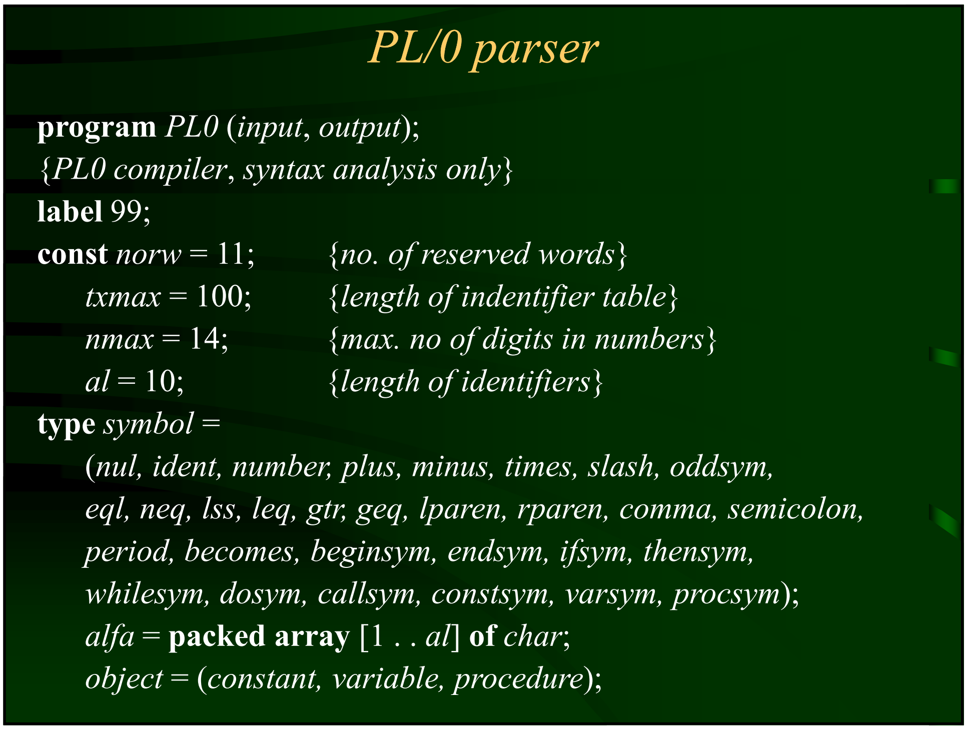
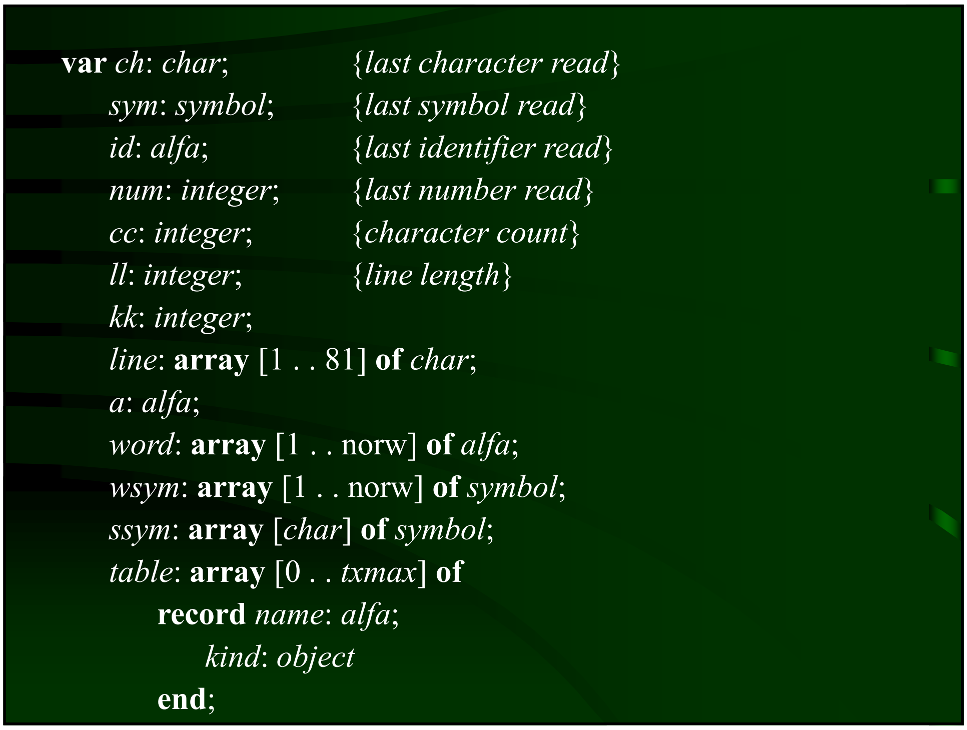
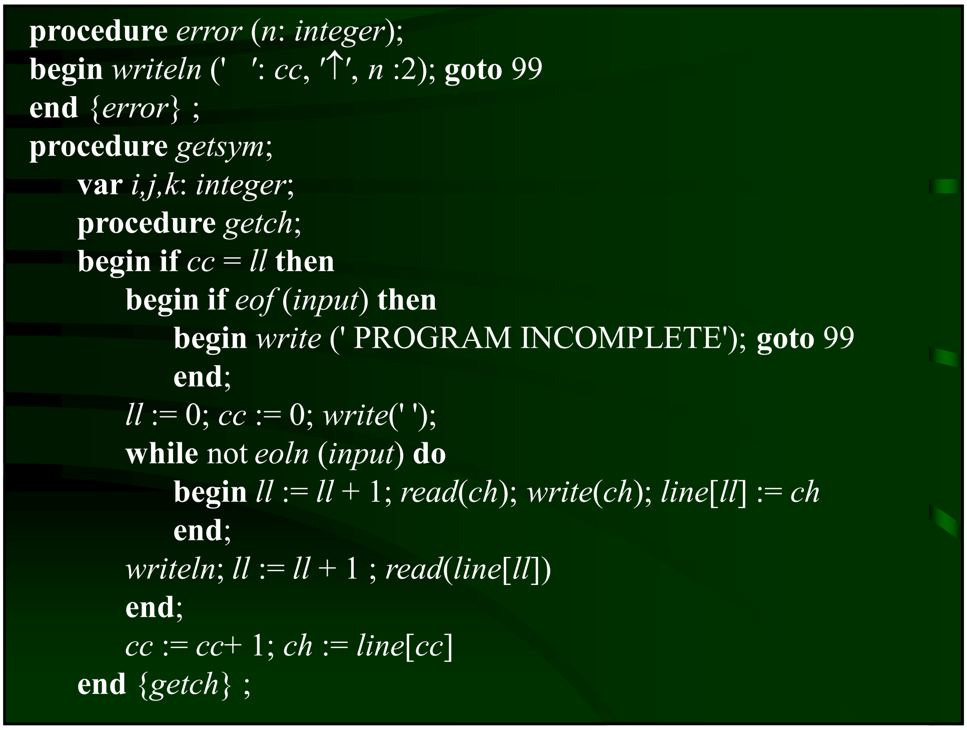


PL/0 parser

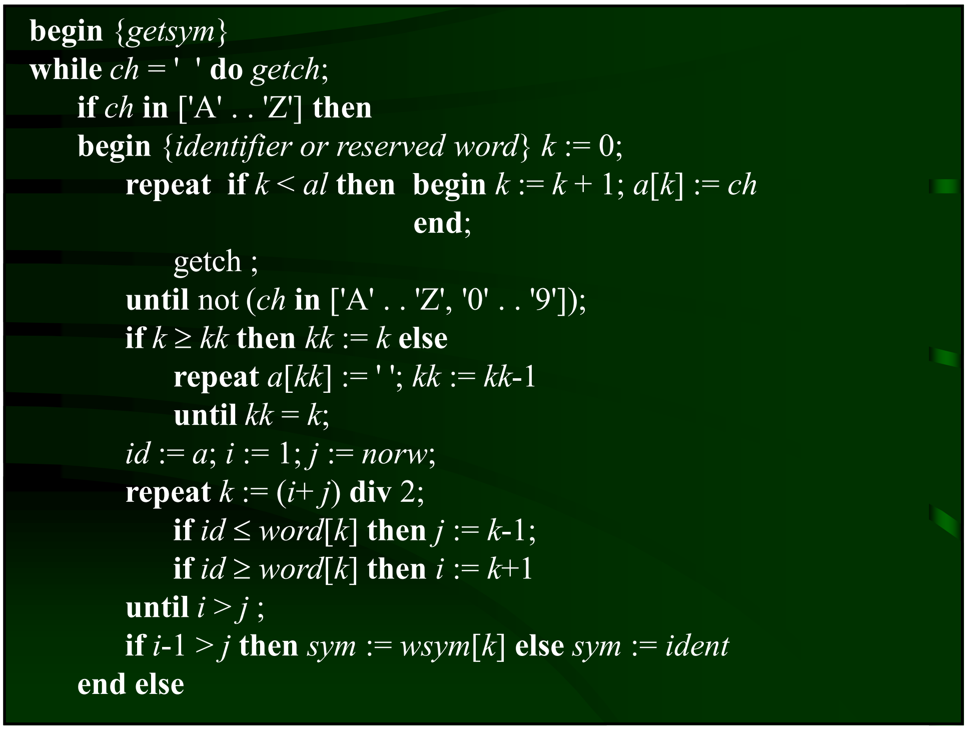
```
program PL0 (input, output);
{PL0 compiler, syntax analysis only}
label 99;
const norw = 11;        {no. of reserved words}
    txmax = 100;        {length of indentifier table}
    nmax = 14;          {max. no of digits in numbers}
    al = 10;            {length of identifiers}
type symbol =
    (nul, ident, number, plus, minus, times, slash, oddsym,
    eql, neq, lss, leq, gtr, geq, lparen, rparen, comma, semicolon,
    period, becomes, beginsym, endsym, ifsym, thensym,
    whilesym, dosym, callsym, constsym, varsym, procsym);
    alfa = packed array [1 . . al] of char;
    object = (constant, variable, procedure);
```



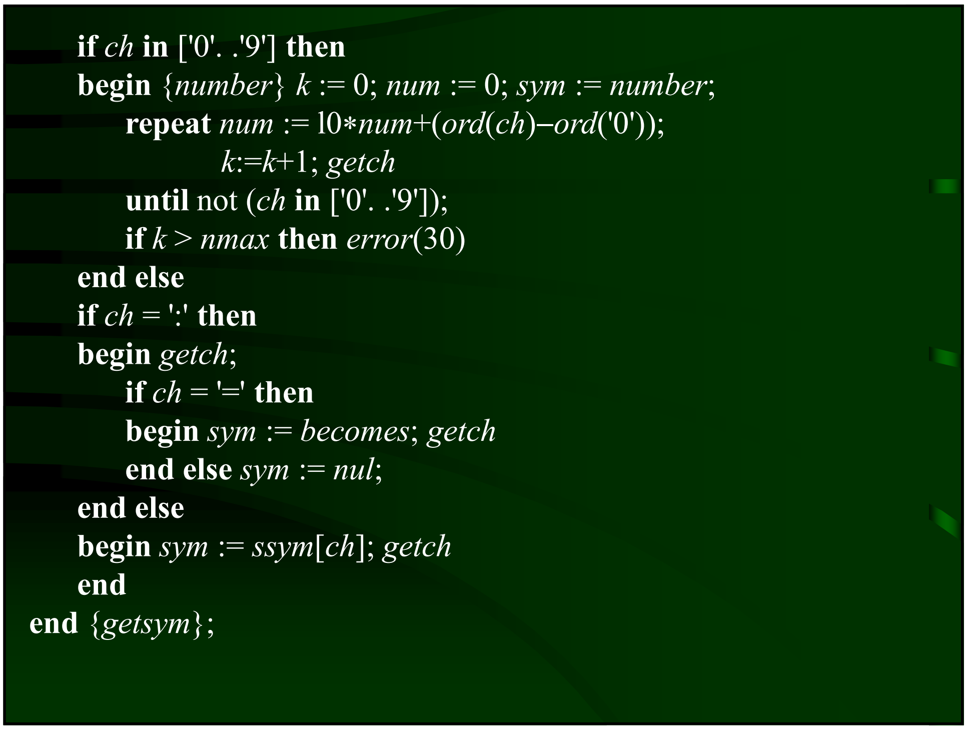
```
var ch: char;           {last character read}
    sym: symbol;        {last symbol read}
    id: alfa;           {last identifier read}
    num: integer;       {last number read}
    cc: integer;        {character count}
    ll: integer;        {line length}
    kk: integer;
    line: array [1 . . 81] of char;
    a: alfa;
    word: array [1 . . norw] of alfa;
    wsym: array [1 . . norw] of symbol;
    ssym: array [char] of symbol;
    table: array [0 . . txmax] of
        record name: alfa;
            kind: object
        end;
```



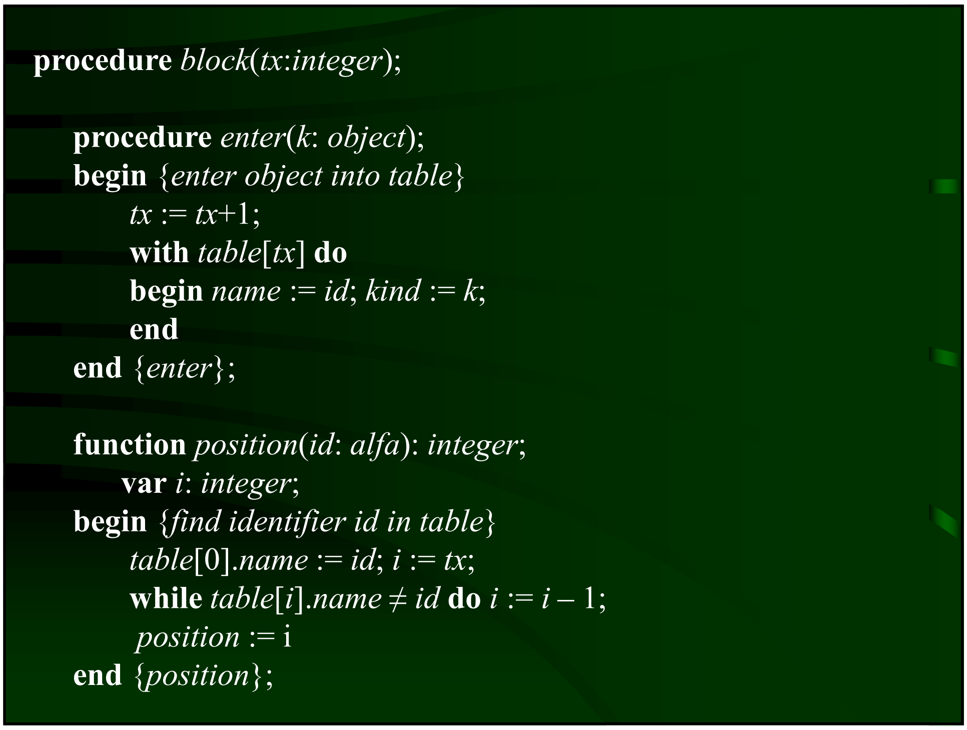
```
procedure error (n: integer);
begin writeln ('    ': cc, '↑', n :2); goto 99
end {error} ;
procedure getsym;
    var i,j,k: integer;
    procedure getch;
    begin if cc = ll then
        begin if eof (input) then
            begin write (' PROGRAM INCOMPLETE'); goto 99
            end;
        ll := 0; cc := 0; write(' ');
        while not eoln (input) do
            begin ll := ll + 1; read(ch); write(ch); line[ll] := ch
            end;
        writeln; ll := ll + 1 ; read(line[ll])
        end;
        cc := cc+ 1; ch := line[cc]
    end {getch} ;
```



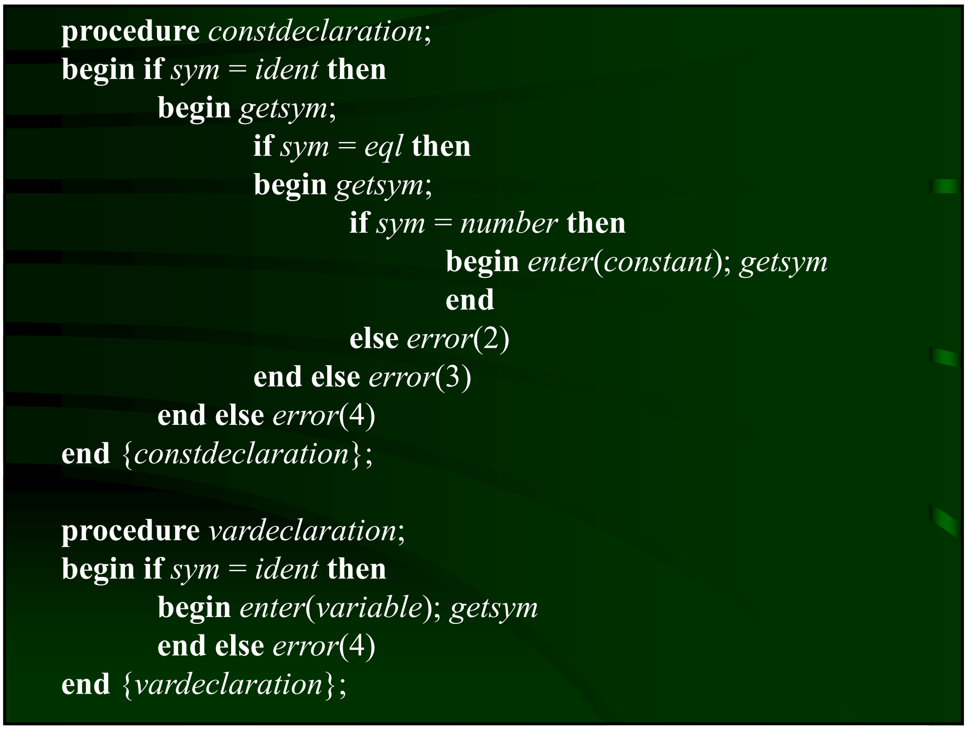
```
begin {getsym}
while ch = ' ' do getch;
    if ch in ['A' . . 'Z'] then
    begin {identifier or reserved word} k := 0;
        repeat  if k < al then  begin k := k + 1; a[k] := ch
                                end;
            getch ;
        until not (ch in ['A' . . 'Z', '0' . . '9']);
        if k ≥ kk then kk := k else
            repeat a[kk] := ' '; kk := kk-1
            until kk = k;
        id := a; i := 1; j := norw;
        repeat k := (i+j) div 2;
            if id ≤ word[k] then j := k-1;
            if id ≥ word[k] then i := k+1
        until i > j ;
        if i-1 > j then sym := wsym[k] else sym := ident
    end else
```



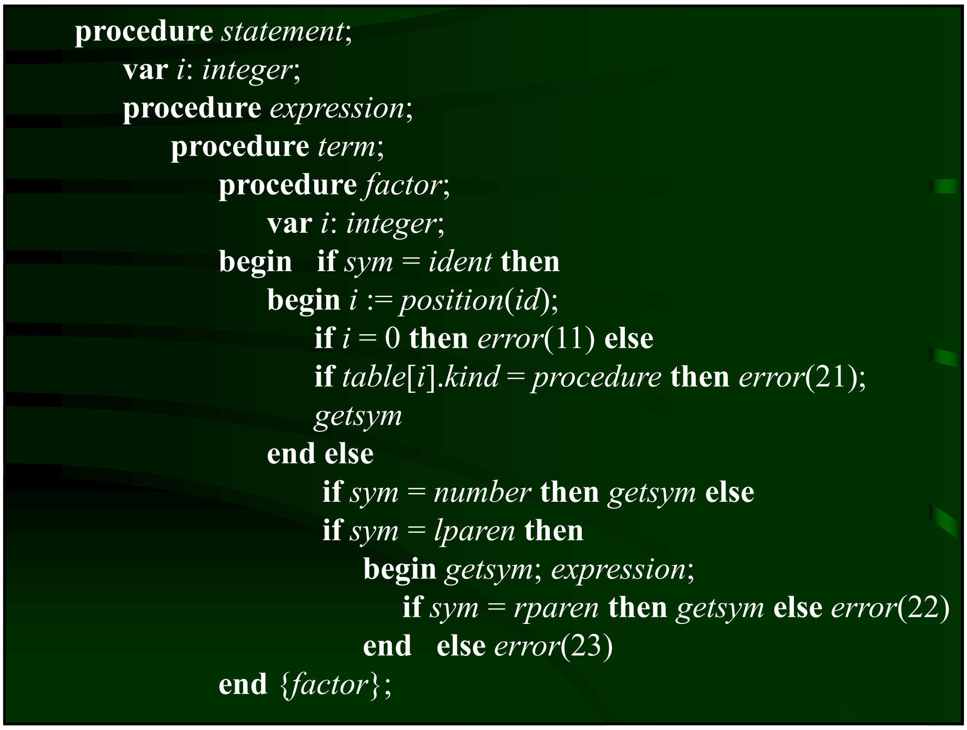
```
    if ch in ['0'. .'9'] then
    begin {number} k := 0; num := 0; sym := number;
        repeat num := 10*num+(ord(ch)–ord('0'));
                k:=k+1; getch
        until not (ch in ['0'. .'9']);
        if k > nmax then error(30)
    end else
    if ch = ':' then
    begin getch;
        if ch = '=' then
        begin sym := becomes; getch
        end else sym := nul;
    end else
    begin sym := ssym[ch]; getch
    end
end {getsym};
```



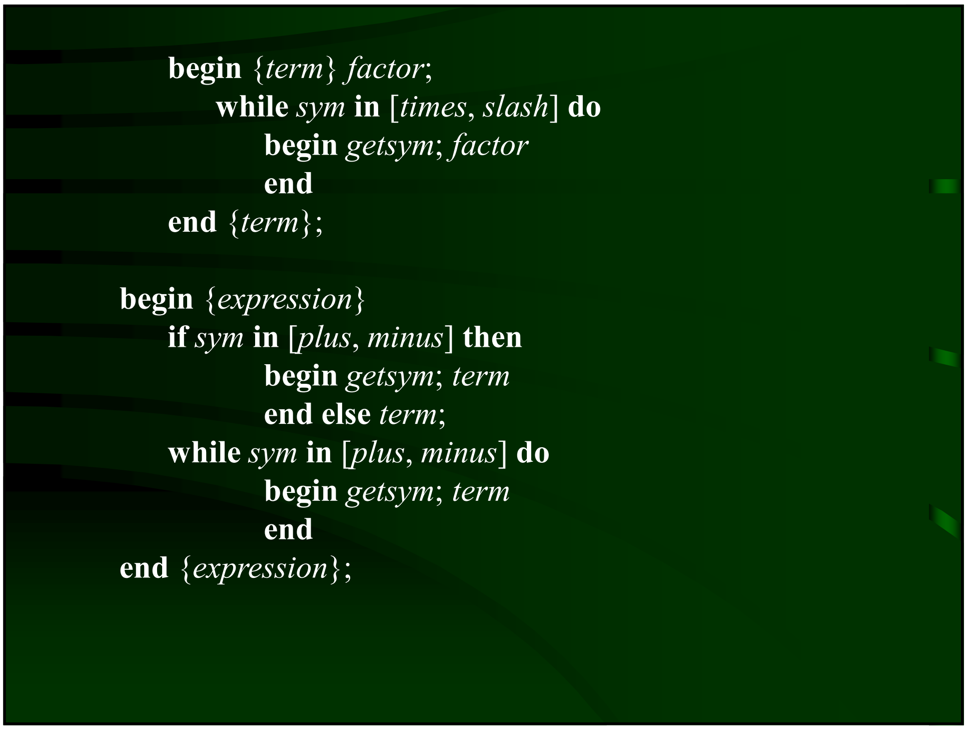
```
procedure block(tx:integer);

    procedure enter(k: object);
    begin {enter object into table}
        tx := tx+1;
        with table[tx] do
        begin name := id; kind := k;
        end
    end {enter};

    function position(id: alfa): integer;
        var i: integer;
    begin {find identifier id in table}
        table[0].name := id; i := tx;
        while table[i].name ≠ id do i := i – 1;
         position := i
    end {position};
```



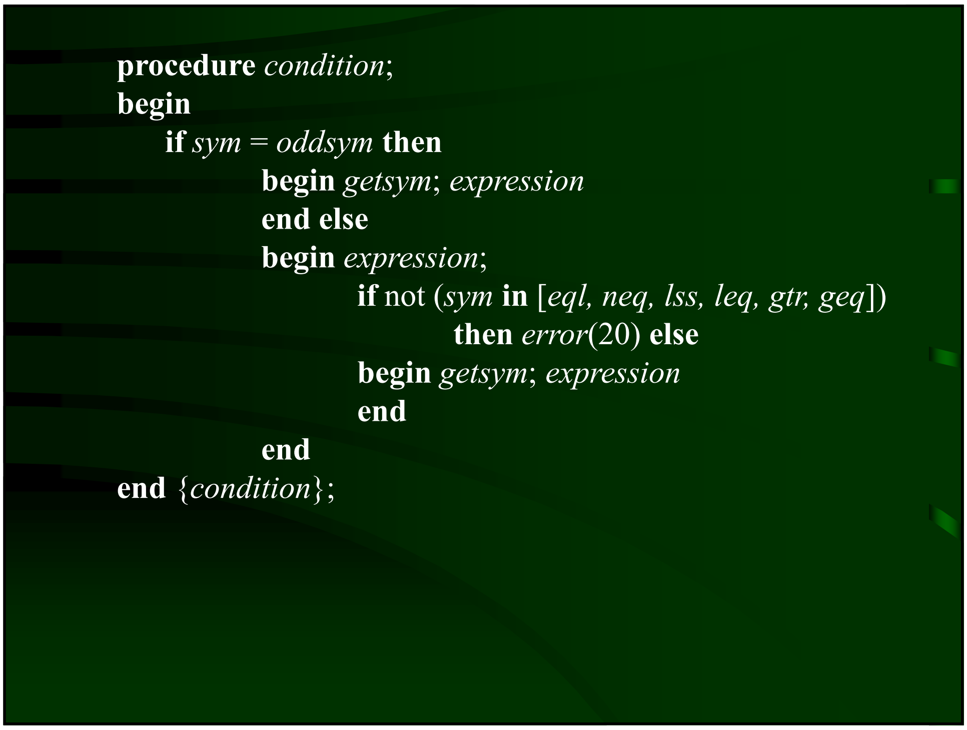
```
procedure constdeclaration;
begin if sym = ident then
        begin getsym;
                if sym = eql then
                begin getsym;
                        if sym = number then
                                begin enter(constant); getsym
                                end
                        else error(2)
                end else error(3)
        end else error(4)
end {constdeclaration};

procedure vardeclaration;
begin if sym = ident then
        begin enter(variable); getsym
        end else error(4)
end {vardeclaration};
```



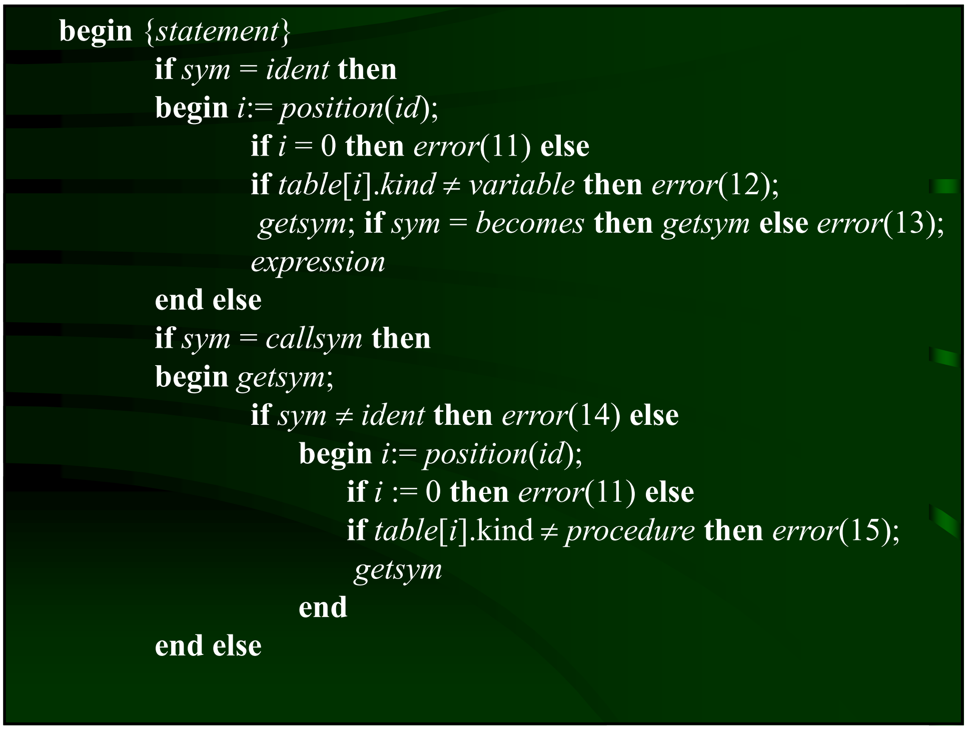
```
procedure statement;
    var i: integer;
    procedure expression;
        procedure term;
            procedure factor;
                var i: integer;
            begin   if sym = ident then
                begin i := position(id);
                    if i = 0 then error(11) else
                    if table[i].kind = procedure then error(21);
                    getsym
                end else
                    if sym = number then getsym else
                    if sym = lparen then
                        begin getsym; expression;
                            if sym = rparen then getsym else error(22)
                        end   else error(23)
            end {factor};
```



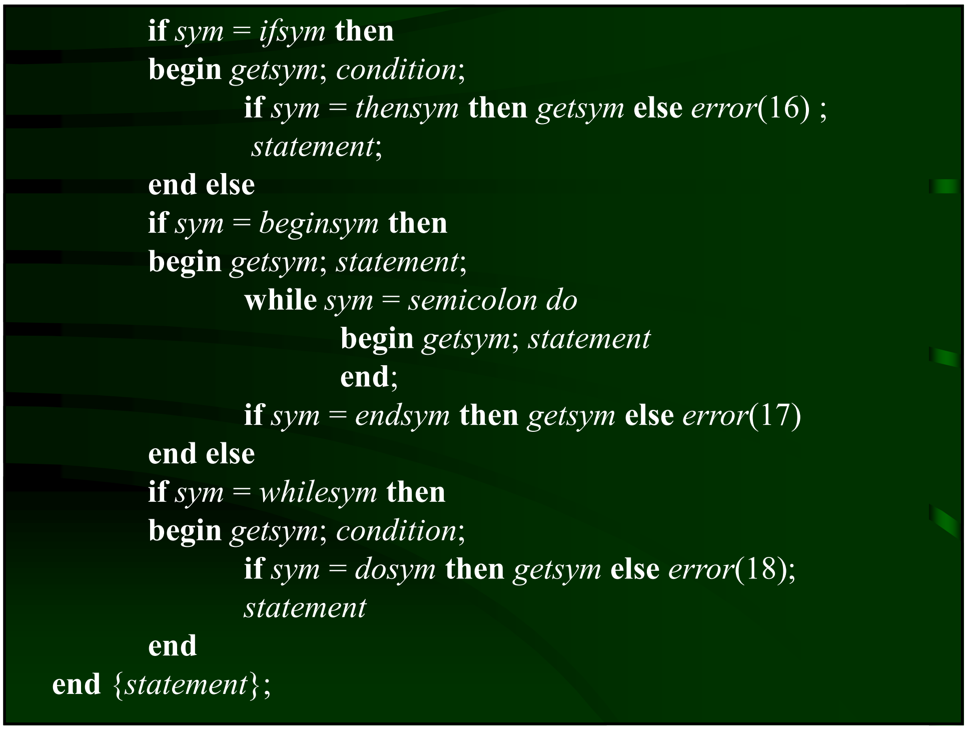
```
      begin {term} factor;
          while sym in [times, slash] do
              begin getsym; factor
              end
      end {term};

  begin {expression}
      if sym in [plus, minus] then
              begin getsym; term
              end else term;
      while sym in [plus, minus] do
              begin getsym; term
              end
  end {expression};
```



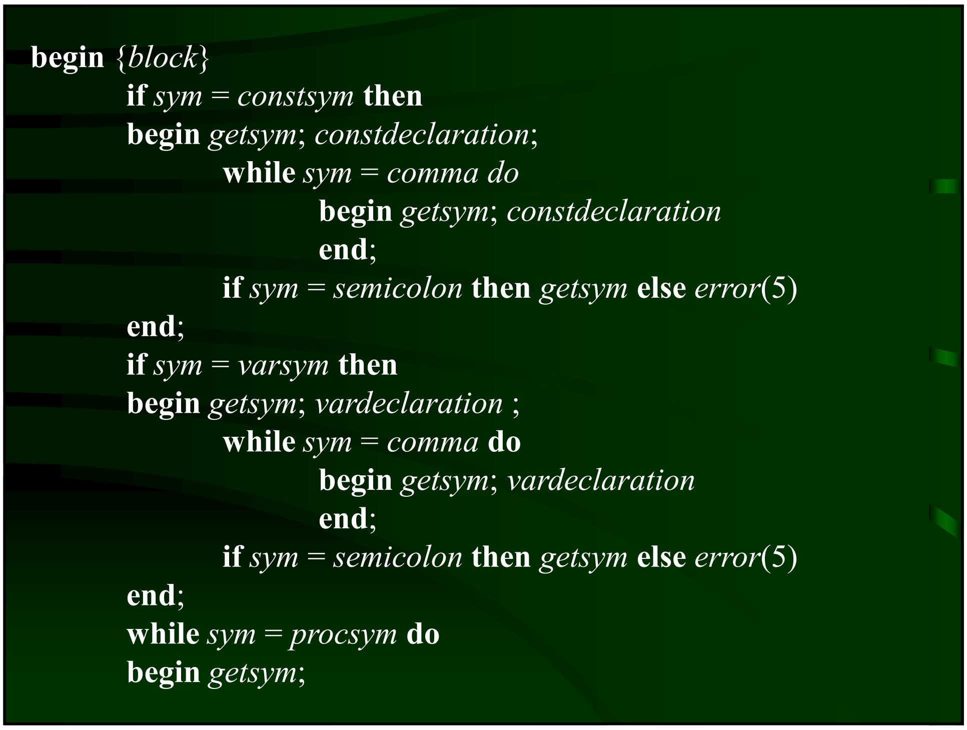
```
procedure condition;
begin
    if sym = oddsym then
            begin getsym; expression
            end else
            begin expression;
                    if not (sym in [eql, neq, lss, leq, gtr, geq])
                            then error(20) else
                    begin getsym; expression
                    end
            end
end {condition};
```



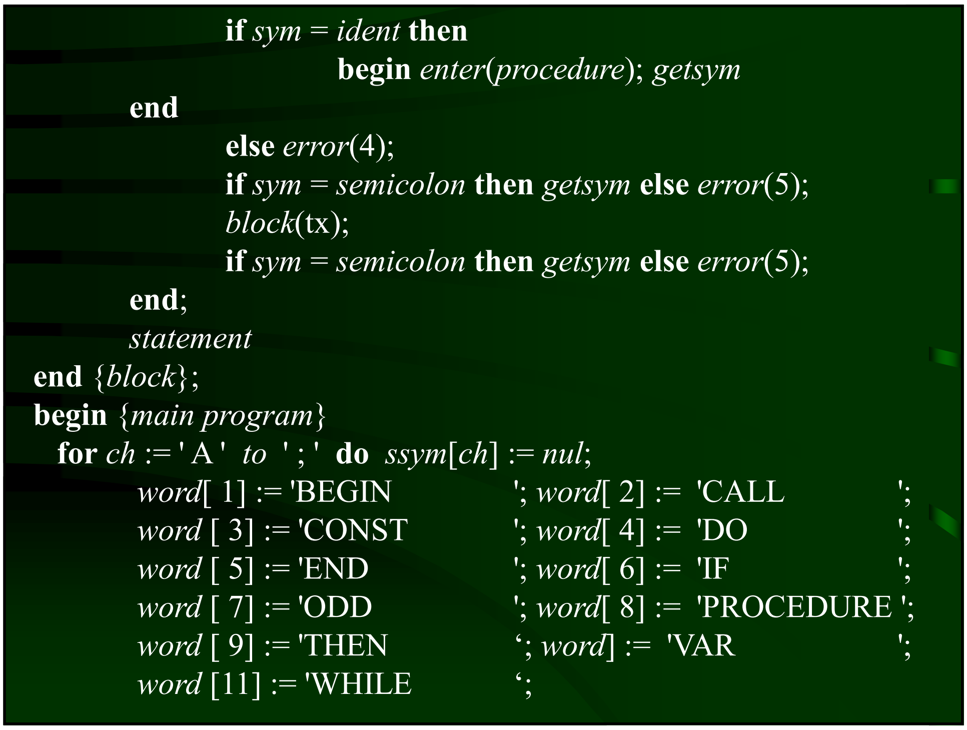
```
begin {statement}
    if sym = ident then
    begin i:= position(id);
        if i = 0 then error(11) else
        if table[i].kind ≠ variable then error(12);
        getsym; if sym = becomes then getsym else error(13);
        expression
    end else
    if sym = callsym then
    begin getsym;
        if sym ≠ ident then error(14) else
          begin i:= position(id);
            if i := 0 then error(11) else
            if table[i].kind ≠ procedure then error(15);
            getsym
          end
    end else
```



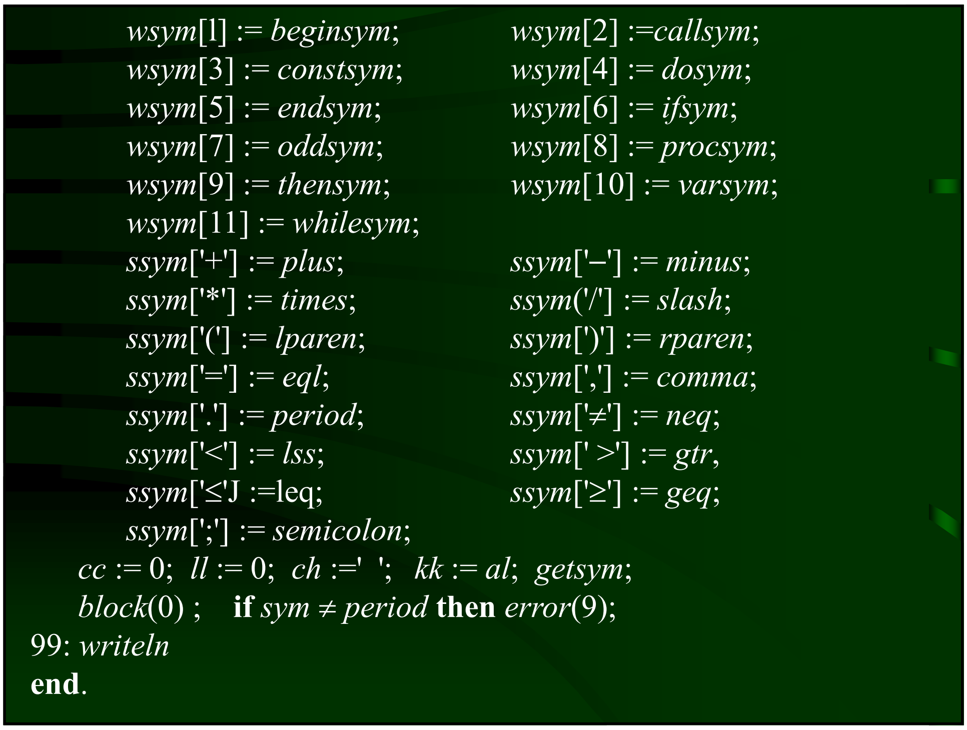
```
    if sym = ifsym then
    begin getsym; condition;
        if sym = thensym then getsym else error(16) ;
        statement;
    end else
    if sym = beginsym then
    begin getsym; statement;
        while sym = semicolon do
            begin getsym; statement
            end;
        if sym = endsym then getsym else error(17)
    end else
    if sym = whilesym then
    begin getsym; condition;
        if sym = dosym then getsym else error(18);
        statement
    end
end {statement};
```



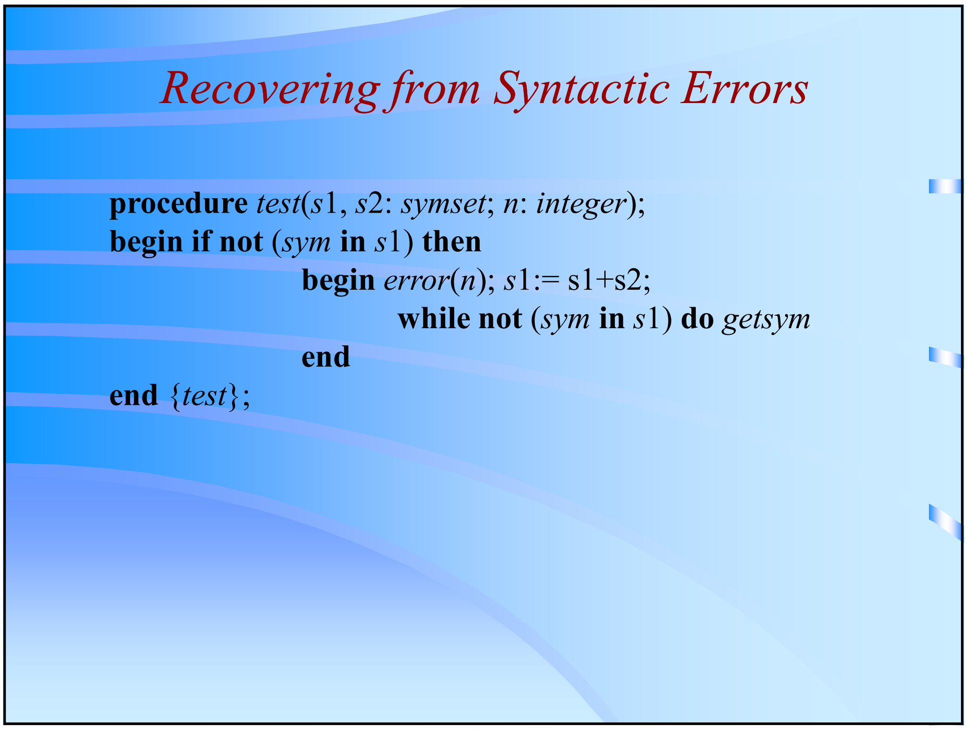
```
begin {block}
    if sym = constsym then
    begin getsym; constdeclaration;
        while sym = comma do
            begin getsym; constdeclaration
            end;
        if sym = semicolon then getsym else error(5)
    end;
    if sym = varsym then
    begin getsym; vardeclaration ;
        while sym = comma do
            begin getsym; vardeclaration
            end;
        if sym = semicolon then getsym else error(5)
    end;
    while sym = procsym do
    begin getsym;
```



```
        if sym = ident then
             begin enter(procedure); getsym
    end
        else error(4);
        if sym = semicolon then getsym else error(5);
        block(tx);
        if sym = semicolon then getsym else error(5);
    end;
    statement
end {block};
begin {main program}
  for ch := ' A '  to  ' ; '  do  ssym[ch] := nul;
     word[ 1] := 'BEGIN           '; word[ 2] :=  'CALL           ';
     word [ 3] := 'CONST          '; word[ 4] :=  'DO             ';
     word [ 5] := 'END            '; word[ 6] :=  'IF             ';
     word [ 7] := 'ODD            '; word[ 8] :=  'PROCEDURE ';
     word [ 9] := 'THEN           ‘; word] :=  'VAR            ';
     word [11] := 'WHILE          ‘;
```

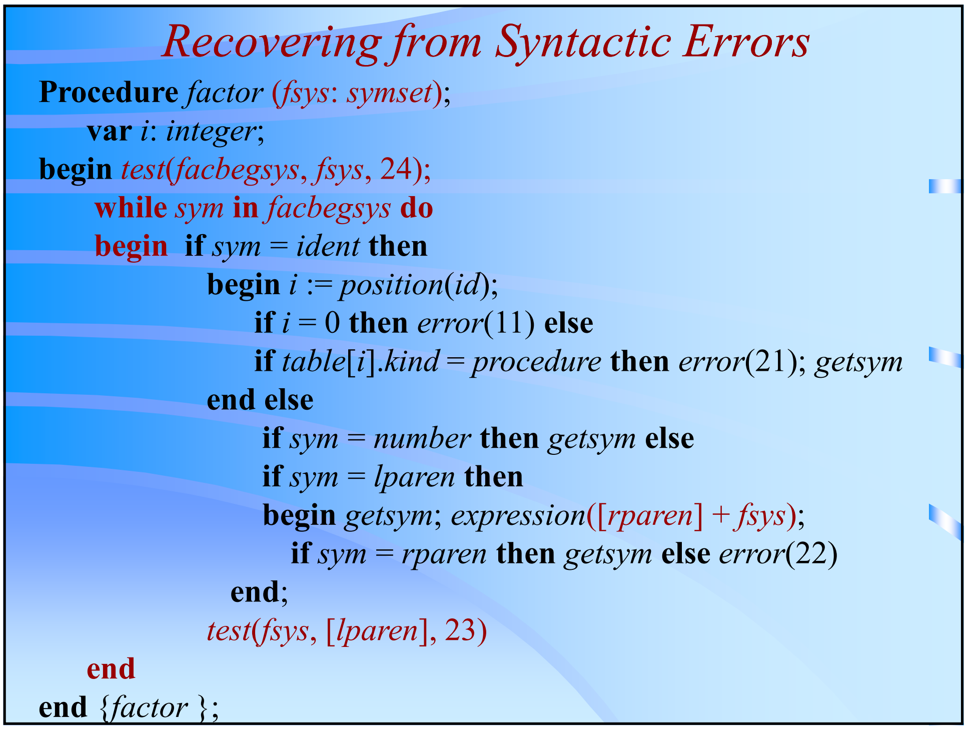


```
        wsym[l] := beginsym;          wsym[2] :=callsym;
        wsym[3] := constsym;          wsym[4] := dosym;
        wsym[5] := endsym;            wsym[6] := ifsym;
        wsym[7] := oddsym;            wsym[8] := procsym;
        wsym[9] := thensym;           wsym[10] := varsym;
        wsym[11] := whilesym;
        ssym['+'] := plus;            ssym['–'] := minus;
        ssym['*'] := times;           ssym('/'] := slash;
        ssym['('] := lparen;          ssym[')'] := rparen;
        ssym['='] := eql;             ssym[','] := comma;
        ssym['.'] := period;          ssym['≠'] := neq;
        ssym['<'] := lss;             ssym[' >'] := gtr,
        ssym['≤'J :=leq;              ssym['≥'] := geq;
        ssym[';'] := semicolon;
    cc := 0;  ll := 0;  ch :=' ';  kk := al;  getsym;
    block(0) ;   if sym ≠ period then error(9);
 99: writeln
 end.
```



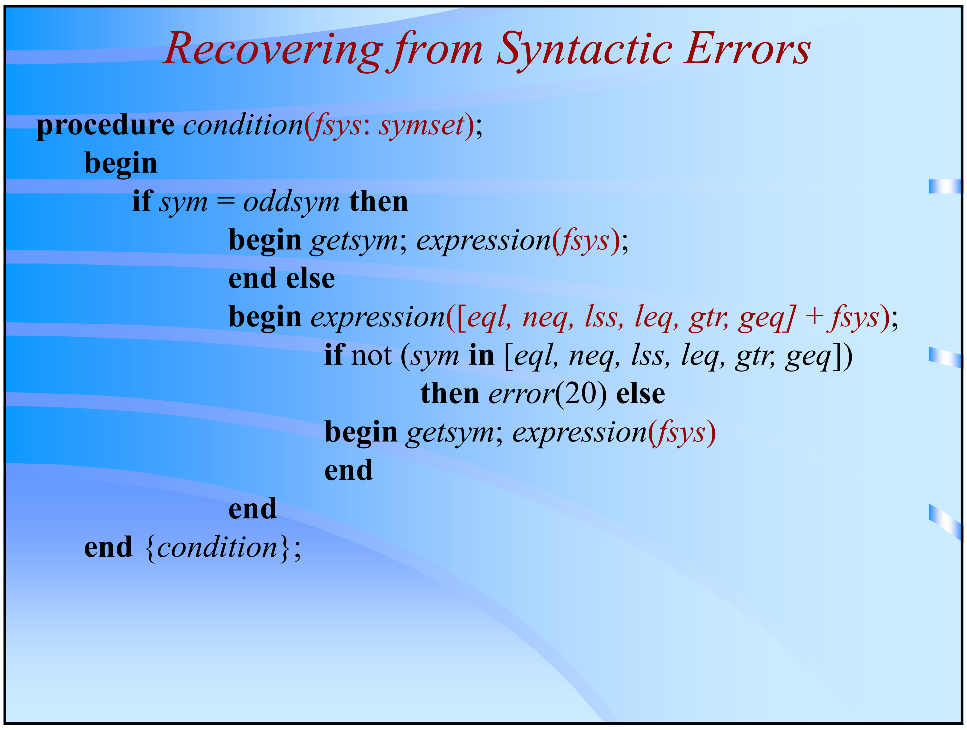
Recovering from Syntactic Errors

```
procedure test(s1, s2: symset; n: integer);
begin if not (sym in s1) then
            begin error(n); s1:= s1+s2;
                  while not (sym in s1) do getsym
            end
end {test};
```



Recovering from Syntactic Errors

```
Procedure factor (fsys: symset);
    var i: integer;
begin test(facbegsys, fsys, 24);
     while sym in facbegsys do
     begin  if sym = ident then
                begin i := position(id);
                    if i = 0 then error(11) else
                    if table[i].kind = procedure then error(21); getsym
                end else
                    if sym = number then getsym else
                    if sym = lparen then
                    begin getsym; expression([rparen] + fsys);
                       if sym = rparen then getsym else error(22)
                  end;
                test(fsys, [lparen], 23)
    end
end {factor };
```



Recovering from Syntactic Errors

```
procedure condition(fsys: symset);
    begin
        if sym = oddsym then
                  begin getsym; expression(fsys);
                  end else
                  begin expression([eql, neq, lss, leq, gtr, geq] + fsys);
                          if not (sym in [eql, neq, lss, leq, gtr, geq])
                                  then error(20) else
                          begin getsym; expression(fsys)
                          end
                  end
    end {condition};
```