Projektová dokumentace

Implementace překladače imperativního jazyka IFJ22

Tým xdobes22, varianta BVS

Miroslav Bálek	(xbalek02)	25 %
Kristián Dobeš	**(xdobes22)**	**25 %**
Martin Otradovec	(xotrad00)	25 %
Maroš Synák	(xsynak03)	25 %

4. prosince 2022

Obsah

1. Úvod

Úkolem bylo vytvořit program v jazyce C, který načte zdrojový kód zapsaný ve zdrojovém jazyce IFJ22 a přeloží jej do cílového jazyka IFJcode22 (mezikód). Jazyk IFJ22 je zjednodušenou podmnožinou jazyka PHP.

Překladač načítá ze standardního vstupu řídící program v jazyce IFJ22 a následně generuje mezikód v jazyce IFJcode22 na standardní výstup. V případě chyby vrací odpovídající chybový kód.

2. Návrh a implementace

Projekt se skládá z několika implementovaných částí, které budou popsány níže.

2.1 Lexikální analýza

Tvorba překladače začala implementací lexikální analýzy. Lexikální analyzátor načítá jednotlivé lexémy a převádí je na tokeny, které reprezentují daný lexém v dalších částech překladače.

Část lexikální analýzy překladače je implementována v modulu `scanner.c` pomocí deterministického konečného automatu. Celý konečný automat je jeden opakující se `switch`, kde každý `case` je jeden stav automatu. Pokud narazí automat na znak, který nezná, vrací chybu v rámci lexikální analízy a ukončí se.

Klíčová slova: `else`, `float`, `function`, `if`, `int`, `null`, `return`, `string`, `void`, `while` jsou definovaná ve scanneru jako `KEYWORD` a každý string se kontroluje, zda neobsahuje jedno z klíčových slov, pokud ano, generuje se jako `TOKEN_KEYWORD` nebo `TOKEN_TYPE` s hodnotou klíčového slova.

2.2 Syntaktická analýza a sémantická analýza

Hlavní tělo syntaktické a sémantické analýzy tvoří zdrojový souborech `parser.c` a `preparser.c`. Jedná se o implementaci metodou rekurzivního sestupu podle pravidel LL - tabulky a LL - gramatiky. Pro každou skupinu pravidel je implementována vlastní funkce. V našem případě se nejedná o LL(1) gramatiku. Při kontrole pravidel u kterých nastal konflikt, se použije funkce `peek_top`, která vrací nadcházející token a to nám umožní volbu korektního pravidla.

Hlavičky funkcí jsou zpracovány dvojím průchodem k umožnění rekurzivního volání. Při prvním průchodu, který provádí `preparser.c`, se do tabulky symbolů doplní definice funkcí, které v daném kódu existují.

2.3 Precedenční syntaktická analýza

Výrazy jsou zpracovány precedenční syntaktickou analýzou implementovanou ve zdrojovém souboru `precedence_expression.c`. Operátory se stejnou prioritou (`+`,`-` a `*`,`/`) byly sjednoceny do jednoho sloupce a řádku.

Pro usnadnění práce s tabulkou je implementována funkce `get_pos_in_t`, která přijímá jako parametr symbol a vrací jeho index v tabulce. Mezi povolené symboly patří pouze symboly uvedené v precedenční tabulce. Symboly, které výraz obsahovat nemůže, jsou v tabulce zastoupeny symbolem `;`. Řádky tabulky reprezentují terminály na vrcholu zásobníku a sloupce symbol v aktuálním tokenu.

2.4 Generování cílového kódu

Generování cílového kódu je implementováno v zdrojových souborech `parser.c`, `precedence_expression.c`, přičemž vestavěné funkce jsou definovány v hlavičkovém souboru `instruction.h`. V hlavičkovém souboru `precedence.h` se nachází pomocná makra pro generaci podmínek a výpočtů ve výrazech. Úkolem této části je vytvářet cílový mezikód IFJcode22.

3. Tabulka symbolů a datové struktury

Tabulka symbolů je implementována jako binární vyhledávací strom v modulu `symtable.c`. Každý uzel stromu obsahuje identifikátor funkce/proměnné. Funkce obsahuje informace zda byla funkce zavolána, definována, obsahuje proměnný počet, má návratovou hodnotu, počet parametrů funkce, ukazatel na tyto parametry, návratová hodnota funkce může být `null` a zdali se jedná o funkci vestavěnou.

Funkce pro práci s tabulkou symbolů jsou implementovány ve zdrojovém souboru `symtable.c`. Pro výpis tabulky symbolu je v souboru `symtable.h` definována funkce `sym_print`.

4. Práce v týmu

4.1 Způsob práce v týmu

Na projektu jsme začali pracovat koncem října, ale nenaplánovali jsme si práci dopředu, takže jsme měli období, kdy jsme na projektu pracovali, a období, kdy jsme se projektu příliš nevěnovali. Většinou začal jeden člověk pracovat na nějaké části a postupně se k němu připojovali další lidé, kteří mu pomáhali. Rozdělení práce také nebylo naplánováno.

4.2 Verzovací systém

Jako verzovací systém jsme zvolili Git, jelikož s ním máme nejvíce zkušeností.

4.3 Komunikace

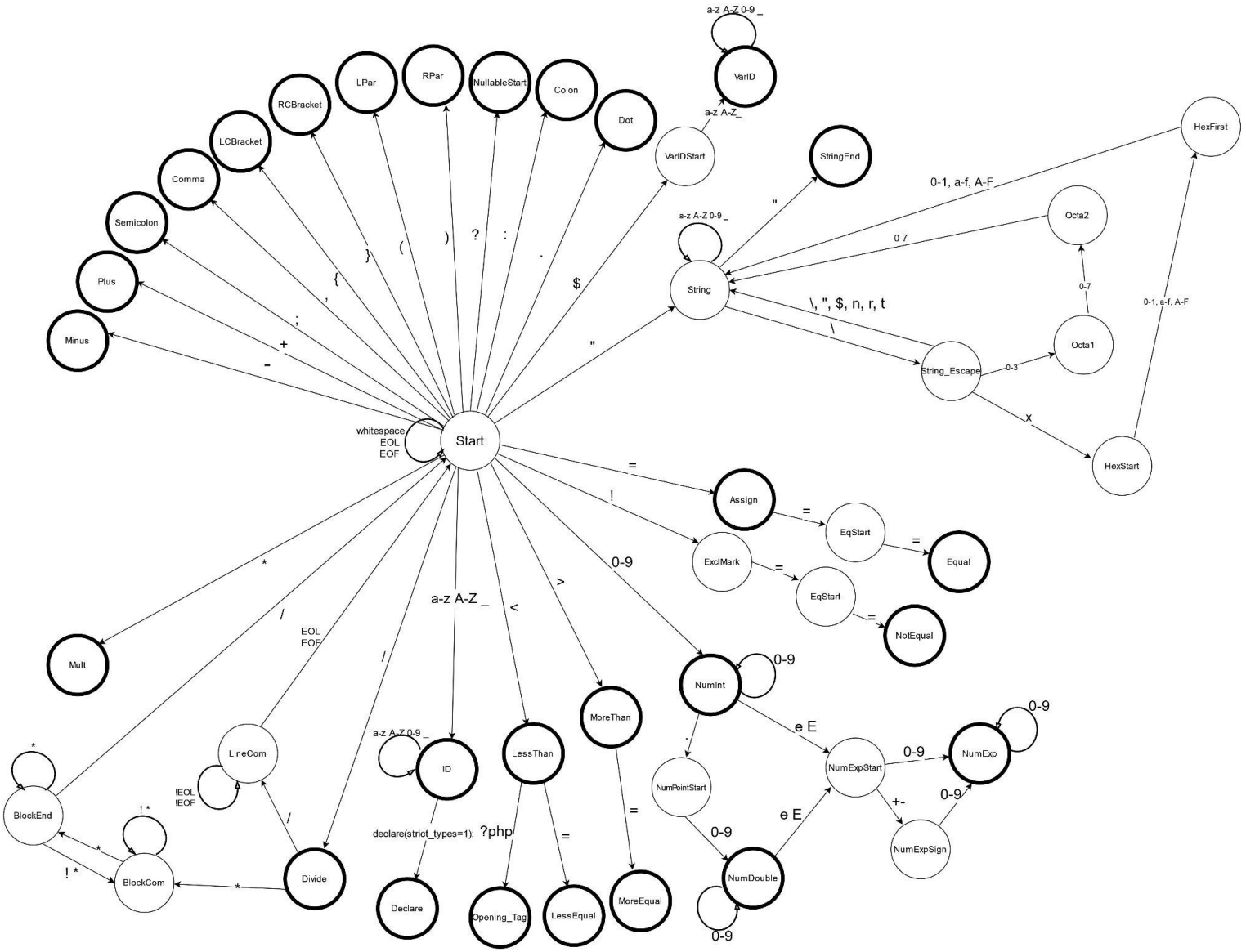
Komunikace mezi členy týmu probíhala výhradně přes Discord, buď písemně nebo v podobě skupinového hovoru.

4.4 Rozdělení práce mezi členy týmu

Práci na projektu jsme si nijak nerozdělili, takže jak je již zmíněno výše, každý se částečně podílel na více částech.

Člen týmu	**Práce na projektu**
Miroslav Bálek	návrh konečného automatu, lexikální analyzátor, tvorba gramatiky, sémantická analýza, generování kódu, testování, dokumentace
Kristián Dobeš	návrh konečného automatu, lexikální analyzátor, tvorba gramatiky, sémantická analýza, testování, dokumentace
Martin Otradovec	návrh konečného automatu, lexikální analyzátor, tvorba gramatiky, syntaktická analýza, sémantická analýza, generování kódu, testování, dokumentace, struktura projektu, kontrola kódu
Maroš Synák	návrh konečného automatu, lexikální analyzátor, tvorba gramatiky, syntaktická analýza, sémantická analýza, generování kódu, testování, dokumentace

Diagram konečného automatu



LL-gramatika

1. <prog> -> <?php declare(strict_types=1); <statement-list> <prog-end>
2. <prog-end> -> ε
3. <prog-end> -> ?>
4. <statement-list> -> <statement> <statement-list>
5. <statement-list> -> ε
6. <statement> -> var_id = <expression>;
7. <statement> -> var_id = id(<arg-list>);
8. <statement> -> id(<arg-list>);
9. <statement> -> function id(<par-list>) : type { <statement-list> }
10. <statement> -> <expression>;
11. <statement> -> return <expression>;
12. <statement> -> while(<expression>){ <statement-list> }
13. <statement> -> if (<expression>) { <statement-list> } else { <statement-list> }
14. <expression> -> var_id
15. <expression> -> const_int
16. <expression> -> string_lit
17. <expression> -> const_float
18. <expression> -> (<expression>)
19. <expression> -> <expression> <expression-next>
20. <expression> -> id(<arg-list>)
21. <expression-next> -> * <expression>
22. <expression-next> -> / <expression>
23. <expression-next> -> - <expression>
24. <expression-next> -> + <expression>
25. <expression-next> -> . <expression>
26. <expression-next> -> < <expression>
27. <expression-next> -> > <expression>
28. <expression-next> -> <= <expression>
29. <expression-next> -> >= <expression>
30. <expression-next> -> === <expression>
31. <expression-next> -> !== <expression>
32. <arg-list> -> <expression> <arg-next>
33. <arg-list> -> ε
34. <arg-next> -> ,<expression> <arg-next>
35. <arg-next> -> ε
36. <par-list> -> <par> <par-next>
37. <par-list> -> ε
38. <par-next> -> ,<par> <par-next>
39. <par-next> -> ε
40. <par> -> type var_id

LL-tabulka

	<?php	declare(strict_types=1);	ε	?>	var_id	`=	<expression>;	id(	);	function	)	:	type	{	}	return	while(<expression>){	if	(<expression>)	else	const_int	string_lit	const_float	(	*	`/	`-	`+	.	<	>	<=	>=	`===	!===	,<expression>	,<par>	$
<prog>	1																																					
<prog-end>			2	3																																		
<statement-list>			5		4		4	4		4						4	4	4																				
<statement>					6,7		10	8		9						11	12	13																				
<expression>					14,19			19,2													15,19	16,19	17,19	18,19														
<expression-next>																									21	22	23	24	25	26	27	28	29	30	31			
<arg-list>			33		32			32													32	32	32	32	32													
<arg-next>			35																																	34		
<par-list>			37										36																									
<par-next>			39																																		38	
<par>													40																									

Precedenční tabulka

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