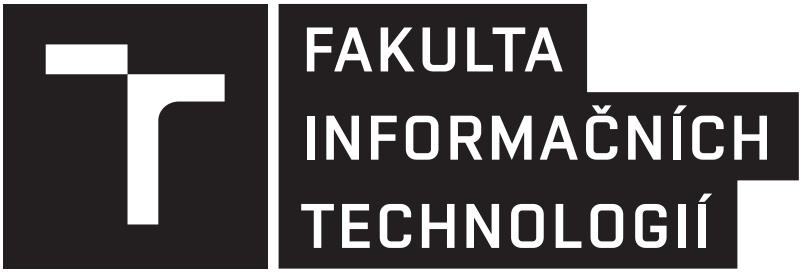
Vysoké učení technické v Brně
Fakulta informačních technologií



Dokumentácia k projektu z IFJ a IAL

Implementácia prekladača jazyka IFJ21

Tým *077*, varianta *I*

Vedúcí tímu:	Bublavý Martin	xbubla02	60 %
Další členovia:	Hučko Adrián	xhucko02	20 %
	Pszota Igor	xpszot00	20 %
	Gablovský Peter	xgablo00	0 %

Obsah

1 Úvod

Cieľom tohoto projektu bolo vytvoriť program v jazyku C, ktorý načíta kód zapísaný v jazyku IFJ21 a preloží ho do cieľového jazyka IFJcode21. Naša varianta č. I zahŕňala riešenie pomocou binárneho vyhĺadávacieho stromu.

2 Práca v tíme

2.1 Rozdelenie práce

- **Bublavý Martin** – parser, symtable, statement_parser, scanner
- **Hučko Adrián** – symtable, dokumentácia, error
- **Pszota Igor** – parser, scanner, dokumentácia
- **Gablovský Peter** – generátor

2.2 Tvorba dokumetácie

Dokumentácia bola založená na štruktúre kódu pomocou, ktorého sme postupne analyzovali jednotlivé časti vypracováneho projektu.

2.3 Príprava na obhajobu

Na obhajobu sme sa pripravovali spoločne, objasnením častí, ktoré vytvoril každý jednotlivec v tíme. K obhajobe bola pripravená prezentácia, ktorá zahŕňa problematiku daného programu.

3 Implementácia interpretu jazyka IFJ21

3.1 Lexikálna analýza

Lexikálny analyzátor(scanner), bola časť projektu, ktorú sme implementovali ako prvú, je navrhnutý pomocou konečného automatu. Scanner načítava znaky zo štandartného vstupu po jednom, a podľa nich rozhoduje o ďalšom stave. Po získaní tokenu nastaví jeho typ.

3.2 Syntaktická a sémantická analýza

3.2.1 Syntaktická analýza

Na implementáciu syntaktického analyzátora bola použitá metóda rekurzívneho zostupu, ktorá vychádza z vytvorenej LL tabuľky(LL gramatiky). Parser ukladá premenné, funkcie, parametre funkcií... do binárnych vyhĺadávacích stromov ,ktoré sú použité v semantickej analýze.

3.2.2 Semantická analýza

Pre semantickú analýzu sme vytvorili štruktúry binárnych stromov, ktoré sa nachádzajú v súbore symtable.c. Jeden strom je určený pre ukladanie funkcí a druhý pre ukladanie premenných. Ukladajú informácie o deklarácií premenných a funkcií, ich parametroch a návratových hodnotách.

3.3 Dátové štruktúry

3.3.1 Token

Pre uloženie aktuálne spracovávaného tokenu sme použili dátovú štruktúru ,ktorá obsahuje jeho reťazec, dĺžku a typ.

3.3.2 String

Slúži pre ukladanie identifikátorov premenných a funkcií pre neskoršie použitie.

3.3.3 Zasobník precedenčnej analýzy

Slúži na ukladanie symbolov a identifikátorov spracovávaného výrazu a podľa pravidiel redukuje uchovávané prvky.

3.4 Binárny strom

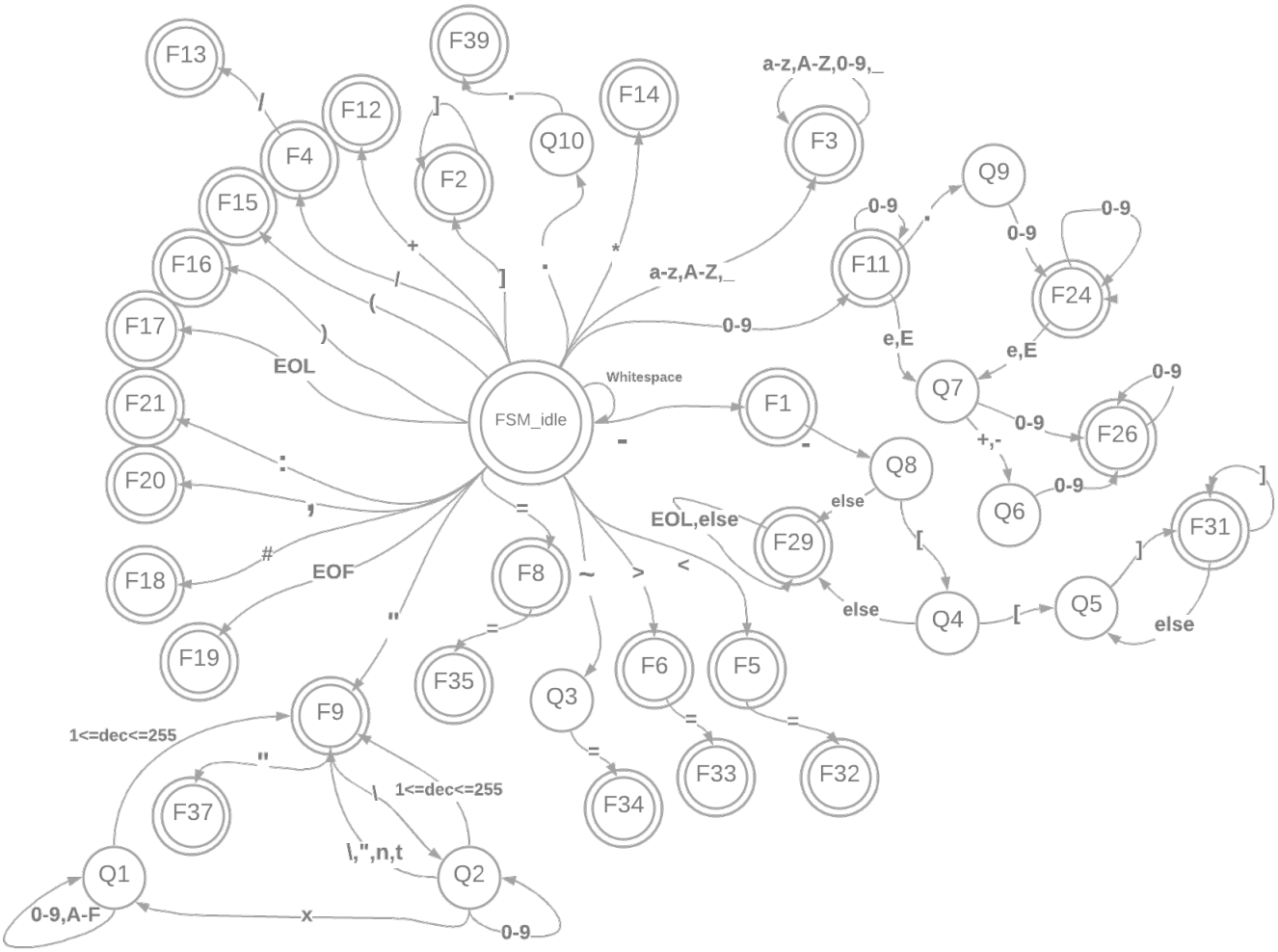
Jeden strom je určený pre ukladanie funkcí a druhý pre ukladanie premenných. Strom určený pre premenné uchováva jej identifikátor, typ a rozsah ,v ktorom bola deklarovaná. Strom určený pre funkcie uchováva identifikátor funkcie, list s parametrami, list s návratovými hodnotami a informácie o deklarováni a volání funkcie.

3.4.1 Zásobník stromu

Zásobník bol z časti implementovaný z predmetu IAL z domácej úlohy č.1. Zároveň ukladá typ a rozsah premennej ,v ktorom bola deklarovaná.

4 Prílohy

4.1 FSM



F1=FSM_Minus
F2=FSM_Block_Comment_End_End
F3=FSM_Identifier_Keyword
F4=FSM_Division
F5=FSM_Less_Than
F6=FSM_More_Than
Q3=FSM_Not_Equal
F8=FSM_Assign
F9=FSM_String
Q10=FSM_Dot
F11=FSM_Integer
F12=FSM_Add
F13=FSM_INT_Divide
F14=FSM_Multiply
F15=FSM_Left_Bracket
F16=FSM_Right_Bracket
F17=FSM_End_Of_Line
F18=FSM_Hashtag
F19=FSM_End_Of_File
F29=FSM_Comment_String
Q5=FSM_Block_Comment_String
F31=FSM_Block_Comment_End
F38=FSM_Block_Comment_End
F24=FSM_Double
Q6=FSM_Exponent_Sign
F26=FSM_Exponent_Finish
Q8=FSM_Comment
Q4=FSM_Block_Comment
F20=FSM_Comma
F21=FSM_Colon
Q9=FSM_Double_Point
F32=FSM_LT_Equals
F33=FSM_BT_Equals
F34=FSM_Not_Equal
F35=FSM_Equals
Q2=FSM_Escape
F37=FSM_String
Q7=FSM_Exponent
F39=FSM_Concat
Q1=FSM_Hex

4.2 LL gramatika

1. <parse> -> EOF <Find_Header> <function_def_list>
2. <Find_header> -> require "ifj21" <parse>
3. <function_def_list> -> global <global_func_def> <function_def_list>
4. <function_def_list> -> function <function_def> <function_def_list>
5. <function_def_list> -> ϵ
6. <global_func_def> -> ID : function (Parameters)
7. <global_func_def> -> ID : function (Parameters) : Returns
8. <global_func_def> -> ϵ
9. <function_def> -> ID (<function_params>) <statement_list_rules>
10. <function_def> -> ID (<function_params>) : <function_returns> <statement_list_rules>
11. <function_params> -> ID : TYPE <function_def>
12. <function_params> -> ID : TYPE , <function_params>
13. <function_params> ->) <function_def>
14. <function_params> -> ϵ
15. <statement_list_rules> -> ID/keywords <statement> <statement_list_rules>
16. <statement_list_rules> -> end/else <function_def>
17. <statement_list_rules> -> ϵ
18. <statement> -> if TYPE/ID/#/(then <statement_list_rules> else <statement_list_rules> end
19. <statement> -> while TYPE/ID/#/(do <statement_list_rules> end
20. <statement> -> local/global ID : TYPE = statement
21. <statement> -> local/global ID : TYPE = ID (<function_call_parameters> <statement_list_rules>
22. <statement> -> local/global ID : TYPE
23. <statement> -> ID = <req_statement>
24. <statement> -> ID = ID (<function_call_parameters> <statement_list_rules>
25. <statement> -> ID , <assign_left> <statement_list_rules>
26. <statement> -> return <return_values>
27. <statement> -> write <write_parameters>
28. <statement> -> ϵ
29. <function_call_parameters> -> ID/TYPE , <function_call_parameters>
30. <function_call_parameters> -> ID/TYPE) <statement>
31. <function_call_parameters> ->) <statement>
32. <function_call_parameters> -> ϵ
33. <assign_left> -> ID , <assign_left>
34. <assign_left> -> ID = <assign_right>
35. <assign_left> -> ϵ
36. <return_values> -> statement , <return_values>
37. <return_values> -> statement <statement>
38. <return_values> -> ϵ
39. <write_parameters> -> ID/TYPE , <write_parameters>
40. <write_parameters> -> ID/TYPE) <statement>
41. <write_parameters> ->) <statement>
42. <write_parameters> -> ϵ
43. <assign_right> -> statement , <assign_right>
44. <assign_right> -> ID (<function_call_parameters> , <assign_right>
45. <assign_right> -> ID (<function_call_parameters> <assign_left>
46. <assign_right> -> statement <assign_left>
47. <function_returns> -> TYPE , <function_returns>
48. <function_returns> -> TYPE <function_def>

4.3 LL tabuľka

<parse>	EOF	<Find_Header>	<function_def_list>					
<Find_header>	require	ifj21	<parse>					
<function_def_list>	global	<global_func_def>	<function_def_list>					
<function_def_list>	function	<function_def>	<function_def_list>					
<function_def_list>	ε							
<global_func_def>	ID	:	function	(	Parameters	)		
<global_func_def>	ID	:	function	(	Parameters	)	:	Returns
<global_func_def>	ε							
<function_def>	ID	(	<function_params>	)	<statement_list_rules>			
<function_def>	ID	(	<function_params>	)	:	<function_returns>	<statement_list_rules>	
<function_params>	ID	:	TYPE	<function_def>				
<function_params>	ID	:	TYPE	,	<function_params>			
<function_params>	)	<function_def>						
<function_params>	ε							
<statement_list_rules>	ID/keywords	<statement>	<statement_list_rules>					
<statement_list_rules>	end/else	<function_def>						
<statement_list_rules>	ε							
<statement>	if	TYPE/ID/#/(	then	<statement_list_rules>	else	<statement_list_rules>	end	
<statement>	while	TYPE/ID/#/(	do	<statement_list_rules>	end			
<statement>	local/global	ID	:	TYPE	=	statement		
<statement>	local/global	ID	:	TYPE	ID	(	<function_call_parameters>	<statement_list_rules>
<statement>	local/global	ID	:	TYPE				
<statement>	ID	=	<req_statement>					
<statement>	ID	=	ID	(	<function_call_parameters>	<statement_list_rules>		
<statement>	ID	,	<assign_left>	<statement_list_rules>				
<statement>	return	<return_values>						
<statement>	write	<write_parameters>						
<statement>	ε							
<function_call_parameters>	ID/TYPE	,	<function_call_parameters>					
<function_call_parameters>	ID/TYPE	)	<statement>					
<function_call_parameters>	)	<statement>						
<function_call_parameters>	ε							
<assign_left>	ID	,	<assign_left>					
<assign_left>	ID	=	<assign_right>					
<assign_left>	ε							
<return_values>	statement	,	<return_values>					
<return_values>	statement	<statement>						
<return_values>	ε							
<write_parameters>	ID/TYPE	,	<write_parameters>					
<write_parameters>	ID/TYPE	)	<statement>					
<write_parameters>	)	<statement>						
<write_parameters>	ε							
<assign_right>	statement	,	<assign_right>					
<assign_right>	ID	(	<function_call_parameters>	,	<assign_right>			
<assign_right>	ID	(	<function_call_parameters>	<assign_left>				
<assign_right>	statement	<assign_left>						
<function_returns>	TYPE	,	<function_returns>					
<function_returns>	TYPE	<function_def>						

4.4 Precedenčná tabuľka

	+	-	*	/	//	<	<=	>	>=	==	~=	#	. .	(	)	ID	INT	STRING	DOUBLE	$
+	>	>	<	<	<	>	>	>	>	>	>	<	X	>	>	>	<	>	>	>
-	>	>	<	<	<	>	>	>	>	>	>	<	X	>	>	>	<	>	>	>
*	>	>	>	>	>	>	>	>	>	>	>	<	X	>	>	>	<	>	>	>
/	>	>	>	>	>	>	>	>	>	>	>	<	X	>	>	>	<	>	>	>
//	>	>	>	>	>	>	>	>	>	>	>	<	X	>	>	>	<	>	>	>
<	<	<	<	<	<	X	X	X	X	X	<	<	X	<	>	<	<	<	<	>
<=	<	<	<	<	<	X	X	X	X	X	<	<	X	<	>	<	<	<	<	>
>	<	<	<	<	<	X	X	X	X	X	<	<	X	<	>	<	<	<	<	>
>=	<	<	<	<	<	X	X	X	X	X	<	<	X	<	>	<	<	<	<	>
==	<	<	<	<	<	X	X	X	X	X	<	<	X	<	>	<	<	<	<	>
~=	<	<	<	<	<	X	X	X	X	X	<	<	X	<	>	<	<	<	<	>
#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	>	X	>	X	>
. .	X	X	X	X	X	X	X	X	X	X	X	X	X	X	>	<	X	<	X	>
(	<	<	<	<	<	<	<	<	<	<	<	<	<	<	=	<	<	<	<	X
)	>	>	>	>	>	>	>	>	>	>	>	X	>	X	>	X	X	X	X	>
ID	>	>	>	>	>	>	>	>	>	>	>	X	>	X	>	X	X	X	X	>
INT	>	>	>	>	>	>	>	>	>	>	>	X	X	X	>	X	X	X	X	>
STRING	>	>	>	>	>	>	>	>	>	>	>	X	>	X	>	X	X	X	X	>
DOUBLE	>	>	>	>	>	>	>	>	>	>	>	X	X	X	>	X	X	X	X	>
$	<	<	<	<	<	<	<	<	<	<	<	<	<	<	X	<	<	<	<	&