN = "begin" then begin

GET TOKEN();
STATEMENT();
while TOKEN = ";" do begin
GET_TOKEN();
STATEMENT();
end;
if TOKEN <> "end" then ERROR ();
GET_TOKEN();
end;
```

a = m;
b = 4;
call ratio;
end.

TOKEN= a	Symbol Table	program()
	m=8; a; b; c; ratio;	block(1) statement(1)
	x; y;	

```
= m;
b = 4;
call ratio;
end.
```

Recursion stack

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
statement(1)
```

b = 4; call ratio; end.

Recursion stack

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
statement(1)
```

b = 4; call ratio; end.

Recursion stack

TOKEN= b	Symbol Table	program()
	m=8; a; b; c; ratio;	block(1) statement(1)
	x; y;	

= 4; call ratio; end.

Recursion stack

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
statement(1)
```

call ratio;end.

Recursion stack

TOKEN= call	Symbol Table	program()
	m=8; a; b; c; ratio;	block(1) statement(1)
	x; y;	statement(1)

ratio; end.

Recursion stack

```
TOKEN=;

Symbol Table

m=8; a; b; c; ratio;
x; y;

program()
block(1)
statement(1)
```

end.

TOKEN= end	Symbol Table	program()
	m=8; a; b; c; ratio;	block(1) statement(1)
	x; y;	

```
else if TOKEN = "begin" then begin

GET TOKEN();

STATEMENT();

while TOKEN = ";" do begin

GET_TOKEN();

—> STATEMENT();

end;

if TOKEN <> "end" then ERROR ();

GET_TOKEN();

end;

end;
```

Recursion stack

TOKEN= end	Symbol Table	program()
	m=8; a; b; c; ratio;	block(1) statement(1)
	x; y;	

• • •

```
TOKEN= . Symbol Table program()

m=8; a; b; c; ratio;
x; y; program()
block(1)
statement(1)
```

```
TOKEN= . Symbol Table program()

m=8; a; b; c; ratio;
x; y;

program()
block(1)
```

```
procedure BLOCK;
begin
    if TOKEN = "const" then CONST-DECL();
    if TOKEN = "var" then VAR-DECL();
    if TOKEN = "procedure" then PROC-DECL();
    STATEMENT;

end;
```

TOKEN= .	Symbol Table	program()
	m=8; a; b; c; ratio;	
	x; y;	

```
procedure PROGRAM;
begin
GET_TOKEN();
BLOCK();
if TOKEN <> "." then ERROR ()

end;
```

Questions?