VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Dokumentácia k projektu z IFJ

Interpret jazyka IFJ20

TÍM 62, VARIANTA II

9. DECEMBRA 2020

ZOZNAM AUTOROV:
MATÚŠ NOSKO (XNOSKO06) 37% - VEDÚCI
LUKÁŠ CHMELO (XCHMEL33) 37%
TOMÁŠ ČECHVALA (XCECHV03) 26%

1 Úvod

Cieľom projektu bolo vytvoriť program v jazyku C. Tento program načíta zdrojový kód napísaný v zdrojovom jazyku IFJ20, ktorý je zjednodušenou podmnožinou jazyka Go a preloží ho do cieľového jazyka IFJcode20 (mezikód). Hlavnou úlohou projektu bola implementácia lexikálneho analyzátora, parsera (syntaktická a sémantická analýza) a generátora kódu.

2 Lexikálna analýza

Lexikálnu analýza sme začali vytvorením funkcie ***getToken***, ktorá číta znak po znaku zo vstupu a pomocou konečného deterministického automatu vráti token. Automat je implementovaný pomocou príkazu switch, kde každý case reprezentuje stav automatu. Podľa aktuálneho znaku sa rozhoduje o ďalšom stave. Token je v našom riešení štruktúra obsahujúca jeho typ a attribute. Postupným dopĺňaním tela funkcie sme dopĺňali aj štruktúru token o ďalšie attributy. Pomocný string **S_Attribute** našeho vlastného typu dstring slúži na ukladanie znakov, ktoré sa pri konečnom stave scanneru prekopíruju do tokenu. Bolo treba vytvoriť aj ďalšie pomocné funkcie k funkcii ***getToken***, ktoré spracúvajú **S_Attribute** na príslušný dátový typ. Najzložitejšia časť bola v stave **ESCAPE_SEQUENCE**, keďže sme nevedeli ako presne sa čítajú znaky zo súboru, ale postupným debugovaním a zisťovaním znaku sme aj tento problém vyriešili. Okrem hlavnej funkcie na zistenie tokenu sme neskôr implementovali aj pomocné funkcie na výpis a inicializáciu tokenu.

3 Syntaktická a sémantická analýza

Jedna z hlavných častí programu, ktorá je založená na LL - gramatike a pomocou pravidiel LL - tabulky prechádza zdrojový súbor zostupnou rekurziou. Rekurzia sa vykoná vyvolaním funkcie **body**, ktorá ako prvé overí základnú podmienku - package main a ak táto podmienka platí presúva sa k ďalším tokenom a na základe tých, ktoré platia sa posunie ďalej hlbšie do **defunc** (vyvolá funkciu **params**, ktorá priradí návratové hodnoty a parametry danej funkcie a taktiež kontroluje sémantickú a syntaktickú platnosť funkcie statement) kde sa overuje syntatiktická platnosť výrazu a presunie sa do tela funkcie. V tele funkcie sa vyvolá funkcia **statements**, ktorá sa zacyklí až dokým nenarazí na koniec bloku a počas toho dokým nenarazí na koniec bloku sa vyvolávajú postupne ďalšie funkcie podľa LL gramatiky na základe prichádzajúcich tokenov. Ak sa nenáchadza koniec vyvolá sa funkcia **statement**, ktorá presne kontroluje syntaktickú stránku jednotlivých výrazov, volá si ďalšie pomocné funkcie podľa LL gramatiky a overuje, či sa v danom programe nenachádzajú vstavané funkcie. Pri inicializácii premennej skúsi nájsť danú premennú v lokálnej tabulke symbolov, ak sa tam nenachádza zavolá funkciu **init**. Inak kontroluje platnosť danej premennej ak sa tam vyskytne prirovnanie. Funkcia **init** zabezpečuje inicializáciu a rozhoduje na základe jednotlivých pravidiel, ktoré môžu nastať. Pri výrazoch si volá výrazovú analýzu, tak isto ako funkcia **assign**. Počas tohto celého procesu kontroluje platnosť všetkých výrazov ak nastane chyba vracia error stav na základe zadanie.

4 Výrazová analýza

Vyrazová analýza sa spúšťa funkciou expression, ktorá si pomocou pomocných funkcií získa najbližší terminál k vrcholu zásobníku, aktuálny token a ziska symbol z precedenčnej tablľky. Ten môže mat 3 varianty a pomocou switchu sa buď pushne na zásobník token a symbol, alebo sa vykoná operácia vo funkcií checkrule, ktorá aplikuje pravidlá. Dôležité bolo najma kontrolovať typy operandov, o čo sa stara funkcia checktype. Výsledkom je buď príslušná chyba, alebo 0. Zároveň sa predá aj aktuálny token, obsahujúci typ výsledku.

5 Generátor kódu

Generáciu kódu sme si nechali ako poslednú časť našeho riešenia a keďže na neho neostalo dostatok času nie je plne funkčný a dokončený. Umožňuje relačné a aritmetické operácie, konkatenanciu stringov, vytváranie framov, výpis hodnôt, ukladanie na zásobník a získanie hodnoty z vrcholu zásobníka. Inštrukcie sa ukladajú do našeho vlastného dátového typu **dstring** a funkcia **code_to_stdout** ho vypíše na štandardný výstup.

6 Práca v tíme

Na projekte sme začali pracovať koncom októbra. Prácu sme delili postupne ako sme pracovali na projekte. Časti, ktoré boli zložitejšie sme riešili spoločne, jednuduchšie časti sme si rozdelili jednotlivo. Dva týždne pred odovzdaním sme začali spájať dokopy všetky časti programu. Pokusných odovzdaní sme sa nezúčastnili, keďže nám vtedy program ešte nefungoval.

6.1 Komunikácia

Komunikáciu v tíme sme riešili pomocou skupiny na messengeri a discord serveru. Stretávali sme sa každý týždeň priebežne a v posledných dvoch týždňoch sme spoločne volali vždy, keď sme pracovali na projekte. Osobné stretnutia neboli z dôvodu pandémie Covid 19.

6.2 Verzovanie

Pre správu súborov sme používali verzovací systém Git a vzdialený repozitár Github. Vďaka tomuto repozitáru sme mohli pracovať na viacerých častiach projektu súčasne. Potom sme jednoducho spojili všetky časti dokopy a každý sme mohli testovať projekt ako celok.

6.3 Rozdelenie práce

- Matúš Nosko 37%: vedenie tímu, testovanie, organizovanie stretnutí, sémantická a syntaktická analýza, diagramy
- Lukáš Chmelo 37%: lexikálna analýza, testovanie, generátor inštrukcií, syntaktická analýza
- Tomáš Čechvala 26%: lexikálna analýza, testovanie, dokumentácia, diagramy

7 Záver

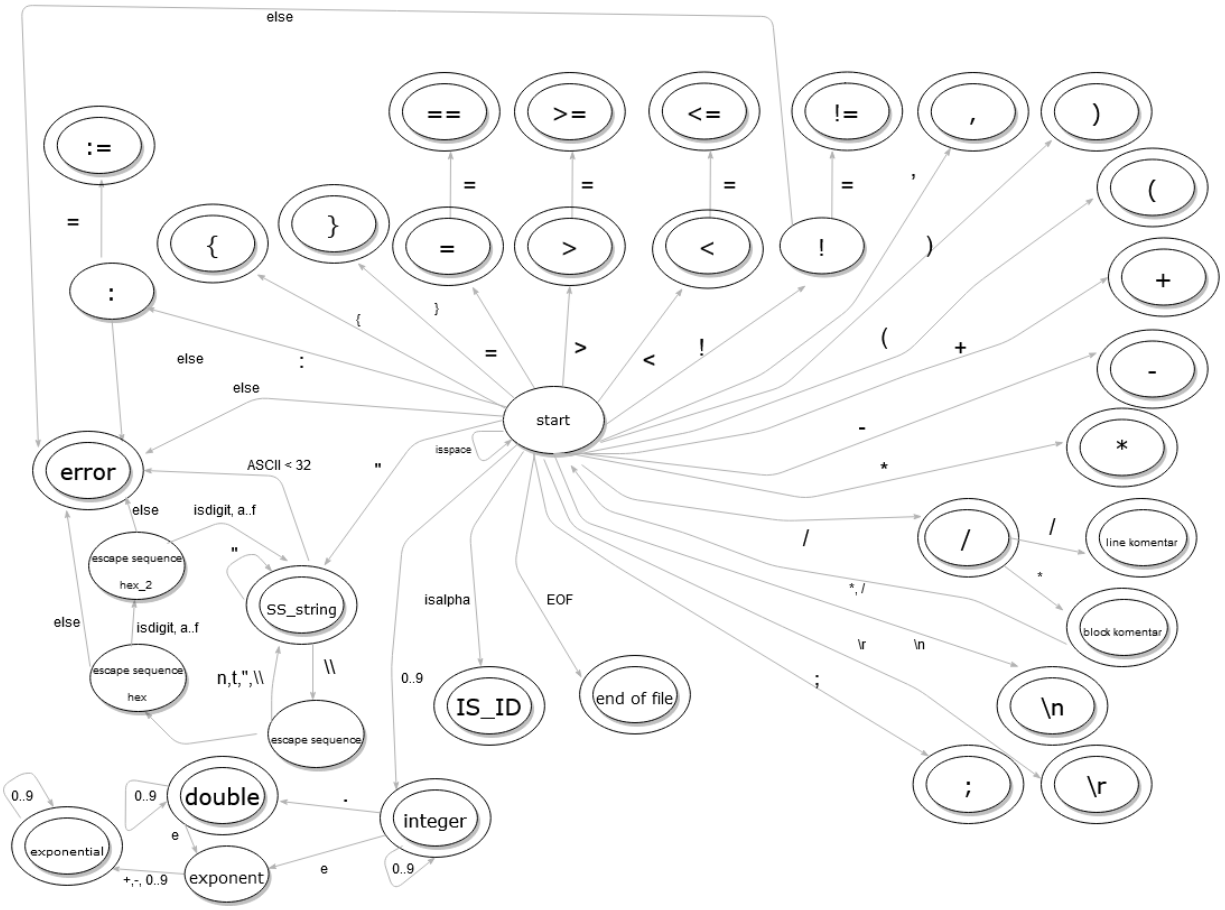
Projekt bol veľmi náročný pre nás, keďže sme boli len 3. Postupne sme ale prichádzali na to, ako jednotlivé časti projektu riešiť a implementovať, hlavne vďaka prednáškam z IFJ, IAL a materiálom, ktoré sú dostupné online. Jednou z hlavných problémov bol nedostatok času pri dokončovaní projektu. Taktiež nám robilo problém pochopiť niektoré časti zadania, na ktoré sme neskôr prišli. Funkčnosť projektu sme overili pomocou automatických testov. Tento projekt nám všetkým priniesol mnoho nových vedomostí a zlepšil naše schopnosti pracovať v tíme a taktiež vylepšil naše programovacie znalosti.

8 Literatúra

B.W. Kernighan & D.M. Ritchie, *Programovací jazyk C, Computer Press, 2019*

9 Tabulky a diagramy

9.1 Deterministický konečný automat



9.2 LL-gramatika

1. <program> -> package main <body> EOF
2. <body> -> FUNC ID (<arguments>) (<datatypes>) (BLOCK_BEGIN EOL <statements> EOL BLOCK_END
3. <body> -> ε
4. <statements> -> <statement> EOL <statement>
5. <statement> -> IF <expression> !EOL BLOCK_BEGIN EOL <statements> EOL BLOCK_END !EOL ELSE !EOL BLOCK_BEGIN EOL <statements> EOL BLOCK_END EOL
6. <statement> -> FOR <init> SEMICOLON <expression> SEMICOLON <assign> BLOCK_BEGIN EOL <statements> EOL BLOCK_END
7. <statement> -> <function> EOL
8. <statement> -> <assign>
9. <statement> -> <init>
10. <statement> -> RETURN <value>
11. <statement> -> RETURN <expression>
12. <statement> -> RETURN <function> ?
13. <statement> -> ε
14. <assign> -> <identif> ASSIGN <function>
15. <assign> -> <identif> ASSIGN <value>
16. <assign> -> <identif> ASSIGN <expression>
17. <assign> -> ε
18. <init> -> ID INIT <value>
19. <init> -> ID INIT <function>
20. <init> -> ID INIT <expression>
21. <init> -> ε
22. <function> -> ID (<identif>)
23. <function> -> inputs (<identif>)
24. <function> -> inputi (<identif>)
25. <function> -> inputf (<identif>)
26. <function> -> print (<identif>)
27. <function> -> int2float (<identif>)
28. <function> -> float2int (<identif>)
29. <function> -> len (<identif>)
30. <function> -> substr (<identif>)
31. <function> -> ord (<identif>)
32. <function> -> chr (<identif>)
33. <arguments> -> <identif> <datatype> <argument>
34. <argument> -> , <identif> <datatype>
35. <argument> -> ε
36. <identif> -> ID <identifs>
37. <identifs> -> , ID <identifs>
38. <identifs> -> ε
39. <expression> -> ID COMPARE||ADD||SUB||MUL||DIV <value>
40. <expression> -> ID COMPARE||ADD||SUB||MUL||DIV <expression>
41. <expression> -> ID COMPARE||ADD||SUB||MUL||DIV <function> ?
42. <value> -> INTEGER
43. <value> -> DECIMAL
44. <value> -> STRING
45. <value> -> ID
46. <datatypes> -> <datatype>
47. <datatypes> -> <datatype>, <datatype>
48. <datatypes> -> ε
49. <datatype> -> int
50. <datatype> -> string
51. <datatype> -> float64
52. <datatype> -> ε

9.3 LL tabulka

	package	FUNC	EOF	ID	BLOCK_BEGIN	EOL	IF	FOR	SEMICOLON	RETURN	?	inputs	inputi	inputf	print	int2float	float2int	len	substr	ord	chr	,	integer	decimal	string	int	float64	$
<program>	1																											
<body>		2	3																									
<statements>				4		4	4	4		4		4	4	4	4	4	4	4	4	4	4							
<statement>				7,8,9		8,9,13	5	6		10,11,12		7	7	7	7	7	7	7	7	7	7							
<expression>				39,40,41																								
<init>				18,19,20		21			21																			
<assign>				14,15,16	17	17																						
<function>				22								23	24	25	26	27	28	29	30	31	32							
<value>				45																			42	43	44			
<identif>				36																								
<arguments>				33																								
<datatype>																						52			50	49	51	
<argument>																						34						
<identifs>					38	38			38		38											37,38			38	38	38	
<datatypes>																									46	46	46	

9.4 Precedenčná tabulka

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