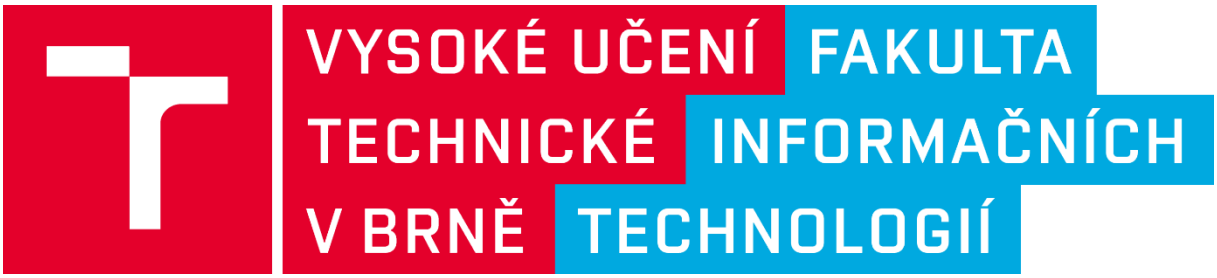


DOKUMENTÁCIA

Implementácia imperatívneho prekladaču jazyka IFJ21

Tým 095, variant 1

LEADER: **Adam Dzurilla (xdzuri00) - 34**

Jakub Kasem (xkasem02) - 33

Martin Mores (xmores02) - 33

8.12.2021

Rozšírenie - FUNEXP

Obsah

1.Úvod

Úlohou je vytvoriť program v jazyku C, ktorý načíta zdrojový kód zapísaný v zdrojovom jazyku IFJ21 a preloží ho do cieľového jazyka IFJcode21

2.Návrh a implementácia

Súbor pozostáva z niekoľkých časti, ktoré spolu pracujú a nadväzujú na seba

2a.Lexikálna analýza

Lexikálny analyzátor dostáva na vstup postupnosť znakov a na zavolanie funkcie get_token vracia posledný ešte nenačítaný token, funkcia môže vrátiť nasledujúce typy tokenov: identifikátor, kľúčové slovo, celočíselný literál, desatinný literál, reťazcový literál, operátor plus, operátor mínus, operátor krát, operátor delenie, operátor celočíselne delenie, relačný operátor väčší, relačný operátor menší, relačný operátor väčší rovný, relačný operátor menší rovný, relačný operátor rovný, relačný operátor rôzny, konkatenacia reťazcových literálov a hashtag

2b.Syntaktická analýza

Syntakticky analyzátor bol navrhnutý podľa dopredu vypracovanej LL Gramatiky, syntakticky analyzátor funguje rekurzívnou metódou .Pre každý neterminál LL Gramatiky bola vytvorená funkcia ktorá ma jasne definovane pravidla pre ďalšie tokeny. V prípade že syntakticky analyzátor dostane token pre ktorý nepozná prechod, vracia syntaktickú chybu (chybový kód 2). V prípade že syntakticky analyzátor ktorý ma epsilon prechod dostane neočakávaný token, vráti ho naspäť lexikálnemu analyzátoru pomocou funkcie return_token, a vracia úspešnosť svojej operácie. Chybový kód sa vracia pomocou parametru return_code nachádzajúci sa v štruktúre SPars_data, ktorá je predávaná medzi funkciami neterminálov. Deklarácia štruktúry SPars_data sa nachádza v súbore "parser.h". Štruktúra obsahuje aj ukazateľ na token do ktorého je načítaný vždy nasledujúci token aby sme nemuseli zakaždým alokovať novy token.

2c.Sémantická analýza

Sémantická analýza bola implementovaná spolu so syntaktickou analýzou vo funkciách neterminálov podľa potreby vykonania sémantickej kontroly. Pomocne premenne sa nachádzajú v štruktúre SPars_data definovanej v súbore "parser.h". Sémantická analýza si pamätá stav v ktorom sa nachádza napr. deklarácia funkcie, definícia funkcie, volanie funkcie a podobne. Všetky stavy sa nachádzajú v enum triede Parser_State deklarovanej v súbore "parser.h". Ďalej si sémantická kontrola pamätá či sa načítavajú typy parametrov alebo typy návratových hodnôt. Táto premenná je používaná pri deklarácii alebo definícii funkcii. Štruktúra ďalej obsahuje globálnu tabuľku symbolov v ktorej sú uložene funkcie programu, a zoznam lokálnych tabuliek symbolov. Lokálna tabuľka symbolov je vytvorená pri definícii funkcie a hneď sú do nej vložene parametri funkcie, ďalšia úroveň tabuľky sa vytvorí pri príkazoch IF alebo WHILE. Štruktúra ešte obsahuje aj zoznam typov do ktorého sa načítavajú postupne typy pri priradení do identifikátorov, tento zoznam je využitý pri kontrole typovej kompatibility vo funkcii func_expression. Pri určení kde sa bude kontrolovať sémantická kontrola sme si pomohli LL Gramatikou.

2d.Generovanie kódu

Generovanie kódu sme implementovali v súboroch code_generator.c a code_generator.h. Pri generovaní kódu sme zvolili prístup využitia funkcii ktoré, generujú kód do symbolového reťazca a ten sa po úspešnej syntaktickej a sémantickej kontrole vypíše na štandardný výstup

3a.Konečný automat pre určovanie tokenov

Konečný automat stavov je navrhnutý aby rozlišoval ci postupnosť znakov na vstupe je validná pre daný jazyk. Rozhoduje o aký typ tokenu sa jedna a vracia daný token syntaktickému analyzátoru spolu s ďalšími atribútmi.

3b.LL Gramatika

LL Gramatika bola navrhnutá podľa zadania v súbore "/docs/ifj2021.pdf". Počiatočný stavom je <prol> kde je vyžadovaná postupnosť tokenov " require \"ifj21\" ". Po načítaní tohto prologu prechádza syntakticky analyzátor do stavu <prog> kde môže načítavať - Deklarácie

funkcii, Definície funkcii, Volania funkcii a token ukončenia súboru. Ukončovací token ukonči syntaktickú analýzu. LL Gramatika sa skladá ešte zo stavov <return_func_var> ktorý pomáha určiť návratové hodnoty funkcie, <params> ktorý pomáha načítať parametre pri definícii funkcie, <param_n> ktorý slúži pre načítanie viac ako jedného parametru funkcie, <statement> ktorý rozširuje načítanie viacerých príkazov - inicializácia premennej (počiatočný token KEYWORD_LOCAL), vracanie hodnôt pri výstupe z tela funkcie (počiatočný token KEYWORD_RETURN), priradenie hodnôt do jedného alebo viacerých identifikátorov (počiatočný token IDENTIFIKATOR), podmienený príkaz if (počiatočný token KEYWORD_IF) a cyklus while (počiatočný token KEYWORD_WHILE). Prechod pre gramatiku pomocou neterminálu <assign_value> ktorá pomáha pri určení ci definícii identifikátoru priradíme hodnotu alebo ho necháme nedefinovaným. Prechod pomocou neterminálu <id_n> určuje ci sme už skončili s načítavaním identifikátorov pre priradenie alebo nie. Prechod pomocou neterminálu <types> určuje ci ideme načítavať postupnosť typov alebo nie. Prechod pomocou neterminálu <type_n> určuje ci sme už skončili s načítavaním typov alebo nie. Prechod pomocou á<type>: očakávame jeden z tokenov KEYWORD_INTEGER, KEYWORD_NUMBER, KEYWORD_STRING alebo KEYWORD_NIL. Prechod pomocou neterminalu <assig_func> určuje ci išlo o volanie funkcie alebo o načítavanie premenných. Cela LL Gramatika sa nachádza buď v súbore "/docs/Dokumentacia.docx" alebo v súbore "/docs/dokumentacia.pdf".

3c.LL tabuľka

LL Tabuľka bola vypracovaná pomocou LL Gramatiky a určuje prechody z neterminálov do iných stavov pomocou tokenov, v prípade že daným tokenom neexistuje prechod jedna sa o syntaktickú chybu. Cela LL Tabuľka sa nachádza buď v súbore "/docs/LL_tabulka.xlsx" alebo v súbore "/docs/dokumentacia.pdf".

4a.Tímová práca

1. Adam Dzurilla - Lexikálna analýza, Syntaktická analýza, Tabuľky symbolov, Sémantická analýza, Testy, Dokumentácia, Štruktúra projektu, LL Gramatika, LL Tabuľka, Konečný automat pre načítavanie tokenov, Zoznamy typov, Zoznamy tabuliek symbolov, Generovanie kódu

2. Jakub Kasem - Syntaktická analýza, Syntaktická analýza výrazov, Sémantická analýza výrazov, LL Gramatika, LL Tabuľka, Dokumentácia, Precedenčná tabuľka, Generovanie kódu

3. Martin Mores - Precedenčna tabuľka, Dokumentácia, Testovanie, Tabuľka symbolov,

4b.Prekážky, s ktorými sme sa stretli

1. Zle sme si odhadli čas, ktorý bude na projekt potreba takže sme ho robili na poslednú chvíľu, čo malo za následok možno nie úplne dokonalú implementáciu projektu

2. Nedostatočne prvotne pochopenia zadania v lexikálnej analýze kde sme prevádzali hashtag na INT číslo hneď v lexikálnej analýze, potom sme sa dozvedeli že to môže byt zadane aj vo forme #s1 kde s1 je identifikátor, ďalej sme pridávali funkciu return_token v lexikálnej analýze, ktorej implementácia nám trošku robila problémy, a pri vracaní tokenu

identifikátoru sme nechtiac vracali aj úvodzovky čo pri následnom načítaní nenačítalo identifikátor ale string a s odhalením tejto chyby sme takisto strávili nejaký čas.

3. Úpravy LL Gramatiky sme našťastie vyriešili veľmi dobrou implementáciou kde najväčšie úpravy bolo treba robiť v komentároch a syntaktická analýza bola pre nás veľmi prehľadná a ľahko upravovateľná na základe pravidiel, ktoré sme si vytvorili

4. S veľa úpravami nám pomohli aj testy, ktoré sme si vytvorili a testovacie súbory, ktoré sme si buď vyčítali z textu, dostali ich alebo sme si vytvorili vlastne

4c.Testovacie súbory

Počas nášho vývoja prekladača sme si vytvorili viacero testov na kontrolu rôznych časti prekladača. Všetky testy sme situovali do zložky "/tests".

1. Testy pre lexikálnu analýzu ("/tests/test_lexical_analysis.c") dostanú súbor a vypisujú tokeny na standardny výstup pod seba spolu s atribútom ak nejaký atribút tieto tokeny majú.

2. Testy pre syntaktickú analýzu ("/tests/test_syntactic_analysis.c") dostanú súbor a zavolajú funkciu func_prol ktorej ako argument dajú štruktúru SPars_data, funkcia prechádza program a syntakticky kontroluje jeho správnosť, v prípade že návratový kód je NO_ERROR (podľa súboru error.h) je vypísaný na standardny výstup text označujúci úspešnosť operácie, v prípade že návratový kód nie je NO_ERROR je vypísaný návratový kód aby sme vedeli o aký typ chyby ide.

3. Testy pre tabuľky symbolov - test vytvorí tabuľku symbolov pre funkcie a tabuľku symbolov pre premenne. Postupne sú otestovane funkcie pracujúce s týmito tabuľkami rovnako ako vymazávania týchto tabuliek.

5. Špeciálne použité techniky a algoritmy

Použili sme naše vedomosti k tvorbe tabuľky symbolov kde sme implementovali binárny vyhľadávací strom (podľa nášho zadania projektu - variant I), pri binárnom strome sme použili funkcie pre - inicializáciu, vkladanie, hľadanie a odstránenie celého stromu. Ostatne funkcie pre prácu s binárnym stromom sme nepotrebovali tak sme ich neimplementovali. Ďalej sme si definovali funkcie pre vytvorenie funkcii a premenných, ktoré alokujú miesto a inicializujú hodnoty. takisto sme si vytvorili aj funkcie pre odstránenie dát z binárneho vyhľadávacieho stromu ktoré korektne uvoľnia alokovaný priestor. Potom sme si ešte implementovali zásobník pre riešenie výrazov pomocou postfixovej notácie

6.Zhrnutie

Projekt nám dal veľa znalosti v oblastiach tímovej práce kde sme sa naučili ako kooperovať pri implementáciách, ako si rozdeliť úlohy medzi sebou a ako komunikovať pri vytváraní rôznych pomocných štruktúr, ako postupovať spoločne pri implementáciách

FUNEXP – rozšírenie

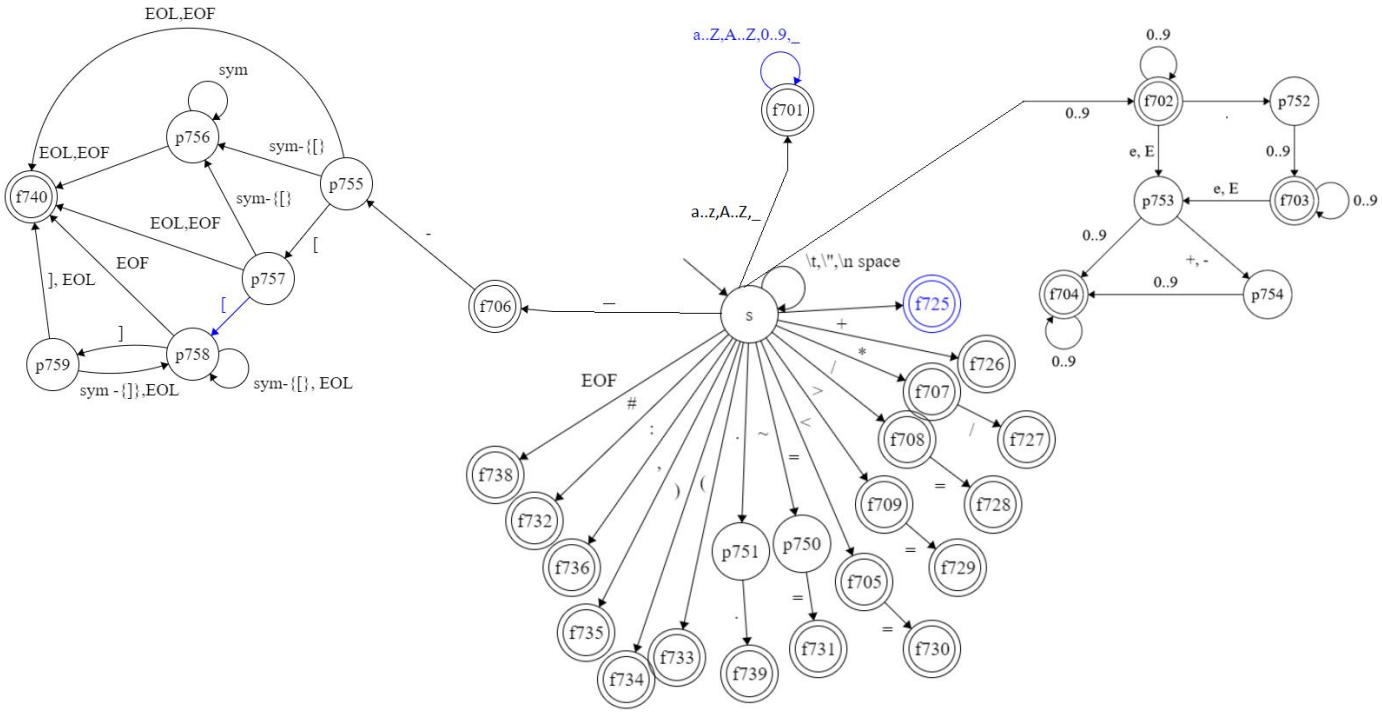
Pri projekte sme si vybrali rozšírenie funexp, rozšírenie sme implementovali primárne v súbore expression.c, kde sme si zvolili pri volaní funkcie ako parametre <expression>, a pri každom parametre voláme funkciu <func_expression> rekurzívne. Pri expression sme si vytvorili pravidlo ktoré môže citát po sebe idúce tokeny id a láva zátvorka, na základe tohto pravidla vieme že ide o volanie funkcie pri expressions

Lexikálny analyzátor	scanner.c	scanner.h
Syntakticky analyzátor	parser.c	parser.h
Sémanticky analyzátor	parser.c	parser.h
Spracovanie vyrazov	expression.c	expression.h
Generovanie kódu	code_generator.c	code_generator.h
Chybové hlásenia	error.c	error.h
Tabuľka symbolov	symtable.c	symtable.h
Zásobník	stack.c	stack.h
Hlavný program	compiler.c	

Precednčná tabuľka

		ZNAK VSTUP																
		+	-	*	..	#	/	//	<	>	<=	>=	"=="	~=	(	)	I	$
ZNAK VRCHOL ZASOBNIKU	+	>	>	<	>	<	<	<	>	>	>	>	>	>	<	>	<	>
	-	>	>	<	>	<	<	<	>	>	>	>	>	>	<	>	<	>
	*	>	>	>	>	<	>	>	>	>	>	>	>	>	<	>	<	>
	..	<	<	<	<	<	<	<	<	<	<	<	<	<	<	>	<	>
	#	>	>	>	>		>	>	>	>	>	>	>	>	<	>	<	>
	/	>	>	>	>	<	>	>	>	>	>	>	>	>	<	>	<	>
	//	>	>	>	>	<	>	>	>	>	>	>	>	>	<	>	<	>
	<	<	<	<	<	<	<	<							<	>	<	>
	>	<	<	<	<	<	<	<							<	>	<	>
	<=	<	<	<	<	<	<	<							<	>	<	>
	>=	<	<	<	<	<	<	<							<	>	<	>
	"=="	<	<	<	<	<	<	<							<	>	<	>
	~=	<	<	<	<	<	<	<							<	>	<	>
	(	<	<	<	<	<	<	<	<	<	<	<	<	<	<	=	<	
	)	>	>	>	>		>	>	>	>	>	>	>	>		>		>
	I	>	>	>	>		>	>	>	>	>	>	>	>		>		>
	$	<	<	<	<	<	<	<	<	<	<	<	<	<	<		<	

Diagram konečného automatu



LL Gramatika

1. <prol> → REQUIRE "ifj21" <prog>
2. <prog> → GLOBAL ID : FUNCTION (<types>) <return_func_var> <prog>
3. <prog> → FUNCTION ID (<params>) <return_func_var> <statement> END <prog>
4. <prog> → MAIN (<expression>) <prog>
5. <prog> → EOF
6. <return_func_var> → : <type> <type_n>
7. <return_func_var> → ε

8. <params> → ID : <type> <param_n>
9. <params> → ε
10. <param_n> → , ID : <type> <param_n>
11. <param_n> → ε
12. <statement> → ε
13. <statement> → LOCAL ID : <type> <assign_value> <statement>
14. <statement> → RETURN <expression> <statement>
15. <statement> → ID <assig_func> <statement>
16. <statement> → IF <expression> THEN <statement> ELSE <statement> END <statement>
17. <statement> → WHILE <expression> DO <statement> END <statement>
18. <assign_value> → ε
19. <assign_value> → = <expression>
20. <id_n> → ε
21. <id_n> → , ID <id_n>
22. <types> → ε
23. <types> → <type> <type_n>
24. <type_n> → ε
25. <type_n> → , <type> <type_n>
26. <type> → INTEGER
27. <type> → STRING
28. <type> → NUMBER
29. <type> → NIL
30. <assig_func> → <id_n> = <expression>
31. <assig_func> → (<expression>)

LL Tabuľka

	REQUIRE	GLOBAL	ID	CONST	:	FUNCTION	MAIN	EOF	LOCAL	=	RETURN	IF	WHILE	INTEGER	STRING	NUMBER	NIL	,	(	ε
<prol>	1																			
<prog>		2				3	4	5												
<return_func_var>					6															7
<params>			8																	9
<param_n>																		10		11
<statement>			15						13		14	16	17							12
<assign_value>										19										18
<id_n>																		21		20
<types>														23	23	23	23			22
<type_n>																		25		24
<type>														26	27	28	29			
<assig_func>																		30	31	30