

Projektová dokumentace  
**Implementace překladače imperativního jazyka IFJ 18**  
Tým 123  
Varianta I

<b>Martina Tučková</b>	<b>(xtucko00)</b>	<b>25 %</b>
Martina Jendrálová	(xjendr03)	25 %
Marek Šťastný	(xstast33)	25%
Martin Janda	(xjanda27)	25 %

# Obsah

<b>1 Úvod a vysvětlení k jazyku IFJ 18</b>	<b>3</b>
<b>2 Implementace a části našeho řešení</b>	<b>3</b>
2.1 – Lexikální analyzátor neboli scanner	
2.2 – Syntaktická a sémantická analýza	
<b>3 Použité algoritmy a speciální datové struktury</b>	
3.1 – Binární strom, implementován rekurzí	4
3.2 – Dynamické řetězce	4
<b>4 Práce v týmu a komunikace</b>	<b>4</b>
<b>5 Rozdělení</b>	<b>5</b>
<b>6 Konečný automat lexikální analýzy</b>	<b>6</b>
<b>7 LL gramatika</b>	<b>7</b>
<b>8 LL tabulka</b>	<b>8</b>
<b>9 Precedenční tabulka</b>	<b>9</b>
<b>10 Závěr</b>	<b>10</b>

# 1 Úvod a zadání našeho projektu

Tato dokumentace slouží k popisu implementace překladače imperativního jazyka IFJ 18. Naším cílem bylo v jazyce C napsat program, jež načítá zdrojový kód. Tento zdrojový kód je zadán v jazyce IFJ 18, jež je zjednodušenou podmnožinou jazyka Ruby 2.0 a přeloží jej do výsledného jazyka IFJcode 18. Program dále vrací návratovou hodnotu dle situace, kde 0 značí, že překlad proběhl v pořádku a vrací jinou návratovou hodnotu v případě některé chyby.

## 2 Implementace a návrh našeho projektu

Náš projekt je sestaven z většího počtu částí, jejich jednotlivá implementace je popsána v následujících podkapitolách.

### 2.1 Lexikální analýza

Jako jednu z prvních částí jsme začali pracovat na lexikálním analyzátoru neboli scanneru. Jeho hlavním úkolem je načítat ze vstupu posloupnosti příchozích znaků (lexémů) a dále je zpracovávat. Takto zpracované lexémy, dále tokeny, jsou v další části předávány syntaktické analýze. Mezi tokeny, jež rozlišujeme patří EOL, EOF, identifikátory, přirovnávací a matematické operátory, desetinné a celé číslo, klíčová slova a další znaky, patřící do části letošního projektu IFJ 18. Hlavní funkcí scanneru v našem případě je funkce getToken.

Analyzátor byl implementován na základě modelu konečného automatu, jež je graficky přidán na konec této dokumentace.

### 2.2 Syntaktická a sémantická analýza

Syntaktický analyzátor, neboli parser, je nejdůležitější částí celého programu. Spouští funkce scanneru a načítá posloupnost příchozích tokenů, jež si dál zpracovává a případně žádá scanner o další token. Takovýto syntaktický analyzátor se řídí LL – gramatikou zakreslené v LL tabulce a v našem případě je použita metoda procházení rekurzivním sestupem seshora dolů. LL gramatika i LL tabulky jsou dále také přiloženy v grafickém zpracování.

Společně se syntaktickou analýzou se provádí i sémantická analýza. Ta kontroluje přijaté symboly ze syntaktické analýzy a přiřazuje jim správný význam. Sémantická kontrola také hlídá, aby nedošlo k implicitnímu deklarování již vytvořené proměnné a pracuje s precedenční gramatikou a precedenční tabulkou, kde jsou jednotlivé symboly znázorněny dle jejich priorit. Se sémantickou analýzou je úzce spjatá podoba ukládání právě oněch přijatých symbolů. To v našem případě bylo implementováno rekurzivní metodou binárního stromu. Ona precedenční tabulka je také dále přiložena v grafické podobě.

### 3 Použité algoritmy a speciální struktury

Během projektu jsme použili i techniky, jež jsme se naučili v předmětu IAL – algoritmy. Například tabulku symbolů jsme mohli implementovat pomocí jednoho ze dvou možných algoritmů. V našem případě na základě zvoleného zadání to bylo metodou binárního stromu.

#### 3.1 Tabulka symbolů – Binary tree

Tabulku symbolů jsme implementovali v souboru symtable. My provedli její implementaci metodou binárního stromu. Binární strom je datová struktura pro ukládání a vyhledávání dat. V našem případě obsahuje řadu funkcí pro rychlejší vyhledávání a práci s hodnotami.

#### 3.2 Dynamické řetězce

Pracovali jsme na souboru strings.c , jež slouží pro práci s řetězci dynamické délky. Nemůžeme s jistotou vědět, jak dlouhý řetězec můžeme očekávat, proto tento pomocný soubor alokuje případné místo pro příchozí řetězce, překopírovává jejich hodnoty a po provedené práci alokované místo opět uvolňuje pro pozdější využití.

### 4 Práce v týmu a komunikace

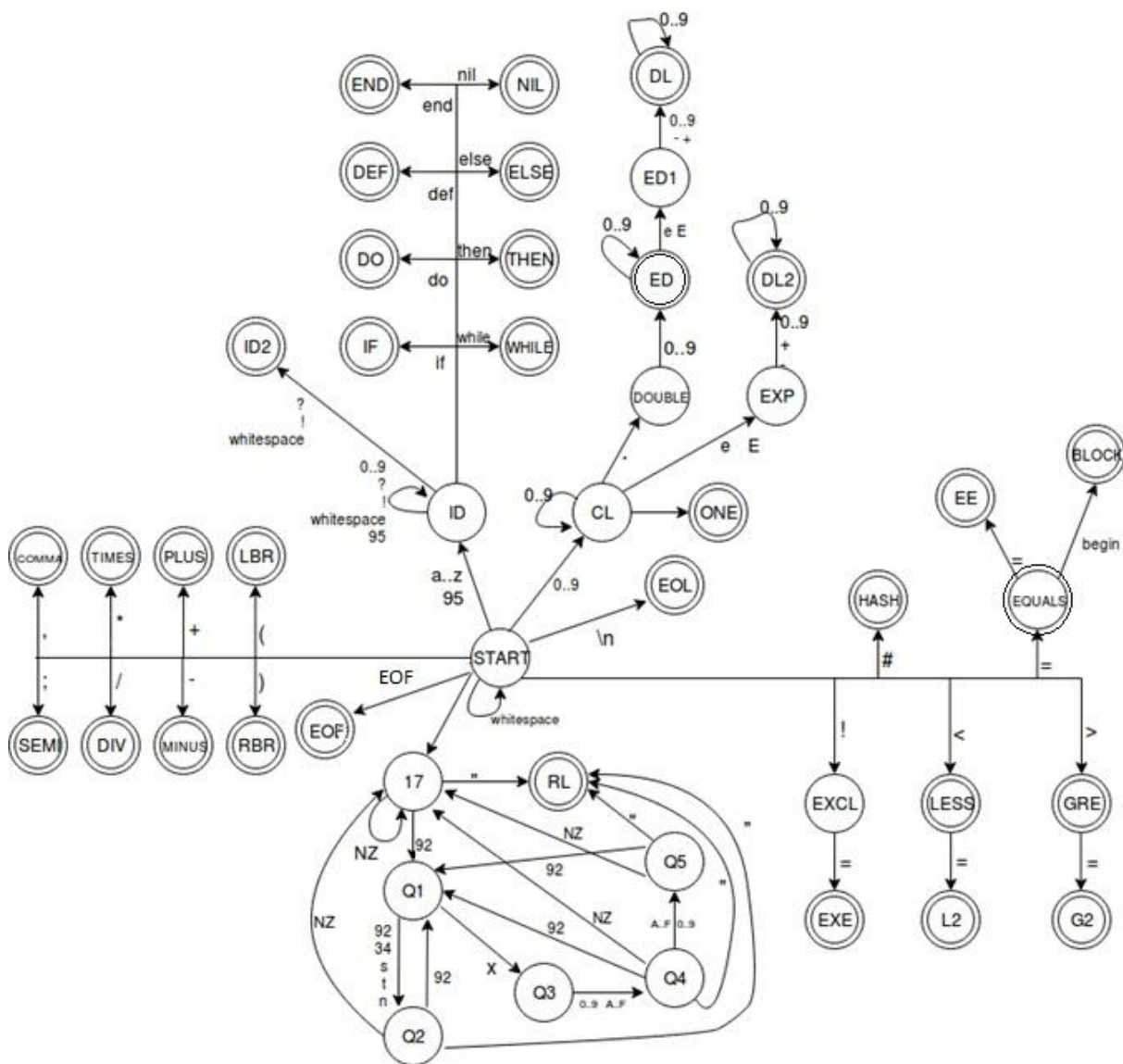
Při první nepracovní schůzce jsme stanovili vedoucího týmu a na další schůzce jsme již začali řešit pracovní rozdělení. Nakonec rozdělení jednotlivých částí probíhalo v průběhu práce na základě toho, co bylo potřeba co nejdříve vypracovat a na každé části pracovala buď dvojice, nebo jednotlivci.

Jako komunikační kanál jsme ze začátku a i později využívali vlastní skupinu na sociální síti facebook, pro sdílení souborů a rychlou spolupráci jsme vytvořili privátní repositář na stránce Github, kde probíhalo sdílení a nahrávání souborů. V průběhu semestru probíhali i déle trvající schůzky, hlavně s blížícím se ukončením projektu za účelem ladění a testování.

## 5 Rozdělení

Martina Tučková	Lexikální analyzátor
Martina Jendrálová	Lexikální analyzátor
Marek Šťastný	Syntaktický analyzátor, expressions.c
Martin Janda	Vedlejší soubory, dokumentace

## 6. Konečný automat lexikálního analyzátoru



## 7. LL Gramatika

1.  $\langle \text{prog} \rangle \rightarrow \langle \text{stat\_list} \rangle \text{ EOF}$
2.  $\langle \text{stat\_list} \rangle \rightarrow \langle \text{stat} \rangle \text{ EOL } \langle \text{stat\_list} \rangle$
3.  $\langle \text{stat\_list} \rangle \rightarrow \epsilon$
4.  $\langle \text{stat} \rangle \rightarrow \epsilon$
5.  $\langle \text{stat} \rangle \rightarrow \text{def id } ( \langle \text{param\_list} \rangle ) \text{ EOL } \langle \text{stat\_list} \rangle \text{ end}$
6.  $\langle \text{param\_list} \rangle \rightarrow \epsilon$
7.  $\langle \text{param\_list} \rangle \rightarrow \text{id } \langle \text{part} \rangle$
8.  $\langle \text{part} \rangle \rightarrow \text{id } \langle \text{part} \rangle$
9.  $\langle \text{part} \rangle \rightarrow \epsilon$
10.  $\langle \text{stat} \rangle \rightarrow \text{exp}$
11.  $\langle \text{stat} \rangle \rightarrow \text{if exp then EOL } \langle \text{stat\_list} \rangle \text{ else EOL } \langle \text{stat\_list} \rangle \text{ end}$
12.  $\langle \text{stat} \rangle \rightarrow \text{while exp do EOL } \langle \text{stat\_list} \rangle \text{ end}$
13.  $\langle \text{stat} \rangle \rightarrow \text{id } \langle \text{assignment} \rangle$
14.  $\langle \text{assignment} \rangle \rightarrow \epsilon$
15.  $\langle \text{assignment} \rangle \rightarrow = \langle \text{assigned} \rangle$
16.  $\langle \text{assigned} \rangle \rightarrow \text{exp}$
17.  $\langle \text{assigned} \rangle \rightarrow \langle \text{f\_call} \rangle$
18.  $\langle \text{f\_call} \rangle \rightarrow \text{id } \langle \text{param\_group} \rangle$
19.  $\langle \text{param\_group} \rangle \rightarrow \langle \text{term\_list} \rangle$
20.  $\langle \text{param\_group} \rangle \rightarrow ( \langle \text{term\_list} \rangle )$
21.  $\langle \text{term\_list} \rangle \rightarrow \epsilon$
22.  $\langle \text{term\_list} \rangle \rightarrow \langle \text{term} \rangle \langle \text{term\_part} \rangle$
23.  $\langle \text{term\_part} \rangle \rightarrow \langle \text{term} \rangle \langle \text{term\_part} \rangle$
24.  $\langle \text{term\_part} \rangle \rightarrow \epsilon$
25.  $\langle \text{term} \rangle \rightarrow \text{id}$
26.  $\langle \text{term} \rangle \rightarrow \text{int}$
27.  $\langle \text{term} \rangle \rightarrow \text{float}$
28.  $\langle \text{term} \rangle \rightarrow \text{string}$
29.  $\langle \text{term} \rangle \rightarrow \text{nil}$

## 8. LL Tabulka

	def	id	if	while	else	end	=	(	)	,	EOL	EOF	int	float	string	nil	exp
<prog>	1	1	1	1							1	1					1
<stat_list>	2	2	2	2	3	3					2	3					2
<stat>	5	13	11	12							4						10
<f_call>		18															
<param_group>		19						20			19		19	19	19	19	
<param_list>		7							6								
<part>									9	8							
<assigned>		17															16
<assignment>							15				14						
<term_list>		22							21		21		22	22	22	22	
<term_part>									24	23	24						
<term>		25											26	27	28	29	



## 9. Precedenční tabulka

	+	-	*	/	<	>	<=	>=	==	!=	(	)	i	\$
+	>	>	<	<	>	>	<	<	<	<	<	>	<	>
-	>	>	<	<	>	>	<	<	<	<	<	>	<	>
*	>	>	>	>	>	>	>	>	>	>	<	>	<	>
/	>	>	>	>	>	>	>	>	>	>	<	>	<	>
<	<	<	<	<							<	>	<	>
>	<	<	<	<							<	>	<	>
<=	<	<	<	<							<	>	<	>
>=	<	<	<	<							<	>	<	>
==	<	<	<	<							<	>	<	>
!=	<	<	<	<							<	>	<	>
(	<	<	<	<	<	<	<	<	<	<	<	=	<	
)	>	>	>	>	>	>	>	>	>	>		>		>
i	>	>	>	>	>	>	>	>	>	>		>		>
\$	<	<	<	<	<	<	<	<	<	<	<		<	

## 10. Závěr

Projekt byl sice značně obtížnější než většina doposud vytvářených projektů, ale to především z důvodu spousty neznámých a nových věcí. Byl to první větší týmový projekt vyžadující vzájemnou komunikaci a spolupráci. Byť to byl projekt náročný a budeme doufat že úspěšný, obdrželi jsme spoustu zkušeností s týmovou spoluprací a vytvářením větších souborů.