VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÝCH TECHNOLÓGIÍ

Dokumentácia k projektu z IFJ/IAL

IMPLEMENTÁCIA PREKLADAČA JAZYKA IFJ19

TÍM 093, VARIANTA I

Členovia tímu:

Krajňák Damián - xkrajn03 25%
Stuchlý Samuel - xstuch06 33%
Feňko Šimon - xfenko01 21 %
Novotný Mlinárcsik Martin - xnovot1r 21%

10.December 2019

OBSAH:

1. ÚVOD

Cieľom projektu bolo vytvorenie prekladača v jazyku C, ktorý prečíta zdrojový kód zapísaný v zdrojovom jazyku IFJ19, ktorý je zjednodušenou podmnožinou jazyka python3 a preloží ho do cieľového jazyka IFJcode19 (medzikód). Vybrali sme si variantu I. a to implementovanie tabuľky symbolov pomocou binárneho vyhľadávacieho stromu.

2. ETAPY PROJEKTU

1. Návrh štruktúry programu:
    a) konečný automat (viď obrázok 1)
    b) precedenčná tabuľka (viď tabuľka 1)
    c) LL tabuľka (viď tabuľka 2)
    d) LL gramatika (viď gramatika)
2. Lexikálna analýza
3. Syntaktická analýza - rekurzívny zostup
4. Precedenčná syntaktická analýza
5. Sémantická analýza
6. Generovanie kódu
7. Testovanie
8. Tvorba Dokumentácie

3. PRÁCA V TÍME

3.1 Komunikácia v tíme

Ako hlavný komunikačný kanál sme zvolili discord (hovory), poprípade messenger. Pravidelne sme sa snažili organizovať stretnutia, na ktorých sme prediskutovali dané úlohy, problematiku a testovali hotové programy.

3.2 Správa kódu

Ako verzovací systém sme použili Git, hostovaný na službe Github. Tento spôsob nám vyhovoval najviac, pretože každý člen tímu mohol pridávať a upravovať zdrojový kód.

3.3 Rozdelenie práce

Damián Krajňák: syntaktická analýza, pomoc s lexikálnou analýzou
Samuel Stuchlý: generovanie kódu, sémantická analýza
Šimon Feňko: lexikálna analýza, úprava zoznamu, testovanie, dokumentácia, úprava stacku
Martin Novotný Mlinárcsik: precedenčná analýza, sémantická analýza

4. IMPLEMENTÁCIA

4.1 LEXIKÁLNA ANALÝZA

Lexikálny analyzátor (LA), bola prvá časť projektu, ktorú sme implementovali ako prvú. Je navrhnutý ako konečný automat, podľa ktorého bol lexikálny analizátor implementovaný. Ten postupne načíta jednotlivé znaky. Zaviedli sme obojsmernú komunikáciu medzi syntaktickcou analyzátorom (SA) a (LA). SA bol potom schopný čítať tokeny a tiež vraciať naspäť do LA. Tokeny boli čítané pri voláni LA zo štandartného vstupu. Pokiaľ sa analyzátor nedostane do požadovaného stavu, vracia lexikálnu chybu (hodnota 1).

4.2 SYNTAKTICKÁ ANALÝZA

Na implementáciu syntaktického analyzátora sme použili metódu rekurzívneho zostupu. Vychádzali sme z LL pravidiel, odvodených y LL gramatiky. (schéma je v prílohe). O ošetrenie validity výrazov sa postarala precedenčná analýza (schéma tabuľky v prílohe). Po nalezení syntaktickej chyby vo vstupnom kóde, vraciame hodnotu 2.

4.2.1 Rekurzívny zostup

Rekurzívny postup sme sa rozhodli použiť z dôvodu jeho doporučenia na prednáškach. Najprv sme zostavili LL gramatiku, na základe ktorej sme podľa množiny Predict vytvorili LL tabuľku. Podľa tabuľky sme ku každému neterminálu vytvorili funkciu, ktorá simulovala dané pravidlá podľa LL gramatiky. Priebežne bola vrámci rekurzívneho zostupu volaná funckia get_next_token(), implementovaná v časti scanner.

4.2.2 Precedenčná analýza

Precedenčná analýza je volaná ak parser narazí na výraz, ktorý je potrebné vyhodnotiť. Tento výraz sa po tokenoch uloží do dvojsmerne viazaného zoznamu, s ktorým následne precedečná analýza pracuje. Algoritmus je založený na algoritme z prednášok, kde sa tokeny postupne ukladajú na stack a na základne precedenčnej tabuľky sa určuje priorita a asociativita operátorov a v akom poradí sa bude výraz vyhodnocovať.

4.3 SÉMANTICKÁ ANALÝZA

Sémantická analýza sa kontroluje v parseri na redefinícue funkcii a sémantika behových chýb sa vykonáva v generátore kódu. Identifikátory premenných a funkcii sú ukladané do globálneho stromu, pričom ak sa v kóde vyskytne identifikátor a jeho označenie sa v globálnom strome nenájde, program skočí s návratovou hodnotou 3.

4.4 IMPLEMENTÁCIA TABUĽKY SYMBOLOV

Tabuľka symbolov je implementovaná pomocou štruktúry binárneho stromu. Kostra pre túto štruktúru bola prevzaná z úlohy na IAL. Vytvárame dva binárne stromy, jeden pre globálne premenné a funkcie a jeden pre lokálne, ktorý sa stále prepisuje. Tabuľka súborov sa využíva aj v generátore na dodatočné kontroly, keďže máme dvojprechodový projekt, čiže tieto štruktúry sú definované ako extern.

4.5 GENEROVANIE KÓDU

Na generovanie kódu používame inštrukcie jazyka IFJcode19. Parser generuje zjednodušený zoznam tokenov pre generátor. Generátor tento zoznam číta a postupne generuje inštrukcie v jazyku IFJcode19 do output_list jednosmerne viazaného

zoznamu. Tento zoznam sa na konci vytiskne na stdout. Časť precedenčnej analýzy vykonávame pomocou generátora, kde prechádzame zoznamom pravidiel, vygenerovaným precanalysis.c a na základe týchto pravidiel generujeme inštrukcie na vyhodnotenie výrazov. Taktiež podporujeme sémantické kontroly za behu programu a pretypovanie int to float.

5. TESTOVANIE

Síce sme to testovali v posledný deň na kratších základných testoch, ale odstránili sme vysoký počet chýb, ako je napríklad delenie nulou alebo printy.

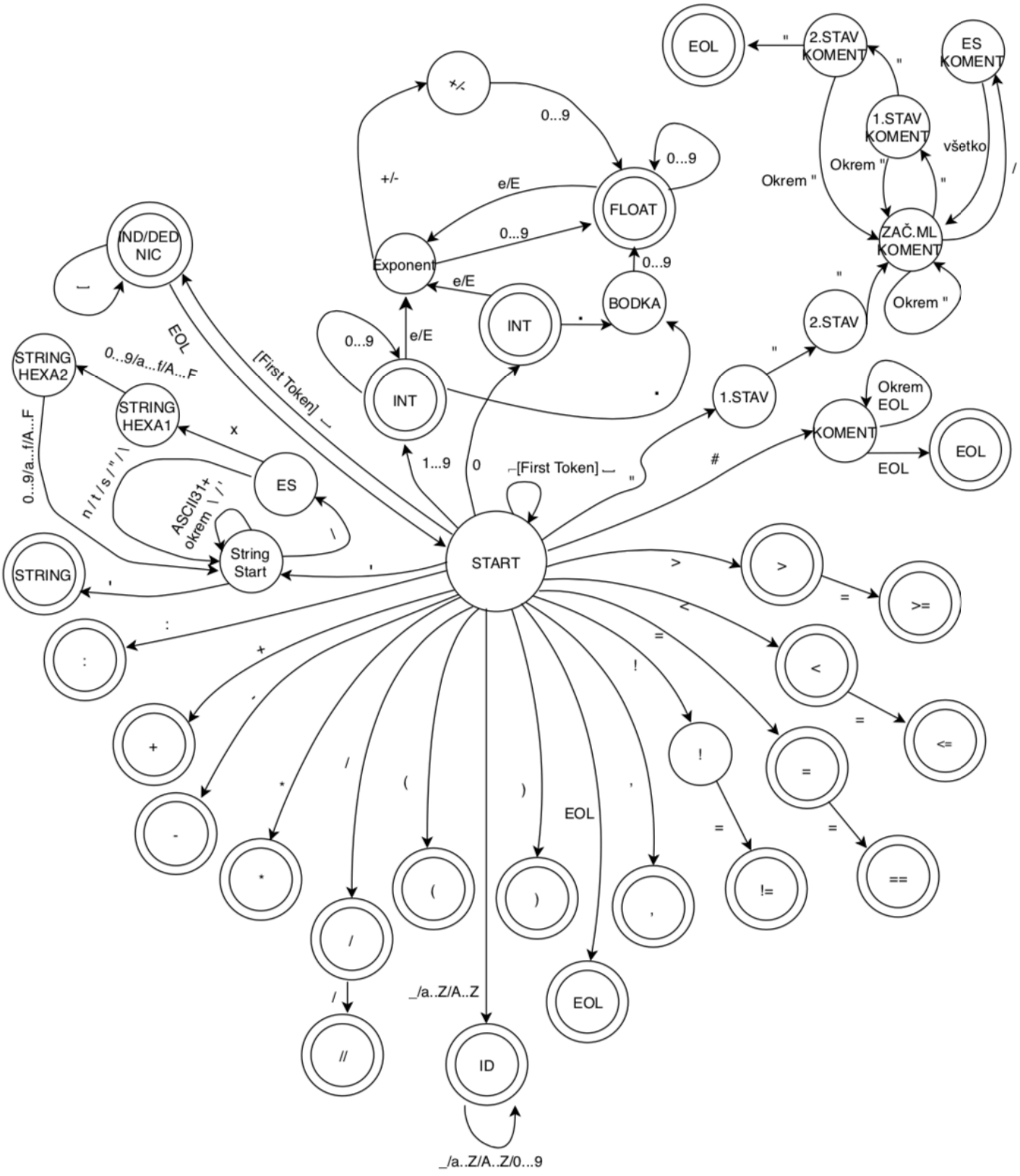
6. ZHRNUTIE

Spočiatku sme mali niektorí určité nedostatky a nezrovnalosti čo sa týkalo projektu. Ale postupom času sa nám všetko ozrejmilo vďaka výkladu z prednášok a aj od vzájomnej pomoci v tíme. Keďže sme využívali ako komunikačný kanál aj hovory na Discorde, tak aj v prípadoch keď sme natrafili na určité problémy a nejasnosti, čo sa týklao zadania, rýchlo sme ich vyriešili.

V priebehu vývoja sme si na pokusnom odovzdaní overili správnosť dovtedy vytvoreného kódu a neskôr na druhom pokusnom odovzdávaní taktiež overili opravené chyby.

Tento projekt nám v konečnom dôsledku určite pomohol, keďže sme získali nejaké nejaké veľa znalostí, čo sa týka fungovania prekladačov. Taktiež nám pomohol aj čo sa týka fungovania a práce v tíme, keďže nie všetci sme doteraz robili nejaký tímový projekt a tým pádom sme nadobudli veľa skúseností.

6: Konečný automat



7. Tabuľka 1: Precedenčná tabuľka

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

8. Tabuľka 2: LL tabuľka

	0	"EOL"	"EOF"	"def"	"ID"	"("	":"	"INDENT"	"FLOAT"	"INT"	"STRING"	"if"	"expression"	"while"	"pass"	"="	")"	","	"DEDENT"	"else"	"return"
<st_list>		2	3	1	1				1	1	1	1		1	1						
<statement>				4	5				5	5	5	5		5	5						
<params>					21												22				
<func_list>		32			31				31	31	31	31		31	31				33		31
<command>					9				6	7	8	10		11	12						
<param_n>																	24	23			
<func_cmd>					37				34	35	36	38		39	40						41
<as/fun/ex>		15				14										13					
<if_list>		26			25				25	25	25	25		25	25				27		
<else_list>		29			28				28	28	28	28		28	28				30		
<if_f_list>		43			42				42	42	42	42		42	42				44		42
<else_f_list>		46			45				45	45	45	45		45	45				47		45
<expr/id>					16				19	18	20		17								

9. Gramatika: LL Gramatika

1. <st-list> -> <statement> EOL <st-list>
2. <st-list> -> EOL <st-list>
3. <st-list> -> EOF
4. <statement> -> def ID (<params> : EOL INDENT <func-list>
5. <statement> -> <command>
6. <command> -> FLOAT
7. <command> -> INT
8. <command> -> STRING
9. <command> -> ID <AS/FUN/EX>
10. <command> -> if (expression) : EOL INDENT <if-list>
11. <command> -> while (expression) : EOL INDENT <else-list>
12. <command> -> pass
13. <AS/FUN/EX> -> = <EXPR/ID>
14. <AS/FUN/EX> -> (<params>
15. <AS/FUN/EX> -> ε
16. <EXPR/ID> -> ID (<params>
17. <EXPR/ID> -> (expression)
18. <EXPR/ID> -> INT
19. <EXPR/ID> -> FLOAT
20. <EXPR/ID> -> STRING
21. <params> -> ID <param_n>
22. <params> ->)
23. <param_n> -> , ID <param_n>
24. <param_n> ->)
25. <if-list> -> <command> EOL <if-list>
26. <if-list> -> EOL <if-list>
27. <if-list> -> DEDENT else : EOL INDENT <command> EOL <else-list>
28. <else-list> -> <command> EOL <else-list>
29. <else-list> -> EOL <else-list>
30. <else-list> -> DEDENT
31. <func-list> -> <func-command> EOL <func-list>
32. <func-list> -> EOL <func-list>
33. <func-list> -> DEDENT
34. <func-command> -> FLOAT
35. <func-command> -> INT
36. <func-command> -> STRING
37. <func-command> -> ID <AS/FUN/EX>
38. <func-command> -> if (expression) : EOL INDENT <if-func-list>
39. <func-command> -> while (expression) : EOL INDENT <else-func-list>
40. <func-command> -> pass
41. <func-command> -> return (expression)
42. <if-func-list> -> <func-command> EOL <if-func-list>
43. <if-func-list> -> EOL <if-func-list>
44. <if-func-list> -> DEDENT else : EOL INDENT <func-command> EOL <else-func-list>
45. <else-func-list> -> <func-command> EOL <else-func-list>
46. <else-func-list> -> EOL <else-func-list>
47. <else-func-list> -> DEDENT