Syntaktická analýza metodou rekurzivního sestupu

Základní vlastnosti

- **Zvládá práci s bezkontextovými gramatikami BKG (nutné pro popis syntaxe programovacích jazyků, nejsou regulární). Obecný tvar pravidel:**

  $A \rightarrow \alpha$ kde $\alpha \in (N \cup T)^*$

- **Deterministická analýza shora dolů (sestup)**
- **Metoda vyjádřená vzájemně se volajícími podprogramy (rekurzivní procedury)**

Pozn.:Obecně je analýza BK jazyka úloha řešitelná metodou s návratem s kubickou složitostí (když se nepodaří generovat/akceptovat daný řetězec, vrátíme se a zkusíme jinou možnost). Pokud ji zvládneme rekurzivním sestupem, pak je složitost lineární.

Princip

- **Každému neterminálnímu symbolu A odpovídá procedura A**
- **Tělo procedury je dáno pravými stranami pravidel pro A**

  A → X11 X12 … X1n | X21 X22 ... X2m | ... | Xp1 Xp2 ... Xpq

  pravé strany musí být rozlišitelné na základě symbolů vstupního řetězce aktuálních v okamžiku uplatnění příslušné pravé strany

- **Je-li rozpoznána pravá strana Xi1 Xi2 ... Xik,**

  pak prováděj pro j = 1 .. k

  1. **Je-li symbol Xij neterminální, vyvolá se v A proceduru Xij**
  2. **Je-li " Xij terminální, ověří A přítomnost Xij ve vstupním řetězci a zajistí přečtení dalšího symbolu ze vstupu**

- **Rozpoznané pravidlo analyzátor oznámí (např. výpisem čísla pravidla)**
- **Chybnou strukturu vstupního řetězce oznámí chybovým hlášením**

Pro rozpoznání správné pravé strany musí platit:

-řetězce derivovatelné z pravých stran začínají různými terminálními symboly

-při prázdné pravé straně se musí navíc lišit i od těch terminálních symbolů, které se mohou vyskytnout v derivacích za neterminálem z levé strany pravidla.

Např. příkaz může začínat *if, while, do, identifikátorem, call,* … tím lze rozlišovat,

ale jak poznat neúplný podm.příkaz od úplného (s *else*)? Prodiskutujte to.

Př. Gramatika přiřazování (použijeme zde metasymboly opakování { + T } a { * F } tzv. iterační zápis gramatiky)

(1,2) S → V = E | if E then S Z

(3,4) Z → else S | e

(5) E → T { + T }

(6) T → F { * F }

(7,8) F → (E) | V

(9) V → a I

(10,11) I → (E) | e

1.Zkuste ji napsat i normálně (bez metasymbolů) a diskutujte možný problém s pravidly

E → E { + E } | E { * E } | (E) | …

2. Zjistěte, jak vypadají množiny *first* terminálních symbolů, kterými začínají řetězce odvoditelné z jednotlivých pravých stran pravidel (dovolí to provést výběr té správné pr.strany na kterou se má expandovat neterminál korespondující dané proceduře)

Symbol	first	follow
S Z E		
T F V		
I		

3.Zjistěte, jak vypadají množiny *follow* terminálních symbolů, které následují ve vstupu po provedení příslušné procedury a zkuste si jak vypadá struktura věty

if a then if a then a = a else a = a

Řešení 1.

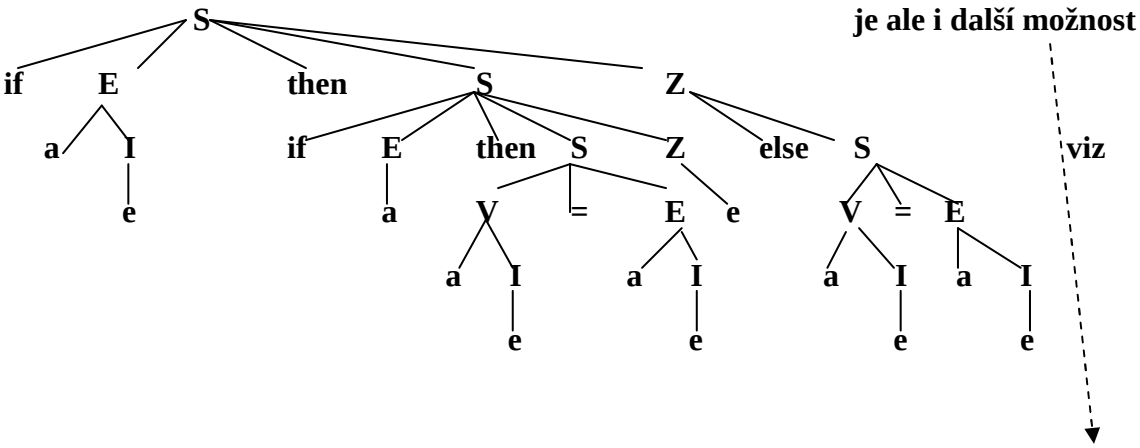
E → T | E + T **T → F | T * F**

a co tohle? **E → T | T + E** **T → F | F * T**

Řešení 2.

Symbol	first	follow
S	**a, if**	**e, else**
Z	**e, else**	**e, else**
E	**a, (**	**), then, e, else**
T	**a, (**	**), then, e, else, +**
F	**a, (**	**), then, e, else, +, ***
V	**a**	**), then, e, else, +, *, =**
I	**e, (**	**), then, e, else, +, *, =**

Řešení 3.



Tohle je vnoření neúplného podm. příkazu do úplného, gramatika umožňuje i opak

Lexikální analýzu bude provádět procedura CTI

Hlášení chyb bude provádět procedura CHYBA

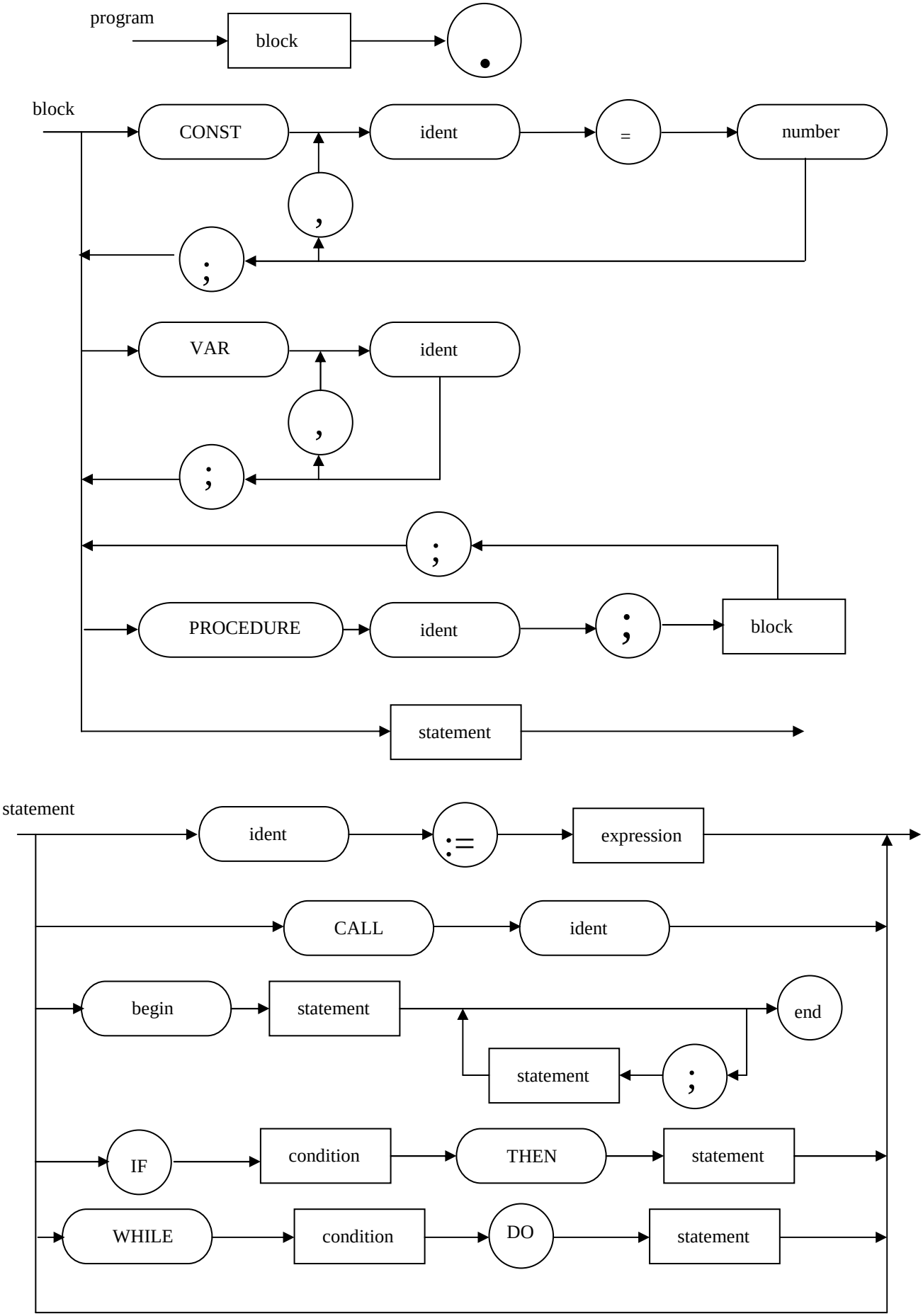
Posloupnost přepisovacích pravidel vypisuje procedura TISK

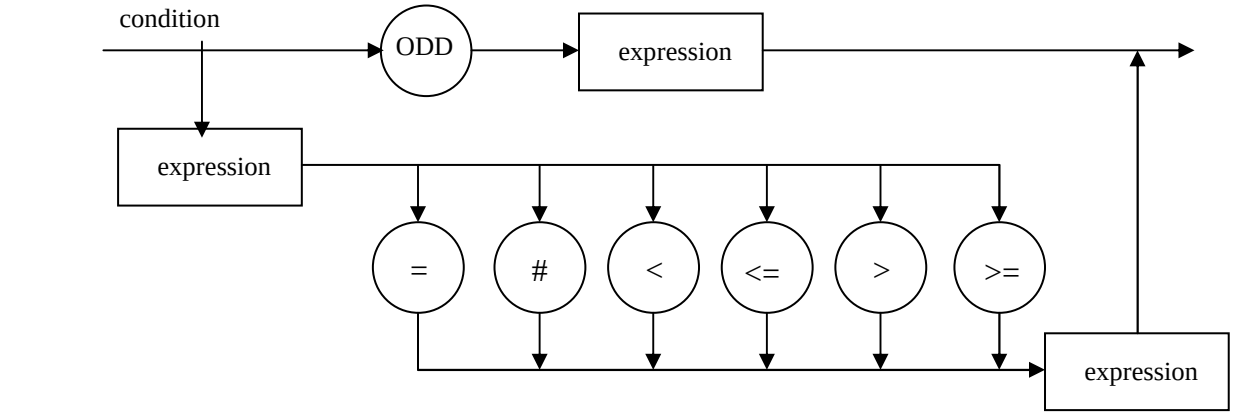
```
program SYNTAKTICKA_ANALYZA
definice vyjmenovaneho typu SYMBOL = (IDENT, PRIRAZ, PLUS, KRAT,  LEVA,
                                                    PRAVA, IFS, THENS, ELSES);
 promenna  N typu SYMBOL;
 procedura CTI(vystupni parametr S typu SYMBOL)   ...
 procedura CHYBA(vstupni parametr CISLO typu integer)  ...
 procedura TISK(vstupni parametr CISLO typu integer)  ...

procedura S
{ if N =  IFS then
              { TISK(2); CTI(N); E;
                  if N ≠ THENS then CHYBA(2);
                  else { CTI(N); S; Z;
                         }
              }
    else
      { TISK(1); V; if N ≠ PRIRAZ then CHYBA(1);
                       else { CTI(N); E;
                              }
      }   /* vstupni řetězec patří do jazyka */
}

 procedura Z
{ if N = ELSES then { TISK(3); CTI(N); S;
                       }
  else TISK(4);         /*bude se chovat tak, jak jsme nakreslili strom, nebo jinak? */
}

 procedura E
{ TISK(5); T;
     while N = PLUS do {  CTI(N); T;
                          }
}
```

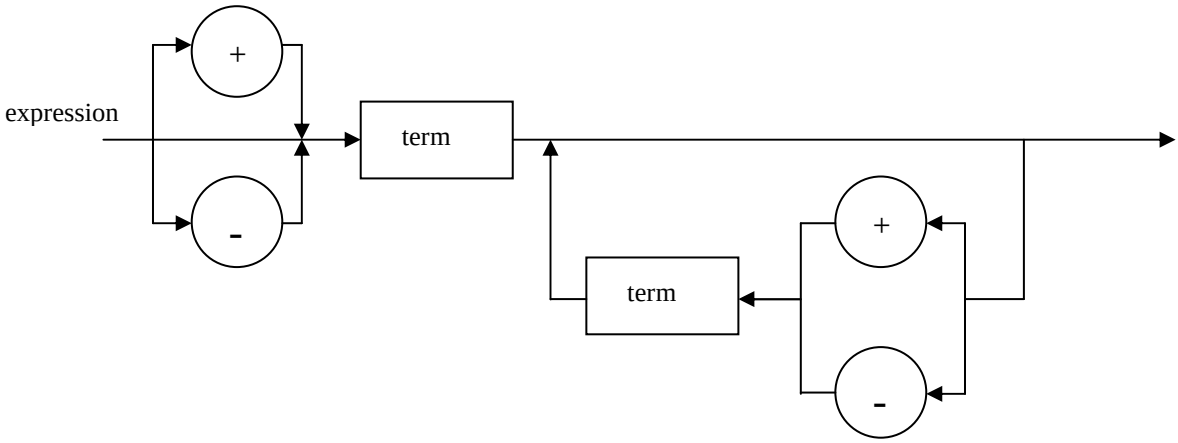
```
procedura T
{ TISK(6); F;
    while N = KRAT do {  CTI(N); F;
                      }
}

procedura F
{ if N = LEVA then { TISK(7); CTI(N); E;
                     if N ≠ PRAVA then CHYBA(7)
                     else CTI(N)
                   }
  else { TISK(8); V; }
}

procedura V
{ if N ≠ IDENT then CHYBA(9)  else { TISK(9);CTI(N);I;
                                     }
}

procedura I
{ if N = LEVA then { TISK(10); CTI(N); E;
                     if N ≠ PRAVA then CHYBA(10)
                                  else CTI(N)
                   }
                 else TISK(11);
}

procedura  MAIN
{  CTI(N); S;
}
```

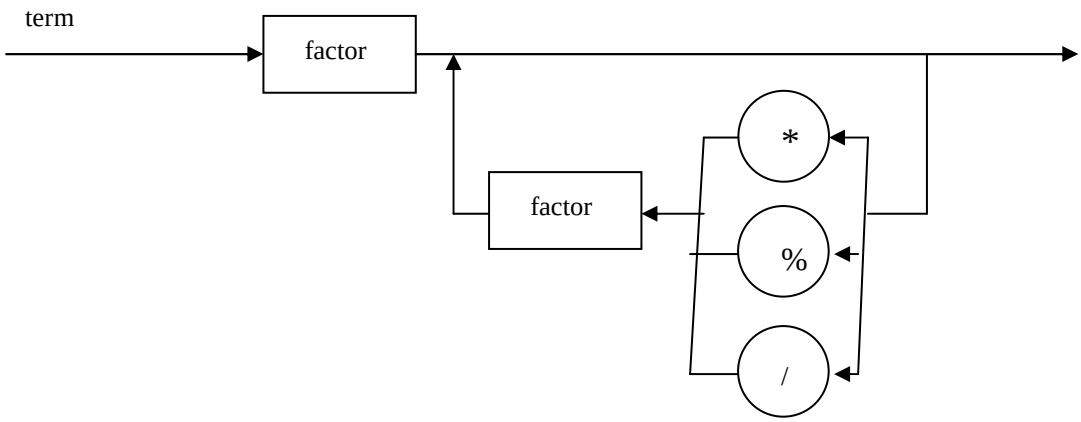
Ověřme na větě: if a then a = a + a

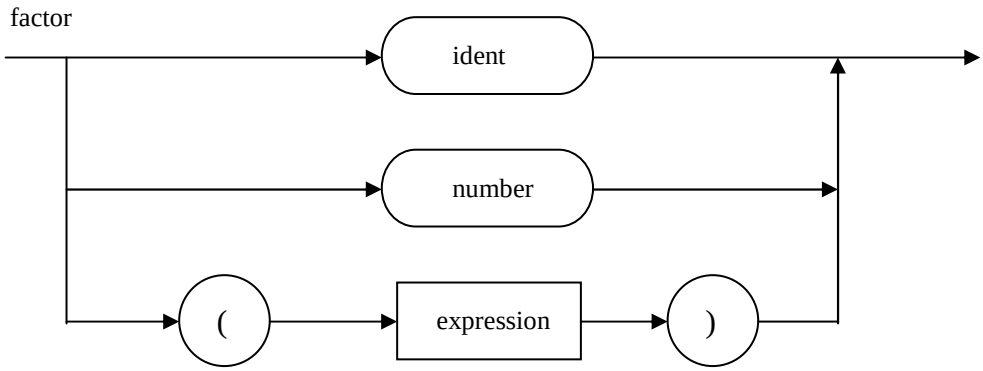
Syntax jazyka PL0




condition
ODD
expression
expression
=
#
<
<=
>
>=
expression


+
expression
term
-
+
term
-


term
factor
*
factor
%
/


factor
ident
number
(
expression
)

X	FIRST (X)	FOLLOW (X)
BLOCK	const , var , procedure ident , call , begin if , while , ε	. , ;
STATEMENT	ident , call , begin if , while , ε	. , ; , end
CONDITION	odd , + , − , ident number , (	then , do
EXPRESSION	(, ident , number , + , −	) , . , ; , = # , < , <= , > , >= then , do , end
TERM	(, ident , number	) , . , ; , = # , < , <= , > , >= then , do , end + , −
FACTOR	(, ident , number	) , . , ; , = # , < , <= , > , >= then , do , end + , − , * , /, %

```
/*C program Lexikální a Syntaktická analýza PL0 – upozorníme jen na hlavní části */

#define NSYM 35              /* pocet rozpoznatelnych symbolu*/
#define NORW 11             /* pocet klicovych slov */
#define TMAX 100            /* velikost tabulky symbolu */
#define NMAX 5             /* maximalni pocet cislic v cisle */
#define AL 10             /* delka identifikatoru */
#define CHSETSIZE 128       /* pocet znaku v mnozine */
#define MAXERR 30           /* maximalni pocet chyb */
#define AMAX 2048           /* nejvyssi adresa */
#define LEVMAX 3            /* maximalni hloubka vnoreni */
#define CXMAX 200           /* velikost prostoru pro kod */
#define STACKSIZE 500   /* vypoctovy zásobník*/

typedef enum {null, ident, number, plus, minus, times, slash, modulo, oddsym, eql,
        neq, lss, leq, gtr, geq, lparen, rparen, comma, semicolon, period,
        becomes, beginsym, endsym, ifsym, thensym, whilesym, dosym, callsym,
        constsym, varsym, procsym} SYMBOL;
typedef char ALFA[AL]; /*pole  k ulozeni textu identifikatoru*/
typedef enum {constant, variable, procedure} OOBJECT; /*druh identifikatoru*/
typedef int SYMSET[NSYM];

char ch;                /*posledni precteny znak*/
FILE *iva, *zdroj;        /*pomocny soubor pro vypis generovaneho kodu*/
                    /*a tabulky symbolu*/

int txpom;              /*pomocna promenna*/
SYMBOL sym;             /*posledni precteny symbol*/
ALFA id;                /*posledni precteny identifikator*/
int num;                /*posledni prectene cislo*/
int cc;                 /*pocet znaku*/
int ll;                 /*delka radku*/
int kk,err;
int cx;                 /*pocitadlo adres; vice bude receno v kap. Pridelovani pameti */
char line[81];          /*nacteny radek*/
ALFA a;
ALFA word[NORW]={"begin", "call", "const", "do", "end", "if", "odd", "procedure",
                 "then", "var", "while"};          /*pole rezervovaných identifikátorů*/
SYMBOL wsym[NORW]={beginsym, callsym, constsym, dosym, endsym, ifsym, oddsym,
                   procsym, thensym, varsym, whilesym}; /*poradi převede na vyctový typ*/
SYMBOL ssym[255]; /*+,-,*,…, jednoznakove oddělovače, plneno v main*/
SYMSET declbegsys, statbegsys, facbegsys;

struct {
   ALFA name;          /*jmeno*/
   OOBJECT kind;       /*druh*/
   union {
      int val;
      struct {
         int level,adr,size;
      } vp;
   } CO;
} TABLE[TMAX+1];

/* nacita ze vstupniho souboru 1 znak do glob. promenne 'ch' a prekroci konec radky */

void getch(void) {   /*sklada znaky do pole line*/
   if (cc == ll) {
```

definice konstant překladače

lex.symboly

atributy identifikatoru

Tabulka symbolů

```
      if (feof(zdroj)) {
         printf("program incompleted");
         exit(2);
      }
      ll = cc = 0;
      printf("      ");

      do {
         fscanf(zdroj,"%c",&ch);
         if ((ch != '\n') && (ch != '\r')) {
            line[ll++]=ch;                    /*pridani znaku do promenne line */
            printf("%c",ch);
         }
      } while ((ch != '\n') && (ch != '\r') && (feof(zdroj) == 0));

      printf("\n");
      line[ll++] = ' ';
   }
   ch = line[cc++];
}   // getch()

/* Lex.analyza nacita ze vstupniho souboru 1 symbol a vrati jeho kod do glob. promenne 'sym' */
void getsym(void) {
int i, j, k;

   while (ch <= ' ') getch();   /* netisknutelne znaky */
   if ((ch >= 'a') && (ch <= 'z')) {   /* identifier or reserved word */
      k = 0;

      do {
         if (k < AL) a[k++] = ch;
         getch();
      } while (((ch >= 'a') && (ch <= 'z')) || ((ch >= '0') && (ch <= '9')));

      a[k] = '\0';
      strcpy(id, a);
      i = 0;
      j = NORW - 1;

      do {
         k = (i + j) / 2;
         if (strcmp(id, word[k]) <= 0) j = k - 1;

         if (strcmp(id, word[k]) >= 0) i = k + 1;
      } while (i <= j);

      if ((i - 1) > j) sym = wsym[k];
      else sym = ident;
   }
   else
      if ((ch >= '0') && (ch <= '9')) {   /* number */
         k = num = 0;
         sym = number;

         /* vypusti pripadne pocatecni nuly u cisla */
        /* while (ch == '0') getch(); */

         do {
            num = 10 * num + (ch - '0');  /*vypocet hodnoty cisla */
```

*/*pulenim intervalu hleda v poli word*/*
*/*zda to je rezervovany identifikátor */*

```
            k++;
            getch();
         } while ((ch >= '0') && (ch <= '9'));
         if (k > NMAX) error(30);
      }
      else
         if (ch == ':') {
            getch();
            if (ch == '=') {                    prirazeni
               sym = becomes;
               getch();
            }
            else sym = null;
         }
         else
            if (ch == '<') {
               getch();
               if (ch == '=') {                 /*mensi roven*/
                  sym = leq;
                  getch();
               }
               else
                  if (ch == '>') {
                     sym = neq;                 /*neroven*/
                     getch();
                  }
                  else
                     sym = lss;                 /*mensi*/
            }
            else
               if (ch == '>') {
                  getch();
                  if (ch == '=') {              /*vetsi roven*/
                     sym = geq;
                     getch();
                  }
                  else sym = gtr;
               }
               else {
                  sym = ssym[ch];               /*jednoznakovy oddělovač viz main*/
                  getch();
               }
} /* konec procedury getsym */
```

```
/* vlozi object do tabulky symbolu
      k    :typ objektu, tj. zda jde o konstantu,promennou,...
      lev :uroven, ve ktere je objekt deklarovan
      dx  :adresa objektu
*/
void enter(OOBJECT k,int *tx,int lev,int *dx) {

   (*tx)++;       /* inkrementuje index  tabulky symbolu */
   txpom = *tx; /* pro vypis TS */
   strcpy(TABLE[*tx].name,id);
   TABLE[*tx].kind=k;

   switch (k) {
      case constant: if (num>AMAX) {
                                 error(31);
                                 num = 0;
                     }
                     TABLE[*tx].CO.val = num;
                     break;

      case variable: TABLE[*tx].CO.vp.level = lev;
                                 TABLE[*tx].CO.vp.adr = (*dx)++;
                                 break;

      case procedure: TABLE[*tx].CO.vp.level = lev;
                                  break;

   }
}   // enter()
```

/*plneni tab. symbolu*/

```
/* vyhleda symbol v tabulce symbolu
      id   :jmeno symbolu
      tx   :ukazovatko na konec tabulky symbolu
   navratova hodnota:
      -1 : symbol nenalezen
     >=0 : adresa symbolu
*/
int position(ALFA id,int tx) {
int i;

   strcpy(TABLE[0].name,id);          /*sentinel*/
   i = tx;                            /*hleda od posledního zarazeneho (respektuje lokalitu*/
   while (strcmp(TABLE[i].name,id)) i--;

return(i);
}   // position()
```

```
/* zpracovani deklarace konstanty ve tvaru:
       ident = hodnota.
       tx   :ukazovatko na volne misto v tabulce symbolu
       lev :uroven, ve ktere je symbol deklarovan
       dx  :adresa - neni pouzita, protoze jde tady o kontantu
*/
void constdeclaration(int *tx,int lev,int *dx) {

if (sym == ident) {
    getsym();
    if ((sym == eql) || (sym == becomes)) {
        if (sym == becomes) error(1);  /*v deklaraci konstant musí byt „=“  */
        getsym();
        if (sym == number) {
            enter(constant,tx,lev,dx);   /*ulozeni konstanty do Tab.Symb, */
            getsym();
        } else error(2);      /*konstante není prirazeno cislo*/
    } else error(3);          /*nenasel = ani prirazeni*/
} else error(4);              /*nenasel identifikátor*/
}   // constdeclaration()

/* zpracovani deklarace promenne
       tx   :ukazovatko na volne misto v tabulce symbolu
       lev :uroven,ve ktere je symbol deklarovan
       dx  :adresa promenne
*/
void vardeclaration(int *tx,int lev,int *dx) {

if (sym == ident) {
    enter(variable,tx,lev,dx);
    getsym();
} else error(4);

}   // vardeclaration()
```

```
void factor(…) {    /*v kompletnim tvaru bude mit parametry*/
int i;

      while (facbegsys[sym]) {   /* facbegsys se naplni v main*/
      if (sym == ident) {
         i = position(id,tx);
         if (i == 0) error(11);      /*nenalezen v Tab.Symb.*/
         else
         getsym();
      } else
         if (sym == number) {
            if (num > AMAX) {
               error(31);
               num = 0;
            }
            getsym();
         } else
            if (sym == lparen) {                  getsym();
               expression(…);
               if (sym == rparen) getsym();
               else error(22);
            }
   }
}   // factor()


void term(…) {           /*v kompletnim tvaru bude mit parametry*/

SYMBOL mulop;
   factor(…);

   while ((sym == times) || (sym == slash) || (sym == modulo)) {
      mulop = sym;
      getsym();
      factor(…);
   }

}   // term()
```

```
void expression(…) {   /*v kompletnim tvaru bude mit parametry*/
SYMBOL addop;

   if ((sym == plus) || (sym == minus)) {  /*unární operatory*/
      getsym();
      term(…);
   }
   else {
      term(…);
   }

   while ((sym == plus) || (sym == minus)) { /*binární operatory*/
      getsym();
      term(…);
   }
}   // expression()


void condition(…) {
SYMBOL relop;

   if (sym == oddsym) {
      getsym();
      expression(…);
   }
   else {
      expression(…);
      if ((sym != eql) && (sym != neq) && (sym != lss) && (sym != gtr) && (sym != leq) &&
         (sym != geq))
         error(22);
      else {
         getsym();
         expression(…);
          }
   }

}   // condition()


void statement(…) {    /*v kompletnim tvaru bude mit parametry*/
int i;

   if  (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);         /*nenasel se identifikátor*/
      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(…);
    }
    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,  …
                                                 else error(15);
               }
                     getsym();
                  }
        }
        else
            if (sym == ifsym) { /*podmineny prikaz*/
               getsym();
               condition(…);

               if (sym == thensym) getsym();
               else error(16);
               statement(…);
            }
            else
               if (sym == beginsym) { /*zacina novy blok*/
                  getsym();
                  statement(…);

                  while  (sym == semicolon) {
                     getsym();
                     else error(10);
                     statement(…);
                  }

                  if (sym == endsym) getsym(); /*konci predchozi blok*/
                  else error(17);
               }
               else
                  if (sym == whilesym) { /*zacina cyklus while*/
                     condition(…);
                     if (sym == dosym) getsym();
                     else error(18);
                     statement(…);
                  }

}   // statement()
```

```
void block(…) {          /*v kompletnim tvaru bude mit parametry*/
   do {
      if (sym == constsym) { /*deklaracni cast konstant*/
         getsym();
         do {
            constdeclaration(…);
            while (sym == comma) {
               getsym();
               constdeclaration(…);
            }
            if (sym == semicolon) getsym();
            else error(5);
         } while (sym==ident);
      }

      if (sym == varsym) { /*deklaracni cast promennych*/
         getsym();
         vardeclaration(…);

         while (sym == comma) {
            getsym();
            vardeclaration(…);
         }
         if (sym == semicolon) getsym();
                  else error(5);
      }

   while (sym == procsym) { /*definice podprogramu*/
         getsym();
         if (sym == ident) {
            enter(procedure);
            getsym();
         } else error(4);
         if (sym==semicolon) getsym();
         else error(5);
         block(…);
         if (sym == semicolon) {
            getsym();
         } else error(5);
      }
   } while (declbegsys[sym]);  /* declbegsys se plni v main*/

   statement(…);

}   // block()
```

```
/*hlavni program*/
main(void) {
char zdrojak[13];

   /* cte jmeno souboru, dokud uzivatel nezada nenulovy retezec */
   do {
      printf("Zadej jmeno souboru obsahujiciho zdrojovy text: ");
      scanf("%s",zdrojak);
   } while (strlen(zdrojak) < 1);

   if ((iva = fopen("TAB.SYM", "w")) == NULL) {
      printf("\nCHYBA! Nepodarilo se otevrit soubor pro zapis tabulky symbolu...\n");
      return(-1);
   }
   /* ...a pak otestuje, jestli soubor existuje */
   if ((zdroj = fopen(zdrojak, "r")) == NULL) {
      printf("\nCHYBA! Nepodarilo se otevrit soubor se zdrojovym textem [%s]...\n",zdrojak);
      return(-1);
   }
   for (ch=' ';ch<='_';ch++) ssym[ch] = null;
   ssym['+'] = plus;
   ssym['-'] = minus;
   ssym['*'] = times;
   ssym['/'] = slash;
   ssym['%'] = modulo;
   ssym['('] = lparen;
   ssym[')'] = rparen;
   ssym['='] = eql;          /*naplneni hodnot jednoznakových oddelovacu*/
   ssym[','] = comma;
   ssym['.'] = period;
   ssym['#'] = neq;
   ssym['<'] = lss;
   ssym['>'] = gtr;
   ssym[';'] = semicolon;
   nuluj(declbegsys);
   nuluj(statbegsys);
   nuluj(facbegsys);
   /*v deklaracni casti se musi zacinat bud 'const','var' nebo 'procedure'*/
   declbegsys[constsym] = declbegsys[varsym] = declbegsys[procsym] = 1;
   /*ve statementu se musi zacinat bud 'begin','call','if','while' nebo ident.*/
   statbegsys[beginsym] = statbegsys[callsym] = statbegsys[ifsym] = statbegsys[whilesym] = 1;
   /*faktor muze byt bud ident., cislo nebo leva zavorka*/
   facbegsys[ident] = facbegsys[number] = facbegsys[lparen] = 1;
   ch = ' ';
   kk = AL;
   getsym();
   block(…); /*zavola preklad programu*/
   if (sym != period) error(9); /*a konci teckou*/
   listtabsym();
   if (err == 0) {
      printf("\nno error in PL/0 program\n");
   }
   else printf("\n %2d error(s) in PL/0 program",err);
return(0);
}
```

Zpracování chyb v PL0

Panický způsob zotavování – triviální strategie:
vynechá text až do místa, kde se snadno vzpamatuje. Snadno se vzpamatuje v místě s významným symbolem.

Předpoklady:

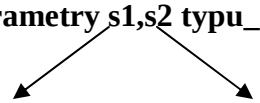
1. **Každý typ příkazu začíná jiným symbolem.**
2. **„ „ deklarace „ „ „ .**
3. **Každá vyvolaná procedura se provede až do konce (žádný chybový výstup).**

Zásady:

1. **Každá procedura má parametr – množinu následujících symbolů.**
2. **Při chybě je přeskočen vstupní text až k legálně následujícímu symbolu za prováděnou procedurou.**
3. **Na konci procedury je proveden Test, který ověří, že příští symbol patří do množiny následovníků.**
4. **Pro zmenšení vynechávaných úseků se do následovníků přidávají symboly ze začátku důležitých konstrukcí (tzv. STOP SYMBOLY). Zdůvodněte si to.**

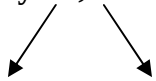
Činnost zajišťuje procedura

Test(parametry s1,s2 typu_množina_symbolů; n celočíselné_označení_chyby)



Následující symboly **Stop symboly**

Procedura Test je využitelná i k ověření akceptovatelnosti symbolů na začátku procedur SA. Symboly s1 , s2 mají pak jiný význam



Počáteční **Následující**

Seznam chyb

1	**pouzito "=" misto ":="**
2	**za "=" musi nasledovat cislo**
3	**za identifikatorem ma nasledovat "="**
4	**za "const", "var", "procedure" musi nasledovat identifikator**
5	**chybi strednik nebo carka**
6	**nespravny symbol po deklaraci procedury**
7	**je ocekavan prikaz**
8	**neocekavany symbol za prikazovou casti bloku**
9	**ocekavam tecku**
10	**nespravny symbol v prikazu**
11	**nedeklarovany identifikator**
12	**prirazeni konstante a procedure neni dovoleno**
13	**operator prirazeni je ":="**
14	**za "call" musi nasledovat identifikator**
15	**volani konstanty nebo promenne neni dovoleno**
16	**ocekavano "then"**
17	**ocekavano "}" nebo ";"**
18	**ocekavano "do"**
19	**nespravne pouzity symbol za prikazem**
20	**ocekavam relaci**
21	**jmeno procedury nelze pouzit ve vyrazu**
22	**chybi uzaviraci zavorka**
23	**faktor nesmi koncit timto symbolem**
24	**vyraz nesmi zacinat timto symbolem**
30	**prilis velke cislo**

Alternativa pro C++

```
/* testovat zda nacteny symbol je v mnozine symbolu 's1'.
   Pokud neni generuje chybu a nacita opakovane ze vstupu
   dokud neni nacten symbol z mnozin 's1' a 's2'
*/
void test(SYMSET s1,SYMSET s2,int n) {
SYMSET pom;

   if (!s1[sym]) {        /*sym není v množině s1 */
      error(n);
      nuluj(pom);
      sjednot(pom,s1);    } /*sjednoti s1, s2 do pom */
      sjednot(pom,s2);

      while (pom[sym] == 0) getsym();  /* pokud sym není v s1∪ s2 cti další */
   }
}   // test()

void expression(SYMSET fsys) {
SYMBOL addop;
SYMSET pom;

   if ((sym == plus) || (sym == minus)) { /*unární plus, minus  */
      addop = sym;
      getsym();
      nuluj(pom);
      sjednot(pom,fsys);
      pom[plus] = pom[minus] = 1;
      term(pom);       /*volame term(fsys ∪ plus ∪ minus ) */
    }
   else {
      nuluj(pom);
      sjednot(pom,fsys);
      pom[plus] = pom[minus] = 1;   /*bez unárního plus minus */
      term(pom);
   }

   while ((sym == plus) || (sym == minus)) {
      addop = sym;
      getsym();
      nuluj(pom);                          /* iterace {+T}  */
      sjednot(pom,fsys);
      pom[plus] = pom[minus] = 1;
      term(pom);

    }
}   // expression()
```

```
void term(SYMSET fsys) {
SYMBOL mulop;
SYMSET pom;

   nuluj(pom);
   sjednot(pom,fsys);
   pom[times] = pom[slash] = pom[modulo] = 1;
   factor(pom);        /*volame factor( followE ∪  follow T */

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

void factor(SYMSET fsys) {
int i;
SYMSET pom;

   test(facbegsys,fsys,24);            /* test na zacatku faktoru */
   while (facbegsys[sym]) {
      if (sym == ident) {
         i = position(id,tx);
         if (i == 0) error(11);        /* ten identifikátor není deklarovany */
         getsym();
      } else
         if (sym == number) {
            if (num > AMAX) {
               error(31);
               num = 0;
            }
            getsym();
         } else
            if (sym == lparen) {
               getsym();
               nuluj(pom);
               sjednot(pom,fsys);
               pom[rparen] = 1;
               expression(pom);
               if (sym == rparen) getsym();/* follow „lparen expression“ je rparen */
               else error(22);             /* paren chybi */
            }
            nuluj(pom);
            pom[lparen] = 1;         /* pokud ses nezotavil drive, preskoc jen k lparen */
            test(fsys,pom,23);       /* test na konci faktoru */
   }
}  // factor()
```