SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Fakulta informatiky a informačných technológií

Dokumentácia k projektu: SMALL

Návrh prekladačov

Autori: **Ján Garaj, Pavol Fábik**
Študijný odbor: *Softvérové inžinierstvo*
Ročník: 1
Školský rok: 2007/2008

Obsah

1 Zadanie projektu

ANALÝZA A NÁVRH

1. Vytvorte niekoľko príkladov viet daného jazyka.
2. Prepíšte gramatiku z BNF (Backus Naur Form) do gramatických pravidiel s alternatívami. Vyznačte miesta, kde gramatika nespĺňa podmienky pre deterministickú analýzu.
3. Transformujte gramatiku tak, aby bola LL(1):
   - odstráňte ľavú rekurziu (ak treba)
   - urobte ľavú faktorizáciu (ak treba)
4. Nájdite
   - množinu FIRST pre každý neterminál v transformovanej gramatike
   - množinu FOLLOW pre každý neterminál v transformovanej gramatike
5. Presvedčte sa, že transformovaná gramatika je LL(1).
6. Pre transformovanú LL(1).gramatiku vytvorte tabuľku prechodov.

IMPLEMENTÁCIA

7. Implementujte lexikálny analyzátor rozpoznávajúci potrebné lexikálne jednotky.
8. Vytvorte tabuľkou riadený syntaktický analyzátor (SA), ktorý
   - analyzuje postupnosť lexikálnych jednotiek na vstupe
   - ak postupnosť zodpovedá vete jazyka, skončí SA prijatím; inak oznámi chybu
   - vypisuje protokol o svojej činnosti obsahujúci informáciu o uskutočnených akciách pri parsovaní vstupu
9. SA testujte na vetách jazyka vytvorených v 1)

PROJEKT#3 – SMALL
SMALL je jednoduchý programovací jazyk zostavený za účelom pedagogiky

```
program ::= 'BEGIN' statement_list 'END' .
statement_list ::= statement {statement} .
statement ::= ident ':=' expression ';' .
statement ::= 'READ' '(' id_list ')' ';' .
statement ::= 'WRITE' '(' expr_list ')' ';' .
statement ::= 'IF' bexpr 'THEN' statement ['ELSE' statement] ';' .
id_list ::= ident {',' ident} .
expr_list ::= expression {',' expression} .
expression ::= [expression op] factor .
factor ::= '(' expression ')' .
factor ::= ident | number .
op ::= '+' | '-' .
bexpr ::= [bexpr 'OR'] bterm .
bterm ::= [bterm 'AND'] bfactor .
bfactor ::= 'NOT' bfactor | '(' bexp ')' | 'TRUE' | 'FALSE' .
ident ::= letter {letter | digit09}.
number ::= [ '+' | '-' ] digit19 {digit09}.
digit09 ::= 0 | .. | 9 .
digit19 ::= 1 | .. | 9 .
letter ::= a | b | .. | z | A | B | .. | Z
```

2 Analýza a návrh

2.1 Príklady viet z jazyka SMALL

Korektné vety

Príklad 1:

```
BEGIN
      READ(a);
      a:=1;
      IF FALSE THEN c:=20;
      WRITE(c);
END
```

Príklad 2:

```
BEGIN
      READ(a,b);
      e:=a+b;
      IF TRUE AND NOT (FALSE) THEN e := e + 20; ELSE e := e + 30;;
      WRITE(e);
END
```

Príklad 3:

```
BEGIN
      READ(a,b);
      IF TRUE OR NOT (TRUE) THEN WRITE(a);
      ELSE WRITE(b);;
END
```

Nekorektné vety

Príklad 1:

```
BEGIN
      READ(a,b);
      a:=1              // chýba znak ';' za 1
      IF TRUE THEN c:=20;
      WRITE(c);
END
```

Príklad 2:

```
BEGIN
      READ(a,b,c,d);
      e:=a+b-c-d;
      IF 1 THEN e := e + 20 ELSE e := e + 30; // chýba znak ';'
      WRITE(e);
END
```

Príklad 3:

```
READ(a,b);                                  // nesprávny začiatok programu
IF TRUE OR NOT (TRUE) THEN WRITE(a)         // chýba znak ; za WRITE()
      ELSE WRITE(b);
END
```

2.2 Prepis gramatiky z BNF do gramatických pravidiel s alternatívami

V tabuľke pravidiel je výsledná gramatika. Skratka R označuje miesta, kde sa vyskytuje ľavá rekurzia a skratka F označuje miesto, kde je potrebné vykonať ľavú faktorizáciu.

Číslo	Nedeterminizmus	Ľavá strana	Pravá strana
1		Program	**'BEGIN'** statement_list **'END'**
2	F 2,3	statement_list	statement
3	F 2,3	statement_list	statement statement_list
4		statement	ident **':='** expression **';'**
5		statement	**'READ'** **'('** id_list **')'** **';'**
6		statement	**'WRITE'** **'('** expr_list **')'** **';'**
7	F 7,8	statement	**'IF'** bexpr **'THEN'** statement **'ELSE'** statement**';'**
8	F 7,8	statement	**'IF'** bexpr **'THEN'** statement **';'**
9	F 9,10	id_list	Ident
10	F 9,10	id_list	ident **','** id_list
11	F 11,12	expr_list	expression
12	F 11,12	expr_list	expression **','** expr_list
13	R	expression	expression op factor
14		expression	Factor
15		factor	**'('** expression **')'**
16		factor	Ident
17		factor	Number
18		Op	**'+'**
19		Op	**'-'**
20		bexpr	Bterm
21	R	bexpr	bexpr **'OR'** bterm
22		bterm	bfactor
23	R	bterm	bterm **'AND'** bfactor
24		bfactor	**'NOT'** bfactor
25		bfactor	**'('** bexp **')'**
26		bfactor	**'TRUE'**
27		bfactor	**'FALSE'**
28	F 28,30	ident	Letter
29	F 28,30	ident	letter ident
30	F 28,30	ident	letter ident_ext
31		ident_ext	digit09 ident_ext
32		ident_ext	**E**
33		number	**+** digit19 number_ext
34		number	**-** digit19 number_ext
35		number	digit19 number_ext
36		number_ext	digit09 number_ext
37		number_ext	**E**
38		digit09	**0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
39		digit19	**1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
40		letter	**a | .. | z | A | .. | Z**

2.3 Transformovaná gramatika LL(1)

V nasledujúcej tabuľke je transformovaná gramatika s odstránenou ľavou rekurziou a faktorizáciou. V prvom stĺpci je uvedené pôvodné číslovanie pravidiel, v nasledujúcom stĺpci je nové číslovanie. Výsledná gramatika spĺňa podmienky LL(1) gramatík.

Číslo	Nové č.	Ľavá strana	Pravá strana
1	1	program	**'BEGIN'** statement_list **'END'**
2	2	statement_list	statement statement_list_ext
3	3	statement_list_ext	statement_list
3a	4	statement_list_ext	**E**
4	5	statement	ident **':='** expression **';'**
5	6	statement	**'READ'** **'('** id_list **')'** **';'**
6	7	statement	**'WRITE'** **'('** expr_list **')'** **';'**
7	8	statement	**'IF'** bexpr 'THEN' statement statement_ext
7a	9	statement_ext	**'ELSE'** statement **';'**
8	10	statement_ext	**E**
9	11	id_list	ident id_list_ext
10	12	id_list_ext	**','** id_list
10a	13	id_list_ext	**e**
11	14	expr_list	expression expr_list_ext
12	15	expr_list_ext	**','** expr_list
12a	16	expr_list_ext	**e**
13	17	expression	factor expression_ext
14	18	expression_ext	op factor expression_ext
14a	19	expression_ext	**e**
15	20	factor	**'('** expression **')'**
16	21	factor	ident
17	22	factor	number
18	23	op	**'+'**
19	24	op	**'-'**
20	25	bexpr	bterm bexpr_ext
21	26	bexpr_ext	**'OR'** bterm bexpr_ext
21a	27	bexpr_ext	**e**
22	28	bterm	bfactor bterm_ext
23	29	bterm_ext	**'AND'** bfactor bterm_ext
23a	30	bterm_ext	**e**
24	31	bfactor	**'NOT'** bfactor
25	32	bfactor	**'('** bexpr **')'**
26	33	bfactor	**'TRUE'**
27	34	bfactor	**'FALSE'**
28	35	ident	letter ident_ext2
29	36	ident_ext2	ident
30	37	ident_ext2	ident_ext
30a	38	ident_ext2	**e**
31	39	ident_ext	digit09 ident_ext
32	40	ident_ext	**e**
33	41	number	**'+'** digit19 number_ext
34	42	number	**'-'** digit19 number_ext
35	43	number	digit19 number_ext
36	44	number_ext	digit09 number_ext
37	45	number_ext	**e**
38	46	digit09	**0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
39	47	digit19	**1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
40	48	letter	**a | .. | z | A | .. | Z**

2.4 Množiny FIRST a FOLLOW

V nasledujúcej tabuľke sú zhrnuté množiny FIRST a FOLLOW pre všetky neterminálne symboly. Symboly *ident* a *number* už ďalej nerozpisujeme, pretože tieto symboly považujeme pri implementácií syntaktického analyzátora za terminálne.

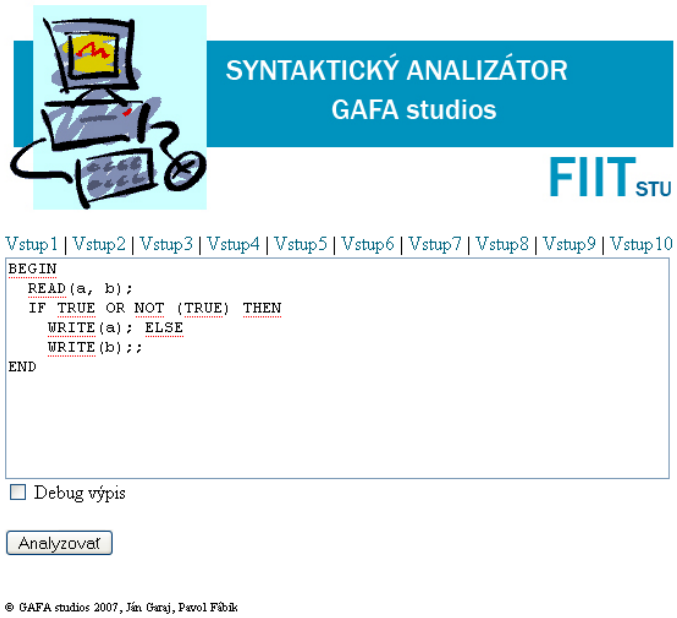
Číslo	Neterminál	FIRST	FOLLOW
1	program	'BEGIN'	e
2	statement_list	ident 'READ' 'WRITE' 'IF'	'END'
3	statement_list_ext	ident 'READ' 'WRITE' 'IF' e	'END'
4	statement	ident 'READ' 'WRITE' 'IF'	'END' ';' 'READ' 'WRITE' 'IF' 'ELSE' ident
5	statement_ext	'ELSE' e	ident'READ' 'WRITE' 'IF'';'
6	id_list	ident	')'
7	id_list_ext	',' e	')'
8	expr_list	'(' ident number	')'
9	expr_list_ext	',' e	')'
10	expression	'(' ident number	';' ')' ','
11	expression_ext	'+' '-' e	';' ')' ','
12	factor	'(' ident number	'+' '-' ';' ')' ','
13	op	'+' '-'	'(' ident number
14	bexpr	'NOT' '(' 'TRUE' 'FALSE'	'THEN' ')'
15	bexpr_ext	'OR' e	'THEN' ')'
16	bterm	'NOT' '(' 'TRUE' 'FALSE'	'THEN' ')' 'OR'
17	bterm_ext	'AND' e	'THEN' ')' 'OR'
18	bfactor	'NOT' '(' 'TRUE' 'FALSE'	'THEN' ')' 'AND' 'OR'

2.5 Tabuľka prechodov

	BEGIN	END	IF	THAN	ELSE	READ	WRITE	AND	OR	NOT	TRUE	FALSE	IDENT	NUMBER	:=	+	-	;	,
program	1																		
statement_list			2			2	2						2						
statement_list_ext		4	3			3	3						3						
statement			8			6	7						5						
statement_ext			10		9	10	10						10					10	
id_list													11						
id_list_ext																			12
expr_list													14	14					
expr_list_ext																			15
expression													17	17					
expression_ext																18	18	19	19
factor													21	22					
op																23	24		
bexpr										25	25	25							
bexpr_ext				27					26										
bterm										28	28	28							
bterm_ext				30				29	30										

3 Implementácia

Dané zadanie sme realizovali v dvoch fázach. V prvej sme navrhli a implementovali Lexikálny analyzátor a v druhej sme sa zamerali na tabuľkou riadený syntaktický analyzátor. Obe časti sú implementované v jazyku php. Zadanie je umiestnené na internetovej stránke http://garaj.positive.sk/np/ . V prvom kroku si používateľ môže vybrať z desiatich preddefinovaných vstupov. Štyri vstupy sú korektné, ostatné obsahujú chyby. Používateľ si môže napísať aj vlastný vstup v textovej ploche. . Stlačením tlačidla analyzovať sa spustí činnosť prekladového automatu na zvolenom vstupe. Výstupom je informácia o úspešnosti sémantickej, syntaktickej analýzy a zoznam vyparsovaných lexikálnych jednodiek. Používateľ si môže zvoliť pri analýze aj debug výpis. Ten mu poskytuje informácie o každom kroku činnosti prekladača.



Obr. 1 Hlavná stránka

Lexikálny analyzátor

Lexikálny analyzátor postupne prechádza vstupným súborom a vyberá z neho znaky jeden za druhým. Počas analýzy z načítavaných znakov vytvára dátové jednotky – tokeny. Tie môžu byť len určitého druhu: kľúčové slová (BEGIN, END, IF, TRUE, ...), operátory (+, -, :=, ..), oddeľovače (‚(' , ‚)' , ‚;') alebo identifikátory – premenné. Takto rozpoznané tokeny sú ukladané do tabuľky LJ[] vo forme stringov. Takúto reprezentáciu sme zvolili preto, aby sa zjednodušila implementácia syntaktického analyzátora, ktorý používa asociatívne indexované polia pri svoje činnosti.

Syntaktický analyzátor

Podstatou syntaktickej analýzy je určenie, či vstup je syntakticky správny vzhľadom na pravidlá pre nejaký zadefinovaný jazyk (či analyzovaná veta patrí do tohto jazyka).

Syntaktický analyzátor požaduje na vstupe tabuľku lexikálnych jednotiek vytvorenú lexikálnym analyzátorom. Svoju činnosť riadi podľa tabuľky prechodov, ktorá je reprezentovaná ako pole veľkosti NxT, kde N je počet neterminálnych symbolov a T je počet terminálnych symbolov. Jednotlivé pravidlá gramatiky sú reprezentované poľom veľkosti rovnej počtu pravidiel. Každý prvok tohto poľa je vlastne reťazec symbolov, teda samotná ľavá strana pravidla.

4 Záver

Úlohou bolo vytvoriť lexikálny a tabuľkou riadený syntaktický analyzátor pre jazyk SMALL. Nami vytvorený program je vlastne prvou časťou procesu prekladu. V prípade potreby je možné rozšíriť našu implementáciu o potrebné funckie k vytvoreniu celého prekladača a tým dokončiť celý prekladový proces. Analyzátor sme testovali na vytvorených príkladoch viet.

Príloha 1

Lexikálny analyzátor

V tejto časti prikladáme dôležité časti implementácie lexikálneho analyzátora.

```
function LexikalnaAnalyza($Data, &$LexikalnaJednotka, &$Pozicia, &$Pocet, $debug) {
   $Riadok = 1;
   $Stlpec = 1;
   $Pocet = 0; //počet nájdených lexikálnych jednotiek
   $Vystup = '';
   $DlzkaData = strlen($Data);
   $AktualnaPozicia = 0;
   $DlzkaJednotky = 0;
   while ($AktualnaPozicia != $DlzkaData) {
    $Znak = substr($Data, $AktualnaPozicia, 1);
    $AktualnaPozicia++;

    $MinPocet = $Pocet;
    if (JeToBielyZnak($Znak) == true) {
     $DlzkaJednotky = 1;
     //oddeľovacie znaky - ignorujeme
     if($debug) $Vystup .= "Nájdený biely znak - ASCII číslo ".ord($Znak)."<br>";
     if (ord($Znak) == 10) {
      $Riadok++; $Stlpec = 1;
     }
     else {
      $Stlpec++;
     }
     continue;
    }
    elseif (JeToPismeno($Znak) == true) {
     //kľúčové slovo / identifikátor
     $Slovo = '';
     $DlzkaSlova = 0;
     while (JeToPismeno($Znak) == true || JeToCislica($Znak) == true) {
      $Stlpec++;
      if ($DlzkaSlova <= MAXIMALNA_DLZKA_KLUCOVEHO_SLOVA) {
       $Slovo .= $Znak;
       $DlzkaSlova++;
       $DlzkaJednotky++;
      }

      $Znak = substr($Data, $AktualnaPozicia, 1); {
      $AktualnaPozicia++;
      }
     }
     $AktualnaPozicia--; //aby som sa vratil naspat o jeden znak zo zistovania slova
     $LexikalnaJednotka[$Pocet] = KlucoveSlovo($Slovo);
     if ($LexikalnaJednotka[$Pocet] === false) {
      //nie je to kľúčové slovo
```

```
        $LexikalnaJednotka[$Pocet] = 'ident';
        if($debug) $Vystup .= "Najdený identifikátor ".$Slovo."<br>";
       }
       else {
        if($debug) $Vystup .= "Najdené kľúčové slovo ".$Slovo."<br>";
       }
       $Pocet++;
      }
      elseif (JeToCislica($Znak) == true) {
       //číslo
       if($debug) $Vystup .= "Najdené číslo ";
       do {
        $Stlpec++;
        $DlzkaJednotky++;
        if($debug) $Vystup .= $Znak;
        $Znak = substr($Data, $AktualnaPozicia, 1); $AktualnaPozicia++; //$Znak = NacitajZnak(Subor);
       } while (JeToCislica($Znak) == true);
       $AktualnaPozicia--;
       if($debug) $Vystup .= "<br>";
       $LexikalnaJednotka[$Pocet] = 'number';
       $Pocet++;
      }
      elseif ($Znak == ':') {
       //priradenie
       $DlzkaJednotky++;
       $Stlpec++;
       $Znak = substr($Data, $AktualnaPozicia, 1); $AktualnaPozicia++; //Znak = NacitajZnak(Subor);
       if ($Znak != '=') {
        //chyba priradenia
        $Vystup .= "<font color=\"red\">Chyba priradenia na riadku ".$Riadok." $Pozícii ".$Stlpec.",
očakávaný znak '='</font><br>";
       }
       else {
        if($debug) $Vystup .= "Nájdené priradenie :=<br>";
        $LexikalnaJednotka[$Pocet] = ':=';
       }
       $Pocet++;
       $Stlpec++;
       $DlzkaJednotky++;
      }
      elseif (JeToSymbol($Znak)) {
       //špeciálny symbol
       switch ($Znak)
       {
        case '+':
         $Znak = '+';
         if($debug) $Vystup .= "Nájdený symbol +<br>";
         break;
        case '-':
         $Znak = '-';
         if($debug) $Vystup .= "Nájdený symbol -<br>";
         break;
        case ';':
         $Znak = ';';
```

```
          if($debug) $Vystup .= "Nájdený symbol ;<br>";
          break;
        case ',':
          $Znak = ',';
          if($debug) $Vystup .= "Nájdený symbol ,<br>";
          break;
        case '(':
          $Znak = '(';
          if($debug) $Vystup .= "Nájdený symbol (<br>";
          break;
        case ')':
          $Znak = ')';
          if($debug) $Vystup .= "Nájdený symbol )<br>";
          break;
      }
      $LexikalnaJednotka[$Pocet] = $Znak;
      $Pocet++;
      $Stlpec++;
      $DlzkaJednotky++;
    }
    else {
      //chybný znak
      $Vystup .= "<font color=\"red\">Nájdený chybný znak ".$Znak." na riadku ".$Riadok." $Pozícii
".$Stlpec."</font><br>";
      $DlzkaJednotky++;
      //return false;
    }

    if ($MinPocet != $Pocet) {
      $Pozicia[$MinPocet] = array('X'=>$Stlpec, 'Y'=>$Riadok, 'start'=>$AktualnaPozicia-$DlzkaJednotky );
    }
  }
  return $Vystup;
 }
```

Príloha 2

Syntaktický analyzátor

Dôležité časti implementácie syntaktického analyzátora.

```
 function SyntaktickaAnalyza($LexikalnaJednotka, &$Pozicia, $Zasobnik, $debug) {
 ...
  $Poz = 0;  //pozicia v lexikalnych jednotkach
  $Vstup;    //aktuálny vstupny znak
  $Zas;     //prvok zo zasobnika
  $Cinnost;  //cinnost ktora sa ide vykovat, 0-redukcia, >0 expanzia, <0 chyba
  $Vystup = '';
  $PocetLJ = sizeof($LexikalnaJednotka);
  array_push($Zasobnik, $NazvySym[0]);  //aby som mal dno zasobnika
```

```
  array_push($Zasobnik, 'program');
  while ($Poz < $PocetLJ) {
   $Vstup = $LexikalnaJednotka[$Poz];
   if($debug) $Vystup .= "<br><b>Vstup</b> ".$Vstup."<br>";

   do {
    $Zas = array_pop($Zasobnik);
    if ($Zas == $Vstup) {
     //redukcia
     $Cinnost = 0;
     if($debug) $Vystup .= "<b>Redukcia</b><br>symbol ".$Vstup."<br>";
    }
    elseif( !isset($Prechody[$Zas]) || !isset($Prechody[$Zas][$Vstup]) ) {
     //syntaktická chyba
     $Cinnost = -150;
    }
    else {
     //zistenie činnosti z tabuľky prechodov
     $Cinnost = $Prechody[$Zas][$Vstup];
     if($Cinnost == "CH") $Cinnost = -150; //chyba
    }
    if ($Cinnost > 0) {
     if($debug) $Vystup .= "<b>Expanzia:</b><br>stack1: ";
     $Expanzia = '';
     foreach($Zasobnik as $key=>$value) {
      $Expanzia = $value." ".$Expanzia;
     }
     if($debug) $Vystup .= $Zas." ".$Expanzia."<br>";
     //natlacim pravidla do zasobnika
     foreach($Pravidla[$Cinnost] as $key=>$value){
      array_push($Zasobnik, $NazvySym[$value]);
     }
     if($debug) $Vystup .= "stack2: ";
     $Expanzia = '';
     foreach($Zasobnik as $key=>$value) {
      $Expanzia = $value." ".$Expanzia;
     }
     if($debug) $Vystup .= $Expanzia."<br>";
    }

    if ($Cinnost < 0) {
     $Vystup .= "<font color=\"red\">Syntaktická chyba na riadku číslo ".$Pozicia[$Poz]['Y'].", na pozícii
".$Pozicia[$Poz]['X']."</font><br>";
     $Pozicia[$Poz]['error'] = 1;
    }

   } while ($Cinnost > 0);
```

```
    $Poz++;
   }

   if ((sizeof($Zasobnik)!=1 && $Zasobnik[0]!=$NazvySym[0]) || preg_match('/<font color="red">/',
$Vystup) ) {
    //v zasobniku nemam iba dno zasobnika
    $Vystup .= "<font color=\"red\"><b>Syntaktická analýza neúspešná</b></font>";
   }
   else {
    $Vystup .="<b>Syntaktická analýza úspešná</b>";
   }
   return $Vystup;
  }
```