Program 5.4. Analizator języka PL/0

```
program PL0;
{kompilator języka PL/0, tylko analiza składniowa}
label 99
const lsz = 11;     {liczba słów zastrzeżonych}
    tmax = 100;   {długość tablicy identyfikatorów}
    cmax = 14;    {maksymalna liczba cyfr w liczbach}
    al = 10     ;   {długość identyfikatorów}
type symbol =
    (żaden, ident, liczba, plus, minus, razy, dziel, oddsym, równe, różne, mniejsze, mnrówne, wieksze, wrówne,
    lnaw, pnaw, przec, średnik, kropka, przypis, begisym, endsym, ifsym, thensym, whilesym, dosym, callsym,
    constsym, varsym, procsym);
    alfa = packed array [1. .al.] of char;
    obiekt = (stała, zmienna, procedura);
var ch: char; {ostatni wczytany znak}
    sym: symbol:  { ostatni wczytany symbol}
    id: alfa:    { ostatni wczytany identyfikator}
    num: integer: { ostatnia wczytana liczba}
    lz: integer;     {licznik znaków}
    dl: integer;     {długość linii}
    kk: integer
    linia: array [1. .81] of char;
    a: alfa;
    słowo: array [1. .lsz] of alfa;
    zsym: array [1. .lsz] of symbol;
    ssym: array [char] of symbol;
    tablica:  array [0. .tmax] of
            record nazwa: alfa;
              rodzaj: obiekt
            end;

procedure błąd(n:integer);
begin writeln(' ': lz, '↑', n: 2); goto 99
end {błąd};

procedure pobsym;
    var i, j, k: integer;
    procedure pobznak;
    begin if lz = dl then
        begin if eof(input) then
                begin write('PROGRAM NIEDOKOŃCZONY'); goto 99
                end;
            dl := 0; lz := 0; write(' ');
            while ¬ eoln(input) do
                begin dl := dl+1; read(ch); write(ch); linia[dl] := ch
                end;
            writeln; dl := dl+1; read(linia[dl])
        end;
        lz := lz+1; ch := linia{lz}
    end {pobznak};

begin {pobsym}
    while ch = ' ' do pobznak;
    if ch in ['A'. .'Z'] then
    begin {identyfikator lub słowo zastrzeżone} k := 0;
        repeat if k < al then
            begin k := k+1; a[k] := ch
```

```
            end;
            pobznak
        until ¬ (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 := lsz;
        repeat k := (i+j) div 2;
            if id ≤ słowo[k] then j := k-1;
            if id ≥ słowo[k] then i := k+1
        until i > j ;
        if i-1 > j then sym := zsym[k] else sym := ident
    end else
    if ch in ['0'. .'9'] then
    begin {liczba} k := 0; num := 0; sym := liczba;
        repeat num := l0*num+(ord(ch)–ord('0'));
            k:=k+1; pobznak
        until ┐ (ch in ['0'. .'9']);
        if k > cmax then błąd(30)
    end else
    if ch = ':' then
    begin pobznak;
        if ch ='=' then
        begin sym := przypis; pobznak
        end else sym := żaden;
    end else
    begin sym := ssym[ch]; pobznak
    end
end {pobsym};

procedure blok(tx:integer);
    procedure wprowadź(k: obiekt);
    begin {wprowadź obiekt do tablicy}
        tx := tx+1;
        with tablica[tx] do
    begin nazwa := id; rodzaj := k;
    end
end {wprowadź};
function pozycja(id: alfa): integer;
    var i: integer;
begin {znajdź identyfikator id w tablicy}
    tablica[0].nazwa := id; i := tx;
    while tablica[i].nazwa ≠ id do i := i – 1;
    pozycja := i
end {pozycja};

procedure deklstałej;
begin if sym = ident then
    begin pobsym;
        if sym = równe then
        begin pobsym ;
            if sym = liczba then
                begin wprowadź(stała); pobsym
                end
            else błąd(2)
        end else błąd(3)
    end else błąd(4)
end {deklstałej};
```

```
procedure deklzmiennej;
begin if sym = idem then
    begin wprowadź(zmienna); pobsym
    end else błąd(4)
end {deklzmiennej};

procedure instrukcja;
    var i: integer;
    procedure wyrażenie;
        procedure składnik;
            procedure czynnik;
                var i: integer;
            begin
                if sym = ident then
                begin i := pozycja(id);
                    if i = 0 then błąd(11) else
                    if tablica[i].rodzaj = procedura then błąd(21);
                    pobsym
                end else
                if sym = liczba then
                begin pobsym
                end else
                if sym = lnaw then
                begin pobsym; wyrażenie ;
                    if sym = pnaw then pobsym else błąd(22)
                end
                else błąd(23)
            end {czynnik};
        begin {składnik} czynnik;
            while sym in [razy, dziel] do
                begin pobsym; czynnik
                end
        end {składnik};
    begin {wyrażenie}
        if sym in [plus, minus] then
            begin pobsym; składnik
            end else składnik;
        while sym in [plus, minus] do
            begin pobsym; składnik
            end
    end {wyrażenie};

    procedure warunek;
    begin
        if sym = oddsym then
        begin pobsym; wyrażenie
        end else
        begin wyrażenie;
            if ¬ (sym in [równe, różne, mniejsze, mnrówne, większe, wrówne])
                then błąd(20) else
            begin pobsym ; wyrażenie
            end
        end
    end {warunek};
begin {instrukcja}
    if sym = ident then
    begin i:=pozycja(id);
```

```
        if i = 0 then błąd(11) else
        if tablica[i].rodzaj ≠ zmienna then błąd(12);
        pobsym; if sym = przypis then pobsym else błąd(13);
        wyrażenie
    end else
    if sym = callsym then
    begin pobsym ;
        if sym ≠ ident then błąd(14) else
            begin i:=pozycja(id);
                if i := 0 then błąd(11) else
                if tablica[i].rodzaj ≠ procedura then błąd(15);
                pobsym
            end
        end else
        if sym = ifsym then
        begin pobsym; warunek;
            if sym = thensym then pobsym else błąd(16) ;
            instrukcja;
        end else
        if sym = beginsym then
        begin pobsym ; instrukcja;
            while sym = średnik do
                begin pobsym; instrukcja
                end;
            if sym = endsym then pobsym else błąd (17)
        end else
        if sym = whilesym then
        begin pobsym ; warunek;
            if sym = dosym then pobsym else błąd(18);
            instrukcja
        end
    end {instrukcja};

begin {blok}
    if sym = constsym then
    begin pobsym; deklstałej;
        while sym=przec do
            begin pobsym; deklstałej
            end;
        if sym = średnik then pobsym else błąd(5)
    end;
    if sym = varsym then
    begin pobsym; deklzmiennej;
        while sym = przec do
            begin pobsym; deklzmiennęj
            end;
        if sym = średnik then pobsym else błąd(5)
    end;
    while sym = procsym do
    begin pobsym;
        if sym = ident then
            begin wprowadź(procedura); pobsym
        end
        else błąd(4);
        if sym = średnik then pobsym else błąd(5);
        blok(tx);
        if sym = średnik then pobsym else błąd(5);
    end;
```

```
    instrukcja
end {blok};

begin {program główny}
    for ch := ' A '  to  ' ; '  do  ssym[ch] := żaden;
    słowo[ 1] := 'BESIN      '; słowo[ 2] :=  'CALL          ';
    słowo[ 3] := 'CONST     '; słowo[ 4] :=  'DO            ';
    słowo[ 5] := 'END        '; słowo[ 6] :=  'IF            ';
    słowo[ 7] := 'ODD        '; słowo[ 8] :=  'PROCEDURE     ';
    słowo[ 9] := 'THEN       '; słowo[10] :=  'VAR           ';
    słowo[11] := 'WHILE     ';
zsym[l] := beginsym; zsym[2] :=callsym;
zsym[3] := constsym;zsym[4] := dosym;
zsym[5] := endsym;       zsym[6] := ifsym;
zsym[7] := oddsym;       zsym[8] := procsym;
zsym[9] := thensym;      zsym[10] := varsym;
zsym[11] := whilesym;
ssym['+'] := plus;          ssym['−'] := minus;
ssym['*'] := razy;          ssym('/'] := dziel;
ssym['('J := lnaw;          ssym[')'] := pnaw;
ssym['='] := równe;         ssym[','] := przec;
ssym['.'] := kropka;        ssym['≠'] := różne;
ssym['<'] := mniejsze;      ssym[' >'] := większe,
ssym['≤'J := mnrówne;       ssym['≥'] := wrówne;
ssym[';'] := średnik;
page(output);
lz := 0;  dl := 0;  ch :='  ';     kk := al;  pobsym;
    blok(0) ;
    if sym ≠ kropka then błąd(9);
99: writeln
end.
```