

Projektová dokumentace  
**Implementace překladače imperativního jazyka IFJ23**  
Tým xhrubo01, varianta TRP-izp

December 6, 2023

Dominik Borek	(xborek12)	25%
<b>Ondřej Hruboš</b>	<b>(xhrubo01)</b>	25%
Radek Jestřábík	(xjestr04)	25%
Ondřej Šatinský	(xsatin03)	25%

# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Návrh a Implementace</b>	<b>2</b>
2.1	Lexikální analýza . . . . .	2
2.1.1	Diagram konečného automatu . . . . .	3
2.2	Syntaktická analýza . . . . .	4
2.2.1	LL-gramatika . . . . .	4
2.2.2	LL-tabulka . . . . .	5
2.3	Sémantická analýza . . . . .	5
2.4	Generátor vnitřního kódu . . . . .	5
2.5	Generátor cílového kódu . . . . .	5
2.6	odchylky od přednášené látky . . . . .	5
2.7	Popis struktur . . . . .	6
2.7.1	AST . . . . .	6
2.7.2	Expression . . . . .	6
2.7.3	Statement . . . . .	6
2.7.4	VarTable . . . . .	6
2.7.5	VarTableStack . . . . .	6
2.7.6	FuncTable . . . . .	6
<b>3</b>	<b>Práce v týmu</b>	<b>7</b>
3.1	Způsob práce v týmu . . . . .	7
3.2	Rozdělení práce . . . . .	7

# 1 Úvod

Cílem projektu do předmětu formální jazyky a překladače (ifj) bylo vytvoření programu v jazyce C, který načte zdrojový kód zapsaný ve zdrojovém jazyce IFJ23 a přeloží jej do cílového jazyka IFJcode23. Jazyk IFJ23 je zjednodušenou podmnožinou jazyka Swift.

Bude se jednat o konzolovou aplikaci. Překladač bude načítat řídicí program ze standardního vstupu a generovat výsledný kód IFJcode23 na standardní výstup. V případě chyby se chybové hlášení vypíše na standardní výstup.

## 2 Návrh a Implementace

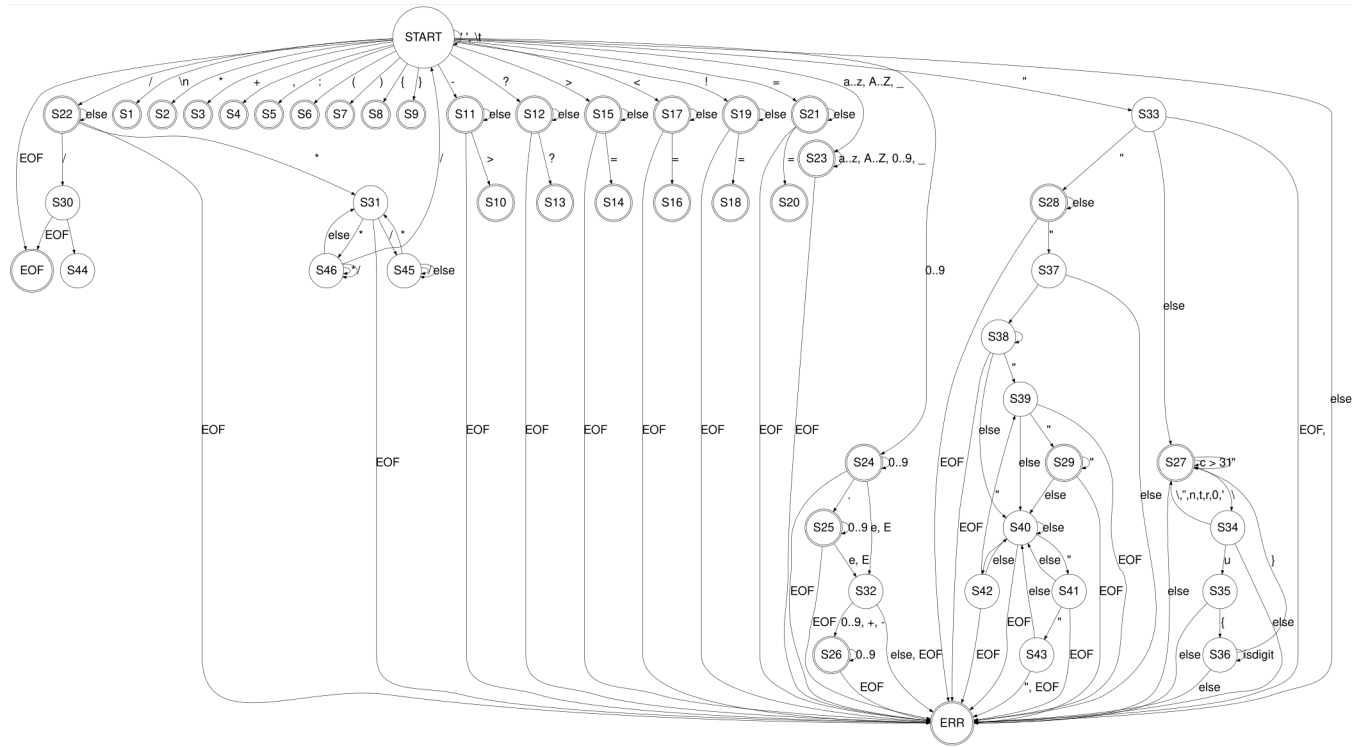
Implementace je provedena pomocí jednotlivých, po sobě jdoucích částí, které jsou detailně popsány v následujících podkapitolách. Mezi tyto části patří: Lexikální analýza, Syntaktická analýza, Sémantická analýza, Generátor vnitřního kódu a Generátor cílového kódu.

### 2.1 Lexikální analýza

Při tvorbě projektu se začalo s lexikálním analyzátozem. Hlavní úkol této analýzy je čtení jednotlivých znaků ze zdrojového souboru a následně se převádí na strukturu tokenů. Typy tokenů jsou konec souboru, základní operace, různé typy závorek, porovnávací operátory, číslo, desetinné číslo, řetězec, nebo identifikátor.

Lexikální analyzátor funguje jako jeden veliký deterministický konečný automat, který je k zobrazení v následující sekci. Tento automat je implementován jako přepínač, ve kterém jsou jednotlivé případy pro jeden stav automatu s možností přejít do dalšího stavu, pokud přijde na tento stav další znak, který mění podstatu tohoto tokenu. Přecházení do stavů bude probíhat, dokud se nenalezne znak EOF. Tyto tokeny se následně po zpracování vloží do tabulky symbolů, se kterou následně pracuje syntaktický analyzátor.

## 2.1.1 Diagram konečného automatu



### Legenda stavů:

STATE.START = START  
 ERR = ERR  
 NEWLINE = S1  
 TOKENTYPE.STAR = S2  
 TOKENTYPE.PLUS = S3  
 TOKENTYPE.COMMA = S4  
 TOKENTYPE.COLON = S5  
 TOKENTYPE.PAR.L = S6  
 TOKENTYPE.PAR.R = S7  
 TOKENTYPE.BRACE.L = S8  
 TOKENTYPE.BRACE.R = S9  
 TOKENTYPE.ARROW = S10  
 STATE.MINUS = S11  
 STATE.QUESTION = S12  
 TOKENTYPE.QUESTIONMARK2 = S13  
 TOKENTYPE.GREATER\_OR\_EQUAL = S14  
 STATE.GREATER = S15  
 TOKENTYPE.LESSER\_OR\_EQUAL = S16  
 STATE.LESSER = S17  
 TOKENTYPE.NOT\_EQUALS = S18  
 STATE.EXCLAMATION = S19  
 TOKENTYPE.EQUALS2 = S20  
 STATE.EQUALS = S21  
 STATE.SLASH = S22  
 STATE.IDENTIF = S23

STATE.NUMBER = S24  
 STATE.NUMBER.DOT = S25  
 STATE.NUMBER.EXPONENT = S26  
 STATE.STRING.BODY = S27  
 STATE.STRING.EMPTY = S28  
 STATE.STRING.3.BODY.QUOTE2 = S29  
 STATE.COMMENT.LINE = S30  
 STATE.COMMENT.BLOCK = S31  
 STATE.NUMBER.E = S32  
 STATE.STRING = S33  
 STATE.STRING.BODY.SLASH = S34  
 STATE.STRING.BODY.U = S35  
 STATE.STRING.BODY.U.B = S36  
 STATE.STRING.3 = S37  
 STATE.STRING.3.BODY.NEWLINE.0 = S38  
 STATE.STRING.3.BODY.QUOTE = S39  
 STATE.STRING.3.BODY = S40  
 STATE.STRING.3.BODY.INV.QUOTE = S41  
 STATE.STRING.3.BODY.NEWLINE = S42  
 STATE.STRING.3.BODY.INV.QUOTE2 = S43  
 TOKENTYPE.NEWLINE = S44  
 STATE.COMMENT.BLOCK.SLASH = S45  
 STATE.COMMENT.BLOCK.STAR = S46

## 2.2 Syntaktická analýza

Syntaktická analýza je nejdůležitější a nejobsáhlejší část celého programu. Sémantická analýza se řídí LL-gramatikou a metodou rekurzivního sestupu pomocí pravidel, které jsou sepsány v LL-tabulce. Token, který přijde do syntaktické analýzy se zpracuje pomocí pravidel definovaných v LL-tabulce. Syntaktická analýza během své práce vytváří simulaci derivačního stromu, kterou předává sémantické analýze.

### 2.2.1 LL-gramatika

0.	epsilon pravidlo	
1.	<exp> ->	<exp1> <exp'>
2.	<exp> ->	<exp1> <exp'>
3.	<exp'> ->	?? <exp>
4.	<exp1> ->	<exp2> <exp1'>
5.	<exp1'> ->	\{==, !=, <, >, <=, >=\} <exp1>
6.	<exp2> ->	<exp3> <exp2'>
7.	<exp2'> ->	\{+, -\} <exp2>
8.	<exp3> ->	<exp4> <exp3'>
9.	<exp3'> ->	\{*, /\} <exp3>
10.	<exp4> ->	<exp5> <exp4'>
11.	<exp4'> ->	!
12.	<exp5> ->	(<exp>), t_int, t_string, t_double, id <args>
13.	<args> ->	( <args_list> )
14.	<args> ->	eps
15.	<arg_list> ->	<e_id> <exp> <arg_list_n>
16.	<arg_list> ->	eps
17.	<e_id> ->	id <exp_id>
18.	<e_id> ->	eps
19.	<arg_list_n> ->	, <arg_list>
20.	<arg_list_n> ->	eps
21.	<exp_id> ->	<exp'>
22.	<exp_id> ->	<exp6'> //CHECK: exp6' ?
23.	<exp_id> ->	<exp'>
24.	<exp_id> ->	( <arg_list> )
25.	<exp_id> ->	eps

#### First(x):

2.	<exp'>	??
4.	<exp1'>	==, !=, <, >, <=, >=
6.	<exp2'>	+, -
8.	<exp3'>	*, \
10.	<exp4'>	!
11.	<exp5>	(, t_int, t_string, t_double, id
12., 13.	<args>	(, eps
15.	<arg_list>	eps
16., 17.	<e_id>	id, eps
18., 19.	<arg_list_n>	', ', eps
23., 24.	<exp_id>	(, eps

### 2.2.2 LL-tabulka

Column1	??	==	!=	<	>	<=	>=	+	-	*	/	!	(	t_int	t_string	t_double	id	,	)	\$
exp														1	1	1	1	1		
exp'	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
exp1														3	3	3	3	3		0
exp1'	0	4	4	4	4	4	4	4	0	0	0	0	0	0	0	0	0	0	0	0
exp2														5	5	5	5	5		0
exp2'	0	0	0	0	0	0	0	0	6	6	0	0	0	0	0	0	0	0	0	0
exp3														7	7	7	7	7		0
exp3'	0	0	0	0	0	0	0	0	0	0	8	8	0	0	0	0	0	0	0	0
exp4														9	9	9	9	9		0
exp4'	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0
exp5														11	11	11	11	11		0
args														12						0
arg_list														11	11	11	11	11		0
e_id																			16	0
arg_list_n																			18	0
exp id														23						0

Table 1: LL – tabulka použitá při syntaktické analýze

### 2.3 Sémantická analýza

Úkol sémantické analýzy je kontrola typů, návratových hodnot, podmínek, cyklů, volání funkcí a kontrola, jestli jsou platné samotné výrazy.

Sémantická analýza je volána v během syntaktické analýzy. Získá výraz, který se zpracuje tak, že se zkontroluje jeho platnost a nastaví se datové typy. Následně se metoda vrací do syntaktické analýzy již sémanticky zkontrolovaný

### 2.4 Generátor vnitřního kódu

Generátor kódu vytváří mezikód, který přijímá AST. Výsledkem vnitřního kódu bude mezikód pro generátor cílového kódu, kterému se tento mezikód předá.

### 2.5 Generátor cílového kódu

Generátor cílového kódu přijímá z generátoru vnitřního kódu mezikód, který zpracuje a předělá se na cílové slovo, které bylo napsáno v jazyce IFJ23. Výsledný kód se generuje na logický výstup po projití veškerých analýz.

### 2.6 odchylky od přednášené látky

tabulka symbolů drží pouze symboly

propagace typů je změněná

sémantická analýza používá tabulku funkcí a zásobník tabulek proměnných

## **2.7 Popis struktur**

### **2.7.1 AST**

### **2.7.2 Expression**

### **2.7.3 Statement**

### **2.7.4 VarTable**

Existuje jeden globální VarTable a následně se na VarTableStack přidávají VarTables jednotlivých rámců.

### **2.7.5 VarTableStack**

### **2.7.6 FuncTable**

## 3 Práce v týmu

### 3.1 Způsob práce v týmu

Na projektu se začalo pracovat na konci září, kdy jsme si rozdělili práce mezi jednotlivé členy týmu, vybrali verzovací systém, testovací nástroj. Přibližně do listopadu na jednotlivých částech se pracovalo převážně samostatně, podle počátečního rozdělení, kdy Radek Jestřábík měl za úkol syntaktickou analýzu, Ondřej Hruboš sémantickou analýzu, Dominik Borek generátor vnitřního a cílového kódu a Ondřej Šatinský lexikální analýzu. Na začátku listopadu jsme se začali pravidelně scházet a řešit kód, který byl do této fáze implementován. V tuto chvíli na jednotlivých částech pracovali většinou 2 členové týmu, případně celý tým, pokud byl problém a odstraňovaly se chyby v jednotlivých částech projektu.

### 3.2 Rozdělení práce

Člen týmu	Přidělená práce
Dominik Borek	sémantická analýza, psaní dokumentace, fungování LL gramatiky, testování
<b>Ondřej Hruboš</b>	rozbor výrazů, tvorba struktur, makefile, testování
Radek Jestřábík	syntaktická analýza, sémantická analýza, testování
Ondřej Šatinský	lexikální analýza, generátor kódu, fungování LL gramatiky, testování

Table 2: Rozdělení práce v týmu mezi jednotlivými členy