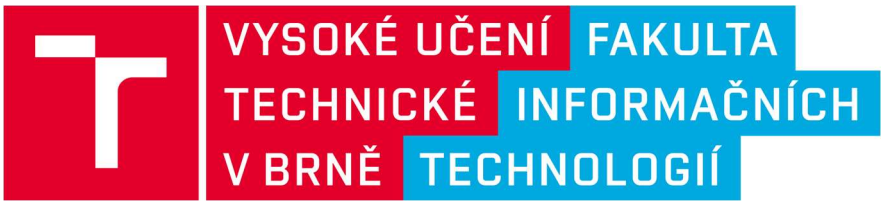


Projektová dokumentácia

Implementácia prekladača imperatívneho jazyka IFJ22

Tím xduric06, varianta TRP

Matúš Ďurica – xduric06 – 25%

Ivan Mahút – xmahut01 – 25%

Dušan Slúka – xsluka00 – 25%

Gabriela Paganíková – xpagan00 – 25%

Úvod

Cieľom tohoto projektu bolo vytvoriť prekladač z jazyka IFJ22 do jazyka IFJcode22 (medzikód). Jazyk IFJ22 reprezentuje zjednodušenú verziu skriptovacieho jazyka PHP. Prekladač funguje ako konzolová aplikácia, ktorá načíta zdorjový program zo štandardného vstupu (stdin) a generuje medzikód na štandardný výstup (stdout). V prípade chyby sa vypíše chybové hlásenie na stderr a vráti sa chybový kód.

1. Implementácia

1.1. Lexikálna analýza

Je obsiahnutá v zdrojovom súbore scanner.c a hlavičkovom súbore scanner.h. Táto analýza ako prvá spracováva zdrojový súbor v jazyku IFJ22 a jej úlohou je správne rozlíšiť medzi typmi lexémov (ďalej už len tokenov).

Pre tvorbu jednotlivých tokenov sme začali návrhom deterministického konečného automatu, ktorý je implementovaný v súbore scanner.c. Stavy prepína na základe premennej transition, do ktorej načítavame nasledujúci znak po momentalne spracovanom.

Komunikácia s lexikálnou analýzou je prostredníctvom funkcie generate_token(), ktorá vracia dátovú štruktúru TOKEN. Táto štruktúra obsahuje hodnotu daného tokenu, ak sa jedná o konštantu, konečný stav a názov, ak sa jedná o premennú. Funkcia vždy vráti jeden token.

Chyby, ktoré sa vyskytnú počas lexikálnej analýzy, sú lexikálne a pri nájdení chyby scanner vracia špeciálny špeciálny chybový token. Niektoré chyby však môžu byť aj syntaktické napr. chýbajúci prolog.

Problémy s ktorými sme sa stretli pri implementácií boli rozsiahlosť a krajné prípady pre jednotlivé stavy, ktoré mohli nastať. Pôvodný automat sme kvôli tomuto museli niekoľkokrát doplniť. Jedným prípadom bol napr. viacriadkový komentár, ktorý mohol pohltiť celý zdrojový súbor a chýbalo prepnutie do chybového stavu ERROR, pokiaľ bol načítaný koniec súboru.

1.2. Syntaktická analýza

Syntaktickú analýzu sme implementovali pomocou metódy rekurzívneho volania. Podkladom bola LL-gramatika (Obrázok 3). Syntaktická analýza je implementovaná v súboroch parser.c a parser.h a riadi celý chod prekladu. S lexikálnou analýzou komunikuje prostredníctvom volania funckie generate_token(), ktorá, ako bolo vyššie spomenuté, vygeneruje nový token, ktorý je následne overený a na základe prijatého tokenu sa program rozhoduje, ktoré pravidlo použiť. Niektoré funkcie zlučujú niekoľko pravidiel gramatiky napr. funkcia stat() zlučuje pravidlá 16. až 22.

1.2.1. Precedenčná analýza

Precedenčná analýza je volaná v prípade, že parser zistí, že za priradením sa nevyskytuje premenná či konštanta. Výraz je následne spracovaný na pravý rozbor a pri použití pravidiel redukcie je generovaný korešpondujúci kód IFJcode22. Tokeny boli postupne spracovávané, určoval sa ich typ a vzťah medzi tokenom na vrchole zásobníku a tokenom na vstupe pomocou navrhnutej precedenčnej tabuľky. Ak bolo možné použiť pravidlo na redukciu, bolo aplikované. Precedenčná analýza je implementovaná v súboroch precedence.c, precedence.h, stack.c a stack.h

1.2.2. Sémantická analýza

Sémantická analýza je obsiahnutá v súboroch sematic.c a semantic.h. Prvým krokom pri vytváraní funkcií pre analýzu, bola sumarizácia všetkých pravidiel, ktoré musia byť dodržané. Z každej triedy je vracaný korešpondujúci chybový kód. Jednou z ťažších častí sémantickej analýzy nebola jej tvorba, ale správne volanie jej funkcií v iných častiach kódu – najmä v syntaktickej analýze. V precedenčnej analýze sa sémantická analýza využíva taktiež, no nevolajú sa jej funkcie. Namiesto nich, sa pri detekcii chyby vracia korešpondujúci chybový kód.

1.3. Tabuľka symbolov

Variantu TRP sme si vybrali, pretože nám to prišlo ako rozumnejšia voľba, keďže podporuje vyhľadávanie podľa zahashovaného identifikátoru premennej/funkcie.

Ako hashovací algoritmus sme zvolili C port algoritmu murmurhash3. Tento algoritmus sme zvolili na základe rýchlosti, jedinečnosti a faktu, že sme nepotrebovali bezpečnostné vlastnosti, ktoré ponúka napr. algoritmus SHA-256. Tento algoritmus je implementovaný v súboroch murmurhash.c a murmurhash.h. Algoritmus sme museli jemne poupraviť, aby nám vracal hodnoty len do maximálnej veľkosti tabuľky symbolov.

Tabuľka samotná je implementovaná v súboroch symtable.c a symtable.h. Dáta v tabuľke, sú uložené v štruktúre, ktorá obsahuje dátový typ, pravdivostnú hodnotu, ktorá hovorí o tom, či je definovaná a ak sa jedná o funkciu, tak aj jednosmerne viazaný zoznam, ktorý obsahuje parametre danej funkcie.

1.4. Generovanie cieľového kódu

Funkcie a makrá generovania inštrukcií, volá parser a precedenčná analýza. Väčšina inštrukcií je generovaná pomocou makier, ktoré sa následne používajú buď priamo v parseri či v precedenčnej analýze alebo vo funkciách na generovanie väčších blokov inštrukcií. Medzi tieto funkcie patrí napr. generovanie konštánt, ktoré musia byť na stdout posielané v špeciálnych formátoch (%d pre int, %a pre float, nahradzovanie bielych znakov, #, \ a netlačiteľných znakov v stringu), generovanie hlavičky funkcie s argumentami či generovanie hlavičky celého programu. Spolu s hlavičkou programu IFJcode22 sa taktiež generujú aj vstavané funkcie. Generátor kódu je implementovaný v súboroch generator.c a generator.h.

2. Práca v tíme

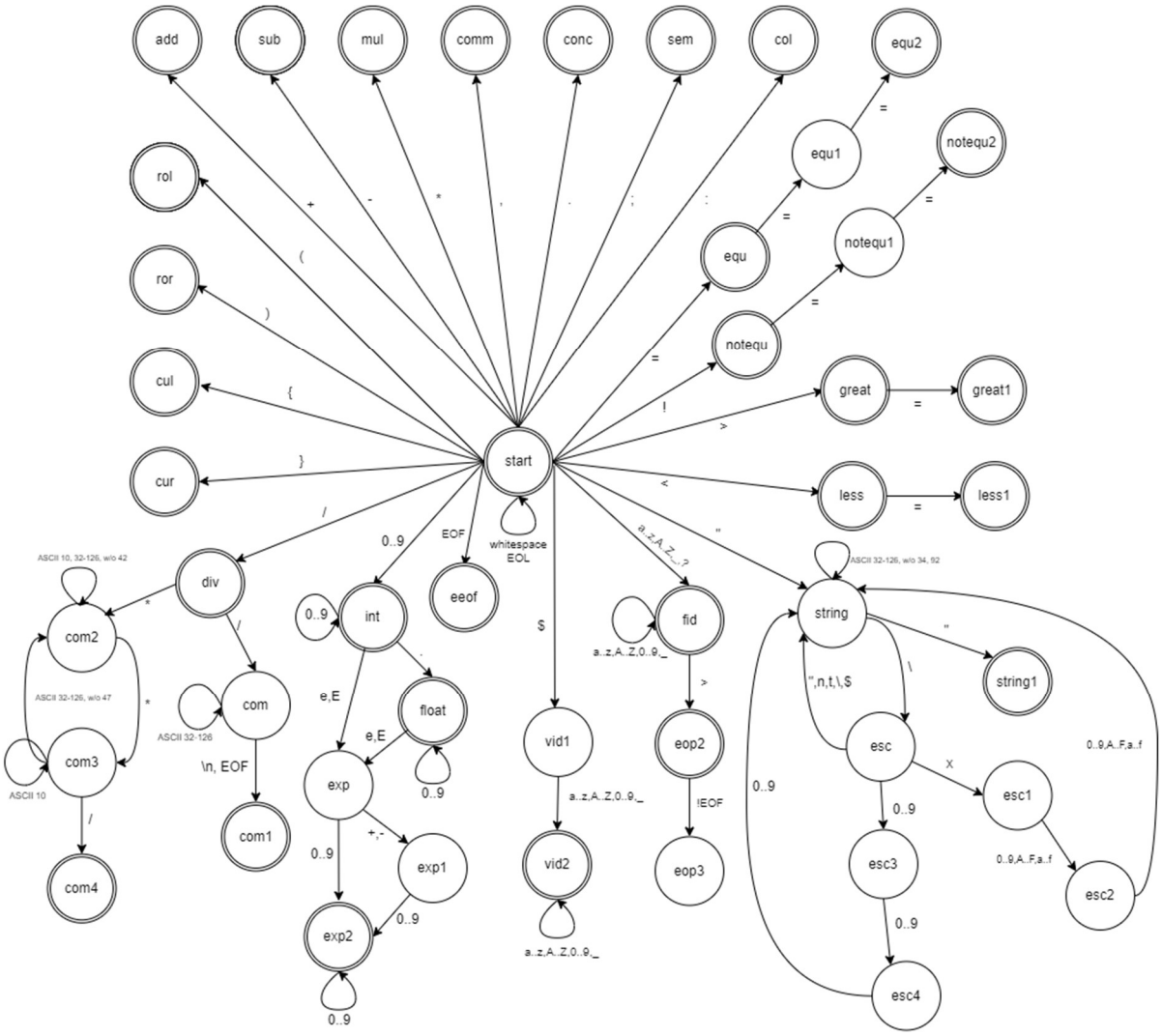
Na riadenie revizií sme použili Git a ako vzdialený repozitár sme použili GitHub. Na komunikáciu nám slúžil Discord, prípadne skupinový chat na Facebooku. Často sme sa však aj stretávali a pracovali na problémoch spoločne.

Rozdelenie práce sme mali nasledovné:

- Matúš Ďurica: tabuľka symbolov, výpomoc pri precedenčnej analýze
- Ivan Mahút: LL-gramatika, parser
- Dušan Slúka: lexikálna analýza, sémantická analýza
- Gabriela Paganíková: návrh konečného automatu a precedenčnej tabuľky, precedenčná analýza

Záver

Projekt bohužiaľ, nebol vypracovaný na 100%, nakoľko sme časovo nestíhali. Dosť problémov nastalo posledné dni, pri snahe o pochopení niektorých častiach zadania, kedy bolo neskoro pokladať otázky na forum. Týmto projektom sme získali skúsenosti v oblasti funkcionality prekladačov, ktoré môžeme využiť v budúcnosti.



Obrázok 1 - Návrh konečného automatu

	+ - .	* /	(	)	=== !==	< <= > >=	i	$
+ - .	>	<	<	>	>	>	<	>
* /	>	>	<	>	>	>	<	>
(	<	<	<	=	<	<	<	E
)	>	>	E	>	>	>	E	>
=== !==	<	<	<	>	>	<	<	>
< <= > >=	<	<	<	>	>	>	<	>
i	>	>	E	>	>	>	E	>
$	<	<	<	E	<	<	<	E

Obrázok 2 - Precedenčná tabuľka

1.<Prolog> -> <?php declare (strict_types=1); < <Program>

2.<Program> -> ?>

3.<Program> -> function function_ID (<Param>):<Ret_type>{<Statement>} < <Program>

4.<Program> -> <Statement> < <Program>

5.<Param> -> <Type> ID <Param_n>

6.<Param> -> eps

7.<Param_n> -> , <Type> <Param_n>

8.<Param_n> -> eps

9.<Type> -> int

10.<Type> -> float

11.<Type> -> string

12.<Ret_type> -> void

13.<Ret_type> -> string

14.<Ret_type> -> int

15.<Ret_type> -> float

16.<Statement> -> if(Expression) {<Statement>} <Else> < <Statement>

17.<Statement> -> while(Expression){<Statement>} < <Statement>

18.<Statement> -> $ID = <Values> < <Statement>

19.<Statement> -> function_ID(<Args>); < <Statement>

20.<Statement> -> return <ReturnVal> < <Statement>

21.<Statement> -> Expression < <Statement>

22.<Statement> -> eps

23.<Else> -> else{<Statement>}

24.<Else> -> eps

25.<Args> -> <Arg> <Args_n>

26.<Args> -> eps

27.<Args_n> -> , <Arg> <Args_n>

28.<Args_n> -> eps

29.<Arg> -> $ID

30.<Arg> -> Int

31.<Arg> -> String

32.<Arg> -> Float

33.<ReturnVal> -> Int

34.<ReturnVal> -> Float

35.<ReturnVal> -> String

36.<ReturnVal> -> $ID

37.<ReturnVal> -> eps

38.<Values> -> String

39.<Values> -> Int

40.<Values> -> Float

41.<Values> -> reads();

42.<Values> -> readi();

43.<Values> -> readf();

44.<Values> -> floatval();

45.<Values> -> intval();

46.<Values> -> strval();

47.<Values> -> strlen();

48.<Values> -> function_ID(<Args>);

49.<Values> -> Expression;

Obrázok 3 - LL-gramatika

	<?php declare (strict_types=1)	>?	function	function_ID	eps	,	int	float	string	void	if	else	while	$	return	expression	reads	readi	readf	floatval	intval	strval	strlen
<prolog>	1																						
<program>		2	3		4																		
<param>					6		5	5	5														
<param_n>					8	7																	
<type>							9	10	11														
<ret_type>							14	15	13	12													
<statement>				19	22						16		17	18	20	21							
<else>					24							23											
<args>					26		25	25	25														
<args_n>					28	27																	
<arg>							30	32	31					29									
<returnVal>					37		33	34	35					36									
<values>				48			39	40	38							49	41	42	43	44	45	46	47

Obrázok 4 - LL-tabuľka