

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ  
Fakulta informačních technologií

Dokumentácia projektu

**Implementace překladače imperativního jazyka  
IFJ17**

Tým 010, varianta II

2. decembra 2017

Rozšírenia: IFTHEN – implementované pomocou stack-u  
a counter-u

Peter Andris – xandri04 – 25%

Adam Petráš – xpetra19 – 25%

Erik Pitko – xpitko00 – 25%

Michal Šajdík – xsajdi00 - 25%

## Obsah

1 Úvod.....	1
2 Vývoj a rozdelenie práce .....	2
2.1 Rozdelenie práce.....	2
2.2 Vývoj.....	2
3 Implementácia .....	3
3.1 Lexikálny analyzátor .....	3
3.2 Syntaktický analyzátor .....	4
3.3 Tabuľka symbolov .....	7
3.4 Generovanie 3-adresného kódu .....	7
3.5 Vstavané funkcie .....	7
4 Testovanie .....	8
5 Zdroje.....	9

# 1 Úvod

Dokumentácia popisuje postupný vývoj a implementáciu prekladača pre jednoduchý imperatívny jazyk IFJ17, ktorý je odvodený z jazyku freeBasic. Dokumentácia je rozčlenená do 3 hlavných kapitol a do niekoľkých podkapitol, v ktorých je rozpísaný vývoj, rozdelenie práce a implementácia všetkých dôležitých častí. Popísané sú všetky podstatné časti prekladaču, použité algoritmy a problémy, ktoré nám našu programátorskú cestu komplikovali. Súčasťou dokumentácie je aj LL-gramatika a LL tabuľka, ktorá je jadrom syntaktického analyzátoru, precedenčná tabuľka a konečný automat lexikálneho analyzátoru.

## 2 Vývoj a rozdelenie práce

Na základe praktických dôvodov, sme sa rozhodli rozdeliť problematiku prekladača na menšie podproblémy, ktoré každý člen riešil samostatne. Bolo nutné implementovať lexikálny analyzátor, vytvoriť teoretické podklady pre syntaktický analyzátor (LL-gramatika, LL tabuľka, precedenčná tabuľka a pravidla pre precedenčnú analýzu), implementovať syntaktický analyzátor, ktorý bol v zdrojovom kóde rozdelený na analyzátor bežného zdrojového kódu (LL-gramatika, LL tabuľka) a analyzátor výrazu (precedenčná tabuľka), sémantickú analýzu, tabuľku symbolov a generovanie 3-adresného kódu. Okrem toho sme implementovali zadané vstavané funkcie jazyka IFJ17, ale aj vlastné pomocné funkcie.

### 2.1 Rozdelenie práce

- Peter Andris – Tabuľka symbolov, testovanie, dokumentácia a LL-gramatika...
- Adam Petráš – Expression parser, tabuľka symbolov, vstavané funkcie a pomocné funkcie...
- Erik Pitko – Syntaktický analyzátor, pomocné funkcie, LL-gramatika, testovanie a generovanie kódu...
- Michal Šajdík – Lexikálny analyzátor, testovanie, generovanie kódu...

### 2.2 Vývoj

Celý projekt sme vyvíjali na webovej službe Github. Vďaka Github-u a pravidelným schôdzkam sme dokázali rýchlejšie riešiť prípadne vzniknuté problémy. Na začiatku vývoja sa nám podarilo vcelku rýchlo implementovať lexikálny analyzátor a pomocné funkcie. Na základe čoho sme predpokladali, že sú správne implementované a že projekt nebude až tak zložitý ako sme spočiatku predpokladali, avšak opak bol pravdou a už po prvých testovaniach sme začali zisťovať nedostatky týchto častí, ktoré sme následne opravovali. Celá implementácia sa začala spomaľovať novými poznatkami, ktoré sme spočiatku buď zle pochopili, alebo na ktoré sme prišli sledovaním fóra projektu. Nakoniec sa nám všetky vzniknuté problémy podarilo vyriešiť z nasadením úsilia všetkých členov tímu.



## 3.2 Syntaktický analyzátor

Syntaktický analyzátor je srdcom celého prekladaču, jeho úlohou je kontrolovať, či vstup zadaný na štandardnom vstupe je syntakticky správne čo znamená, že kontroluje či postupnosť prichádzajúcich tokenov z lexikálneho analyzátoru odpovedá pravidlám nami vytvorenej LL-gramatiky a LL tabuľky. Ďalej ukladá všetky deklarované funkcie, parametre funkcií a premenné do tabuľky symbolov. Syntaktický analyzátor je implementovaný metódou zhora-dole, ale druhá časť syntaktického analyzátora slúžiaca na spracovanie výrazov je implementovaná zdola-hore. V poslednom rade krokov, syntaktický analyzátor volá funkcie určené na vygenerovanie 3-adresného kódu.

	+	-	*	/	\	=	<	>	<=	>=	<>	(	)	EOL	Variable
+	>	>	<	<	<	>	>	>	>	>	>	<	>	>	<
-	>	>	<	<	<	>	>	>	>	>	>	<	>	>	<
*	>	>	>	>	>	>	>	>	>	>	>	<	>	>	<
/	>	>	>	>	>	>	>	>	>	>	>	<	>	>	<
\	>	>	>	>	>	>	>	>	>	>	>	<	>	>	<
=	<	<	<	<	<	>	<	<	<	<	>	<	>	>	<
<	<	<	<	<	<	>	>	>	>	>	>	<	>	>	<
>	<	<	<	<	<	>	>	>	>	>	>	<	>	>	<
<=	<	<	<	<	<	>	>	>	>	>	>	<	>	>	<
>=	<	<	<	<	<	>	>	>	>	>	>	<	>	>	<
<>	<	<	<	<	<	>	<	<	<	<	>	<	>	>	<
(	<	<	<	<	<	<	<	<	<	<	<	<	=	Err	<
)	>	>	>	>	>	>	>	>	>	>	>	Err	>	>	Err
EOL	<	<	<	<	<	<	<	<	<	<	<	<	Err	Err	<
Variable	>	>	>	>	>	>	>	>	>	>	>	Err	>	>	Err

Obrázok 2: Precedenčná tabuľka

Pravidlá pre precedenčnú analýzu:

- E E + E
- E E - E
- E E \* E
- E E / E
- E E \ E
- E E = E
- E E < E
- E E > E
- E E <= E
- E E >= E
- E E <> E
- E ( E )
- E EOL
- E variable

## LL-gramatika:

1: <prog_body>	=>	EOF
2: <prog_body>	=>	EOL <prog_body>
3: <prog_body>	=>	Scope EOL <main_body> <prog_body>
4: <prog_body>	=>	Declare Function fncName ( <par_list> As <var_type> EOL <prog_body>
5: <prog_body>	=>	Function id ( <par_list> As <var_type> EOL <fnc_body> EOL <prog_body>
6: <main_body>	=>	<command><main_body>
7: <main_body>	=>	End Scope
8: <fnc_body>	=>	<command><fnc_body>
9: <fnc_body>	=>	End Function
10: <if_body>	=>	<command><if_body>
11: <if_body>	=>	ElseIf EXP Then EOL <if_body>
12: <if_body>	=>	Else <else_body>
13: <if_body>	=>	End If
14: <else_body>	=>	<command><else_body>
15: <else_body>	=>	End If
16: <while_body>	=>	<command><while_body>
17: <while_body>	=>	Loop
18: <par_list>	=>	)
19: <par_list>	=>	id As <var_type> <par_next>
20: <par_next>	=>	, id As <var type> <par_next>
21: <par_next>	=>	)
22: <arg_list>	=>	)
23: <arg_list>	=>	id <arg_next>
24: <arg_list>	=>	const <arg_next>
25: <arg_next>	=>	)
26: <arg_next>	=>	, <arg_next2>
27: <arg_next2>	=>	id <arg_next>
28: <arg_next2>	=>	const <arg_next>
29: <var_type>	=>	Integer
30: <var_type>	=>	Double
31: <var_type>	=>	String
32: <print_exp>	=>	EXP; <print_exp>
33: <print_exp>	=>	EOL
34: <command>	=>	Input id EOL
35: <command>	=>	If EXP Then EOL <if_body> EOL
36: <command>	=>	id = <assignment> EOL
37: <command>	=>	Do While EXP EOL <while_body> EOL
38: <command>	=>	Print EXP ; <print_exp>
39: <command>	=>	Dim id As <var_type> <dim_end>
40: <command>	=>	Return EXP EOL

41: <command> => EOL  
 42: <dim\_end> => EOL  
 43: <dim\_end> => =  
 <assignment> EOL  
 44: <assignment> => EXP  
 45: <assignment> => id ( <arg\_list>

# **LL tabul'ka:**

	<i>EOF</i>	<i>EOL</i>	<i>SCOPE</i>	<i>DECLARE</i>	<i>FUNCTION</i>	<i>&lt;command&gt;</i>	<i>END</i>	<i>ELSEIF</i>
<i>&lt;prog_body&gt;</i>	1	2	3	4	5			
<i>&lt;main_body&gt;</i>						6	7	
<i>&lt;fnc_body&gt;</i>						8	9	
<i>&lt;if_body&gt;</i>						10	13	11
<i>&lt;else_body&gt;</i>						14	15	
<i>&lt;while_body&gt;</i>						16		
<i>&lt;par_list&gt;</i>								
<i>&lt;par_next&gt;</i>								
<i>&lt;arg_list&gt;</i>								
<i>&lt;arg_next&gt;</i>								
<i>&lt;arg_next2&gt;</i>								
<i>&lt;var_type&gt;</i>								
<i>&lt;print_exp&gt;</i>		33						
<i>&lt;command&gt;</i>	41							
<i>&lt;dim_end&gt;</i>		42						
<i>&lt;assignment&gt;</i>								

<i>ELSE</i>	<i>LOOP</i>	<i>ID</i>	<i>,</i>	<i>)</i>	<i>CONST</i>	<i>INTEGER</i>	<i>DOUBLE</i>	<i>STRING</i>	<i>EXP</i>	<i>INPUT</i>	<i>IF</i>	<i>DO</i>
12												
	17											
		19		18								
			20	21								
		23		22	24							
			26	25								
		27			28							
						29	30	31				
									32			
		36								34	35	37
		45							44			



### 3.3 Tabuľka symbolov

Tabuľku symbolov sme implementovali podľa zadania čiže pomocou hashovacej funkcie. Behom implementácie projektu sme mnohokrát zmenili rozhodnutie ohľadne toho čo všetko bude tabuľka symbolov obsahovať a aké pomocné funkcie budú s ňou pracovať. Do tabuľky symbolov ukladáme potrebné informácie. Pomocné funkcie pracujúce s tabuľkou symbolov využíva syntaktická analýza aj sémantická analýza.

### 3.4 Generovanie 3-adresného kódu

Pre generovanie 3-adresného kódu sme využili metódu, pri ktorej ho priamo syntaktický analyzátor vygeneruje. Z toho dôvodu sme vytvorili množstvo pomocných funkcií, ktoré si syntaktický analyzátor volá.

### 3.5 Vstavané funkcie

V zadaní sme dostali za úlohu do prekladaču implementovať niekoľko vstavaných funkcií, ktoré sme implementovali podľa zadania. Implementovali sme nasledujúce funkcie:  
Len

- `Length(s As String) As Integer`
- `SubStr(s As String, i As Integer, n As Integer) As String`
- `Asc(s As String, i As Integer) As Integer`
- `Chr(i As Integer) As String`

## 4 Testovanie

V počiatočných fázach implementácie sme si testovanie riešili každý sám, ale po prvom pokusnom odovzdaní sme sa rozhodli vytvoriť sadu 50 testovacích vstupov, na ktorých sme automatizovane testovali náš projekt. Vzhľadom k tomu, že podmienka využitia druhého pokusného odovzdania, bolo mať aspoň 50 testovacích vstupov tak sme aj toto pokusné odovzdanie využili. Po druhom pokusnom odovzdaní sme vytvorili sadu 128 testovacích vstupov, s pomocou ktorých sme doladľovali náš projekt.

## **5 Zdroje**

- Prednášky IFJ
- Democvičenia IFJ
- Študijná opora – Formálne jazyky a prekladače
- Študijná opora – Algoritmy