

Sémantické zpracování (při Synt.Anal. prováděné rekurzivním sestupem)

Atributy = vlastnosti gramatických symbolů nesoucí sémantickou informaci (hodnota, adresa, typ, apod.)

Sémantické zpracování zahrnuje vyhodnocení atributů symbolů v derivačním stromu

Způsoby vyhodnocení (**pro praxi je zajímavé jednorůchodové vyhodnocování**) :

1. procházením stromem od listů ke kořenu se vypočtou tzv. syntetizované atributy
2. „ „ „ od rodiče k potomkovi, od staršího bratra k mladšímu se vypočtou tzv. dědičné atributy

Rekurzivním sestupem řízený překladač provádí výpočet atributů v čase zpracování pravých stran pravidel formou:

- Zpracování neterminálu \equiv vyvolání procedury \equiv expanze neterminálu při derivaci,
- srovnávání terminálů (ze vstupu a z pravé strany pravidla)

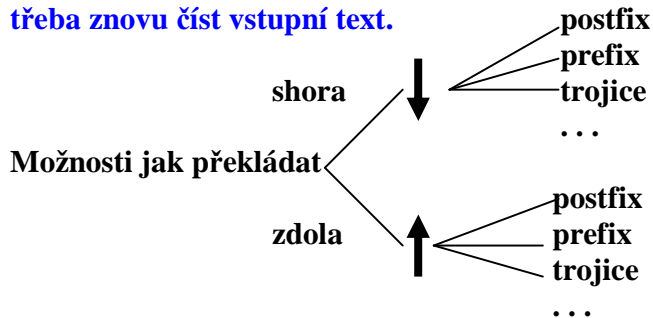
Nutno doplnit procedury LA i SA takto:

-LA bude předávat s přečteným vstupním symbolem i jeho atributy

-procedury SA pro neterminály doplnit o:

- ❖ parametry vstupní, odpovídající dědičným atributům,
- ❖ parametry výstupní odpovídající syntetizovaným atributům,
- ❖ zavést lokální proměnné pro uložení atributů pravostranných symbolů,
- ❖ před vyvoláním procedury korespondující neterminálu z pravé strany vypočítat hodnoty jeho dědičných atributů,
- ❖ na konec procedury popisující pravou stranu pravidla zařadit příkazy vyhodnocující syntetizované atributy levostranného symbolu.

Snažíme se vymyslet to tak, aby se vyhodnocení provedlo jedním průchodem a nebylo třeba znovu číst vstupní text.



Překlad příkazů

Překlad přiřazovacího příkazu do čtveřice shora dolů (rekurzivním sestupem)

Gramatika: $S \rightarrow a := E ;$
 $E \rightarrow T \{ + T \}$
 $T \rightarrow F \{ * F \}$
 $F \rightarrow (E) | a$

Atributy symbolů E, T, F jsou:
PA = počáteční adresa pro mezivýsledky
KA = koncová adresa (již nezabraná)
AH = adresa hodnoty

Přiřazení atributů symbolům:	Atributy	Symboly které ho mají
Dědičný	PA	S, E, T, F
Syntetizovaný	AH	a, E, T, F
Syntetizovaný	KA	S, E, T, F

Budeme generovat čtveřice tvaru:
 (s významem (operátor, operand, operand, výsledek))

(+, adr1, adr2, adr3)
 (*, adr1, adr2, adr3)
 (=, adr1, adr2, 0)

Vstupem jsou symboly zjištěné LA spolu s jejich atributy

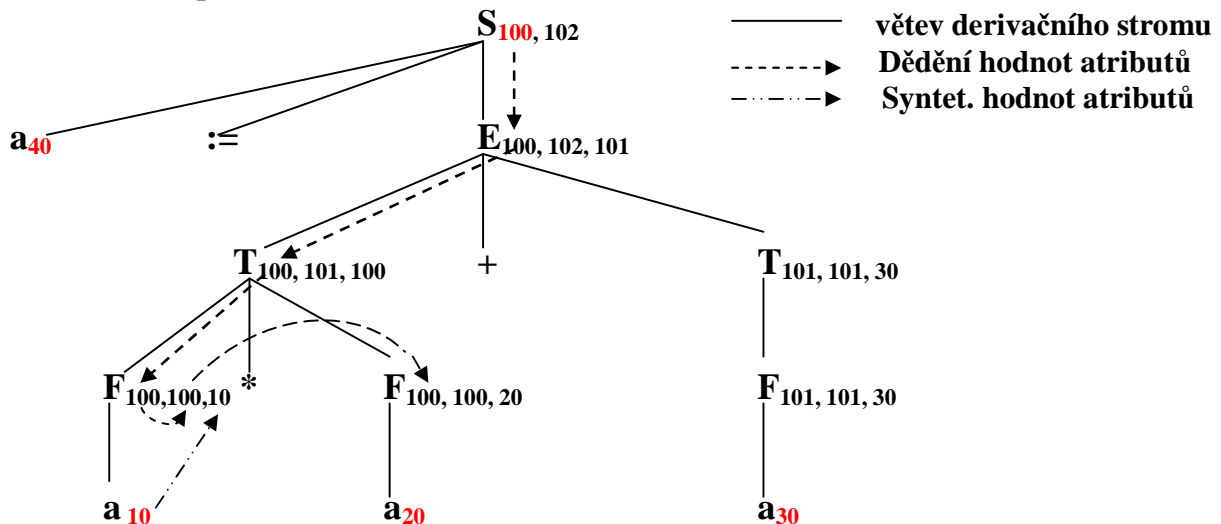
Výstupem je posloupnost instrukcí vnitřního jazyka (atributy jsou jejich součástí)

Mezivýsledky budeme v příkladu ukládat od adresy 100

Např. $a_{40} := a_{10} * a_{20} + a_{30}$ přeloží se na *, 10, 20, 100
 +, 100, 30, 101
 =, 101, 40, -

Indexy znamenají adresu

Atributy uzlů stromu jsou v pořadí PA, KA, AH, lze je vyhodnotit procházením stromem.



Symbolický zápis programu překladače přiřazovacího příkazu do čtveřic

program PREKLAD bude používat tyto proměnné a podprogramy:

```
typ SYMBOL = záznam TYP:(IDENT, PRIRAZ, PLUS, KRAT,  
                    STREDNIK, LEVA, PRAVA);  
                    ADRESA: INTEGER;  
                    konec záznamu;  
procedura CTI(S) /*cte lexikální symbol do promenne S typu SYMBOL*/  
procedura CHYBA ...  
procedura VYSTUP(... /*tiskne instrukce tvaru řetězec, adresa, adresa, adresa*/  
  
promenna N: SYMBOL;  
  
procedura S {  
  promenna L: SYMBOL; KA, AH: integer;  
  CTI(L);  
  if L.TYP <> IDENT then CHYBA;  
  CTI(N);  
  if N.TYP <> PRIRAZ then CHYBA;  
  CTI(N);  
  E(100, KA, AH); /*před vyvoláním procedury se určí hodnota dědičných atributů */  
  if N.TYP <> STREDNIK then CHYBA;  
  VYSTUP('=', AH, L.ADRESA, 0);  
}  
      vstupní param.      výstupní parametry  
procedura E(hodnota PA: integer; adresa KA, AH: integer) {  
  promenna AH2: integer;  
  T(PA, KA, AH);  
  while N.TYP = PLUS DO  
  { CTI(N);  
    T(KA, KA, AH2);  
    VYSTUP('+', AH, AH2, KA);  
    AH ← KA;  
    KA ← KA + 1;  
  }  
}
```


S	E	T	F
<p>čte a(10), :=, a(20) tj. L <- a(10) N <- a(20) volá E(100,KA,AH)</p>	<p>volá T(100,KA,AH)</p> <p>návratem AH <- 20 KA <- 100 čtením N <- a(30) volá T(100,KA,AH2)</p> <p>návratem AH2 <- 30 KA <- 100 výstup (+,20,30,100) AH <- 100 KA <- 101</p>	<p>volá F(100,KA,AH)</p> <p>návratem AH <- 20 KA <- 100</p> <p>volá F(100,KA,AH)</p> <p>návratem AH <- 30 KA <- 100</p>	<p>AH <- 20 čtením N <- PLUS KA <- 100</p> <p>AH <- 30 čtením N <- STREDNIK KA <- 100</p>
<p>návratem AH <- 100 KA <- 101 výstup (=, 100, 10, 0)</p>			

Rozšíření jazyka na seznam příkazů a příkaz if. < > označují neterminální symbol

G[<SP>]:

<SP> --> S ; <SP>

<SP> --> e

S --> a := E

S --> if <BE> then S

↑ zde budeme generovat instrukci
pro přeskok S při <BE> false, ve tvaru
JIF, AH, místo_za_S. Nevíme zatím ale kam, tak si číslo instrukce
zapamatujeme a dodatečně pak kompletujeme

↑ Vygeneruje čtveřici, která uloží
do AH hodnotu booleovského výrazu

pracuje také s následujícími proměnnými a podprogramy:

promenna N: SYMBOL;

C, KA: integer; /* C bude citac instrukci, tj. radek výstupního kodu */

procedura DOPLN(hodnota KAM, CO: integer) ... /*kompletuje čtveřici*/ {

/*... není rozepsana */ }

procedura S(hodnota PA: integer; adresa KA: integer) {

promenna L: SYMBOL;

AH, KA1, PC: integer; /* PC slouzi pro zapamatovani citace instrukci C */

if N.TYP = IFS then

{ CTI(N);

BE(PA, KA, AH);

VYSTUP('JIF', AH, 0, 0);

PC ← C; /* C je čítač instrukcí, inkrementovaný procedurou VYSTUP */

if N.TYP <> THENS then CHYBA; CTI(N);

S(KA, KA1); //paměť obsazuje od KA, KA1 je již neobsazené místo

DOPLN(PC-1, C);

KA ← KA1;

}

else

{ if N.TYP <> IDENT then CHYBA;

L ← N;

CTI(N); if N.TYP <> PRIRAZ then CHYBA; /*prirazovací prikaz */

CTI(N); E(PA, KA, AH);

VYSTUP('=', AH, L.ADRESA, 0);

}

}

procedura SP(hodnota PA: integer; adresa KA: integer) {

promenna KA1: integer;

if N <> PRAZDNY_SYMBOL then /*prázdným symbolem bude eof */

{ S(PA, KA);

if N <> STREDNIK then CHYBA;

CTI(N); SP(KA, KA1); KA ← KA1;

}

}

procedura BE(hodnota PA: integer; adresa KA, AH: integer) ...

/*je podobná E, pracuje s or, and, not, <, >, =, ... */

/* další procedury pro E, T, F, VYSTUP */

procedura MAIN {

C ← 1; (* C je dále inkrementováno procedurou VYSTUP *)

CTI(N);

SP(100, KA); /* pomocne promenne pro mezivysledky zacinaji na adr 100 */

}

Např.

C := A * B ; if A < B then C := A + B ;
30 10 20 10 20 30 10 20

se přeloží na:

(1) '*', 10, 20, 100

(2) '=', 100, 30, 0

(3) '<', 10, 20, 101

(4) 'JIF', 101, 7, 0

(5) '+', 10, 20, 102

(6) '=', 102, 30, 0

(7)

procedura DOPLN(7, 4) doplni adresu 7

Programový zápis používá následující proměnné a podprogramy

```
promenna N: SYMBOL; /*dodá lexikální analyzátor*/
                C; integer; /*čítač umístění další generované instrukce*/
...
procedura S {
    promenna PC: integer; /*pomocný čítač pro zapamatování C*/
    if N.TYP = IDENT then
        { VYSTUP('TA', N.ADRESA);
          CTI(N);
          if N.TYP <> PRIRAZ then CHYBA;
          CTI(N);
          E;
          VYSTUP('ST', 0); /*druhý parametr je nevýznamový*/
        }
    else
        { if N.TYP <> IFS then CHYBA;
          else
            { CTI(N);
              E;
              PC ← C;
              if N.TYP <> THENS then CHYBA;
              VYSTUP('IFJ', 0);
              CTI(N);
              S;
              DOPLN(PC, C);
            }
          }
    }
}
```

```
procedura SP {  
    if N.TYP <> PRAZDNY_SYMBOL then  
        {  
            S;  
            if N.TYP <> STREDNIK then CHYBA;  
            CTI(N);  
            SP;  
        }  
}
```

```
procedura E {  
    T;  
    while N.TYP = PLUS do  
        {  
            CTI(N);  
            T;  
            VYSTUP('PLUS', 0) /*0 znamená, že u PLUS je parametr nevýznamový*/  
        }  
}
```

```
procedura T {  
    F;  
    while N.TYP = KRAT do  
        {  
            CTI(N);  
            F;  
            VYSTUP('KRAT', 0);  
        }  
}
```

```

procedura F {
  if N.TYP = IDENT then
    { VYSTUP(TA, N.ADR);
      VYSTUP(DR, 0);
      CTI(N);
    }
  else if N.TYP <> LEVA then CHYBA
    else { CTI(N);
      E;
      if N.TYP <> PRAVA then CHYBA
        else CTI(N);
      }
  }

```

```

procedura MAIN
{
  C ←-1;
  CTI(N);
  SP;
}

```

Např. $a_{10} := a_{20} + a_{30}$ se přeloží na

TA 10
TA 20
DR
TA 30
DR
PLUS
ST

Sémantické zpracování v PL0

Instrukce postfixového zápisu

lit 0,A :ulož konstantu A do zásobníku

opr 0,A :proved instrukci A

1	unarni minus
2	+
3	-
4	*
5	div
6	mod
7	odd
8	=
9	<>
10	<
11	>=
12	>
13	<=

lod L,A :ulož hodnotu proměnné z adr. L,A na vrchol zásobníku

sto L,A :zapiš do proměnné s adr. L,A hodnotu z vrcholu zásob.

cal L,A :volej proceduru s adresou kódu A z úrovně L

ret 0,0 :return, v některých verzích opr 0,0

ing 0,A :zvyš obsah Top-registru zásobníku o hodnotu A
v některých verzích int 0,A

jmp 0,A :proved' skok na adresu kódu A

jpc 0,A :proved' podmíněný skok na adresu kódu A

Následující kód je lepší si prohlédnout z obrazovky, než-li jej tisknout. Části týkající se překládání jsou vybarvené.

```
/*postfixove instrukce jsou ve tvaru*/
```

```
typedef struct {  
    FCT f; /*kod instrukce = postfixový operátor=funkce instrukce*/  
    int l; /*uroven vnoreni, je parametrem instrukce*/  
    int a; /*cast adresy nebo kod operace, je parametrem instrukce*/  
} INSTRUCTION;
```

```
/* generovani instrukci do pole code = řetězec postfixových instrukci
```

```
    x :kod instrukce tj. postfixovy operátor  
    y :hladina  
    z :adresni cast
```

```
*/
```

```
void gen(FCT x, int y, int z) {
```

```
    if (cx>CXMAX) { /* cx je globální proměnná, citac instrukci*/  
        printf("program too long");  
        exit(1);  
    }
```

```
    code[cx].f = x; /*druh=funkce instrukce*/
```

```
    code[cx].l = y; /*hladina = level*/
```

```
    code[cx++].a = z; /*adresa*/
```

```
} // gen()
```

```
void factor(SYMSET fsys,int tx,int lev) { /*násl. symb., konec tab. symb., hladina*/
```

```
int i;
```

```
SYMSET pom;
```

```
test(facbegsys,fsys,24); /*poč. symboly, násl. symboly, číslo chyby*/
```

```
while (facbegsys[sym]) {
```

```
    if (sym == ident) {
```

```
        i = position(id,tx); /*najdi ho v tabulce symbolů */
```

```
        if (i == 0) error(1);
```

```
        else
```

```
            switch (TABLE[i].kind) {
```

```
                case constant: gen(lit,0,TABLE[i].CO.val);
```

```
                    break;
```

```
                    /*identifikátor jako faktor, může být konstanta nebo proměnná*/
```

```
                case variable: gen(lod,lev-TABLE[i].CO.vp.level,TABLE[i].CO.vp.adr);
```

```
                    break; /*lev je úroveň místa použití, dále je úroveň deklarace a ofset */
```

```
                    /*ale nemůže být jménem podprogramu*/
```

```
                case procedure: error(21);
```

```
                    break;
```

```
            }
```

```
            getsym();
```

```
        } else
```

```
            if (sym == number) { /*faktor tske může být pojmenovaným číslem*/
```

```
                if (num > AMAX) {
```

```
                    error(31);
```

```
                    num = 0;
```

```
                }
```

```
                gen(lit,0,num);
```

```

    getsym();
} else
    if (sym == lparen) { /*faktor může být také závorkovaný výraz*/
        getsym();
        nuluj(pom);
        sjednot(pom,fsys);
        pom[rparen] = 1; /*do follow přidáme) */
        expression(pom,tx,lev);
        if (sym == rparen) getsym();
        else error(22);
    }
    nuluj(pom);
    pom[lparen] = 1;
    test(fsys,pom,23); /*následující symb., stop symb., číslo chyby*/
}
} // factor()

```

/ následuje generování kódu pro term, parametry znamenají*

fsys :mnozina follow symbolu
tx :ukazatel na konec tabulky symbolu
lev :hladina místa termu

**/*

```

void term(SYMSET fsys,int tx,int lev) {
    SYMBOL mulop;
    SYMSET pom;

```

```

    nuluj(pom);
    sjednot(pom,fsys);
    pom[times] = pom[slash] = pom[modulo] = 1;
    factor(pom,tx,lev);

```

```

while ((sym == times) || (sym == slash) || (sym == modulo)) {
    mulop = sym;
    getsym();
    sjednot(pom,fsys);
    pom[times] = pom[slash] = pom[modulo] = 1;
    factor(pom,tx,lev);

```

```

    switch(mulop) {

```

```

        case times: gen(opr,0,mul); /*viz Operation na 1.str programu: mul je 4, di 5, mod 6 */
            break;

```

```

        case slash: gen(opr,0,di);
            break;

```

```

        case modulo: gen(opr,0,mod);
            break;
    }
}
} // term()

```

```

/* generovani kodu pro aritm. vyraz
   fsys      :mnozina follow symbolu vyrazu
   tx       :ukazatel na konec tabulky symbolu
   lev      :hladina mista vyrazu
*/
void expression(SYMSET fsys,int tx,int lev) {
SYMBOL addop;
SYMSET pom;

  if ((sym == plus) || (sym == minus)) {
    addop = sym;
    getsym();
    nuluj(pom);
    sjednot(pom,fsys);
    pom[plus] = pom[minus] = 1;
    term(pom,tx,lev);
    if (addop == minus) gen(opr,0,neg); /* neg ma v Operation pořadí 1 */
  }
  else {
    nuluj(pom);
    sjednot(pom,fsys);
    pom[plus] = pom[minus] = 1;
    term(pom,tx,lev);
  }

  while ((sym == plus) || (sym == minus)) {
    addop = sym;
    getsym();
    nuluj(pom);
    sjednot(pom,fsys);
    pom[plus] = pom[minus] = 1;
    term(pom,tx,lev);

    if (addop == plus) gen(opr,0,add); /* pořadí add 2, sub 3 */
    else gen(opr,0,sub);
  }
} // expression()

```

```

/* generovani kodu pro logicky vyraz
   fsys      :follow symboly vyrazu
   tx       :ukazatel na konec tabulky symbolu
   lev      :hladina mista vyrazu
*/
void condition(SYMSET fsys,int tx,int lev) {
SYMBOL relop;
SYMSET pom;

  if (sym == oddsym) {
    getsym();
    expression(fsys,tx,lev);
  }
}

```

```

    gen(opr,0,odd);
}
else {
    nuluj(pom);
    sjednot(pom,fsys);
    pom[eql] = pom[neq] = pom[lss] = pom[gtr] = pom[leq] = pom[geq] = 1;
    expression(pom,tx,lev);
    if ((sym != eql) && (sym != neq) && (sym != lss) && (sym != gtr) && (sym != leq) && (sym !=
geq))
        error(22);
    else {
        relop = sym;
        getsym();
        expression(fsys,tx,lev);

        switch (relop) {
            case eql: gen(opr,0,eq);    // 8
                break;

            case neq: gen(opr,0,ne);   // 9
                break;

            case lss: gen(opr,0,lt);    // 10
                break;

            case geq: gen(opr,0,ge);    // 11
                break;

            case gtr: gen(opr,0,gt);    // 12
                break;

            case leq: gen(opr,0,le);    // 13
                break;

        }
    }
}
} // condition()

```

```

/* generovani kodu pro statement
   fsys      :follow symbolu statementu
   tx       :ukazatel na konec tabulky symbolu
   lev      :uroven mista statementu
*/
void statement(SYMSET fsys,int tx,int lev) {
    int i, cx1, cx2;
    SYMSET pom;

    if ((fsys[sym] == 0) && (sym != ident)) {

```

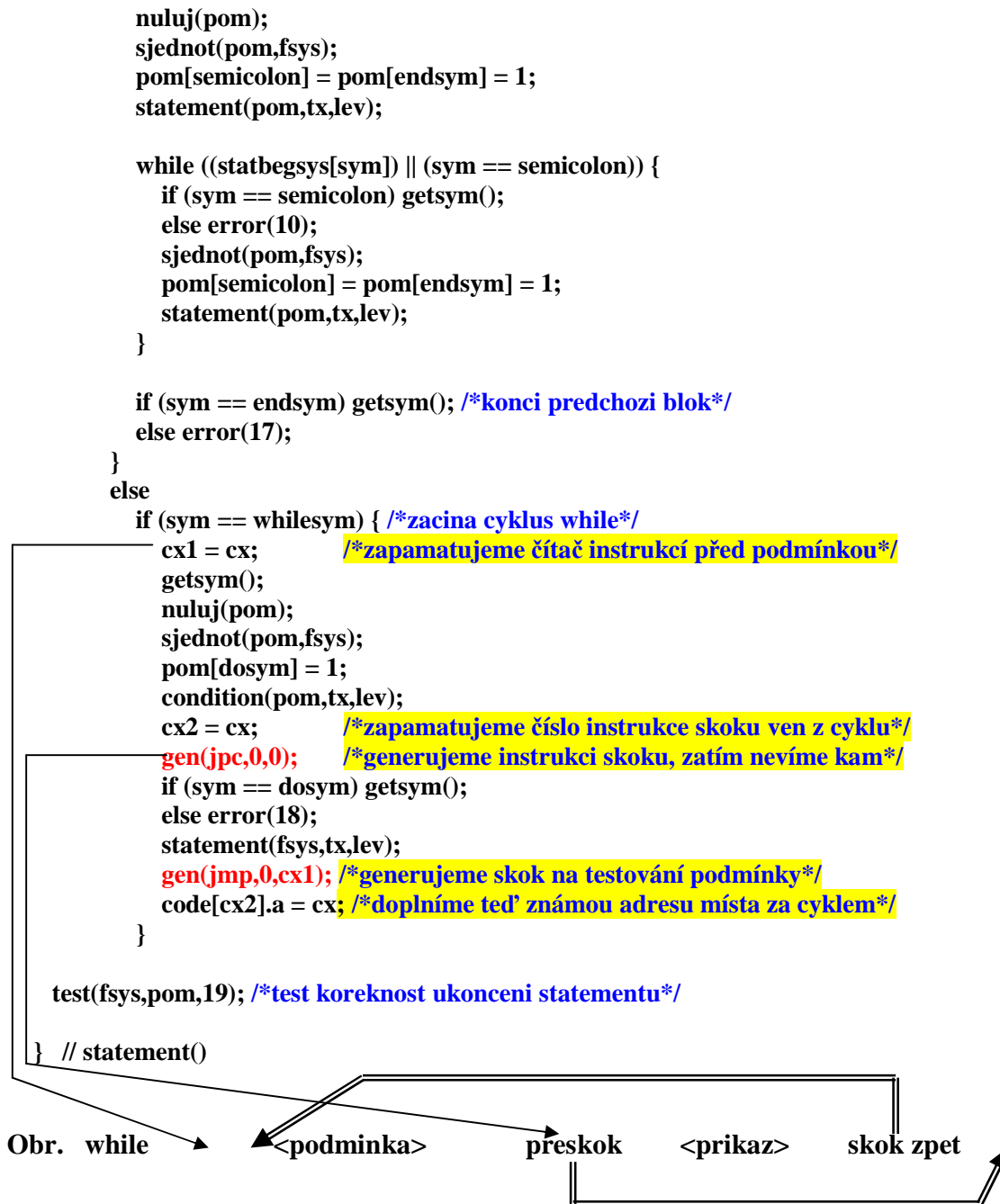


```

error(10);
do
    getsym();
    while (fsys[sym] == 0);
}
if (sym == ident) { /*nalezen prikaz prirazeni*/
    i = position(id,tx);
    if (i == 0) error(11);
    else
        if (TABLE[i].kind!=variable) { /*prirazeni do jineho ident. nez promenna*/
            error(12);
            i = 0;
        }
    getsym();
    if (sym == becomes) getsym();
    else error(13);
    expression(fsys,tx,lev);
    if (i) gen(sto,lev-TABLE[i].CO.vp.level,TABLE[i].CO.vp.adr);
} /*hladina místa použití minus hladina deklarace, offset*/
else
    if (sym == callsym) { /*nalezeno volani podprogramu*/
        getsym();
        if (sym != ident) error(14);
        else {
            if ((i = position(id,tx)) == 0) error(11);
            else {
                if (TABLE[i].kind == procedure) gen(cal,lev-
TABLE[i].CO.vp.level,TABLE[i].CO.vp.adr); /*rozdíl úrovní, adresa začátku kódu*/
                else error(15);
            }
        }
        getsym();
    }
}
else
    if (sym == ifsym) { /*podmineny prikaz*/
        getsym();
        nuluj(pom);
        sjednot(pom,fsys);
        pom[thensym] = pom[dosym] = 1;
        condition(pom,tx,lev);

        if (sym == thensym) getsym();
        else error(16);
        cx1 = cx; /*zapamatování čítače*/
        gen(jpc,0,0); /*generování neúplné instrukce*/
        statement(fsys,tx,lev);
        code[cx1].a = cx; /*doplnění adresy*/
    }
    else
        if (sym == beginsym) { /*zacina nový blok*/
            getsym();

```



```

var i,j;
procedure p;
    begin i := i-1; if i>1 then call p;
    end;
begin
    i:=3;
    call p;
end.

```

generovaný kod:

0	JMP	0	13	skok na začátek hl progr, přeskok deklarací
1	JMP	0	2	
2	INT	0	3	začátek procedury p. Rezervace 3 míst, viz kap.5
3	LOD	1	3	natažení i (1 je rozdíl místa použití a deklarace)
4	LIT	0	1	natažení konstanty 1
5	OPR	0	3	odečtení
6	STO	1	3	uložení i
7	LOD	1	3	natažení i
8	LIT	0	1	natažení 1
9	OPR	0	12	větší
10	JMC	0	12	některé verze používají místo JPC JMC
11	CAL	1	2	1 je rozdíl místa použití a místa deklarace, 2 začátek
12	RET	0	0	návrat, některé verze překladače používají OPR 0 0
13	INT	0	5	rezervace míst pro stat a dyn ukaz, návř adr, i, j
14	LIT	0	3	natažení konstanty 3
15	STO	0	3	uložení trojky do i
16	CAL	0	2	vyvolání procedury p
17	RET	0	0	

tabulka symbolů:

1	name: i	var	lev=	0	adr=	3	size=	0
2	name: j	var	lev=	0	adr=	4	size=	0
3	name: p	proc	lev=	0	adr=	2	size=	3

Složitější příklad s vnořeným podprogramem:

```
var a,aa;procedure p1;  
    var b;  
    procedure p2;  
        begin a := 10; b:=20;  
    end;  
    begin call p2; a:=b*b;  
    end;  
begin call p1;aa:=a  
end.
```

generovaný kod:

```
0 JMP 0 16  
1 JMP 0 9  
2 JMP 0 3  
3 INT 0 3  
4 LIT 0 10  
5 STO 2 3  
6 LIT 0 20  
7 STO 1 3  
8 RET 0 0  
9 INT 0 4  
10 CAL 0 3  
11 LOD 0 3  
12 LOD 0 3  
13 OPR 0 4  
14 STO 1 3  
15 RET 0 0  
16 INT 0 5  
17 CAL 0 9  
18 LOD 0 3  
19 STO 0 4  
20 RET 0 0
```

tabulka symbolů:

1	name: a	var	lev= 0	adr= 3	size= 0
2	name: aa	var	lev= 0	adr= 4	size= 0
3	name: p1	proc	lev= 0	adr= 9	size= 4
4	name: b	var	lev= 1	adr= 3	size= 0
5	name: p2	proc	lev= 1	adr= 3	size= 3