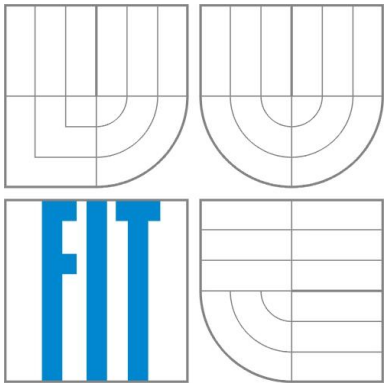


Dokumentace projektu

Implementace překladače imperativního jazyka IFJ17

Číslo týmu: 014, Varianta II

Rozšíření FUNEXP, BASE

6. prosince 2017

Vedoucí týmu:	**Adam Venger**	**(xvenge00)**	**25%**
Členové:	Peter Babka	(xbabka01)	25%
	Ondřej Doseděl	(xdosed08)	25%
	Vojtěch Randýsek	(xrandy00)	25%

Obsah

1 Úvod

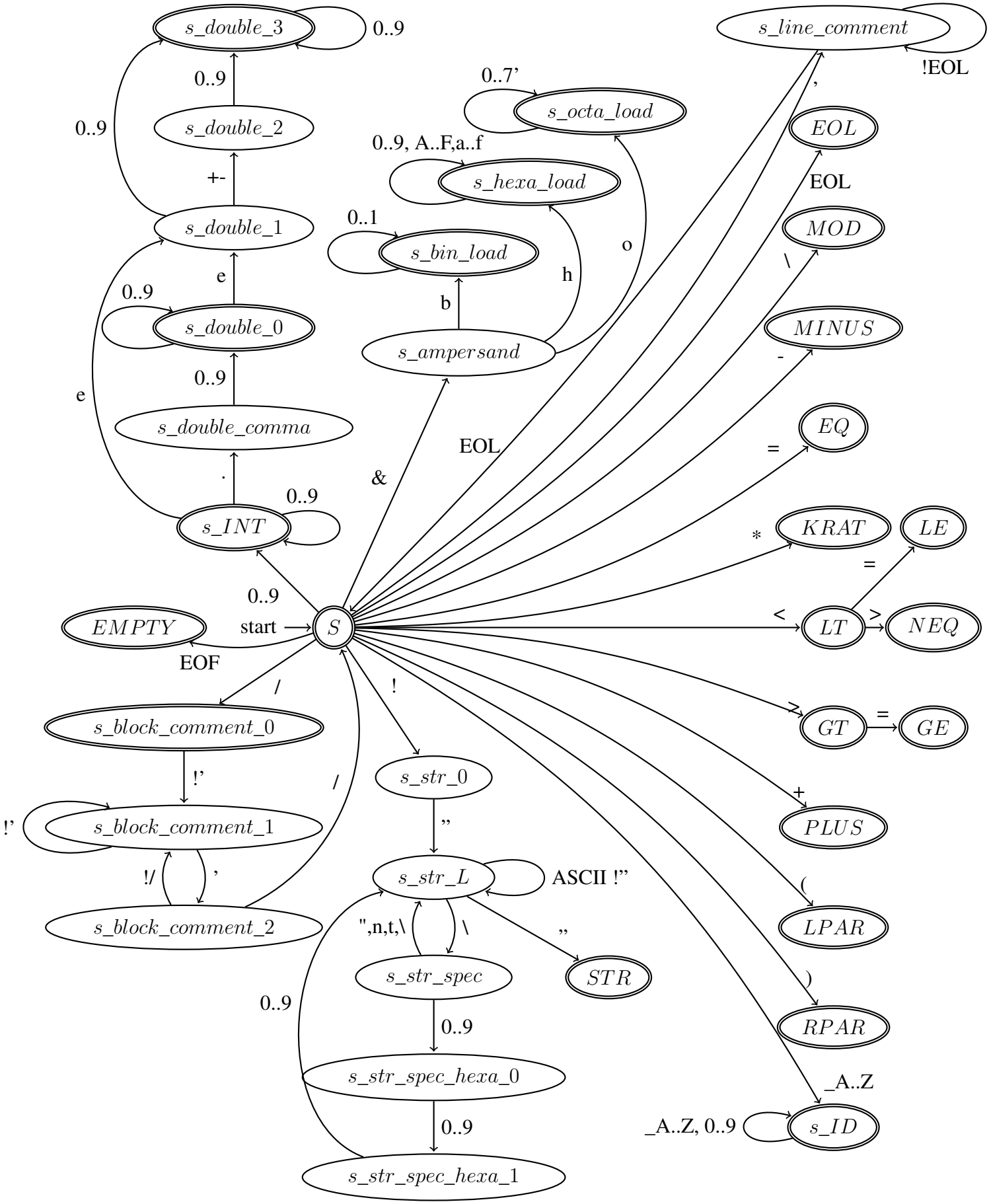
Tato práce pojednává o překladači imperativního jayzka IFJ17. Práce byla zadaná jako týmový projekt do předmětů IAL a IFJ. Dokumentace sestává z popisu jednotlivých částí překladače, doplněných o schémata, grafy a tabulky. V závěrečné části se věnujeme zpracování projektu jako celku a rozdělení práce v týmu.

2 Lexikální analýza

Lexikální analyzátor (dále jen LA) je v pomyslném řetězu zpracování programu na prvním místě. Jeho hlavním úkolem je čtení zdrojového souboru a rozdělení prvků souboru na jednotlivé lexikální části, tzv. lexémy, na základě lexikálních pravidel jazyka.

Pro svoji činnost potřebuje pouze vstupní soubor obsahující program, který chceme zpracovávat. Na žádost syntaktického analyzátoru pak přečte další lexém a předá ho. Spolu se samotným lexémem předá i načtenou hodnotu (například číslo,řetezec).

Princip fungování LA je založen na konečném automatu, jehož stav řekne, jaký lexém(token) byl detekován. Po vytvoření je token uložen na zásobník.


s_double_3
0..9
0..9
s_double_2
0..9
+-
s_double_1
e
0..9
s_double_0
0..9
e
s_double_comma
.
0..9
s_INT
0..7'
s_octa_load
0..9, A..F,a..f
s_hexa_load
0..1
s_bin_load
o
h
b
s_ampersand
&
EOL
s_line_comment
!EOL
,
EOL
EOL
MOD
\
MINUS
-
EQ
=
KRAT
*
LE
=
EMPTY
start
S
<
LT
>
NEQ
EOF
/
!
>
GT
=
GE
s_block_comment_0
!'
s_block_comment_1
!'
!/
'
/
s_block_comment_2
s_str_0
"
s_str_L
ASCII !"
",n,t,\
\
„
s_str_spec
STR
0..9
s_str_spec_hexa_0
0..9
s_str_spec_hexa_1
+
PLUS
(
LPAR
)
RPAR
_A..Z
_A..Z, 0..9
s_ID

3 Syntaktická analýza

Syntaktická analýza se dělí na dvě části - obecnou analýzu jazykových konstrukcí a analýzu výrazů. První zmíněná je implementována pomocí metody rekurzivního sestupu, řízeného LL gramatikou. Syntaktická správnost výrazů je potom vyhodnocována pomocí precedenční syntaktické analýzy.

4 Sémantická analýza

Hlavním úkolem této fáze je sémantická kontrola vstupního textu. Kompilátor kontroluje, zda všechny operandy mají správné typy, nebo jestli jdou implicitně transformovat (int -> double).

Během sémantické analýzy dochází k ukládání informací, které jsou uvedené v deklaracích do vnitřních datových struktur. V našem případě se jedná o vyhledávací tabulku s rozptýlenými položkami.

5 LL-gramatika

1. *PARSE* → declare *FUNCTION* eol *PARSE*
2. *PARSE* → *FUNCTION* eol *FUNBODY* end function eol *PARSE*
3. *PARSE* → scope eol *SCOPEBODY* end scope
4. *FUNCTION* → function id (*PARAM*) as type
5. *PARAM* → id as type *NPARAM*
6. *PARAM* → ε
7. *NPARAM* → , id as type *NPARAM*
8. *NPARAM* → ε
9. *FUNBODY* → dim id as type eol *FUNBODY*
10. *FUNBODY* → dim id as type = *EXPR* eol *FUNBODY*
11. *FUNBODY* → id = *EXPR* eol *FUNBODY*
12. *FUNBODY* → inut id eol *FUNBODY*
13. *FUNBODY* → print *EXPR* eol*FUNBODY*
14. *FUNBODY* → return *EXPR*
15. *FUNBODY* → if *EXPR* eol *FUNBODY ELSE* end if eol *FUNBODY*
16. *ELSE* → eol *FUNBODY*
17. *ELSE* → ε
18. *FUNBODY* → ε
19. *FUNBODY* → do while *EXPR* eol *FUNBODY*
20. *FUNBODY* → loop eol
21. *SCOPEBODY* → dim id as type eol *SCOPEBODY*
22. *SCOPEBODY* → dim id as type = value eol *SCOPEBODY*
23. *SCOPEBODY* → id as = value eol *SCOPEBODY*
24. *SCOPEBODY* → inut id eol *SCOPEBODY*
25. *SCOPEBODY* → print *EXPR* eol*SCOPEBODY*
26. *SCOPEBODY* → if *EXPR* eol *SCOPEBODY ELSE* end if eol *SCOPEBODY*
27. *SCOPEBODY* → ε
28. *SCOPEBODY* → do while *EXPR* eol *SCOPEBODY*
29. *SCOPEBODY* → loop eol
30. *ELSE* → eol *SCOPEBODY*

6 LL-Tabulka

	declare	function	scope	id	dim	input	print	return	if	do	loop	eol
PARSE	1	2	3									
FUNCTION			4									
PARAM				5								
NPARAM				7								
FUNBODY				11	10	12	13	14	15	19	20	
ELSE												16
SCOPEBODY				21	23	24	25		26	28	29	

7 Precedenční analýza výrazů

Precedenční analýza (dále jen PA) slouží k vyhodnocování výrazů. Kdykoliv syntaktický analyzátor narazí na výraz, zavolá PA. Precedenční analýza využívá ke své činnosti lexikální analyzátor, od něhož si může vyžádat další token.

Základem PA je precedenční tabulka. V této tabulce jsou definována syntaktická pravidla pro všechny výrazy. Při vyhodnocování a redukování výrazu používáme jako pomocnou strukturu zásobník.

	+	-	*	/	\	(	)	id	<	<=	>	>=	<>	=	$	f	,
+	>	>	<	<	<	<	>	<	>	>	>	>	>	>	>	<	>
-	>	>	<	<	<	<	>	<	>	>	>	>	>	>	>	<	>
*	>	>	>	>	>	<	>	<	>	>	>	>	>	>	>	<	>
/	>	>	>	>	>	<	>	<	>	>	>	>	>	>	>	<	>
\	>	>	<	<	>	<	>	<	>	>	>	>	>	>	>	<	>
(	<	<	<	<	<	<	=	<	<	<	<	<	<	<		<	=
)	>	>	>	>	>		>		>	>	>	>	>	>	>		>
id	>	>	>	>	>		>		>	>	>	>	>	>	>		>
<	<	<	<	<	<	<	>	<							>	<	>
<=	<	<	<	<	<	<	>	<							>	<	>
>	<	<	<	<	<	<	>	<							>	<	>
>=	<	<	<	<	<	<	>	<							>	<	>
=	<	<	<	<	<	<	>	<							>	<	>
<>	<	<	<	<	<	<	>	<							>	<	>
$	<	<	<	<	<	<		<	<	<	<	<	<	<		<	
f						=											
,	<	<	<	<	<	<	=	<	<	<	<	<	<	<		<	=

Tabulka 1: Precedenční tabulka

8 Tabulka symbolů

Hashovací tabulka je abstraktní datové struktura, která sdružuje záznamy do skupin na základě shodnosti hashe jejich unikátního klíče. V naší implementaci používáme pro zahashování klíče Bernsteinovu funkci[1].

Každá tabulka je z počátku inicializována na 8 řádků. Při obsazenosti 75% dojde ke zvětšení počtu řádků tabulky na dvojnásobek. Zvětšení tabulky probíhá ve třech krocích - nejpve je vytvořena nová, větší tabulka, následně jsou do ní vloženy všechny prvky z původní tabulky, která je nakonec smazána.

9 Rozšíření

Z možného výběru rozšíření, které jsme mohli implementovat, jsme se rozhodli pro FUNEXP, které přidává možnost volání funkcí jako součásti výrazů a možnost volání

[1] k dispozici na https://www.strchr.com/hash_functions

funkcí v parametrech funkcí. Rozšíření znamenalo přidání pravidla do precedenční tabulky.

Další rozšíření které jsme implementovali bylo rozšíření BASE, jehož implementace nám přišla jednoduchá. Stačilo přidat příslušné stavy do automatu lexikální analýzy.

10 Vývoj a práce v týmu

Vzhledem k rozsahu projektu bylo nutné ustanovit určitá pravidla pro práci a zacházení se zdrojovými soubory. Jako verzovací systém jsme používali Git a privátní repozitář umístěný na serveru Github. Jako hlavní komunikační kanál jsme používali Slack.

Projekt jsme se snažili řešit iterativně a agilně - první týdny jsme rozdělili do Sprintů, přičemž každý týdenní Sprint začínal meetingem, na kterém jsme prošli hotovou práci a určili úkoly na další Sprint. Od tohoto postupu jsme v pozdějších fázích upustili, neboť jsme převážně opravovali chyby v dříve implementovaných částech projektu.

Co se zdrojů týče, čerpali jsme hlavně z přednášek, ale vzhledem k termínu pokusného odevzdání, které jsme chtěli stihnout (a také jsme úspěšně stihli), bylo velkým zdrojem informací samostudium.

11 Rozdělení práce

Adam Venger	sémantická analýza, generování kódu, testování
Peter Babka	lexikální analýza,sémantická analýza, generování kódu
Ondřej Doseděl	sémantická analýza, dokumentace
Vojtěch Randýsek	syntaktická analýza, tabulka symbolů

12 Závěr

Díky obavám, vyvolaných historkami starších spolužáků, jsme s řešením projektu začali velmi brzy. Většinu problémů jsme tak řešili o dost dřív, než byly probírány na přednáškách. I přes komplikace, kterým jsme se mohli snadno vyhnout, kdybychom s řešením o chvíli počkali, byl projekt připraven pro první pokusné odevzdání, které nás nasměrovalo k úspěšnému doladění finální verze.