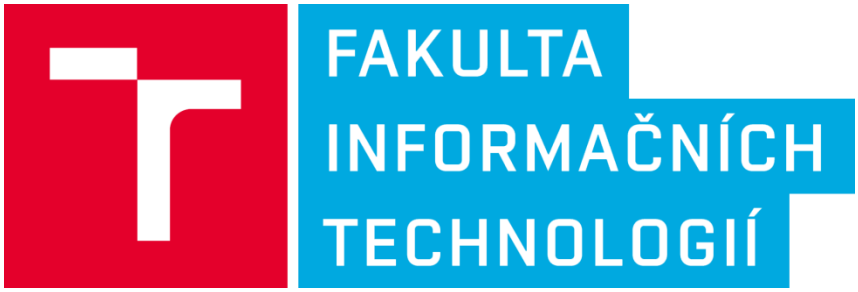
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta Informačných technológií



Dokumentácia k projektu do predmetov IFJ a IAL

Implementácia prekladača imperatívneho jazyka IFJ18

Tím 060, varianta II
30.11.2018

Ján Zauška(vedúci)	xzausk00	25%
Ján Vavro	xvavro05	25%
Július Marko	xmarko17	25%
Martin Ročkár	xrocka00	25%

Obsah

1 Úvod

Zadaním projektu bolo implementovať prekladač imperatívneho jazyka IFJ18. Prekladač načíta program napísaný v jazyku IFJ18 zo štandardného vstupu a preloží ho do cieľového jazyka IFJcode18(medzikód) na štandardný výstup.

Projekt sa skladá zo štyroch základných častí:

- Lexikálny analyzátor
- Syntaktický analyzátor
- Sémantický analyzátor
- Generátor medzikódu IFJcode18

1.1 Popis jazyka IFJ18

Jazyk IFJ18 je zjednodušenou podmnožinou jazyka Ruby 2.0, čo je dynamicky typovaný objektovo-orientovaný jazyk. V jazyku záleží na veľkosti písmen („case-sensitive“).

2 Štruktúra projektu

2.1 Lexikálny analyzátor

Lexikálny analyzátor alebo „scanner“ je vstupnou časťou prekladača. Úlohou lexikálneho analyzátora je rozdeliť postupnosť znakov ktoré reprezentujú zdrojový program zo štandardného vstupu na lexémy (tokeny) s ktorými následne pracuje syntaktický analyzátor. Princíp „scanneru“ je založený na konečnom automate. V praxi je tento automat implementovaný v jazyku C pomocou konštrukcie „switch“ kde každý „case “ reprezentuje jeden stav automatu . Hlavnou funkciou lexikálnej analýzy je funkcia „getTokenFromInput“ ktorá číta znak po znaku a vracia typ a hodnotu lexému. Funkcia ignoruje biele znaky a komentáre pretože nie sú potrebné v ďalších krokoch prekladu. Pokiaľ načítaný znak alebo reťazec nesúhlasí so žiadnym stavom konečného automatu, funkcia vracia chybu s hodnotou 1 (lexikálna chyba). Identifikátory sú od kľúčových slov odlišované pomocou funkcie „checkKeywords“ ktorá buď vracia hodnotu konkrétneho rezervovaného slova alebo hodnotu identifikátora.

2.2 Syntaktický analyzátor

Syntaktická analýza je najdôležitejšou časťou prekladača pretože kontroluje korektnú syntax vstupného programu pomocou nižšie uvedenej LL gramatiky. Pri implementácií syntaktického analyzátora sme použili metódu rekurzívneho zostupu. Ten je realizovaný pomocou jedného priechodu zdrojového kódu. Výrazy vyhodnocujeme pomocou precedenčnej analýzy ktorú riadi nižšie uvedená precedenčná tabuľka.

2.3 Sémantický analyzátor

Sémantická analýza je vykonávaná zároveň so syntaktickou analýzou. Sémantická analýza využíva na ukladanie dát dve tabuľky s rozptýlenými prvkami, jednu pre údaje o funkciách a druhú pre údaje o premenných kvôli prehľadnejšiemu rozdeleniu dát. Typové kontroly sú generované do trojadresného kódu a sú vykonávané až počas interpretácie kódu.

2.4 Generátor medzikódu IFJcode18

Generovanie medzikódu IFJcode18 prebieha v rámci syntaktickej a sémantickej analýzy. Jednotlivé inštrukcie sú postupne ukladané do jednosmerne viazaného zoznamu kvôli jednoduchému vkladaniu ďalších inštrukcií. Po overení lexikálnej, syntaktickej a sémantickej korektnosti zdrojového programu sú inštrukcie vypísané na štandardný výstup.

3 Použité algoritmy

3.1 Tabuľka s rozptýlenými prvkami

Základ tabuľky symbolov tvorí tabuľka s rozptýlenými prvkami čo bola aj požiadavka riešenia. Túto variantu sme si vybrali hlavne kvôli rýchlosti vyhľadávania. Tabuľka pozostáva z poľa ukazateľov na jednosmerne viazané zoznamy. Každá položka musí obsahovať špecifický kľúč, dáta a ukazateľ na ďalšiu položku. Naše položky v tabuľke obsahujú okrem toho aj informáciu o tom či položka je funkcia alebo premenná. Vyhľadávanie v tabuľke je vykonávané pomocou hashovacej funkcie ktorej vstupný parameter je kľúč a výstup funkcie je index do hashovacej tabuľky. Tabuľka symbolov je implementovaná v súboroch ‘symtable.c’ a ‘symtable.h’.

3.2 Jednosmerne viazaný zoznam

Lineárny jednosmerný zoznam je dynamická štruktúra obsahujúca položky rovnakého typu. Každý prvok zoznamu musí obsahovať ukazateľ na nasledujúci prvok v zozname. Jednosmerne viazaný zoznam využívame pri generovaní medzikódu do ktorého sú ukladané inštrukcie. Táto štruktúra sa nám osvečila hlavne vďaka jednoduchému vkladaniu prvkov.

3.3 Zásobník

Zásobník je implementovaný ako jednosmerne viazaný zoznam. Pri vkladaní prvku sa položka uloží na jeho vrchol. Táto dátová štruktúra sa označuje ako LIFO(last-in-first-out) čiže posledná vkladaná položka bude spracovávaná ako prvá. Zásobník ukazuje na vrchnú položku v zozname. V našom prekladači je zásobník použitý pri precedenčnej analýze a je implementovaný v súboroch ‘stack.c’ a ’stack.h’.

4 Práca v tíme

Projekt bol riešený štyrmi ľuďmi, čo vyžadovalo dôsledné rozdelenie a prípravu pred začatím vývojového procesu. Prvé tímové stretnutie bolo asi dva týždne po zverejnení projektu kde sme si rozdelili prácu. Na konkrétnych častiach projektu pracovali väčšinou jednotlivci alebo dvojice. Počas riešenia projektu sme mávali schôdzky raz týždenne na ktorých sme diskutovali o problémoch na ktoré sme narazili pri vývoji.

4.1.1 Komunikácia v tíme

Na komunikáciu v tíme sme používali buď osobné stretnutia alebo aplikáciu Slack. V aplikácií sme mali vytvorené kanály pre všetky časti projektu v ktorých medzi sebou komunikovali konkrétni užívatelia pracujúci na danej časti.

4.1.2 Verzovací systém

Pre správu súborov sme sa rozhodli používať verzovací systém „git“ hlavne kvôli predchádzajúcej skúsenosti všetkých členov tímu so systémom. Toto nám dovoľovalo pracovať na projekte súčasne vo viacerých vetvách.

4.2 Rozdelenie práce

Člen tímu	login	Vykonaná práca
Ján Zauška	xzaus00	Syntaktický analyzátor, generovanie medzikódu
Ján Vavro	xvavro05	Syntaktický analyzátor výrazov, generovanie medzikódu
Július Marko	xmarko17	Sémantický analyzátor, generovanie medzikódu, tabuľka symbolov
Martin Ročkár	xrocka00	Lexikálny analyzátor, dokumentácia

Tabuľka 1 Rozdelenie práce

5 Záver

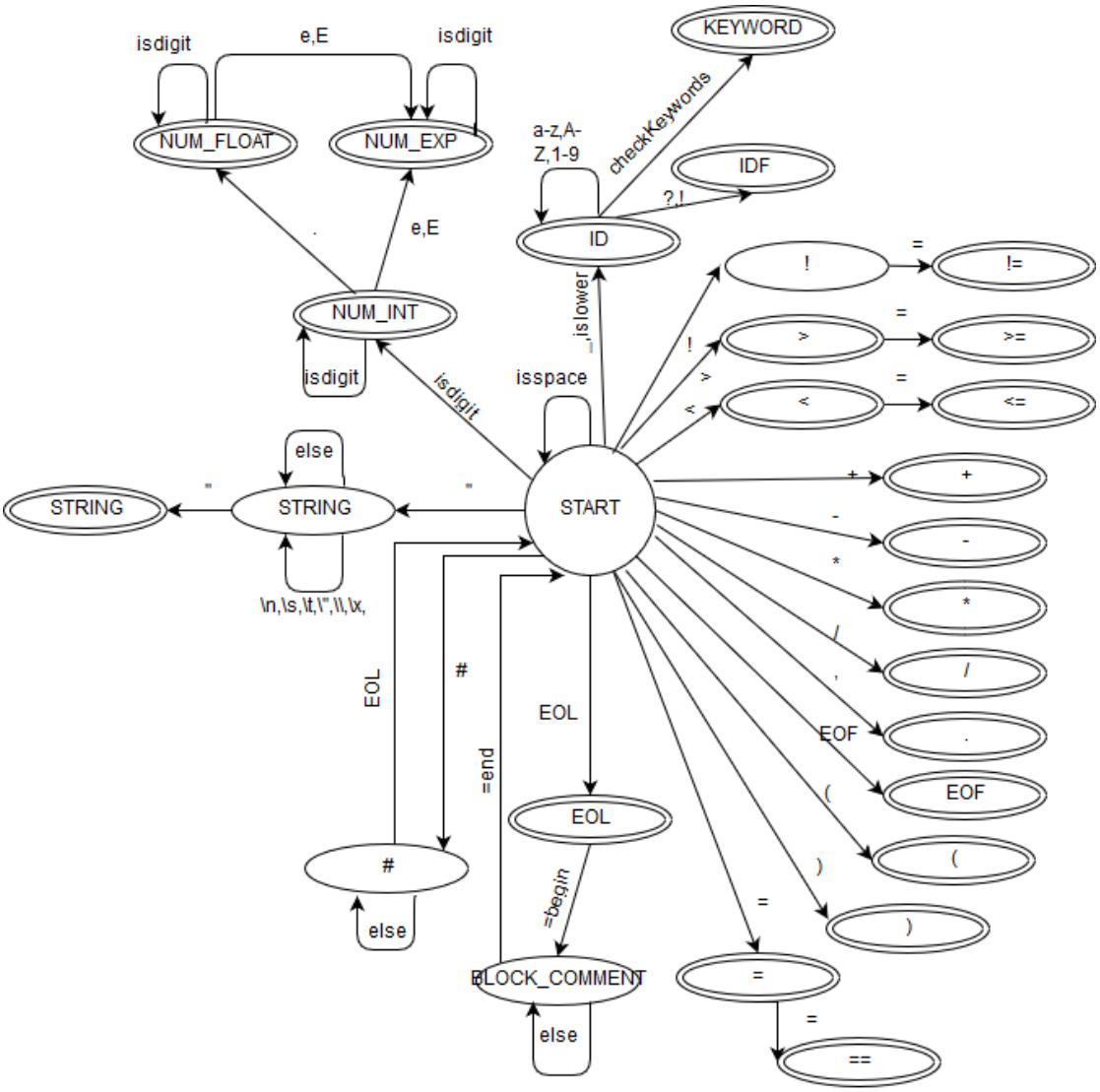
Po prečítaní zadania nás projekt prekvapil pretože sme sa na FIT-e zatiaľ nestretli s projektom s podobným rozsahom a zložitosťou. No po čase, keď sme nabrali vedomosti o tvorbe prekladačov sme začali s vývojovým procesom. Projekt sa nám podarilo dokončiť do predpokladaného termínu. S prácou v tíme a komunikáciou sme nemali žiadne problémy pretože tím sme mali zostavený už pred zadaním projektu a všetci členovia sa osobne poznáme. V priebehu vývoja sme narazili na pár problémov ale podarilo sa nám ich skupinovo vyriešiť. Projekt nám priniesol množstvo skúseností napríklad pre prácu v tíme a prakticky nám objasnil látku preberanú v predmetoch IFJ a IAL.

6 Prílohy

1.	<PROG> -> <FUN_DCLR> <PROG>
2.	<PROG> -><STAT_LIST><PROG>
3.	<PROG> -> EOF
4.	<FUN_DCLR> ->DEF<ID> <FUN_PARAMS> EOL <STAT_LIST>
5.	<STAT_LIST> -> <STAT><STAT LIST>
6.	<STAT_LIST>->**ε**
7.	<STAT LIST>->EOF
8.	<FUN_PARAMS>-><ITEM><FUN_PARAMS>
9.	<FUN_PARAMS>->,<ITEM><FUN_PARAMS>
10.	<FUN_PARAMS>->)
11.	<CALL_PARAMS>-><ITEM><CALL_PARAMS>
12.	<CALL_PARAMS>->EOL
13.	<ITEM>->VALUE
14.	<STAT> -> IF<EXPR> THEN EOL<STAT_LIST> ELSE EOL <STAT_LIST> END
15.	<STAT> -> WHILE <EXPR> DO EOL <STAT_LIST> END
16.	<STAT> -> PRINT(<ARG_LIST>)
17.	<STAT> -> ID
18.	<STAT> -> LENGTH(<ITEM>)
19.	<STAT> -> SUBSTR(<ITEM>,<ITEM>,<ITEM>)

20.	<STAT> -> ORD(<ITEM>,<ITEM>)
21.	<STAT> -> CHR(<ITEM>)
22.	<STAT> -> EOF
23.	<STAT> -> EOL
24.	<STAT> -> NIL
25.	<STAT> -><ASSIGN>
26.	<STAT>-><FUN_CALL>
27.	<ASSIGN> -><ID> = <VALUE>
28.	<STAT>-><VALUE>
29.	<FUN_CALL>->ID(<FUN_PARAMS>)
30.	<FUN_CALL>->ID<FUN_PARAMS>
31.	<FUN_CALL>-> INPUTS()
32.	<FUN_CALL>-> INPUTF()
33.	<FUN_CALL>-> INPUTI()
34.	<ARG_LIST>-><ITEM><ARG_LIST>
35.	<ITEM>->STRING
36.	<ITEM>->INT
37.	<ITEM>->FLOAT
38.	<ITEM>->ID
39.	<ITEM>->NIL
40.	<CALL_PARAMS>->,<ITEM><ARG_LIST>
41.	<VALUE>-><FUNC_CALL>
42.	<VALUE>-><EXPR>

Tabuľka 2 LL gramatika



Obrázok 1 Konečný automat lexikálnej analýzy

	+,-	*,/	rel	(	)	ID	$
+,-	>	<	>	<	>	<	>
*,/	>	>	>	<	>	<	>
rel	<	<		<	>	<	>
(	<	<	<	<	=	<	
)	>	>	>		>		>
id	>	>	>		>		>
$	<	<	<	<		<	

Tabuľka 3 Precedenčná tabuľka

	ID	NUM	STRING	EOL	COMMA	ROUND	ASSIGN	PLUS,MINUS,MUL,DIV	"==,>=,<=,<,>"	EOF	INPUTS	INPUTF	INPUTI	PRINT	ORD	CHR	SUBSTR	LENGTH	DEF	DO	ELSE	END	IF	NIL	THEN	WHILE
<PROG>	2	2	2	2		2				3	2	2	2	2	2	2	2	2	1				2	2		2
<FUN_DCLR>																			4							
<STAT_LIST>	5	5	5	5		5				7	5	5	5	5	5	5	5	5					5	5		5
<FUN_PARAMS>	8	8	8		9	10																		8		
<STAT>	17	28	28			28				22	26	26	26	26	26	26	26	26					14	42		15
<ITEM>	38	36	35																					39		
<FUN_CALL>	29										31	32	33													
<CALL_PARAMS>	11	11	11		40																					
<ASSIGN>	27	27	27			27																				
<VALUE>	42	42	42			42			42	42	41	41	41	41	41	41	41	41						42		

Tabuľka 4 LL tabuľka