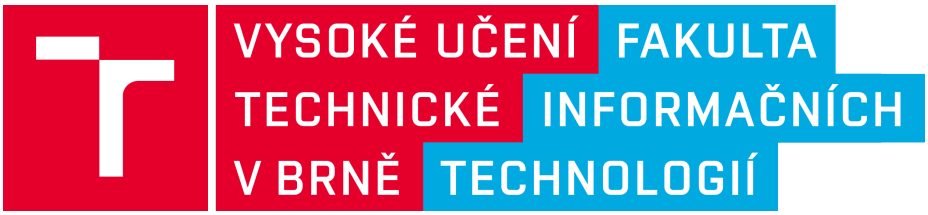


Projektová dokumentace
Implementace překladače jazyka IFJ21

Tým 047, Varianta I

Členové týmu:

Tomáš Bártů	**(xbartu11)**	25 %
Šimon Vacek	(xvacek10)	25 %
Vít Janeček	(xjanec30)	25 %
Tony Pham	(xphamt00)	25 %

8. prosince 2021

Obsah

1 Řešení projektu

1.1 Lexikální analýza

Jádro lexikální analýzy tvoří funkce ***get_token***, která je implementována jako konečný automat. Vstupem jsou 2 parametry: struktura token a zdrojový kód jazyka ifj21. Po průchodu funkcí se do struktury token uloží ***ID*** [typ tokenu: řetězec, klíčové slovo, číslo, EOL, EOF,. . .] a ***VALUE*** [value může být typu: string, integer, keyword, double]). Funkce čte soubor znak po znaku na jehož základě rozhoduje do jakého stavu má jít využitím switche.

1.2 Syntaktická a sémantická analýza

Syntaktickou analýzu shora dolů jsme implementovali rekurzivně a nachází se v ***parser.c*** a analýzu zdola nahoru precedenčně v ***expression.c***. Vstupem jsou tokeny generované scannerem.

Pro kontrolu syntaxe výrazů jsme implementovali zásobík na terminály a neterminály v souboru ***term_stack.c***

Sématická analýza stejně jako syntaktická je implementována v ***parser.c*** a ***expression.c***.

Tabulka symbolů je implementovaná jednosměrně vázaným seznamem, do kterého lze přidávat rámce pouze na vrchol. Položka (rámec) v seznamu odpovídá jednomu bloku programu. Tento rámec má ukazatel na vrchol binárního vyhledávacího stromu, ve kterém už jsou identifikátry proměnných a funkcí. Struktura tabulky obsahuje ukazatel na globální rámec (***GlobalElement***) s identifikátory funkcí a nejvyšší rámec (***TopLocalElement***) s identifikátory proměnných tohoto rámce. Nejvyšší rámec dál ukazuje na předchozí rámce, které překryl.
Uzel stromu s identifikátorem obsahuje data k němu se vztahující, tzn. např. datový typ u proměnné a seznam s parametry a návratovými hodnotami u funkce.

Pro zjednodušení jsme se rozhodli neimplementovat vícenásobné přiřazení hodnot do proměnných. Rovněž neimplementujeme funkce s více návratovými hodnotami.
Tabulka symbolů je k nalezení v souboru ***symtable.c*** .

1.3 Generování cílového kódu

Rozhodli jsme se generovat kód bez optimalizací a to tak, že tiskneme přímo instrukce na standardní výstup.

Výpis kódu vestavěných funkcí (reads, readi, readn, write, tointeger, substr, ord, chr) je v ***buildIn.c*** . Dále jsme vytvořili pomocné funkce pro výpis instrukcí, které se často opakují a nachází se v ***generate_code.c***.

Samotné generování ifjcode21 a volání výše zmíněných funkcí je umístěné v ***parser.c*** a ***expression.c***. Za zmínku stojí vyřešení stínění tím, že kopírujeme proměnné z nižšího rámce datového zásobníku do následujícího rámce. Pokud se hodnota dané proměnné změní, propaguje se změno do nižšího rámce.

Na generování kódu pracovali všichni členové týmu a proto jsme zavedli konvence pro názvy návěští a proměnných.

2 Práce v týmu

2.1 Způsob práce v týmu

Práci v týmu jsme pojali tak, že členové týmu si rozdělili úkoly na právě zpracovávané části tzn. nedělali jsme moduly. Na projektu jsme začali pracovat v půlce října.

2.2 Verzovací systém

Pro správu našich souborů jsme zvolili verzovací systém Git, který jsme hostovali na platformě GitHub. Zvolili jsme přístup pull-requestů na repozitář vedoucího týmu.

2.3 Komunikace

Jako komunikační prostředky jsme zvolili discord a messenger, kde jsme konzultovali aktuální témata k projektu. Později jsme se začali scházet ve školních prostorách, abychom mohli aktivněji řešit aktuální problémy.

2.4 Rozdělení práce mezi členy týmu

Tomáš Bártů:
Návrh automatu, Korekce, Syntaktická analýza zdola nahoru, Sématická analýza, Abstraktí datové struktury, Debug, Generování kódu

Šimon Vacek:
Testy, LL gramatika, Syntaktická analýza shora dolu, Sématická analýza, Debug, Generování kódu

Vít Janeček:
Lex. analýza, Abstraktí datové struktury včetně tabulky symbolů, Debug, Generování kódu

Tony Pham:
Lex. analýza, Dokumentace, Oprava leaků, Debug, Generování kódu

3 Problémy při vývoji

Během implementace scanneru jsme v lexikální analýze museli řešit hned několik problému např.: přeskočení čtení znaku u některých stavů (pomohla nám funkce ***ungetc***), chybný návrh automatu (museli jsme často přidávat / opravovat stavy).

Během syntaktické analýzy shora dolů se také projevily naše chabé dovednosti plánování. Pravidla LL gramatiky byly upraveny téměř jedenáctkrát a to i 3 dny před odevzdáním.

Podcenili jsme časovou náročnost projektu, sice jsme začali poněkud brzy a ze začátku šlo vše bez problému, ale po dokončení lexikální analýzy se vývoj projektu zabrzdil.

Při příštím týmovém projektu bychom se více zaměřili na tvorbu testů.

U sémantické analýzy jsme narazili na problém kontroly datových typů. Ten jsme vyřešili implementací zásobníků datových typů v souboru ***type_stack.c***.

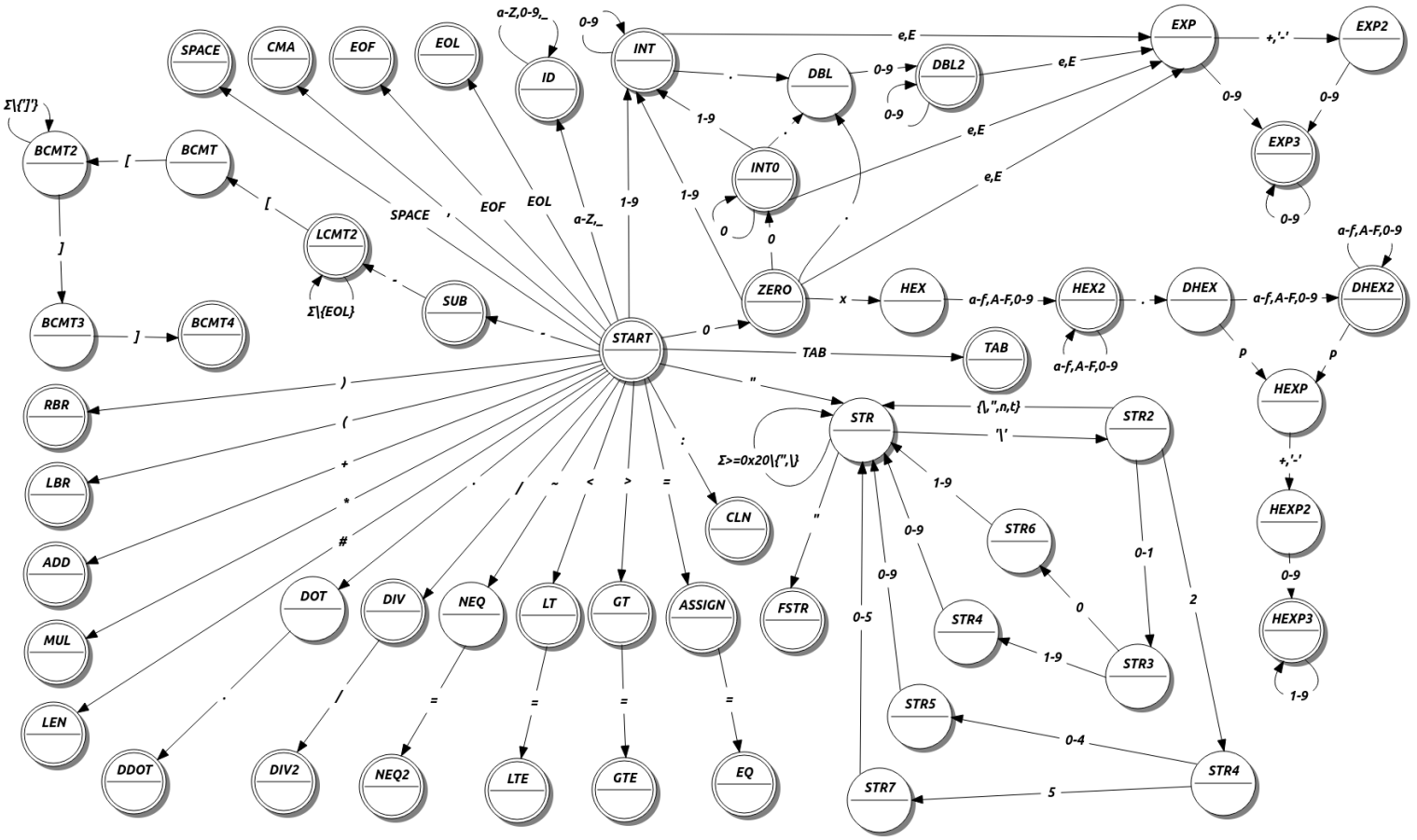
Většinu paměti se nám podařilo uvolnit, ale problémem byla neuvolněná paměť ve struktuře Token.

S větsující se složitostí struktury bylo obtížnější správně uvolnit všechnu paměť.

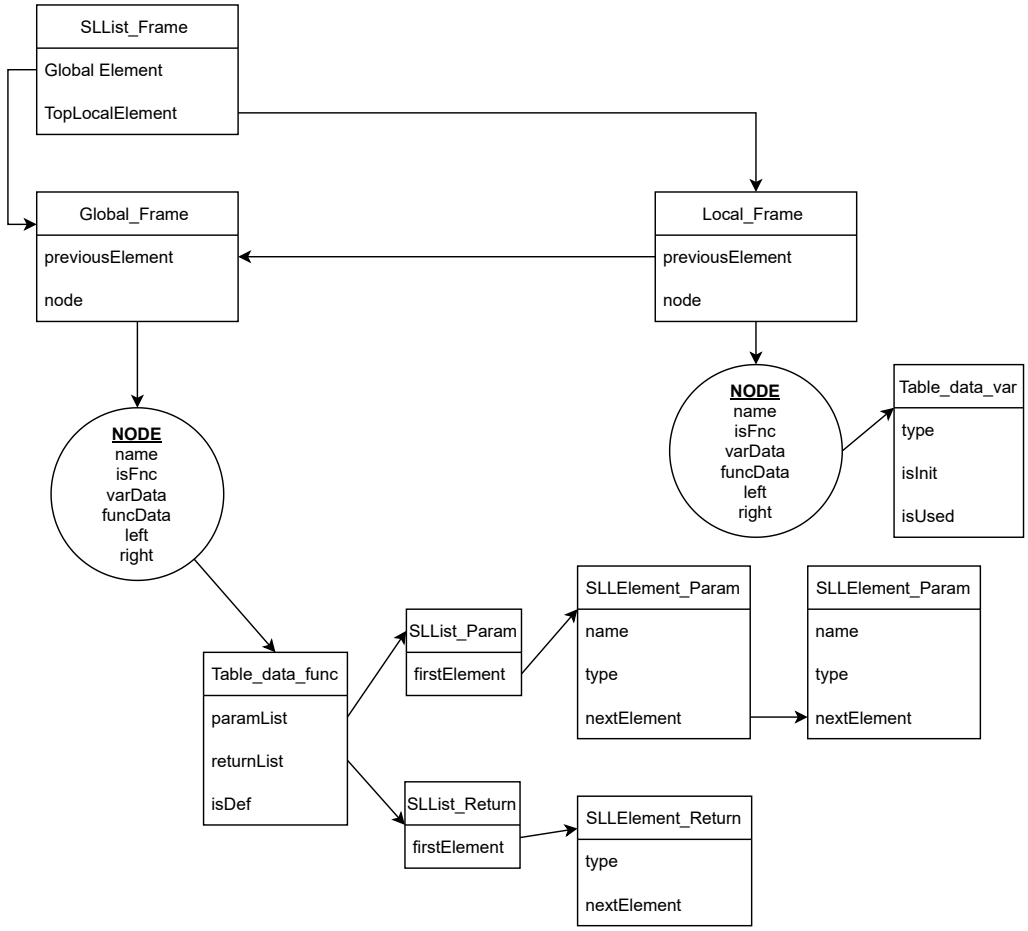
4 Použité nástroje, programy a literatura

- Prof. RNDr. Alexandr Meduna CSc. a Ing. Roman Lukáš Ph.D. - Formální jazyky a překladače, prezentace k přednáškám
- Ing. Bohuslav Křena Ph.D., Ing. Ivana Burgetová Ph.D. - Algoritmy, prezentace k přednáškám
- Ing. Zbyněk Křivka Ph.D. - Demonstrační cvičení IFJ, Implementace překladače IFJ21
- Prof. RNDr. Milan Češka, CSc.- Teoretická informatika, Učební texty
- Clion, https://www.jetbrains.com/clion/
- GitHub, https://github.com/paetricc/IFJ-Project
- Vim, https://www.vim.org
- QFSM, http://qfsm.sourceforge.net

5 Diagram automatu



6 Tabulka symbolů



7 Precedenční tabulka

Terminal na vrcholu zasobniku	Vstupni token																
	+	-	*	/	//	<	>	<=	>=	~=	"=="	(	)	I	#	..	$
+	>	>	<	<	<	>	>	>	>	>	>	<	>	<	<	>	>
-	>	>	<	<	<	>	>	>	>	>	>	<	>	<	<	>	>
*	>	>	>	>	>	>	>	>	>	>	>	<	>	<	<	>	>
/	>	>	>	>	>	>	>	>	>	>	>	<	>	<	<	>	>
//	>	>	>	>	>	>	>	>	>	>	>	<	>	<	<	>	>
<	<	<	<	<	<	>	>	>	>	>	>	<	>	<	<	<	>
>	<	<	<	<	<	>	>	>	>	>	>	<	>	<	<	<	>
<=	<	<	<	<	<	>	>	>	>	>	>	<	>	<	<	<	>
>=	<	<	<	<	<	>	>	>	>	>	>	<	>	<	<	<	>
~=	<	<	<	<	<	>	>	>	>	>	>	<	>	<	<	<	>
"=="	<	<	<	<	<	>	>	>	>	>	>	<	>	<	<	<	>
(	<	<	<	<	<	<	<	<	<	<	<	<	=	<	<	<	E
)	>	>	>	>	>	>	>	>	>	>	>	E	>	E	E	<	>
I	>	>	>	>	>	>	>	>	>	>	>	E	>	H	<	>	>
#	>	>	>	>	>	>	>	>	>	>	>	<	>	<	E	>	>
..	<	<	<	<	<	>	>	>	>	>	>	>	E	<	<	<	>
$	<	<	<	<	<	<	<	<	<	<	<	<	E	<	<	<	E

8 Gramatika pro výrazy

1. EXP → i
2. EXP → EXP + EXP
3. EXP → EXP - EXP
4. EXP → EXP * EXP
5. EXP → EXP / EXP
6. EXP → EXP // EXP
7. EXP → EXP .. EXP
8. EXP → (EXP)
9. EXP → EXP < EXP
10. EXP → EXP > EXP
11. EXP → EXP <= EXP
12. EXP → EXP >= EXP
13. EXP → EXP ~= EXP
14. EXP → EXP == EXP
15. EXP → #EXP

9 Pravidla LL gramatiky

1. <start> -> require "ifj21"<program>

2. <program> -> <fnc_dec> <program>
3. <program> -> <fnc_call> <program>
4. <program> -> <fnc_def> <program>
5. <program> -> ε

6. <fnc_dec> -> global id_fnc : function(<params_dec>) <return_type>
7. <params_dec> -> ε
8. <params_dec> -> <data_type> <params_dec2>
9. <params_dec2> -> ε
10. <params_dec2> -> ,<data_type><params_dec2>
11. <return_type> -> : <data_type>
12. <return_type> -> ε
13. <data_type> -> integer
14. <data_type> -> number
15. <data_type> -> string
16. <data_type> -> nil

17. <fnc_call> -> id_fnc (<value>)
18. <value> -> ε
19. <value> -> <value_last> <value2>
20. <value2> -> ,<value_last> <value2>
21. <value2> -> ε
22. <value_last> -> id_var
23. <value_last> -> integer_value
24. <value_last> -> number_value
25. <value_last> -> string_value
26. <value_last> -> nil

27. <fnc_def> -> <fnc_head> <fnc_def2> end
28. <fnc_head> -> function id_fnc (<params_def>)
29. <fnc_def2> -> <fnc_body>
30. <fnc_def2> -> : <data_type><fnc_body>
31. <params_def> -> ε
32. <params_def> -> <var_def> <params_def2>
33. <params_def2> -> ε
34. <params_def2> -> ,<var_def> <params_def2>
35. <var_def> -> id_var : <data_type>

36. <return> -> return <return2>
37. <return2> -> ε
38. <return2> -> <expr>

39. <fnc_body> -> <statement><fnc_body>
40. <fnc_body> -> <return>
41. <fnc_body> -> ε

42. <statement> -> <var_dec>
43. <statement> -> id_var = <var_assign>
44. <statement> -> <fnc_call>
45. <statement> -> <if>
46. <statement> -> <loop>

47. <var_dec> -> local <var_def> <var_dec_init>
48. <var_dec_init> -> = <var_assign>
49. <var_dec_init> -> ε

50. <var_assign> -> <expr>
51. <var_assign> -> <fnc_call>

52. <if> -> if <expr> then <statements> else <statements> end
53. <loop> -> while <expr> do <statements> end

54. <statements> -> ε
55. <statements> -> <statement> <statements>
56. <statements> -> <return>

POZN.: Slova a znaky psaná černě jsou neterminály.

POZN.2: Znak mezery v pravidlech je irelevantní a nic neznačí, slouží pouze k zpřehlednění.

POZN.3: Neterminál <expr> není rozšířen a zpracuje ho analýza shora dolů.

10 LL tabulka

	require	global	id_fce	function	return	end	(	)	integer	number	string	id_var	,	:	int_value	num_value	str_value	nil	local	=	#	if	else	while	$
<start>	1.																								
<program>		2.	3.	4.																					5.
<fnc_dec>		6.																							
<params_dec>								7.	8.	8.	8.														
<params_dec2>								9.					10.												
<return_type>		12.	12.	12.										11.											13.
<data_type>									13.	14.	15.							16.							
<fnc_call>			17.																						
<value>								18.				19.			19.	19.	19.	19.							
<value2>								21.					20.												
<value_last>												22.			23.	24.	25.	26.							
<fnc_def>				27.																					
<fnc_head>				28.																					
<fnc_def2>			29.		29.	29.						29.		30.					29.			29.		29.	
<params_def>								31.				32.													
<params_def2>								33.					34.												
<var_def>												35.													
<return>					36.																				
<return2>						37.	38.					38.			38.	38.	38.	38.			38.				
<fnc_body>			39.		40.	41.						39.							39.			39.		39.	
<statement>			44.									43.							42.			45.		46.	
<var_dec>																			47.						
<var_dec_init>			49.		49.	49.						49.							49.	48.		49.	49.	49.	
<var_assign>			51.				50.					50.			50.	50.	50.	50.			50.				
<if>																						52.			
<loop>																								53.	
<statements>			55.		56.	54.						55.							55.			55.	54.	55.	