

The Parsing Problem

- Given a **string of symbols** in a language (tokens) and a **grammar** for that language => 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).



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.

```
<program>      ::= <block> "." .

<block>        ::= <const-decl> <var-decl> <proc-decl> <statement> .

<const-decl>   ::= const <const-assignment-list> ";"
                |      ε

<const-assignment-list> ::= <ident> = <number>
                          |      <const-assignment-list> "," <ident> "=" <number> .

<var-decl>     ::= "var" <ident-list> ";"
                |      ε .

<ident-list>   ::= <ident> | <ident-list> "," <ident> .

<proc-decl>    ::= <proc-decl> "procedure" <ident> ";" <block> ";"
                |      ε .

<statement>    ::= <ident> "!=" <expression>
                |      "call" <ident>
                |      "begin" <statement-list> "end"
                |      "if" <condition> "then" <statement>
                |      "while" <condition> "do" <statement>
                |      ε .

<statement-list> ::= <statement> | <statement-list> ";" <statement> .
```

`<condition> ::= "odd" <expression> | <expression> <relation> <expression> .`

`<relation> ::= "=" | "<>" | "<" | ">" | "<=" | ">=" .`

`<expression> ::= <term> | <adding-operator> <term>
 | <expression> <adding-operator> <term>`

`<adding-operator> ::= "+" | "-"`

`<term> ::= <factor> | <term> <multiplying-operator> <factor>`

`<multiplying-operator> ::= "*" | "/"`

`<factor> ::= <ident> | <number> | (<expression>)`

PL/0 Grammar


Terminals

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

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

Non-Terminals

<program> <block> <const-
decl> <var-decl> <proc-decl>
<statement> <const-
assignment-list> <ident>
<number > <ident-list>
<expression> <statement-
list> <condition> <relation>
<term> <adding-operator>
<factor> <multiplying-
operator>



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

<program> ::= <block> .

```
procedure PROGRAM;
```

```
begin
```

```
    GET_TOKEN();
```

```
    BLOCK();
```

```
    if TOKEN <> "." then ERROR (No Period at end of file)
```

```
end;
```

<block> Procedure

$\langle \text{block} \rangle ::= \langle \text{const-decl} \rangle \langle \text{var-decl} \rangle \langle \text{proc-decl} \rangle \langle \text{statement} \rangle$

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

$\langle \text{var-decl} \rangle ::= \text{var } \langle \text{ident-list} \rangle ; \mid \text{e}$
 $\langle \text{ident-list} \rangle ::= \langle \text{ident} \rangle \mid \langle \text{ident-list} \rangle , \langle \text{ident} \rangle$

```
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

<proc-decl> ::= <proc-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;

<statement> Procedure

$\langle \text{statement} \rangle ::= \langle \text{ident} \rangle := \langle \text{expression} \rangle \mid \text{call } \langle \text{ident} \rangle$
 $\mid \text{begin } \langle \text{statement-list} \rangle \text{ end} \mid \text{if } \langle \text{condition} \rangle \text{ then } \langle \text{statement} \rangle$
 $\mid \text{while } \langle \text{condition} \rangle \text{ do } \langle \text{statement} \rangle \mid \text{e}$
 $\langle \text{statement-list} \rangle ::= \langle \text{statement} \rangle \mid \langle \text{statement-list} \rangle ; \langle \text{statement} \rangle$

```
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  
end
```

<statement> Procedure

$\langle \text{statement} \rangle ::= \langle \text{ident} \rangle := \langle \text{expression} \rangle \mid \text{call } \langle \text{ident} \rangle$
 $\mid \text{begin } \langle \text{statement-list} \rangle \text{ end} \mid \text{if } \langle \text{condition} \rangle \text{ then } \langle \text{statement} \rangle$
 $\mid \text{while } \langle \text{condition} \rangle \text{ do } \langle \text{statement} \rangle \mid \text{e}$
 $\langle \text{statement-list} \rangle ::= \langle \text{statement} \rangle \mid \langle \text{statement-list} \rangle ; \langle \text{statement} \rangle$

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;

...

<statement> Procedure

$\langle \text{statement} \rangle ::= \langle \text{ident} \rangle := \langle \text{expression} \rangle \mid \text{call } \langle \text{ident} \rangle$
 $\mid \text{begin } \langle \text{statement-list} \rangle \text{ end} \mid \text{if } \langle \text{condition} \rangle \text{ then } \langle \text{statement} \rangle$
 $\mid \text{while } \langle \text{condition} \rangle \text{ do } \langle \text{statement} \rangle \mid \text{e}$
 $\langle \text{statement-list} \rangle ::= \langle \text{statement} \rangle \mid \langle \text{statement-list} \rangle ; \langle \text{statement} \rangle$

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;

...

<statement> Procedure

$\langle \text{statement} \rangle ::= \langle \text{ident} \rangle := \langle \text{expression} \rangle \mid \text{call } \langle \text{ident} \rangle$
 $\mid \text{begin } \langle \text{statement-list} \rangle \text{ end} \mid \text{if } \langle \text{condition} \rangle \text{ then } \langle \text{statement} \rangle$
 $\mid \text{while } \langle \text{condition} \rangle \text{ do } \langle \text{statement} \rangle \mid \text{e}$
 $\langle \text{statement-list} \rangle ::= \langle \text{statement} \rangle \mid \langle \text{statement-list} \rangle ; \langle \text{statement} \rangle$

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>

```
procedure CONDITION;  
begin  
    if TOKEN = "odd" then begin  
        GET_TOKEN();  
        EXPRESSION();  
    else begin  
        EXPRESSION();  
        if TOKEN <> RELATION then ERROR (relational operator missing in conditional statement);  
        GET_TOKEN();  
        EXPRESSION();  
    end  
end;  
end;
```


<expression> Procedure

$\langle \text{expression} \rangle ::= \langle \text{term} \rangle \mid \langle \text{adding-operator} \rangle \langle \text{term} \rangle$
 $\mid \langle \text{expression} \rangle \langle \text{adding-operator} \rangle \langle \text{term} \rangle$

```
procedure EXPRESSION;  
begin  
  if TOKEN = ADDING_OPERATOR then GET_TOKEN();  
  TERM();  
  while TOKEN = ADDING_OPERATOR do begin  
    GET_TOKEN();  
    TERM();  
  end  
end;  
end;
```

<term> Procedure

$\langle \text{term} \rangle ::= \langle \text{factor} \rangle \mid \langle \text{term} \rangle \langle \text{multiplying-operator} \rangle \langle \text{factor} \rangle$

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

<factor> Procedure

$\text{<factor>} ::= \text{<ident>} \mid \text{<number>} \mid (\text{<expression>})$

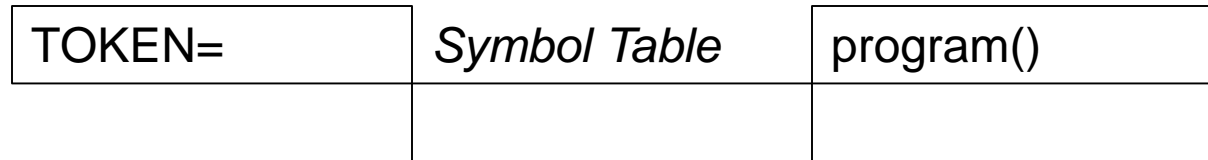
```
procedure FACTOR;  
begin  
    if TOKEN = IDENTIFIER then  
        GET_TOKEN();  
    else if TOKEN = NUMBER then  
        GET_TOKEN();  
    else if TOKEN = "(" then begin  
        GET_TOKEN();  
        EXPRESSION();  
        if TOKEN <> ")" then ERROR( left ( has not been closed );  
   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

Small Example

```

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.

```



```

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

```

Small Example

```

    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	<i>Symbol Table</i>	<i>Recursion stack</i> program()

```

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

```

Small Example

```

    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	<i>Symbol Table</i>

Recursion stack

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;

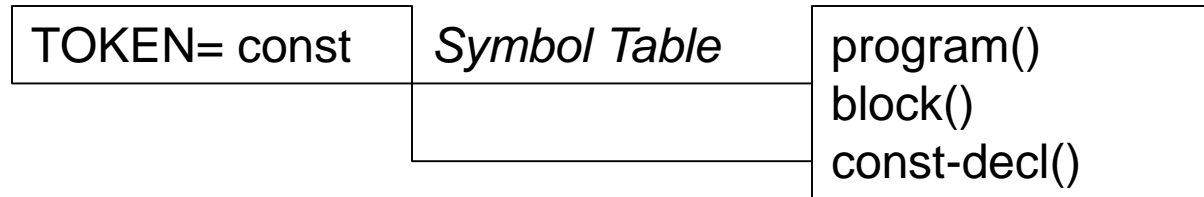
```

Small Example

```

    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;

```

Small Example

```

    = 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

Recursion stack

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;

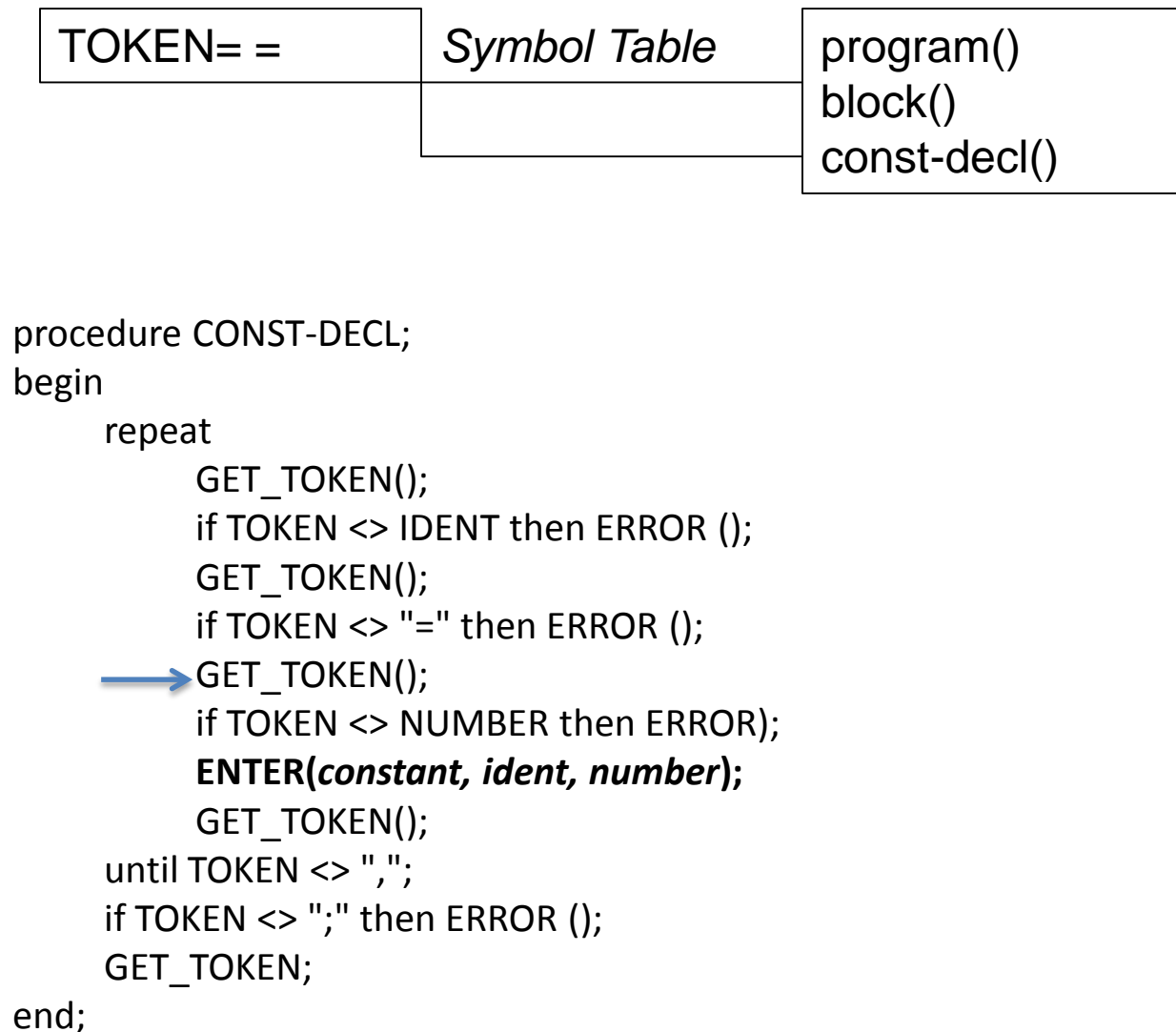
```


Small Example

```

      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.

```



Small Example

```

;
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

Recursion stack

```

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;

```

Small Example

```

;
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	Recursion stack
	m=8;	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;

```

Small Example

```

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= ;	<i>Symbol Table</i>
m=8;	

Recursion stack

```

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;

```

Small Example

```

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;

Recursion stack

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;**

Small Example

```

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;

Recursion stack

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;

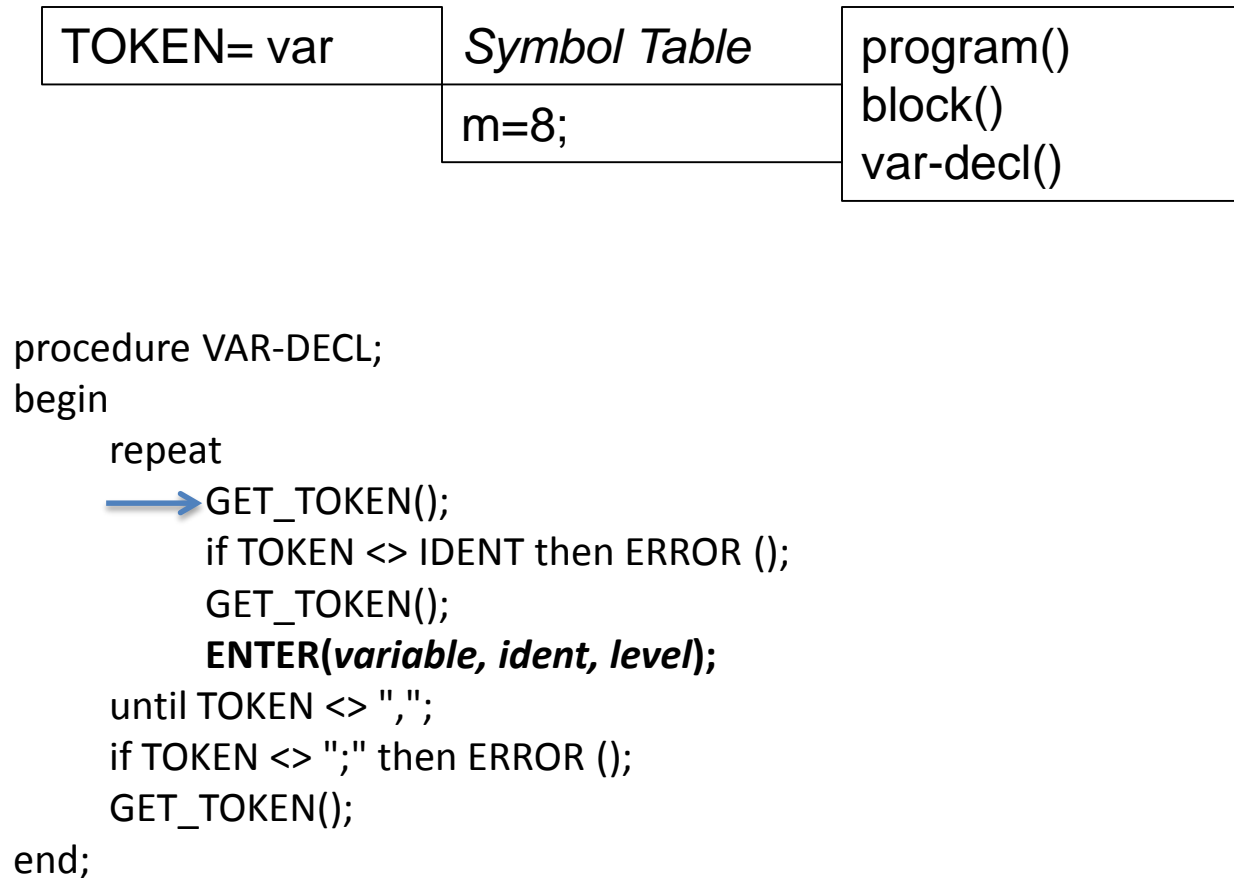
```

Small Example

```

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.

```



Small Example

```

    , 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	Recursion stack
	m=8;	program() 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;

```


Small Example

```

    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 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= ,	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8;	program() block() var-decl()

Small Example

```

    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= ,	<i>Symbol Table</i>
	m=8; a;

Recursion stack

```

program()
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;

```

Small Example

```

, 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 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= b	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a;	program() block() var-decl()

Small Example

```

        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 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= ,	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a;	program() block() var-decl()

Small Example

```

        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 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= ,	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b;	program() block() var-decl()

Small Example

```

;
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();
        ENTER(variable, ident, level);
    until TOKEN <> ",";
    if TOKEN <> ";" then ERROR ();
    GET_TOKEN();
end;

```

TOKEN= c	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b;	program() block() var-decl()

Small Example

Recursion stack

TOKEN= ;	<i>Symbol Table</i>	program()
	m=8; a; b;	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();
        → ENTER(variable, ident, level);
    until TOKEN <> ",";
    if TOKEN <> ";" then ERROR ();
    GET_TOKEN();
end;

```

Small Example

Recursion stack

TOKEN= ;	<i>Symbol Table</i>	program()
	m=8; a; b; c;	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();
        ENTER(variable, ident, level);
    until TOKEN <> ",";
    if TOKEN <> ";" then ERROR ();
    → GET_TOKEN();
end;

```


Small Example

Recursion stack

TOKEN= procedure

Symbol Table

m=8; a; b; c;

program()
block()
var-decl()

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();

ENTER(*variable, ident, level*);

until TOKEN <> ",";

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

GET_TOKEN();

→ **end;**

Small Example

Recursion stack

TOKEN= procedure

Symbol Table

m=8; a; b; c;

program()
block()

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 BLOCK;

begin

if TOKEN = "const" then CONST-DECL();

if TOKEN = "var" then VAR-DECL();

→ if TOKEN = "procedure" then PROC-DECL();
STATEMENT;

end;

Small Example

Recursion stack

TOKEN= procedure

Symbol Table

m=8; a; b; c;

program()
block()
proc-decl()

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 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;
```

Small Example

Recursion stack

TOKEN= ratio

Symbol Table

m=8; a; b; c;

program()
block()
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.
```

```
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;
```

Small Example

Recursion stack

TOKEN= ratio	<i>Symbol Table</i>	program() block() proc-decl()
	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.

```

```

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;

```

Small Example

Recursion stack

TOKEN= ;	<i>Symbol Table</i>	program() block() proc-decl()
	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.

```

```

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;

```

Small Example

Recursion stack

TOKEN= var	<i>Symbol Table</i>	program()
	m=8; a; b; c; ratio;	block() 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;

```

Small Example

Recursion stack

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;

```


Small Example

Recursion stack

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;

```

Small Example

```

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	<i>Symbol Table</i>
	m=8; a; b; c; ratio;

Recursion stack

```

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

```

Small Example

```

    , 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= x	<i>Symbol Table</i>
	m=8; a; b; c; ratio;

Recursion stack

```

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

```

Small Example

```

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= ,	<i>Symbol Table</i>
	m=8; a; b; c; ratio;

Recursion stack

```

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

```

Small Example

```

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= ,	<i>Symbol Table</i>
	m=8; a; b; c; ratio; x;

Recursion stack

```

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

```

Small Example

Recursion stack

TOKEN= y

Symbol Table

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

program()
block(1)
proc-decl(1)
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;

```

Small Example

TOKEN= ;

Symbol Table

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

Recursion stack

program()
block(1)
proc-decl(1)
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;

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) 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;

```


Small Example

TOKEN= begin


Symbol Table

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

Recursion stack

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

```
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;
```

Small Example

TOKEN= begin

Symbol Table

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

Recursion stack

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

```
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;
```

Small Example

TOKEN= begin	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2)

```

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;

```

Small Example

TOKEN= begin

Symbol Table

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

Recursion stack

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

```
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 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;
  end;
...
```

Small Example

TOKEN= x	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(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;
...
    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;
...

```

Small Example

TOKEN= x

Symbol Table

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

Recursion stack

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
    ...

```

Small Example

TOKEN= =	<i>Symbol Table</i>	<i>Recursion stack</i>
	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
    ...

```

Small Example

TOKEN= a

Symbol Table

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

Recursion stack

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

```

```

procedure STATEMENT;
begin
    if TOKEN = IDENT then begin
        GET_TOKEN();
        If TOKEN <> "!=" then ERROR ();
        GET_TOKEN();
        → EXPRESSION();
    end
    ...

```


Small Example

Recursion stack

TOKEN= a	<i>Symbol Table</i>	program()
	m=8; a; b; c; ratio; x; y;	block(1)
		proc-decl(1)
		block(2)
		statement(2)
		statement(2)
		expression(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.

```

```

procedure EXPRESSION;
begin
    if TOKEN = ADDING_OPERATOR then GET_TOKEN();
    → TERM();
    while TOKEN = ADDING_OPERATOR do begin
        GET_TOKEN();
        TERM();
    end
end;

```

Small Example

TOKEN= a

Symbol Table

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

Recursion stack

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

```

```

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

```

Small Example

TOKEN= a	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) expression(2) term(2) factor(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.

```

```

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;

```

Small Example

Recursion stack

TOKEN= ;

Symbol Table

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

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

```

```

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;

Small Example

Recursion stack

TOKEN= ;

Symbol Table

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

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

```

```

procedure TERM;
begin

```

```

    FACTOR();

```

```

    while TOKEN = MULTIPLYING_OPERATOR do begin

```

```

        GET_TOKEN();

```

```

        FACTOR();

```

```

    end

```

→ end;

Small Example

Recursion stack

TOKEN= ;

Symbol Table

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

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

```

```

procedure EXPRESSION;
begin
    if TOKEN = ADDING_OPERATOR then GET_TOKEN();
    TERM();
    while TOKEN = ADDING_OPERATOR do begin
        GET_TOKEN();
        TERM();
    end
end;
→ end;

```

Small Example

TOKEN= ;

Symbol Table

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

Recursion stack

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

```

```

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

```

Small Example

TOKEN= ;

Symbol Table

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

Recursion stack

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.

```

```

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;
...

```


Small Example

TOKEN= ;

Symbol Table

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

Recursion stack

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.

```

```

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;
...

```

Small Example

Recursion stack

TOKEN= y

Symbol Table

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

program()
block(1)
proc-decl(1)
block(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;
...
    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;
...

```

Small Example

TOKEN= y	<i>Symbol Table</i>	<i>Recursion stack</i>
	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
    ...

```

Small Example

TOKEN= =	<i>Symbol Table</i>	<i>Recursion stack</i>
	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
    ...

```

Small Example

TOKEN= b	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(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
  ...

```

Small Example

TOKEN= b	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) expression(2)

```

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

```

```

procedure EXPRESSION;
begin
    if TOKEN = ADDING_OPERATOR then GET_TOKEN();
    → TERM();
    while TOKEN = ADDING_OPERATOR do begin
        GET_TOKEN();
        TERM();
    end
end;

```

Small Example

TOKEN= b

Symbol Table

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

Recursion stack

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

```

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

```

```

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

```

Small Example

TOKEN= b	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) expression(2) term(2) factor(2)

```

;
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;
end;

```


Small Example

Recursion stack

TOKEN= ;

Symbol Table

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

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

```

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;**

Small Example

TOKEN= ;

Symbol Table

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

Recursion stack

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

```

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

```

```

procedure TERM;
begin

```

```

  FACTOR();

```

```

  while TOKEN = MULTIPLYING_OPERATOR do begin

```

```

    GET_TOKEN();

```

```

    FACTOR();

```

```

  end

```

→ **end;**

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) expression(2)

```

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

```

```

procedure EXPRESSION;
begin
  if TOKEN = ADDING_OPERATOR then GET_TOKEN();
  TERM();
  while TOKEN = ADDING_OPERATOR do begin
    GET_TOKEN();
    TERM();
  end
end;

```



Small Example


TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(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
    ...

```

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) 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;
...
    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;
...

```

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) 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;
...
    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;
...

```

Small Example

TOKEN= if	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() 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;
...

```

Small Example

TOKEN= if	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	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.

```

```

procedure STATEMENT;
...
    else if TOKEN = "if" then begin
        → GET_TOKEN();
        CONDITION();
        if TOKEN <> "then" then ERROR ();
        GET_TOKEN();
        STATEMENT();
    end;
...

```


Small Example

TOKEN= b	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2)

```

    > 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 = "if" then begin
    GET_TOKEN();
    → CONDITION();
    if TOKEN <> "then" then ERROR ();
    GET_TOKEN();
    STATEMENT();
  end;
...

```

Small Example

TOKEN= b	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) condition(2)
<pre> > a then begin x = b; y = a; end c = x / y; end; begin a = m; b = 4; call ratio; end. </pre>	<pre> procedure CONDITION; begin → if TOKEN = "odd" then begin GET_TOKEN(); EXPRESSION(); else begin EXPRESSION(); if TOKEN <> RELATION then ERROR (); GET_TOKEN(); EXPRESSION(); end end; </pre>	

Small Example

Recursion stack

TOKEN= b	<i>Symbol Table</i>	program()
	m=8; a; b; c; ratio; x; y;	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;
end;

```

Small Example

Recursion stack

TOKEN= >	<i>Symbol Table</i>	program()
	m=8; a; b; c; ratio; x; y;	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;
end;

```

Small Example

Recursion stack

TOKEN= a	<i>Symbol Table</i>	<i>Recursion stack</i> program() block(1) proc-decl(1) block(2) statement(2) statement(2) condition(2)
	m=8; a; b; c; ratio; x; y;	

```

    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;

```

Small Example

Recursion stack

TOKEN= then

Symbol Table

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

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

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;
→ end;
```

Small Example

TOKEN= then	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2)

```

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

procedure STATEMENT;
...
    else if TOKEN = "if" then begin
        GET_TOKEN();
        CONDITION();
        if TOKEN <> "then" then ERROR ();
        → GET_TOKEN();
        STATEMENT();
    end;
...

```

Small Example

TOKEN= begin

Symbol Table

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

Recursion stack

program()
block(1)
proc-decl(1)
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.

Small Example

TOKEN= begin

Symbol Table

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

Recursion stack

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

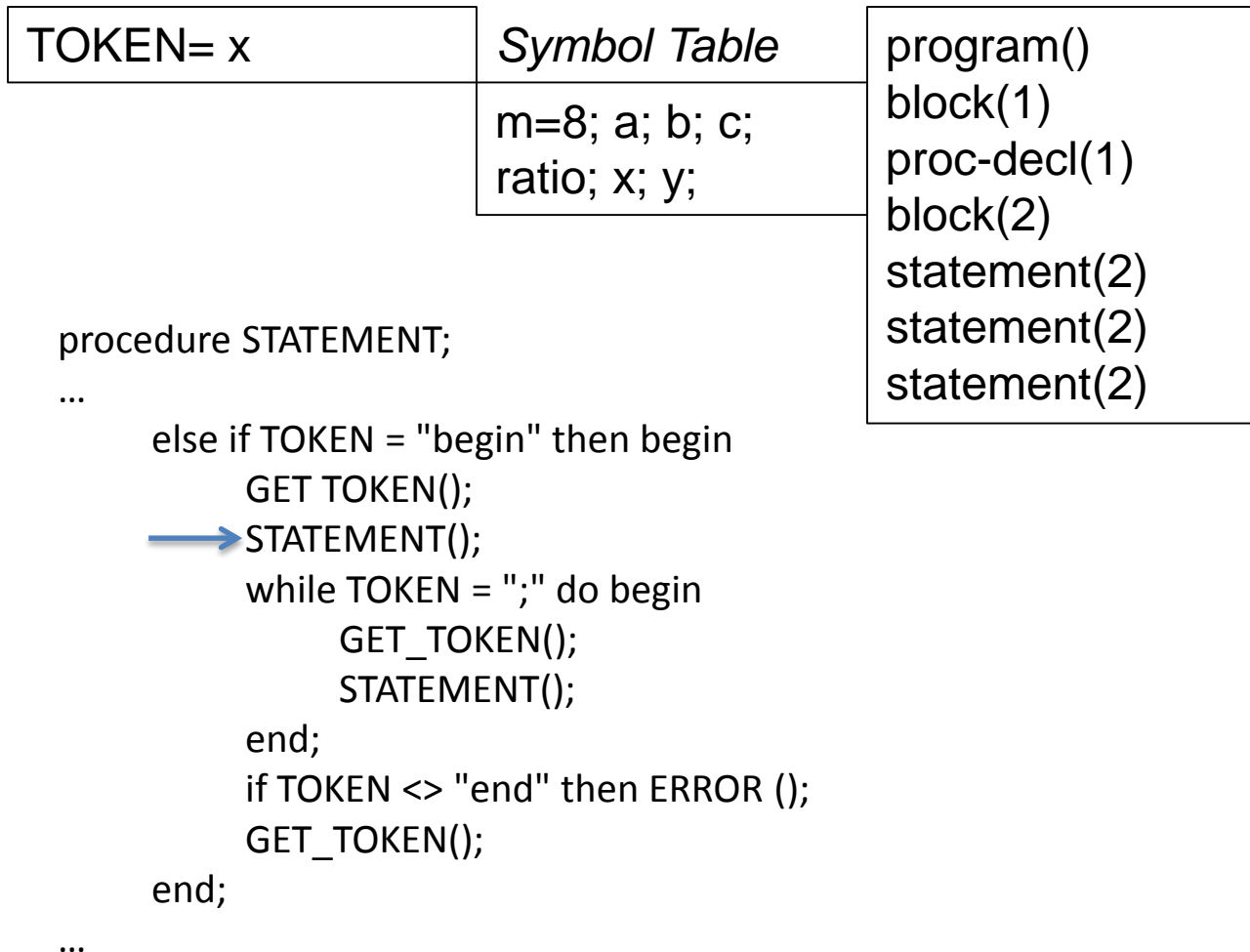
GET_TOKEN();

end;

...

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

Small Example



Small Example

TOKEN= ;

Symbol Table

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

Recursion stack

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

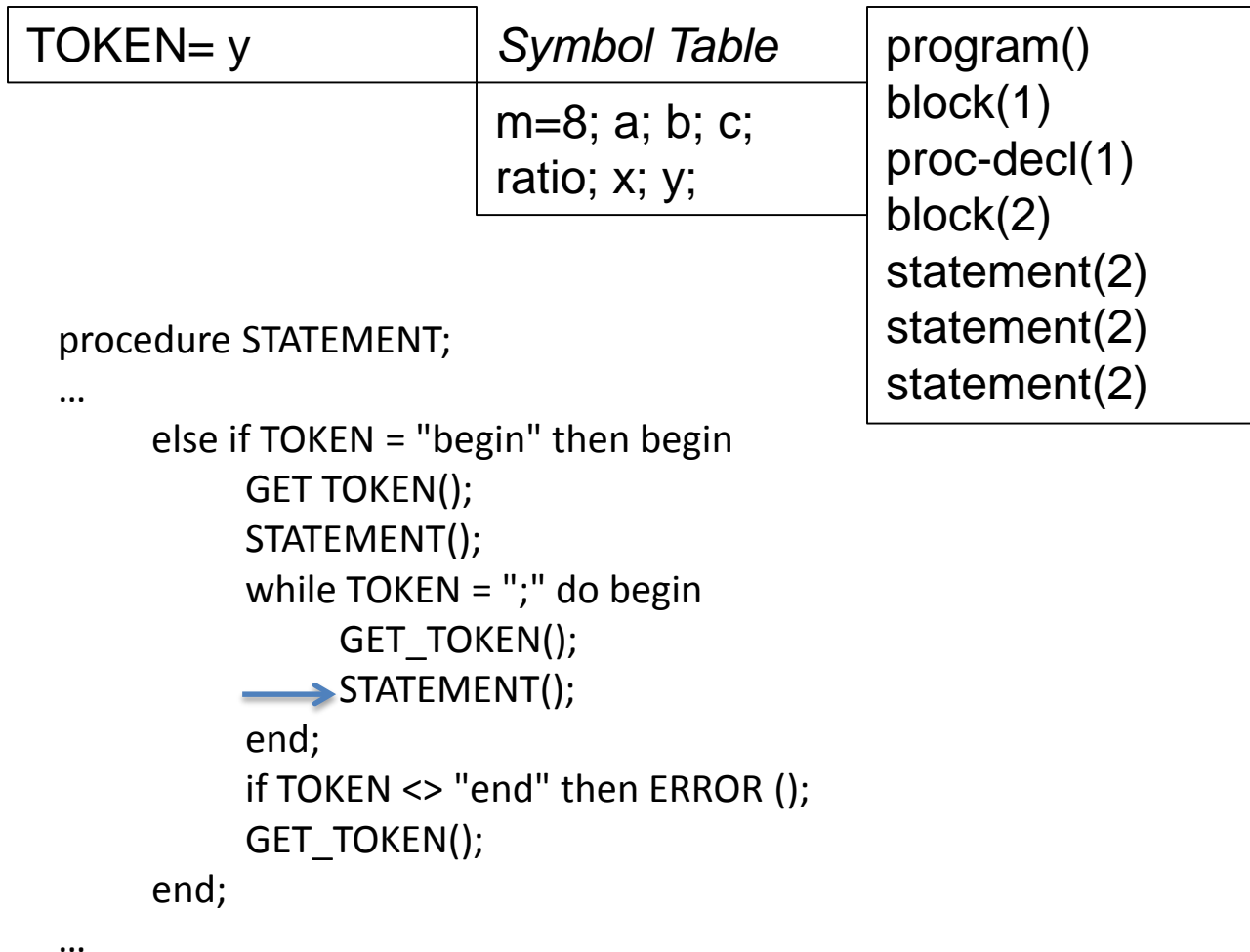
GET_TOKEN();

end;

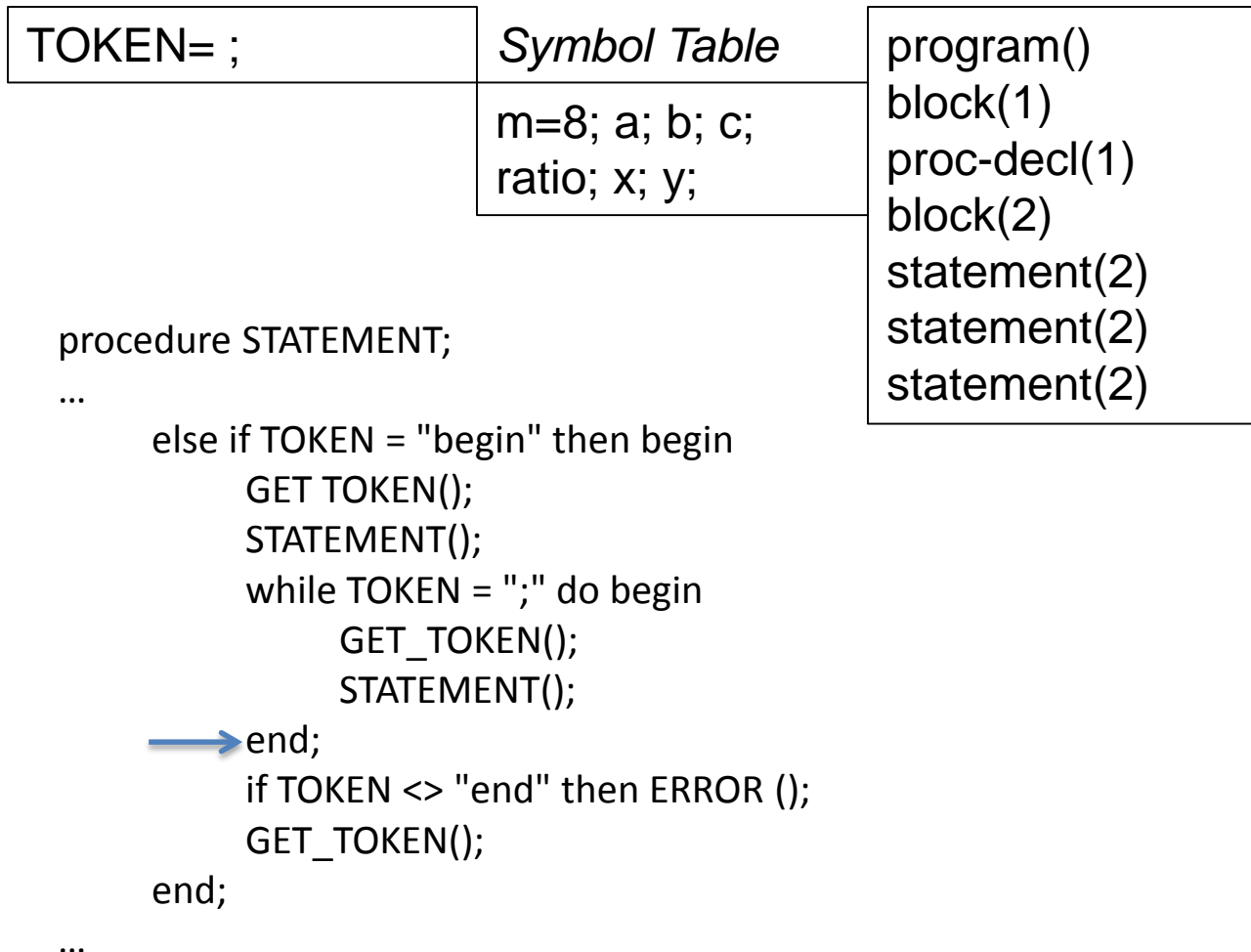
...

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

Small Example



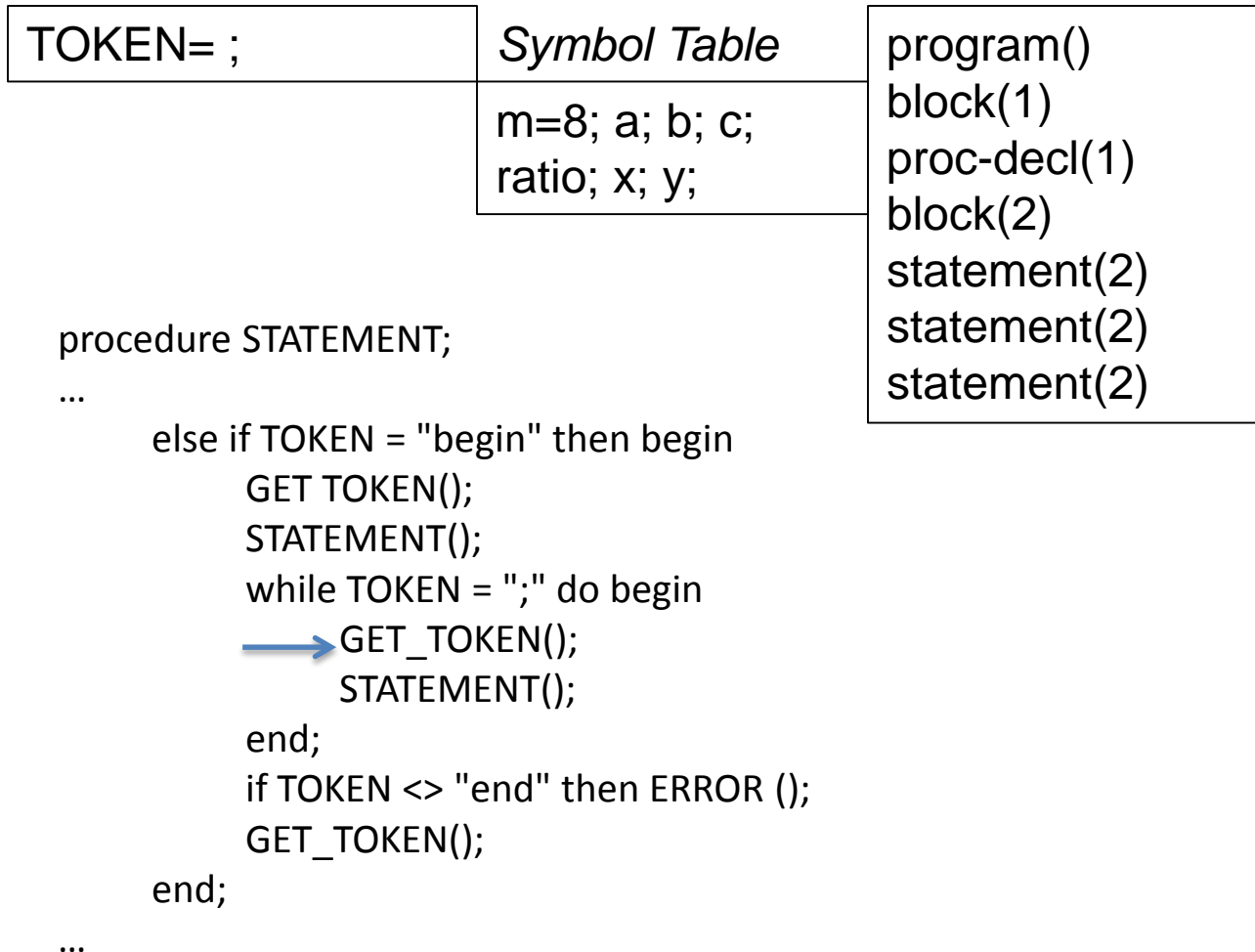
Small Example



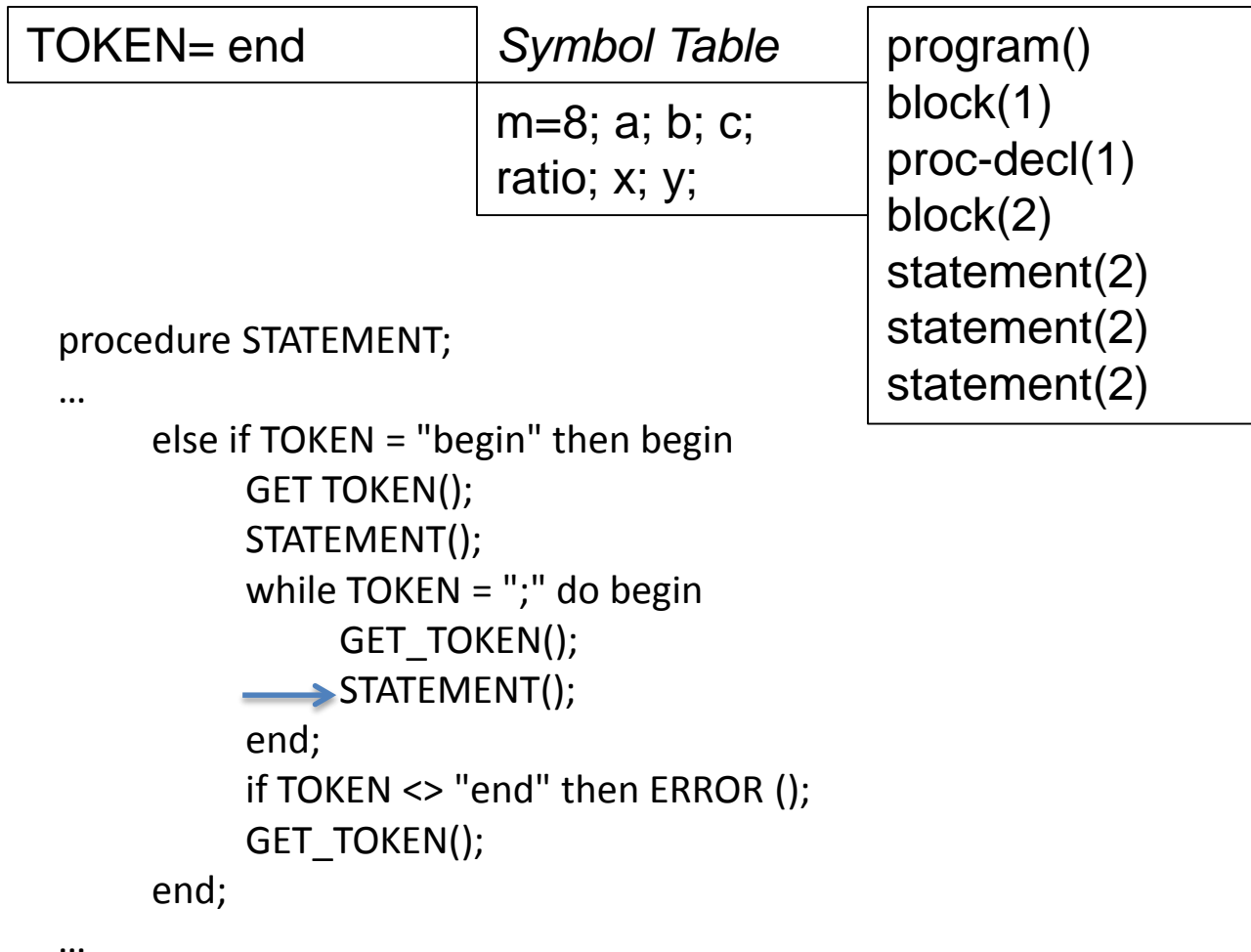
```

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

Small Example



Small Example



Small Example

TOKEN= end

Symbol Table

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

Recursion stack

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

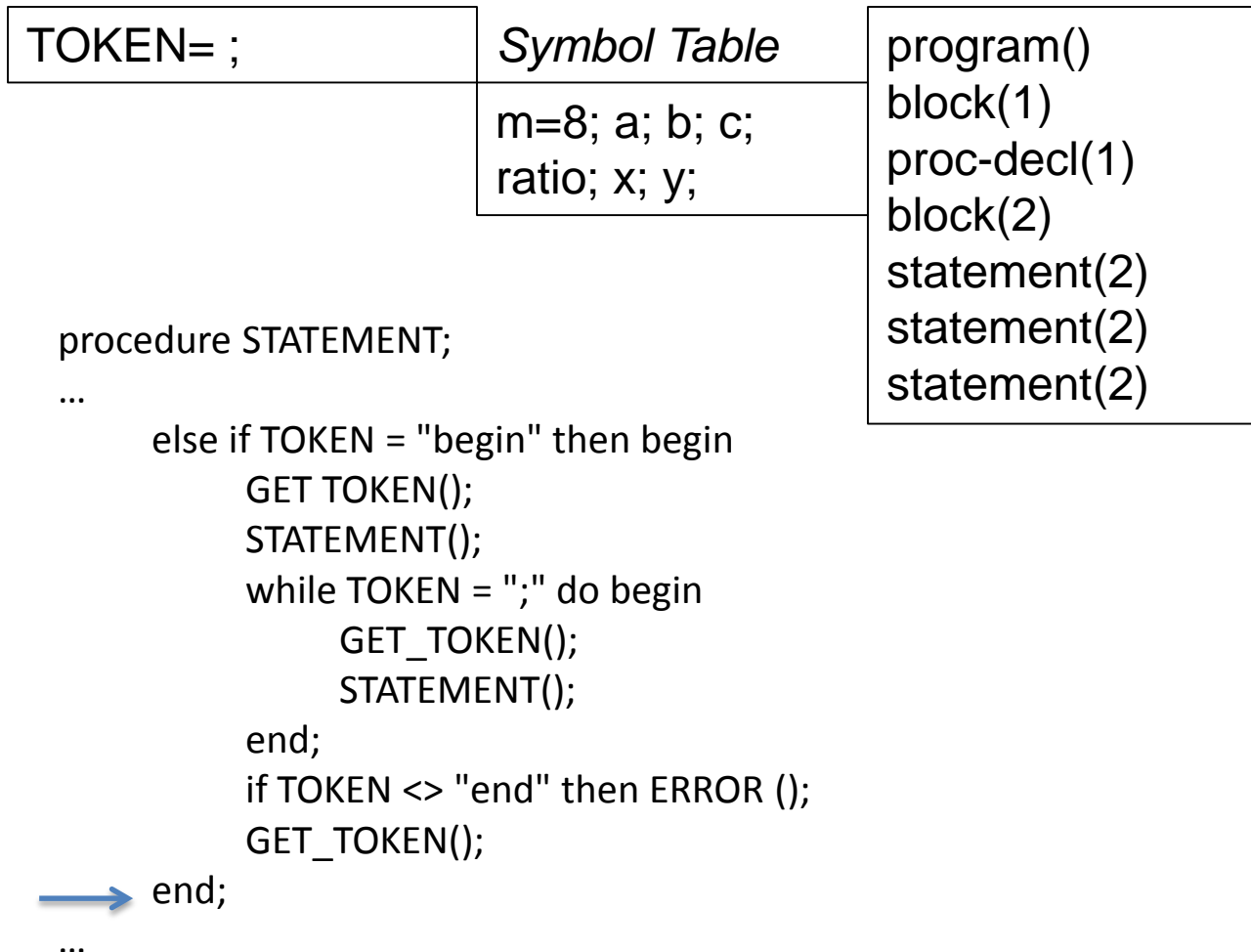
→ GET_TOKEN();

end;

...

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

Small Example



```

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

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) 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.

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() 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;

...

```

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

```

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() 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;

...

```

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

```

Small Example

TOKEN= c	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() 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;

...

```

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

```

Small Example

TOKEN= c	<i>Symbol Table</i>	<i>Recursion stack</i>
	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

```

```

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

```

Small Example

TOKEN= =	<i>Symbol Table</i>	<i>Recursion stack</i>
	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

```

```

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

```

Small Example

TOKEN= x	<i>Symbol Table</i>	<i>Recursion stack</i>
	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
end;
begin
    a = m;
    b = 4;
    call ratio;
end.
    / y;

```


Small Example

TOKEN= x	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) 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;
end;
begin
    a = m;
    b = 4;
    call ratio;
end.

```

Small Example

TOKEN= x	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) expression(2) term(2)

```

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

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

```

Small Example

TOKEN= x	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) expression(2) term(2) factor(2)

```

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;

```

Small Example

TOKEN= /	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) expression(2) term(2) factor(2)

```

        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;
→ end;

```

Small Example

TOKEN= /	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) expression(2) term(2)

```

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

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

```

Small Example

TOKEN= y	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) expression(2) term(2)

```

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

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

```

Small Example

TOKEN= ;

Symbol Table

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

Recursion stack

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

procedure TERM;
begin

FACTOR();

while TOKEN = MULTIPLYING_OPERATOR do begin

GET_TOKEN();

FACTOR();

end

end;

begin

a = m;

b = 4;

call ratio;

end.

→ end;

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1) block(2) statement(2) statement(2) expression(2)

```

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

```

```

procedure EXPRESSION;
begin
    if TOKEN = ADDING_OPERATOR then GET_TOKEN();
    TERM();
    while TOKEN = ADDING_OPERATOR do begin
        GET_TOKEN();
        TERM();
    end
end;

```

→ end;

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	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

```

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

```

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() 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;

...

end;

begin

a = m;

b = 4;

call ratio;

end.

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() 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;

...

end;

begin

a = m;

b = 4;

call ratio;

end.

Small Example

TOKEN= end	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() 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;

...

```

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

```

Small Example

TOKEN= end	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() 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;

...

```

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

```

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() 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;

...

begin

a = m;

b = 4;

call ratio;

end.

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	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.

```

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1)

```
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;
```


Small Example

TOKEN= begin	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) proc-decl(1)

```

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;

```

Small Example

TOKEN= begin	<i>Symbol Table</i>	<i>Recursion stack</i>
	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.

Small Example

TOKEN= begin	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

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.

Small Example

TOKEN= a	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

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;

...

 = m;

 b = 4;

call ratio;

end.

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

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;

...

b = 4;

call ratio;

end.

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

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;

...

b = 4;

call ratio;

end.

Small Example

TOKEN= b	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

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;

...

= 4;

call ratio;

end.

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

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;

...

call ratio;

end.

Small Example

TOKEN= call	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

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;

...

ratio;

end.

Small Example

TOKEN= ;	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

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.

end;

...

Small Example

TOKEN= end	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

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;

...

Small Example

TOKEN= end	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

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;

...

Small Example

TOKEN= .	<i>Symbol Table</i>	<i>Recursion stack</i>
	m=8; a; b; c; ratio; x; y;	program() block(1) statement(1)

```

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;
...

```

Small Example

TOKEN= .	<i>Symbol Table</i>	<i>Recursion stack</i>
	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;

Small Example

	<i>Symbol Table</i>	<i>Recursion stack</i>
TOKEN= .	m=8; a; b; c; ratio; x; y;	program()

```

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