



# Dokumentace

Implementace překladače imperativního jazyka IFJ23

Varianta - vv-BVS

Podporovaná rozšíření: FUNEXP

Tým xeffen00

**vedoucí** Marek Effenberger (xeffen00) 25%

Samuel Hejníček (xhejni00) 25%

Dominik Horut (xhorut01) 25%

Adam Valík (xvalik05) 25%

Brno, 5. prosince 2023

# Obsah

<b>1</b>	<b>Práce v týmu</b>	<b>2</b>
1.1	Rozdělení práce mezi jednotlivé členy týmu . . . . .	2
1.2	Organizace práce . . . . .	3
<b>2</b>	<b>Implementace překladače</b>	<b>3</b>
2.1	Tabulka symbolů . . . . .	3
2.2	Datové typy . . . . .	3
2.2.1	token_t . . . . .	3
2.2.2	forest_node . . . . .	3
2.2.3	vector . . . . .	3
2.3	Datové struktury . . . . .	3
2.3.1	Fronta . . . . .	3
2.3.2	Zásobník tokenů . . . . .	3
2.3.3	Zásobník cnt . . . . .	3
2.3.4	AVL strom . . . . .	4
2.3.5	Obousměrně vázaný seznam . . . . .	4
2.4	Lexikální analýza . . . . .	4
2.4.1	Implementace lexikálního analyzátoru . . . . .	4
2.4.2	Diagram konečného automatu . . . . .	4
2.4.3	Legenda diagramu konečného automatu . . . . .	5
2.5	Syntaktická a sémantická analýza . . . . .	6
2.5.1	Parser . . . . .	6
2.5.2	LL gramatika . . . . .	6
2.5.3	LL tabulka . . . . .	8
2.5.4	Expression parser . . . . .	8
2.5.5	Precedenční tabulka . . . . .	8
2.5.6	Gramatika výrazů . . . . .	8
2.6	Generování kódu . . . . .	9

# 1 Práce v týmu

## 1.1 Rozdělení práce mezi jednotlivé členy týmu

### **Marek Effenberger**

- Návrh automatu pro lexikální analýzu
- Návrh LL gramatiky
- Implementace tabulky symbolů
- Implementace syntaktického analyzátoru
- Generování cílového kódu

### **Adam Valík**

- Návrh automatu pro lexikální analýzu
- Návrh LL gramatiky
- Implementace tabulky symbolů
- Implementace syntaktického analyzátoru
- Generování cílového kódu

### **Dominik Horut**

- Implementace lexikálního analyzátoru
- Návrh gramatiky výrazů a precedenční tabulky
- Implementace syntaktického analyzátoru výrazu
- Generování cílového kódu výrazů

### **Samuel Hejníček**

- Návrh automatu pro lexikální analýzu
- Implementace lexikálního analyzátoru
- Návrh gramatiky výrazů a precedenční tabulky
- Implementace syntaktického analyzátoru výrazu
- Generování cílového kódu výrazů

## 1.2 Organizace práce

Před samotným začtkem vývoje projektu se tým rozdělil na dvě skupiny po dvou členech - první se zaměřila na tvorbu parseru, druhá nejprve implementovala lexikální analyzátor, a poté analyzátor výrazů. Obě větve vývoje se následně spojily a celý tým pak pracoval na generaci cílového kódu. Při vývoji byl použit verzovací systém Git a komunikace probíhala skrze platformu Discord a osobní setkání členů týmu.

## 2 Implementace překladače

### 2.1 Tabulka symbolů

### 2.2 Datové typy

#### 2.2.1 `token_t`

Datový typ `token_t` reprezentuje token načtený lexikálním analyzátozem a je implementován v souboru `scanner.h`. Obsahuje datový typ tokenu a jeho hodnotu.

#### 2.2.2 `forest_node`

*forest\_node* představuje uzel v AVL stromu, je implemetován v souboru `forest.h`. Obsahuje ukazatel na svého rodiče a také na své potomky, současně uchovává i informaci o počtu svých potomků.

#### 2.2.3 `vector`

*vector* reprezentuje dynamický řetězec a je implementován v souboru `string_vector.h`, funkce, které s tímto datovým typem pracují jsou pak implementovány v souboru `string_vector.c`. Nejvíce je tento typ používán v lexikálním analyzátoru při načítání jmen identifikátorů, klíčových slov a řetězců.

### 2.3 Datové struktury

#### 2.3.1 Fronta

Fronta je implementována v souboru `queue.c`. PROC A NA CO SE POUZIVA

#### 2.3.2 Zásobník tokenů

Používá se při vyhodnocování výrazů pomocí precedenční tabulky. Je implementován v souboru `token_stack.c` a kromě klasických operací se zásobníkem jako jsou *push()*, *pop()* nebo *top()*, obsahuje také funkci *stack\_top\_terminal()*, která navrací nejvrchnější terminál na zásobníku.

#### 2.3.3 Zásobník cnt

Zásobník je implementován v souboru `cnt_stack.c`. PROC A NA CO SE POUZIVA

### 2.3.4 AVL strom

Slouží k lepší reprezentaci hierarchie mezi uzly tabulky symbolů a abstraktním syntaktickým stromem. Implementován je v souboru `forest.c`. CO JINEHO

### 2.3.5 Obousměrně vázaný seznam

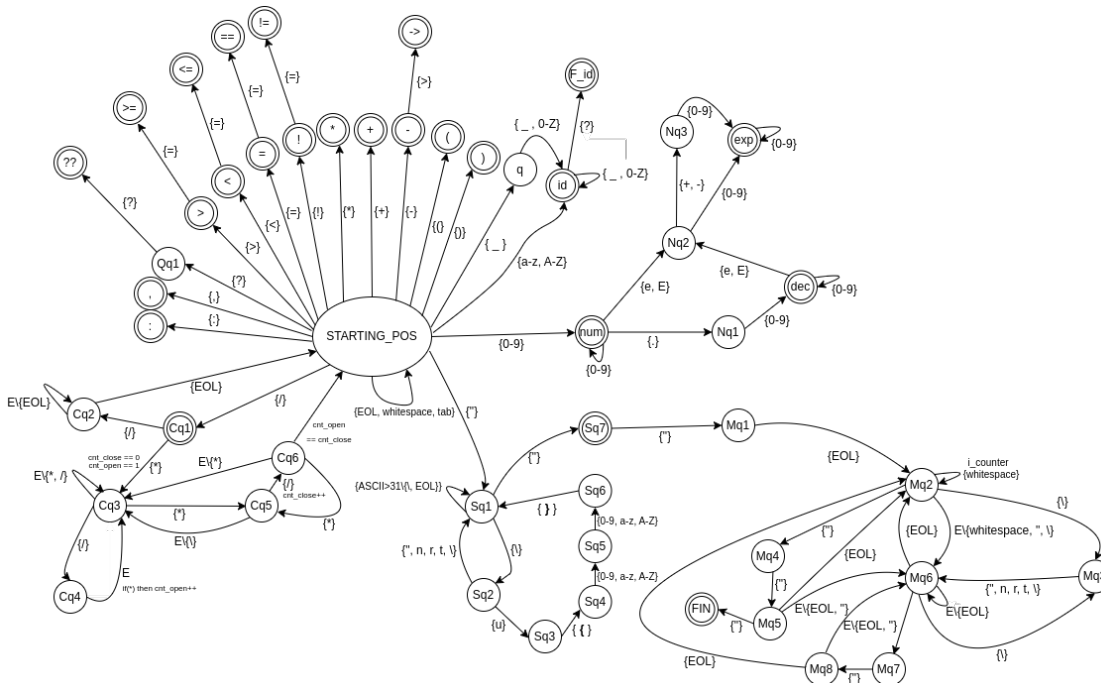
Používá se pro správné generování cílového kódu. Je implementován v souboru `codegen.c`

## 2.4 Lexikální analýza


































### 2.4.1 Implementace lexikálního analyzátoru

Lexikální analyzátor je implementován v souboru `scanner.c` a sestává ze tří pomocných funkcí a hlavní funkce `get_me_token()`, která implementuje jeho vnitřní logiku. Pomocí cyklu `while` v ní dochází k načítání znaků ze standardního vstupu a na základě tohoto vstupu k přechodu do odpovídajícího stavu. Cyklus se opakuje do okamžiku, kdy se automat dostane do koncového stavu. V takovémto případě je navracena hodnota typu `token_t*`, která obsahuje důležité atributy získaného tokenu. U identifikátorů se poté ještě zjišťuje, zda se nejedná o klíčové slovo. V případě, že sekvence znaků nevede ke koncovému stavu, dojde k lexikální chybě. Komentáře, stejně jako bílé znaky, jsou ignorovány. U vnořených komentářů je pak navíc kontrolován odpovídající počet otevírajících (`/*`) a uzavírajících (`*/`) znaků.

### 2.4.2 Diagram konečného automatu



### 2.4.3 Legenda diagramu konečného automatu

 S_COLON	 S_LESS_EQ	 S_MINUS
 S_COMMA	 S_EQ	 S_RET_TYPE
 S_QM	 S_EQEQ	 S_LPAR
 S_DOUBLE_QM	 S_EXCLAM	 S_RPAR
 S_GTR	 S_EXCLAMEQ	 S_UNDERSCORE
 S_GTR_EQ	 S_MULTIPLY	 S_ID
 S_LESS	 S_PLUS	 S_ID_QM
 S_NUM	 S_DIVIDE	
 S_NUM_DOT	 S_SL_COM	
 S_DEC	 S_NESTED_COM	
 S_NUM_E	 S_SLASH	
 S_NUM_E_SIGN	 S_NESTED_CLOSE	
 S_EXP	 S_NESTED_END	

$\textcircled{\text{Sq1}}$ S_START_QUOTES	$\textcircled{\text{Mq1}}$ S_THREE_QUOTES
$\textcircled{\text{Sq2}}$ S_START_ESC_SEQUENCE	$\textcircled{\text{Mq2}}$ S_START_MULTILINE
$\textcircled{\text{Sq3}}$ S_START_HEX	$\textcircled{\text{M3}}$ S_MULTI_ESC_SEQUENCE
$\textcircled{\text{Sq4}}$ S_FIRST_HEX	$\textcircled{\text{Mq4}}$ S_ML_END_START
$\textcircled{\text{Sq5}}$ S_SECOND_HEX	$\textcircled{\text{Mq5}}$ S_PRE_END_MULTILINE
$\textcircled{\text{Sq6}}$ S_END_HEX	$\textcircled{\text{Mq6}}$ S_IS_MULTILINE
$\textcircled{\text{Sq7}}$ S_STR_EMPTY	$\textcircled{\text{Mq7}}$ S_ML_EOL_END
$\textcircled{\text{FIN}}$ S_END_MULTILINE	$\textcircled{\text{Mq8}}$ S_FAKE_END_MULTILINE

## 2.5 Syntaktická a sémantická analýza

### 2.5.1 Parser

#### 2.5.2 LL gramatika

1.  $\langle prog \rangle \rightarrow EOF$
2.  $\langle prog \rangle \rightarrow \langle func\_def \rangle \langle prog \rangle$
3.  $\langle prog \rangle \rightarrow \langle body \rangle \langle prog \rangle$
4.  $\langle func\_def \rangle \rightarrow func\ id\ ( \langle params \rangle ) \langle ret\_type \rangle \{ \langle local\_body \rangle \}$
5.  $\langle params \rangle \rightarrow \langle par\_name \rangle \langle par\_id \rangle : \langle type \rangle \langle params\_n \rangle$
6.  $\langle params \rangle \rightarrow \varepsilon$
7.  $\langle par\_name \rangle \rightarrow \_$
8.  $\langle par\_name \rangle \rightarrow id$
9.  $\langle par\_id \rangle \rightarrow \_$
10.  $\langle par\_id \rangle \rightarrow id$
11.  $\langle type \rangle \rightarrow Int$
12.  $\langle type \rangle \rightarrow Int?$
13.  $\langle type \rangle \rightarrow Double$
14.  $\langle type \rangle \rightarrow Double?$

15.  $\langle type \rangle \rightarrow String$
16.  $\langle type \rangle \rightarrow String?$
17.  $\langle params\_n \rangle \rightarrow, \langle params \rangle$
18.  $\langle params\_n \rangle \rightarrow \varepsilon$
19.  $\langle ret\_type \rangle \rightarrow \rightarrow \langle type \rangle$
20.  $\langle ret\_type \rangle \rightarrow \varepsilon$
21.  $\langle local\_body \rangle \rightarrow \langle body \rangle \langle local\_body \rangle$
22.  $\langle local\_body \rangle \rightarrow \varepsilon$
23.  $\langle body \rangle \rightarrow \langle var\_def \rangle$
24.  $\langle body \rangle \rightarrow \langle condition \rangle$
25.  $\langle body \rangle \rightarrow \langle cycle \rangle$
26.  $\langle body \rangle \rightarrow \langle assign \rangle$
27.  $\langle body \rangle \rightarrow \langle func\_call \rangle$
28.  $\langle body \rangle \rightarrow \langle ret \rangle$
29.  $\langle body \rangle \rightarrow \varepsilon$
30.  $\langle ret \rangle \rightarrow return \langle exp \rangle$
31.  $\langle ret \rangle \rightarrow return$
32.  $\langle var\_def \rangle \rightarrow let\ id \langle opt\_var\_def \rangle$
33.  $\langle var\_def \rangle \rightarrow var\ id \langle opt\_var\_def \rangle$
34.  $\langle opt\_var\_def \rangle \rightarrow: \langle type \rangle$
35.  $\langle opt\_var\_def \rangle \rightarrow \langle assign \rangle$
36.  $\langle opt\_var\_def \rangle \rightarrow: \langle type \rangle \langle assign \rangle$
37.  $\langle assign \rangle \rightarrow = \langle exp \rangle$
38.  $\langle assign \rangle \rightarrow = \langle func\_call \rangle$
39.  $\langle func\_call \rangle \rightarrow id(\langle args \rangle)$
40.  $\langle args \rangle \rightarrow \varepsilon$
41.  $\langle arg \rangle \rightarrow \langle exp \rangle$
42.  $\langle arg \rangle \rightarrow id : \langle exp \rangle$



43.  $\langle args\_n \rangle \rightarrow, \langle arg \rangle \langle args\_n \rangle$
44.  $\langle args\_n \rangle \rightarrow \varepsilon$
45.  $\langle condition \rangle \rightarrow if \langle exp \rangle \{ \langle local\_body \rangle \} else \{ \langle local\_body \rangle \}$
46.  $\langle condition \rangle \rightarrow if let id \{ \langle local\_body \rangle \} else \{ \langle local\_body \rangle \}$
47.  $\langle cycle \rangle \rightarrow while \langle exp \rangle \{ \langle local\_body \rangle \}$

Pozn. "*exp*" je označení pro výraz

### 2.5.3 LL tabulka

#### 2.5.4 Expression parser

Je implementován v souboru `expression_parser.c` a je volán parserem skrze funkci `call_expr_parser()` pokaždé, když je potřeba vyhodnotit výraz. Vyhodnocení pak probíhá pomocí precedenční syntaktické analýzy. K tomu využívá zásobník tokenů, kde ihned po inicializaci vloží ukončovač výrazu `$` a pomocí precedenční tabulky a gramatiky výrazů celý výraz vyhodnocuje, dokud na zásobníku nezůstane pouze ukončovač výrazu `$` a výsledek výrazu. V rámci sémantické analýzy je výsledek podroben kontrole, zda odpovídá očekávanému datovému typu, který byl předán pomocí parametru funkce. Během vyhodnocování výrazu navíc probíhá sémantická kontrola typů pomocí funkce `check_types()`.

#### 2.5.5 Precedenční tabulka

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

#### 2.5.6 Gramatika výrazů

1.  $E \rightarrow i$
2.  $E \rightarrow (E)$

3.  $E \rightarrow E + E$
4.  $E \rightarrow E - E$
5.  $E \rightarrow E * E$
6.  $E \rightarrow E / E$
7.  $E \rightarrow E < E$
8.  $E \rightarrow E > E$
9.  $E \rightarrow E <= E$
10.  $E \rightarrow E >= E$
11.  $E \rightarrow E == E$
12.  $E \rightarrow E! = E$
13.  $E \rightarrow E!$
14.  $E \rightarrow E??E$

## 2.6 Generování kódu