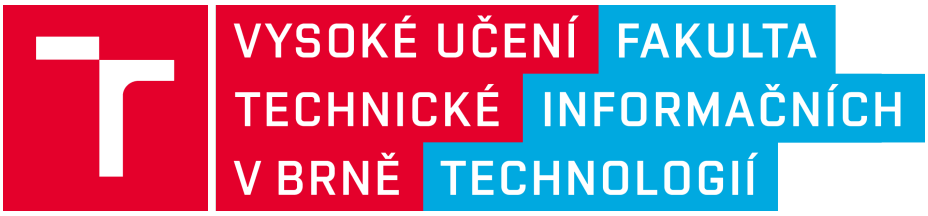


Projektová dokumentace
Implementace překladače jazyka IFJ21

Tým 047, Varianta I

Členové týmu:

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

8. prosince 2021

Obsah

1 Řešení projektu

1.1 Lexikální analýza

Při implementaci překladače jsme začali prvně s tvorbou lexikální analýzy. Hlavní funkce je ***get_token***, která je implementována jako konečný automat a žádá 2 parametry: strukturu token (po průchodu funkcí uloží do struktury ***ID*** [typy tokenů: řetězec, klíčové slovo, číslo, EOL, EOF ...] a ***VALUE*** [value se dále rozděluje na: string, integer, keyword, double]) a vstupní soubor. Funkce čte soubor znak po znaku a rozhoduje do jakého stavu má jít pomocí využití switche a casu pro každý stav, kde je potřeba.

1.2 Syntaktická analýza

Syntaktickou analýzu jsme implementovali rekurzivně a nachází se v ***parser.c*** (top-down) a ***expression.c*** (bottom-up) Vstupem jsou jednotlivé tokeny z lexikální analýzy. Syntaktická analyzátor využívá LL-Gramatiku (LL-Tabulku a LL-Pravidla) k tomu, aby zjistil, zda jde vstup programu syntakticky správně.

Symtable.c je implementovaná pomocí binárního stromu, zásobníku a jednosměrně vázaných seznamů. Základem je sktruktura ***SLList_Frame*** , která tvoří počátek vázaného seznamu. Tato struktura dále obsahuje další 2 vázané seznamy: ***topLocalElement*** a ***globalElement***.

GlobalElement ukazuje na globální rámce (funkce) a tento rámec je struktura obsahující: ***node*** a ***previousElement***.

TopLocalElement ukazuje na lokální rámce(tělo funkce, cyklus, while) a tento rámec je struktura obsahující: ***node*** a ***previouElement***.

Node je ukazatel na kořen binárního stromu, kde jeho jednotlivé noty jsou struktury, které obsahují další vázané seznamy s informacemi o rámcích.

Pro zjednodušení jsme se rozhodli neimplementovat vícenásobné přiřazení hodnot do proměnných. Rovněž neimplementujeme funkce s více návratovými hodnotami.
Kvůli nejasnostem v zadaní jsme nil nebrali jako datový typ a pouze jako hodnotu. Deklarované proměnné je implicitně nastavena hodnota nil a nedá se použít, dokud není inicializována.

1.3 Sémantická analýza

Sématická analýza stejně jako syntaktická analýza je implementována v ***parser.c*** a ***expression.c***. Při implementaci bylo nutné rozdělit jeden z neterminálu na 2 funkce které dělají různé sémantické kontroly, ale syntaxi kontrolují stejně.

1.4 Generování cílového kódu

Generování kódu pro vestavěné funkce (reads, readi, readn, write, tointeger, substr, ord, chr) je v ***buildIn.c*** . Dále jsme vytvořili pomocné funkce pro generování kódu, který se často opakuje a nachází se ve ***generate_code.c***.

Samotné generování ifjcode21 je umístěné v ***parser.c*** a ***expression.c***. Zde se generuje hlavička souboru, funkce programu, logika if a while atd. . Za zmínku stojí řešení stínění, které jsme vyřešili tak, že připisujeme proměnné ze spodku zásobníku rámců do rámce nad ním až na top.

2 Práce v týmu

2.1 Způsob práce v týmu

Na projektu jsme začali pracovat v půlce října. Ze začátku jsme si rozdělili základní úkoly a po dokončení jsme se vždy domluvili co má daný jedinec dělat dál.

2.2 Verzovací systém

Pro správu našich souborů jsme zvolili verzovací systém Git. Ten nám umožnil zpracovávat více souborů zároveň.

2.3 Komunikace

Jako komunikační prostředky jsme zvolili discord a massanger, kde jsme konzultovali aktuální témata k projektu. Později jsme se začali scházet ve školních prostorech, aby jsme 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

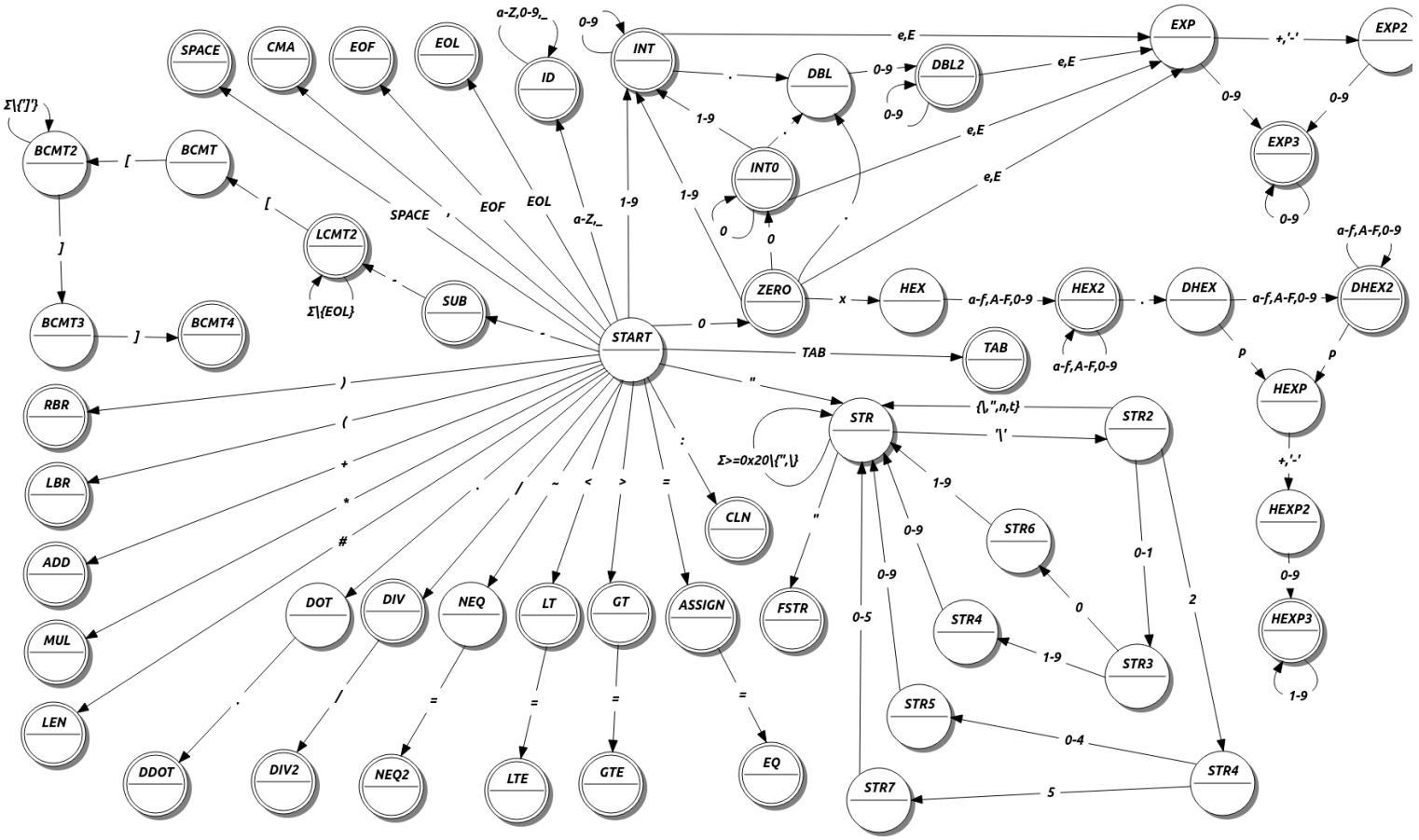
3 Problémy při vývoji

Během implementace scanneru jsme v syntaktické 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 projevila 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 při implementaci Syntaktické analýzy se vývoj projektu velmi zpomalil.

4 Diagram automatu



5 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. <id_fnc> -> identifikátor_funkce
8. <params_dec> -> ε
9. <params_dec> -> <data_type> <params_dec2>
10. <params_dec2> -> ε
11. <params_dec2> -> ,<data_type><params_dec2>
12. <return_type> -> : <data_type>
13. <return_type> -> ε
14. <data_type> -> integer
15. <data_type> -> number
16. <data_type> -> string

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. <id_var> -> identifikátor_proměnné

27. <fnc_def> -> <fnc_head> <fnc_def2> end
28. <fnc_head> -> function <id_fnc>(<params_def>)
29. <fnc_def2> -> <fnc_body> <return_void>
30. <fnc_def2> -> : <data_type><fnc_body><return>
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 <expr>

37. <return_void> -> return
38. <return_void> -> ε

39. <fnc_body> -> <if><fnc_body2>
40. <fnc_body> -> <loop><fnc_body2>
41. <fnc_body> -> <statement><fnc_body2>
42. <fnc_body2> -> ε
43. <fnc_body2> -> <fnc_body>

44. <statement> -> <var_dec>
45. <statement> -> <id_var> = <var_assign>
46. <statement> -> <fnc_call>

47. <var_dec> -> local <var_def> <var_dec_init>
48. <var_dec_init> -> = <var_dec_init2>
49. <var_dec_init> -> ε
50. <var_dec_init2> -> <expr>
51. <var_dec_init2> -> <fnc_call>

52. <var_assign> -> <expr>
53. <var_assign> -> <fnc_call>

54. <if> -> if <expr> then <statements> else <statements> end

55. <loop> -> while <expr> do <statements> end

56. <statements> -> ε
57. <statements> -> <statement> <statements>

POZN.: Slova a znaky psaná černě jsou neterminály.

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

6 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

7 Precedenční tabulka

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

8 Použité nástroje, programy a literatura

- Alexandr Meduna a Roman Lukáš, Formální jazyky a překladače, prezentace k přednáškám
- Bohuslav Křena, Ivana Burgetová, Algoritmy, prezentace k přednáškám
- Zbyněk Křivka, Demonstrační cvičení IFJ, Implementace překladače IFJ21
- Clion, https://www.jetbrains.com/clion/
- GitHub, https://github.com/paetricc/IFJ-Project
- Vim, https://www.vim.org
- QFSM, http://qfsm.sourceforge.net

9 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.														
<fnc_call>			16.																						
<value>								17.				18.			18.	18.	18.	18.							
<value2>								20.					19.												
<value_last>												21.			22.	23.	24.	25.							
<fnc_def>				26.																					
<fnc_head>				27.																					
<fnc_def2>			28.		28.	28.						28.		29.					28.			28.		28.	
<params_def>								30				31.													
<params_def2>								32.					33.												
<var_def>												34.													
<return>					35.																				
<return_void>					36.	37.																			
<fnc_body>			40.		41.	41.						40.							40.			38.		39.	
<statement>			44.									43.							42.						
<var_dec>																			45.						
<var_dec_init>			47.		47.	47.						47.							47.	46.		47.	47.	47.	
<var_assign>			49.				48.					48.			48.	48.	48.	48.			48.				
<if>																						50.			
<loop>																								51.	
<statements>			53.			52.						53.							53.				52.		