PROJEKT Z PREDMETOV IFJ A IAL

IMPLEMENTÁCIA PREKLADAČA IMPERATÍVNEHO JAZYKA IFJ19

TÍM 36, VARIANTA 02

Vedúci: Žitňanský Adam, xzitna02 (26%)

Otčenáš Matej, xotcen01 (26%)

Dúdik Samuel, xdudik01 (26%)

Žitňanská Lívia, xzitna03 (22%)

Obsah:

1. Úvod:

Úlohou projektu je načítať zdrojový kód Ifj2019 a generovať výsledný medzikód v IFJcode2019 aj s návratovou hodnotou, kde jazyk Ifj2019 je podmnožina jazyka Python 3^2 .

2. Návrh riešenia:

Táto kapitola popisuje návrh riešenia, časti prekladača a predávanie informácií medzi nimi rovnako ako náš prístup a spôsob riešenia.

2.1. Lexikálna analýza:

Lexikálna analýza bola prvá časť, na ktorej sa začalo pracovať. Jej implementácia sa nachádza v súbore **scanner.s** s hlavičkovým súborom **scanner.h**. a vychádza z návrhu konečného automatu (viz príloha.č.1). Scanner číta vstupný súbor po znakoch. Z nich vytvorí tzv. tokeny. Tieto tokeny prechádzajú funkciou *read_token,* v ktorej sa v nekonečnom cykle while vyhodnotí, či daný token patrí medzi operátory, operandy, identifikátory, kľúčové slová, názov premennej či funkcie, alebo predstavuje číselnú hodnotu. Na základe tohto vyhodnotenia sa tok riadenia posúva do príslušného stavu, vykonajú sa požadované inštrukcie, vráti sa spracovaný token a presunie sa na ďalší, pre ktorý celý proces zopakuje. Spracované tokeny si vyžiada Parser na vykonianie syntaktickej analýzy. Dôležitá pre Scanner je dátová štruktúra zásobník **stack**, ktorá sa využíva hlavne pri rátaní počtu odsadení a generovaní príslušného tokene pre ne, a knižnica **string_lib**, ktorá slúži na prácu s reťazcami.

2.2. Syntaktická analýza:

Syntaktická analýza sa riadi LL gramatikou a je reprezentovaná súbormi **parser.c** a **expr_parser.c**. V hlavnom tele Parseru, *start_compiler*, sa zavolá inicializácia parseru a tabuliek pre lokálne a globálne symboly, potom sa vráti spracovaný príkaz cez funkciu *statement*. V tejto funkcii Parser žiada Scanner o tokeny, zostavuje „strom programu“ a pravidlami LL gramatiky (príloha č.2) ho redukuje. Parser je implementovaný pomocou odporúčanej metódy rekurzívneho zostupu. Pre prácu s tokenmi sa zaviedol typ zodpovedajúci symbolom, čo uľahčilo prevod tokenu na symbol. Niektoré si automaticky zodpovedali, iné bolo treba manuálne konvertovať, na čo sa použil podmienený príkaz „if“. Okrem toho sme sa rozhodli v ňom implementovať pomocné makrá.

2.2.1. Vyhodnocovanie výrazov

Na vyhodnocovanie výrazov pre Parser slúži Expr_parser, ktorý je Parserom zavolaný v prípade, že sa za premennou nevyskytuje ľavá zátvorka – teda sa na vstupe očakáva výraz, nie funkcia. V hlavnom tele *expression* sa po inicializácií Výrazového parseru v nekonečnom cykle while sa spracovávajú tokeny prislúchajúce výrazu. Keďže pre určenie, či sa jedná

a funkciu alebo výraz sa v Parseri musia načítať dva tokeny, Výrazovému parseru sú tiež predané dva, miesto jedného. Na základe precedentnej tabuľky, ktorá je uložená v matici 16x16, definuje Výrazový parser prednosť jednotlivých symbolov a shiftuje či redukuje daný symbol.

2.3. Sémantická analýza:

Sémantická analýza je úzko zviazaná so Syntaktickou analýzou a s generovaním kódu. Funkcie sú uložené v globálnej tabuľke symbolov, aby sa kedykoľvek dali zavolať. Parametre a premenné sú uložené v lokálnej tabuľke prvkov.

3. Implementácia

3.1. Tabulka symbolov

Implementácia tabuľky symbolov sa nachádza v súbore **symtable.c**. Inicializuje sa spolu s Parserom a to až dvakrát – ako tabuľka lokálnych a ako tabuľka globálnych symbolov. Po inicializácii sa do tabuľky pridajú vstavané funkcie ako inputs, print a iné. Ako alokačnú veľkosť tabuľky sme zvolili prvočíslo, aby sme sa vyhli zoskupovaniu hodnôt a zabezpečili tak rovnomernejšie distribuovanú tabuľku. Hashovaciu funkciu sme zvolili pre jej rýchlosť a malé množstvo kolízii, čo zvyšuje jej efektivitu.

3.2. Generovanie kódu

3.3. Zásobník

Popri inicializácii scanneru sa súčasne vytvára aj dátová štruktúra zásobník a spoločne so scannerom je aj uvoľnený. Jeho hlavná úloha je pri lexikálnej analýze, a to na kontrolu počtu odsadení a jeho zmeny, aby sme správne generovali tokeny pre indent a dedent a tým poslali syntaktickej analýze správu o zanoreniach príkazov. Podstatnú rolu však táto dátová štruktúra zohráva aj pri spracovaní výrazov. Z tohto dôvodu bolo nutné, aby bol schopný podporovať viacero dátových typov - na vyjadrenie veľkosti odsadenia podporuje dátový typ integer a pri spracovaní výrazov pre **expr_parser** zas štruktúru symbol.

4. Práca v tíme:

4.1. Komunikácia:

Práca na projekte začala v septembri. Ako komunikačný kanál sme zvolili Discord vzhľadom na predošlé skúsenosti s prácou v ňom. Preferovali sme osobné stretnutia a skupinovú prácu - všetci členovia tímu sa stretli a na mieste sa rozdelili úlohy a spoločne

pracovali. Takýto prístup povzbudzoval pracovnú morálku a mal za výsledok, že všetci členovia tímu bol aspoň čiastočne zapojený pri vývoji každej časti prekladača a jednoduchšie sme mohli konzultovať svoje nápady a prípadne si objasniť intention spôsobu riešenia.

4.2. Verzovací systém:

Vzhľadom na predošlé skúsenosti sme sa rozhodli pre prácu s Githubom, kvôli jeho prehľadnosti. Repozitár bol založený začiatkom septembra a každý člen tímu postupne pridával svoju prácu a prípadné úpravy do vetiev, ktoré sa po istej dobe alebo určitom počte úprav spájali v hlavnú.

4.3. Rozdelenie úloh:

Žitňanský Adam – lexikálna analýza, testy, zásobník, spracovanie výrazov
Otčenáš Matej – syntaktická analýza, tabuľka symbolov
Dúdik Samuel - generovanie kódu, testy
Žitňanská Lívia – dokumentácia, testy

Členovia sa súčasne podieľali na viacerých častiach implementácie, konzultácie , rady a návrhy

5. Záver

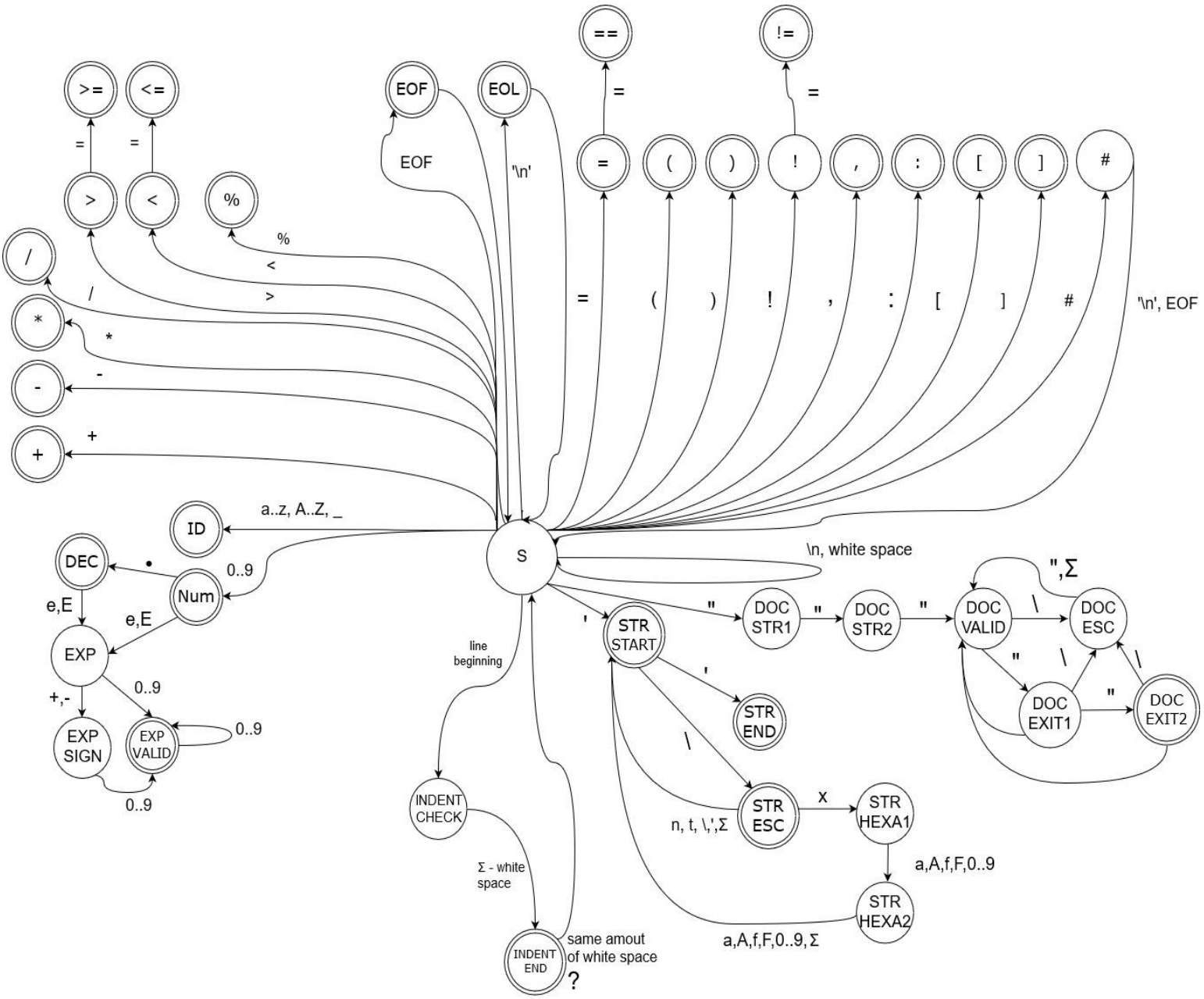
Projekt hodnotíme veľmi kladne, keďže náš tým bol rovnako zložený aj na predmete IVS, bolo jednoduchšie nadviazať na predošlý systém práce a využiť nadobudnuté skúsenosti. Prípadné problémy a nejasnosti zadania sme riešili osobne vzájomnou diskusiou. Pracovať sme začali dostatočne skoro, takže sme stihli projekt dokončiť do pokusného odovzdania a vďaka nemu a prebežnému testovaniu sme zvládli dodatočne vylepšiť naše riešenie. Okrem komunikácie a spoločnej práce, sme si zlepšili svoje schopnosti a počas implementácie sme aktívne využívali vedomosti z predmetov IFJ a IAL.

LITERATÚRA:

Wikipédia (adam dodá odkaz)

6. Prílohy:

6.1. Konečný automat lexikálneho analyzátoru



6.2. LL Gramatika

1. <statement> -> EOF

2. <statement> -> EOL <statement>

3. <statement> -> DEF ID (<params>): EOL INDENT <statement_inside> <end> DEDENT <statement>

4. <statement> -> IF <expression_start>: EOL INDENT <statement_inside> EOL DEDENT ELSE: EOL INDENT <statement_inside> <end> DEDENT <statement>

5. <statement> -> WHILE <expression_start>: EOL INDENT <statement_inside> <end> DEDENT <statement>

6. <statement> -> ID = <expression_start> <end> <statement>

7. <statement> -> PASS <end> <statement>

8. <statement> -> PRINT (<arg>) <end> <statement>

9. <statement> -> <value> <end> <statement>

10. <statement_inside> -> IF <expression>: EOL INDENT <statement_inside> EOL DEDENT ELSE: EOL INDENT <statement_inside> EOL DEDENT <statement>

11. <statement_inside> -> WHILE <expression_start>: EOL INDENT <statement_inside> EOL DEDENT <statement>

12. <statement_inside> -> ID = <expression_start> EOL <statement_inside>

13. <statement_inside> -> RETURN <expression_start> EOL <statement_inside>

14. <statement_inside> -> PASS EOL <statement_inside>

15. <statement_inside> -> PRINT (<arg>) EOL

16. <end> -> EOF

17. <end> -> EOL

18 <params> -> ID <next_params>

19. <params> -> eps

20. <next_params> -> , ID <next_params>

21. <next_params> -> eps

22. <expression_start> -> <value> <expression_next>

23. <expression_next> eps

24. <expression_next> OPERATOR <expression_start>

25. <value> ID

26. <value> INT

27. <value> DOUBLE

28. <value> STRING

29. <value> NONE

30. <value> -> ID (<arg>)

31. <value> -> INPUTS()

32. <value> -> INPTUF()

33. <value> -> INPUTI()

34. <value> -> LEN (<value>)

35. <value> -> SUBSTR (<value>, <value>, <value>)

36. <value> -> ORD (<value>, <value>)

37. <value> -> CHR (<value>)

38. <arg> -> eps

39. <arg> -> <value> <arg>

6.3. LL tabuľka

	EOF	EOL	DEF	IF	WHILE	,	ID	PASS	PRINT
<statement>	1	2	3	4	5		6	7	8
<statement_inside>				10	11		12	14	15
<end>	16	17							
<params>							18		
<next_params>						20			
<expression_start>									
<value>							25, 30		
<arg>									

- využitie pri syntaktickej analýze

6.4. Precedentná tabuľka

	+	-	*	/	//	>	==	!=	<=	>=	<	(	)	v	id	$
+	<	<	>	>	>	<	<	<	<	<	<	>	<	>	>	<
-	<	<	>	>	>	>	<	<	<	<	<	>	<	>	>	<
*	<	<	<	<	<	<	<	<	<	<	<	>	<	>	>	<
/	<	<	<	<	<	<	<	<	<	<	<	>	<	>	>	<
//	<	<	<	<	<	<	<	<	<	<	<	>	<	>	>	<
>	>	>	>	>	>						<	>	<	>	>	<
==	>	>	>	>	>						<	>	<	>	>	<
!=	>	>	>	>	>						<	>	<	>	>	<
<=	>	>	>	>	>						<	>	<	>	>	<
>=	>	>	>	>	>						<	>	<	>	>	<
<	>	>	>	>	>						<	>	<	>	>	<
(	>	>	>	>	>	>	>	>	>	>	>	>	>	>	>	<
)	<	<	<	<	<	<	<	<	<	<	<		<			<
v	<	<	<	<	<	<	<	<	<	<	<		<			<
id	<	<	<	<	<	<	<	<	<	<	<		<			<
$	>	>	>	>	>	>	>	>	>	>	>	>		>	>	

- využitie v expr_parseri