VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Dokumentácia k spoločnému projektu pre predmety IFJ a IAL

Implementacia prekladača imperativného jazyka IFJ17

Tým 75, Varianta I.

2017

Zoznam autorov:

Matej Kašťák – veduci tímu	xkasta02	[45%]
Viktória Červeňanska	xcerve23	[6%]
Jozef Dráb	xdrabj00	[34%]
Christian Farkaš	xfarka06	[15%]

Obsah

1. Úvod

Táto dokumentácia obsahuje stručný návrh a implementáciu prekladača jazyka IFJ17, ktorý je zjednodušenou podmnožinou jazyka FreeBASIC. Implementovaný je v jazyku C.

Zadaný interpreter zo zdrojového súboru načíta príkazy a kontroluje chyby v následnom poradí. Kontrola lexikálna, syntaktická a zároveň sémantická. V prípade odhalenia chyby zadaný interpret vypíše chyby na štandardný chybový výstup a ukončí svoje vykonávanie s návratovou hodnotou danej chyby. Popísane sú aj použité algoritmy, popis práce na projekte, testovanie a implementácia modulov. Obsahom tejto dokumentácie je aj diagram konečného automatu znázorňujúci deterministický konečný automat, LL-gramatiku a precedenčnú tabuľku, ktoré boli použité v syntaktickej analýze.

2. Implementácia modelov a algoritmov

2.1 Lexikálna analýza

Lexikálna analýza bola implementovaná v podobe konečného automatu, pričom začína počiatočným stavom závislá načítaným znakom zdrojového súboru. Po tomto stave sa lexikálny analyzátor posunie o jedno písmeno do ďalšieho stavu, pričom sa kontroluje správnosť daného lexému.

V prípade načítaného irelevantného znaku, ktorý nepatrí do aktualného lexému, alebo sa jedná o neočakávaný znak, analyzátor vráti chybu, pričom je program ukončený návratovou hodnotou 1.

Po spracovaní korektného lexému je lexém zapísaný v štruktúre dát tokenu, ktorý je predaný syntaktickému analyzátoru k syntaktickej a sémantickej analýze.

2.2 Syntaktická a sémantická analýza

Syntaktický analyzátor volá funkciu get_token(), ktorá je implementovaná v lexikálnom module. Daná funkcia predá syntaktickému analyzátoru token získaný zo zdrojového súboru.

Identifikátory premenných, funkcií a parametrov funkcií sú ukladané do rekurzívnych binárnych stromov. Lokálne premenné tela programu, parametre funkcií a premenné funkcií sa spolu s informáciami ukladajú do lokálneho stromu.

Jazykové konštrukcie sú overované pomocou rekurzivného zostupu. Výrazy pomocou precedenčnej sémantickej analýzy. Behom sémantickej analýzy dochádza ku kontrole typov premenných, návratových hodnôt funkcií

a porovnávanie dátových typov parametrov funkcií. V prípade bezchybnej syntaktickej a sémantickej analýzy dochádza ku generovaniu trojadresného kódu, ktorý inštrukcie ukladá do inštrukčného listu.

2.3 Implementácia binárnych stromov

Binárne stromy sú implementované ako stromy s radenými položkami. Vkladanie prvkov do stromu je implementované podľa názvu identifikátora a to alfabeticky. Obsahom jedného uzlu je názov identifikátora, dátový typ identifikátora, ukazateľ na ľavý a pravý podstrom, informácia či je daný identifikátor funkcia a či bol identifikátor inicializovaný.

K vyhľadávaniu a slúži kľúč, ktorým je názov identifikátora funkcie alebo premennej. Použité boli dva stromy, globálny a lokálny. V globálnom strome sú behom programu ukladané globálne premenné a funkcie, ktoré navyše majú v štruktúre uzlu ukazateľ na dvojsmerne viazaný lineárny zoznam parametrov.

3. Práca na projekte

Na tímových stretnutiach sa diskutovalo o implementácii modulov, návrhu a rozdelenie práce. V priebehu implementácie sa tím niekoľko krát stretol k diskutovaniu o vývoji, problematike a testovaniu a prípadných zmenách v implementácii.

V prípade nevyhovujúcej možnosti sa stretnúť náš tím komunikoval za pomoci spoločných chatov ako Discord. Plánovanie a rozvrhnutie sa uskutočňovalo za pomoci Trello planning. K zálohovaniu súborov a sledovaniu jednotlivých verzií sa využíval GitHub.

4. Testovanie

Každý modul behom implementácie bol testovaný a vylepšovaný. Každý modul bol testovaný najprv zvlášť, neskôr s ostatnými modulmi. Po odkazovaní chýb na daný modul bol modul zbavený chýb a následne znovu otestovaný.

Po dokončení všetkych modelov a zostavení projektu bol project testovaný ako celok.

5. Rozdelenie práce

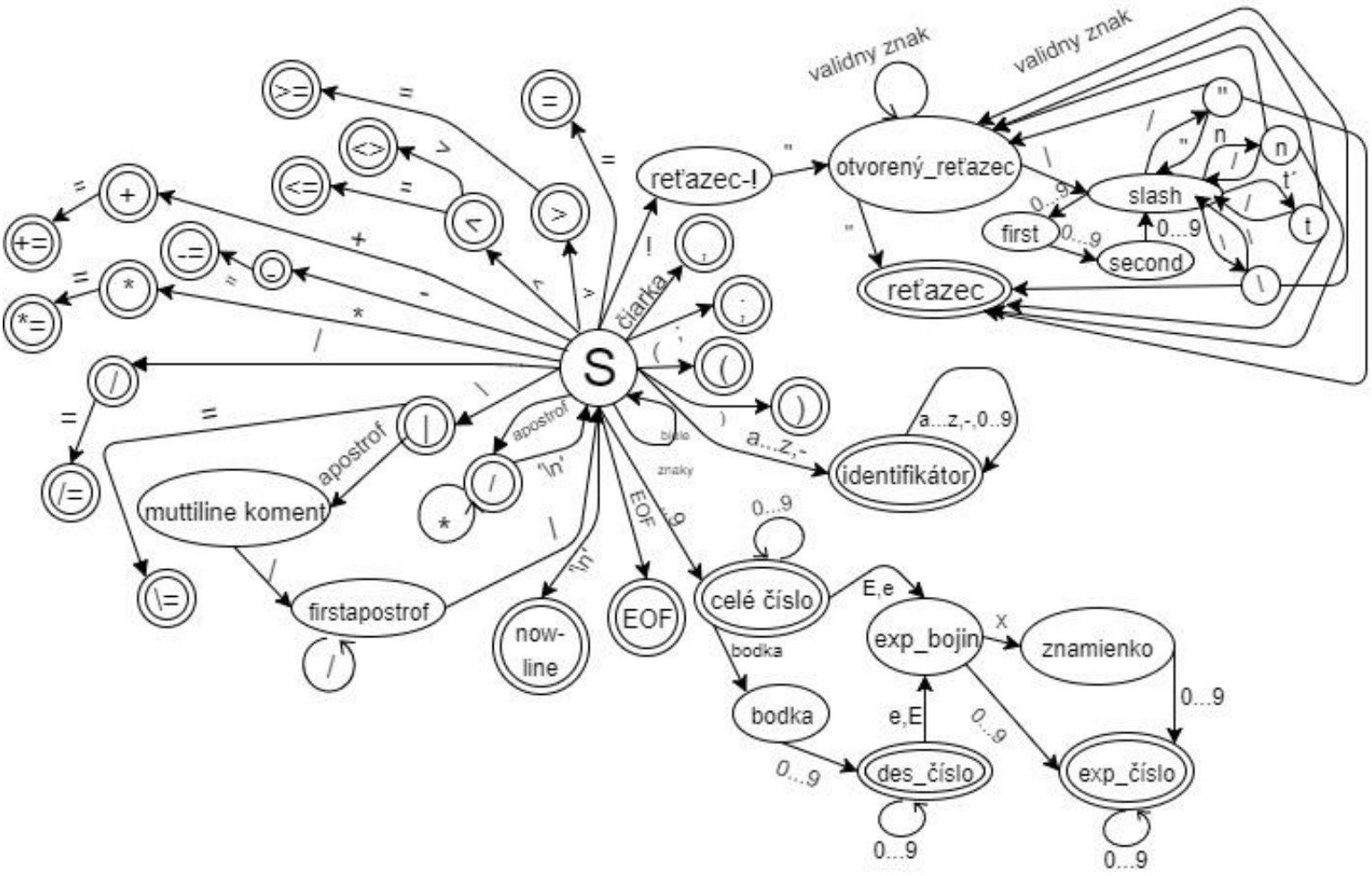
Matej Kašťák (xkasta02) – vedúci tímu: lexikálny analyzátor, syntaktický analyzátor, sémantický analyzátor, generovanie trojadresného kódu, administrácia, návrh
Jozef Dráb (xdrabj00): lexikálny analyzátor, binárne stromy, pomocné funkcie syntaktického analyzátora, dokumentácia
Viktoria Červeňanská (xcerve23): dokumentácia, digitalizácia konečného automatu

Christian Farkaš (xfarka06): pomocné funkcie pre generovanie trojadresného kódu
Body boli rozdelené podľa zdieľaného dokumentu, pričom jednotlivé moduly boli percentuálne hodnotené váhou obtiažnosti.

6. Záver

Týmto projektom sme si overili svoje znalosti v programovaní jazyku C a mnoho iných. Mohli sme si aplikovať nové teoretické znalosti v oblasti predmetu IFJ a IAL. Práca na tomto projekte nás taktiež obohatila o skúsenosť pri práci v malom tíme a komunikáciu medzi ostatnými členmi tímu.

7. Diagram konečného automatu



8. Precedená tabuľka

	+	-	*	/	\	neg	=	>	<	>=	<=	<>	(	)	function	,	var	$
+	>	>	<	<	<	<	>	>	>	>	>	>	<	>	<	>	<	>
-	<	>	<	<	<	<	>	>	>	>	>	>	<	>	<	>	<	>
*	>	>	>	>	>	<	>	>	>	>	>	>	<	>	<	>	<	>
/	>	>	>	>	>	<	>	>	>	>	>	>	<	>	<	>	<	>
\	>	>	>	>	>	<	>	>	>	>	>	>	<	>	<	>	<	>
neg	>	<	<	<	<	<	<	<	<	<	<	<	<	>	<	>	<	>
=	<	<	<	<	<	<	>	>	>	>	>	>	<	>	<	>	<	>
<	<	<	<	<	<	<	>	>	>	>	>	>	<	>	<	>	<	>
>	<	<	<	<	<	<	>	>	>	>	>	>	<	>	<	>	<	>
>=	<	<	<	<	<	<	>	>	>	>	>	>	<	>	<	>	<	>
<=	<	<	<	<	<	<	>	>	>	>	>	>	<	>	<	>	<	>
<>	<	<	<	<	<	<	>	>	>	>	>	>	<	>	<	>	<	>
(	<	<	<	<	<	<	<	<	<	<	<	<	<	=	<	=	<	E
)	>	>	>	>	>	<	<	<	<	<	<	<	E	>	E	>	E	>
function	E	E	E	E	E	E	E	E	E	E	E	E	=	E	E	E	E	E
,	<	<	<	<	<	<	<	<	<	<	<	<	<	=	<	=	<	E
var	>	>	>	>	>	>	>	>	>	>	>	>	E	>	>	>	E	>
$	<	<	<	<	<	<	<	<	<	<	<	<	<	E	<	>	<	>

9. LL-tabulka

	declare	function	scope	id	)	,	dim	=	eol	input	print	if	do	return	else	end	loop	E	String	Double	Integer
PROG	1	1	1																		
DECLPART	2	2	3																		
MAINSCOPE			4																		
FUNCTION	5	6	7																		
DEFLIST				8	8																
DEFLISTCOMMA					11	10															
DEFINE				12																	
DEFINELOCAL							13														
INIT								14	15												
SLIST				16						16	17	16	16	16	17	17	17				
SMT							26				20	20	24	25							
VYPIS									26	19								25			
TYP																			29	30	31

10. LL-tabulka

1. PROG	->	DECLPART MAINSCOPE EOF
2. DECLPART	->	FUNCTION DECLPART
3. DECLPART	->	□
4. MAINSCOPE	->	scpoe SMT end scope
5. FUNCTION	->	declare function_keyword id (DEFLIST) asi typ EOL
6. FUNCTION	->	function_keyword id (DEFLIST) as typ EOL SLIST end function
7. FUNCTION	->	□
8. DEFLIST	->	DEFINE DEFLISTCOMMA
9. DEFLIST	->	□
10. DEFLISTCOMMA	->	,D DEFLISTCOMMA
11. DEFLISTCOMMA	->	□
12. DEFINE	->	id as ty
13. DEFINELOCAL	->	dim D INIT
14. INIT	->	
15. INIT	->	□
16. SLIST	->	SMT EOL SLIST
17. SLIST	->	□
18. SMT	->	id = E
19. SMT	->	input id
20. SMT	->	print E; VYPIS
21. SMT	->	if E then EOL SLIST else EOL SLIST end if
22. SMT	->	do while E EOL SLIST loop
23. SMT	->	return E
24. SMT	->	DEFINELOCAL
25. VYPIS	->	E;
26. VYPIS	->	□
27. TYP	->	String
28. TYP	->	Double
29. TYP	->	Integer