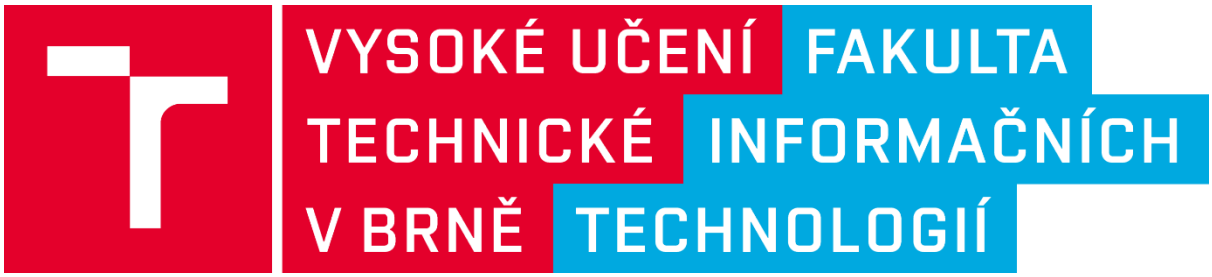


Implementácia prekladača imperatívneho jazyka IFJ22

Tým xkniaz00, varianta BVS

Daniil Kniazkin	(xkniaz00)	25% - vedúci tímu
Kotvitskiy Nikita	(xkotvi01)	25%
Hrubý Erik	(xhruby30)	25%
Hricovová Alžbeta	(xhrico00)	25%

7.12.2022

Obsah

1. Úvod

Cieľom projektu bolo vytvoriť program v jazyku C, ktorý načíta zdrojový kód zapísaný v zdrojovom jazyku IFJ22 a preloží ho do cieľového jazyka IFJcode22. Jazyk IFJ22 je zjednodušenou podmnožinou jazyka PHP. Zdrojový kód je načítavaný zo štandardného vstupu a výsledný kód sa generuje na štandardný výstup.

2. Implementačné riešenie

V tejto kapitole sú uvedené stručné popisy jednotlivých častí implementácie a ich vzájomná spolupráca.

2.1. Lexikálna analýza

Prvou časťou prekladača bolo vytvorenie skenera, ktorý má za úlohu rozpoznávať jednotlivé lexikálne jednotky a prevádzať ich na príslušné tokeny. Implementácia sa nachádza v súboroch *scanner.c* a *scanner.h* a je realizovaná deterministickým konečným automatom.

Hlavnou funkciou skeneru je *readProgram*, ktorá opakovane volá funkciu *getToken*, pokiaľ nenarazí na token značiaci koniec súboru alebo token uzatváracej značky ?>. *getToken* ďalej opakovane volá funkciu *processChar,* ktorá obsahuje konštrukciu switch, kde každý case predstavuje stav automatu. Pre odlíšenie kľúčového slova od identifikátorov hľadáme jeho výskyt v tabuľke kľúčových slov realizovanej ako binárny vyhľadávací strom.

Štruktúra tokenu je nasledovná:

type - určuje druh tokenu
textData - rôzne textové údaje o tokene(napr. meno premennej, identifikátor funkcie...)
numericData - int/float hodnota tokenu

Ďalšie časti prekladača pracujú so štruktúrou *program_t,* ktorá obsahuje zoznam a počet tokenov naplnených v *readProgram*.

2.2. Syntaktická analýza

V súboroch *parser.c* a *parser.h* sa nachádza syntaktická analýza, ktorá sa volá cez funkciu *parseProgram* a je typu zhora-dole. Analýza je založená na syntaktických pravidlách popísaných pomocou LL gramatiky. Pravidlá sa načítavajú v súbore *data.c* zo súboru *rules.txt* a ukladajú sa vo formáte štruktúr typu *rule_t* do binárneho vyhľadávacieho stromu. Analýza sa vždy začína spracovaním pravidla <*prog*>, ktoré sa nachádza na vrchole BVS. Niektoré pravidlá majú viacero variant. Varianty sa spracovávajú vo funkcii *checkRuleVariant*. Ak niektorý token programu nezodpovedá žiadnej variante pravidla, jedná sa o syntaktickú chybu. Syntaktická analýza je ukončená po zpracovaní pravidla <*prog*>.

2.3. Sémantická analýza

Sémantická analýza sa nachádza v súboroch sematic.c a semantic.h a je riešená dvomi priechodmi. Každý priechod je realizovaný ako deterministický konečný automat cez zoznam tokenov. Funkcia *getFunTable* vykonáva prvý priechod, ktorý naplní tabuľku symbolov *funcTable* pre hlavné telo programu symbolmi pre deklarované funkcie.

Funkcia *semanticControl* vykonáva druhý už úplný priechod. Pre telo definovaných funkcií sa vytvára nová lokálna tabuľka symbolov *localTable.*

Funkcia *funCallToken* kontroluje volanie funkcie – či existuje funkcia v tabuľke symbolov *funcTable*, správny typ a počet parametrov.

Výraz sa spracováva v súboroch *expr.c* a *expr.h* ako precedenčná syntaktická analýza.

Funkcia *varToken* slúži na uloženie konečného typu výrazu do premennej a následne sa aktualizuje/uloží premenná do tabuľky symbolov.

2.4. Generovanie jazyka

V súboroch *generator.c* a *generator.h* sa nachádza generovanie jazyka IFJcode22. Generovanie sa začína volaním funkcie *generateProgram* a skladá sa z troch krokov.

V prvom kroku sa generuje kostra programu, ktorá sa skladá z povinného úvodného riadku, definícií globálnych pomocných premenných a definícií vstavaných funkcií.

V druhom kroku sa vykonáva priechod celým programom a preklad definícií užívateľských funkcií (funkcia *functionDef*). Všetky funkcie sa prekladajú takým spôsobom, že ich návratové hodnoty sa vkladajú na vrchol zásobníka.

Tretí krok je ďalším priechodom celého programu, v ktorom sa vykonáva preklad hlavného tela programu. Pre každú konštrukciu jazyka IFJ22 existuje odpovedajúca funkcia. Generátor zároveň rieši implicitnú konverziu typov vo výrazoch pomocou univerzálnych konštrukcií predpísaných v jazyku IFJcode22, ktoré sú uložené v súbore *generator.h* ako makrá.

2.5. Tabuľka symbolov – varianta BVS

Tabuľka symbolov je implementovaná v súboroch *symtable.c* a *symtable.h* a je riešená ako binárny vyhľadávací strom, kde kľúče prvkov sú identifikátori funkcií alebo premenných zahashované pomocou *Jenkins hash function* [1].

Štruktúra symbolu tabuľky je nasledovná:

type - určuje či je to symbol premennej alebo funkcie
dtype - určuje typ pre symbol premennej a informácie o návratovom type a parametroch pre symbol funkcie

3. Rozdelenie práce medzi členmi tímu

Práca bola rozdelená rovnomerne, preto každý dostal 25% hodnotenia práce. Riešenie každej časti projektu bolo konzultované medzi všetkými členmi tímu.

- Daniil Kniazkin : vedenie tímu, tabuľka symbolov, generovanie kódu, testovanie
- Kotvitskiy Nikita : lexikálna analýza, syntaktická analýza, generovanie kódu
- Hrubý Erik : dokumentácia, sémantická analýza
- Hricovová Alžbeta : dokumentácia, sémantická analýza

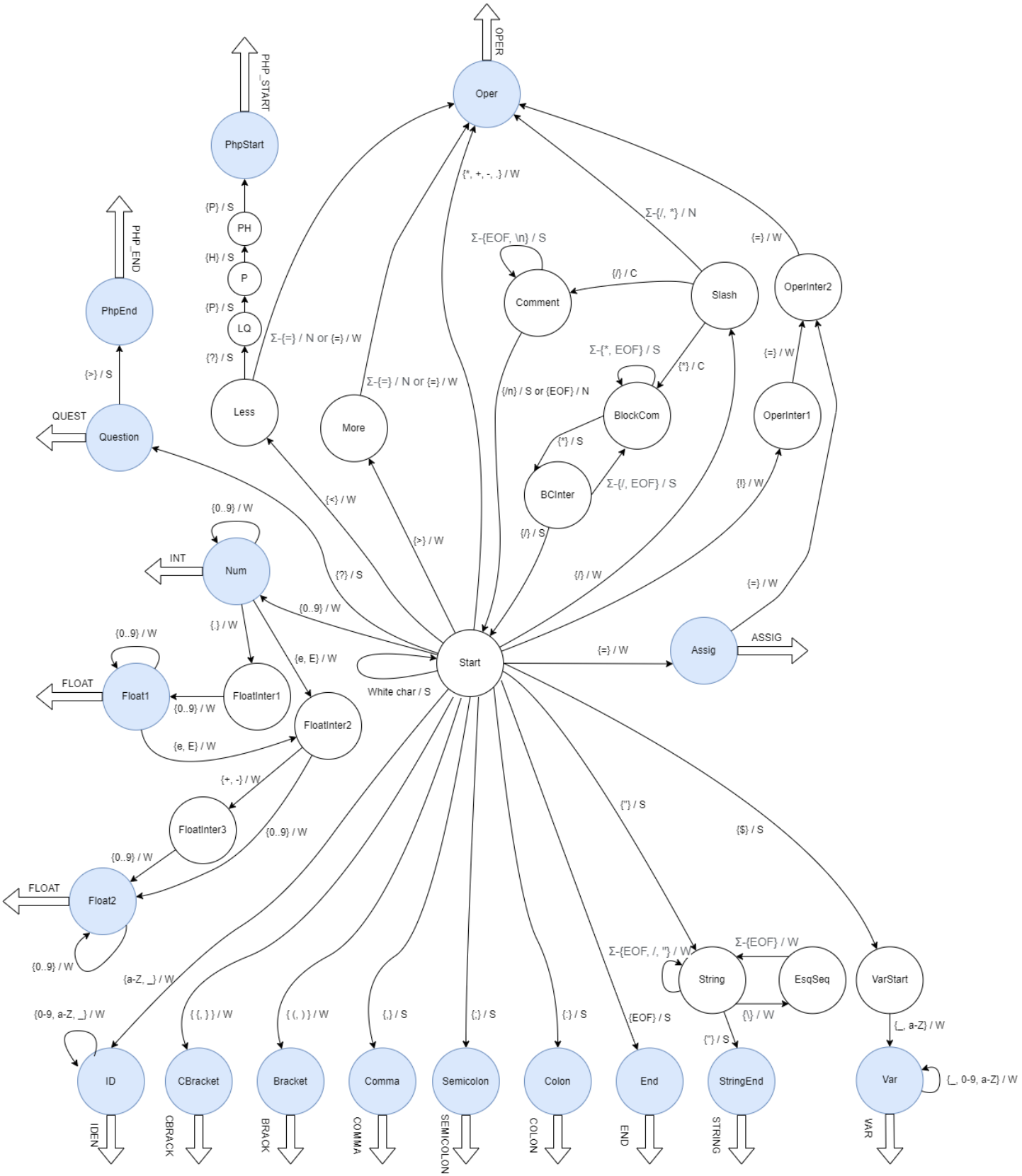
4. Zdroje

[1] Jenkins hash function, https://en.wikipedia.org/wiki/Jenkins_hash_function

[2] Formální jazyky a překladače, http://www.fit.vutbr.cz/study/courses/IFJ/public/materials/

[3] PHP manuál, https://www.php.net/manual/en/index.php

5. Diagram deterministického konečného automatu pre lexikálnu analýzu



6. LL gramatika

```
1   <prog>              -> <begin> <st-list>                                        |   |
2   <begin>             -> START declare ( strict_types = 1 ) ;                     |   |
3   <st-list>           -> EOF                                                      |   |
4   <st-list>           -> ?>                                                       |   |
5   <st-list>           -> return <return>                                          |   |
6   <st-list>           -> <prog-block> <st-list>                                   |   |
7   <prog-block>        -> varID = <assig>                                          |   |
8   <prog-block>        -> funID ( <params>                                         |   |
9   <prog-block>        -> if ( <expr-br> { <constr-st-list> else { <constr-st-list>|   |
10  <prog-block>        -> while ( <expr-br> { <constr-st-list>                     |   |
11  <prog-block>        -> function funID ( <dec-params> : <func>                   |   |
12  <prog-block>        -> <expr>                                                   |   |
13  <dec-params>        -> )                                                        |   |
14  <dec-params>        -> type varID <dec-params-2>                                |   |
15  <dec-params>        -> ? type varID <dec-params-2>                              |   |
16  <dec-params-2>      -> , type varID <dec-params-2>                              |   |
17  <dec-params-2>      -> , ? type varID <dec-params-2>                            |   |
18  <dec-params-2>      -> )                                                        |   |
19  <func>              -> ? type { <constr-st-list>                                |   |
20  <func>              -> type { <constr-st-list>                                  |   |
21  <constr-st-list>    -> }                                                        |   |
22  <constr-st-list>    -> <constr-block> <constr-st-list>                          |   |
23  <constr-block>      -> varID = <assig>                                          |   |
24  <constr-block>      -> funID ( <params>                                         |   |
25  <constr-block>      -> if ( <expr-br> { <constr-st-list> else { <constr-st-list>|   |
26  <constr-block>      -> while ( <expr-br> { <constr-st-list>                     |   |
27  <constr-block>      -> return <return>                                          |   |
28  <constr-block>      -> <expr>                                                   |   |
29  <return>            -> ;                                                        |   |
30  <return>            -> <expr>                                                   |   |
31  <assig>             -> funID ( <params>                                         |   |
32  <assig>             -> <expr>                                                   |   |
33  <params>            -> ) ;                                                      |   |
34  <params>            -> <params-2> ;                                             |   |
35  <params-2>          -> const <params-3>                                         |   |
36  <params-2>          -> varID <params-3>                                         |   |
37  <params-2>          -> null <params-3>                                          |   |
38  <params-3>          -> )                                                        |   |
39  <params-3>          -> , <params-2>                                             |   |
40  <expr>              -> const <expr-2>                                           |   |
41  <expr>              -> varID <expr-2>                                           |   |
42  <expr>              -> null <expr-2>                                            |   |
43  <expr>              -> ( <expr-br> <expr-2>                                     |   |
44  <expr-2>            -> ;                                                        |   |
45  <expr-2>            -> oper <expr>                                              |   |
46  <expr-br>           -> const <expr-br-2>                                        |   |
47  <expr-br>           -> varID <expr-br-2>                                        |   |
48  <expr-br>           -> null <expr-br-2>                                         |   |
49  <expr-br>           -> ( <expr-br> <expr-br-2>                                  |   |
50  <expr-br-2>         -> )                                                        |   |
51  <expr-br-2>         -> oper <expr-br>                                           |   #
```

7. LL tabuľka

	START	declare	(	strict_types	=	1	)	;	EOF	?>	return	varID	funID	if	{	else	while	function	:	type	?	,	}	const	null	oper
<prog>	1																									
<begin>	2																									
<st-list>			6						3	4	5	6	6	6			6	6						6	6	
<prog-block>			12									7,12	8	9			10	11						12	12	
<dec-params>							13													14	15					
<dec-params-2>							18															16,17				
<func>																				20	19					
<constr-st-list>			22								22	22	22	22			22						21	22	22	
<constr-block>			28								27	23,28	24	25			26							28	28	
<return>			30					29				30												30	30	
<assig>			32									32	31											32	32	
<params>							33					34												34	34	
<params-2>												36												35	37	
<params-3>							38															39				
<expr>			43									41												40	42	
<expr-2>								44																		45
<expr-br>			49									47												46	48	
<expr-br-2>							50																			51

8. Precedenčná tabuľka

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