Projektová dokumentace

Implementace překladače imperativního jazyka IFJ23

Tým xhrubo01, varianta TRP-izp

Rozšíření: FUNEXP

December 6, 2023

Dominik Borek	(xborek12)	22%
Ondřej Hruboš	**(xhrubo01)**	22%
Radek Jestřabík	(xjestr04)	22%
Ondřej Šatinský	(xsatin03)	26%

Obsah

1 Úvod

Cílem projektu do předmětu formalní 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 řídící program ze standardního vstupu a generovat výsledný kód IFJcode23 na standardní výstup. V případě chyby se chybové hlášení vypíše na standardní výstup.

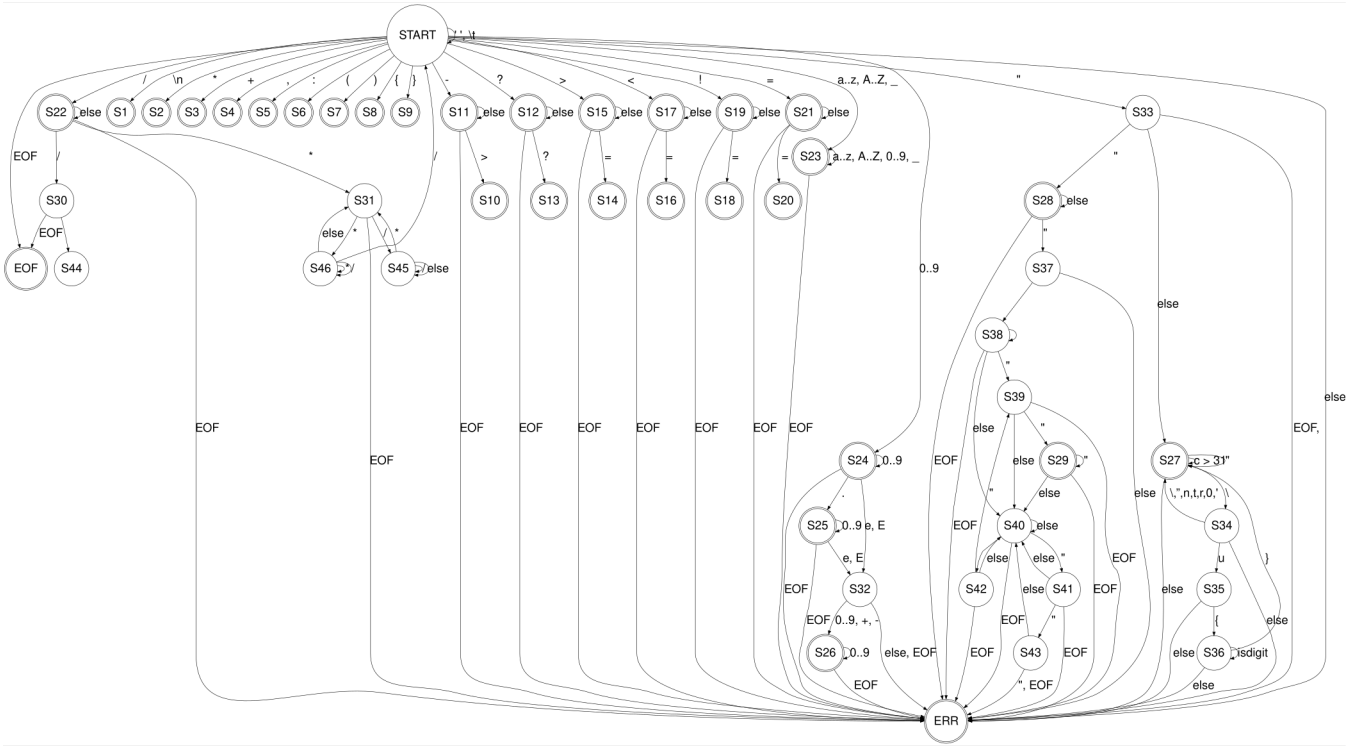
2 Návrh a Implementace

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

Lexikární analyzátor se nachází v souboru `scanner.c` Při tvorbě projektu se začalo s lexikálním analyzátorem. 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, porovnávající 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 se nachází v souboru `parse.c`. Tabulka pravidel je v souboru `rules.c`. 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 příjde 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

	int	string	double	id	nil	,	(	??	==	!=	>	>=	+	-	*	/	!	)	$	<=	:	<
exp	1	1	1	1	1	1	1															
exp_								2	0	0	0	0	0	0	0	0	0	0	0	0		0
exp1	3	3	3	3	3	3	3												0			
exp1_	0	0	0	0		0	0	0	4	4	4	4	0	0	0	0	0	0	0	4		0
exp2	5	5	5	5	5	5	5												0			
exp2_	0	0	0	0		0	0	0	0	0	0	0	6	6	0	0	0	0	0	0		
exp3	7	7	7	7	7	7	7												0			
exp3_	0	0	0	0		0	0	0	0	0	0	0	0	0	8	8	0	0	0	0		0
exp4	9	9	9	9	9	9	9												0			
exp4_	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	10	0	0	0		0
exp5	11	11	11	11		11												0	0			
args	0	0	0	0		0	12	0	0	0	0	0	0	0	0	0	0	0	0	0		0
args_list	14	14	14	14		14	14	14	14	14	14	14	14	14	14	14	14	0	14	14		
e_id	0	0	0	16		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	24	0
arg_list_n	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0
exp_id	0	0	0	20		0	0	20	0	0	0	0	20	20	20	20	20	0	0	0	0	0

Table 1: LL – tabulka použitá při syntaktické analýze

2.3 Sémantická analýza

Sémantická analýza se nachází v souboru `semantic.c` 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 se nachází v souboru `ir.c`
Generátor kódu vytváří mezikód, který přijímá AST. Výsledkem vnitřního kódu bude mezikód pro generátor cílového kódu, kterému se temto mezikód předá.

2.5 Generátor cílového kódu

Generátor kódu je implementován v souboru `code_generator.c`. 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

Je to struktura ukládající tabulku symbolů a seznam příkazů, které simulují strukturu programu.

2.7.2 Expression

Je struktura, které seskupuje všechny potřebné data pro vyhodnocování výrazů.

2.7.3 Statement

Simuluje dílčí příkaz.

2.7.4 VarTable

Existuje jedena globální tabulka proměnných a následně se na zásobník přidávají tabulky jednotlivých rámců.

2.7.5 VarTableStack

Je zásobník tabulek ramců.

2.7.6 FuncTable

V celém programu je jedna globální tabulka funkcí, která ukládá všechny uživatelsky vytvořené a vestavěné funkce.

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 Jestřabík měl za úkol syntakticou 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
Ondřej Hruboš	rozbor výrazů, tvorba struktur, makefile, testování
Radek Jestřabík	syntaktická analýza, sémantická analýza, testování
Ondřej Šatinský	lexikální analýza, generátor kódu, fungování LL gramatiky, testování

Table 2: Rozdělení práce v týmu mezi jednotlivými členy