`dokumentace.pdf`, 3-5 stran A4.

vývojový cyk-lus, způsob práce v týmu, speciální použité techniky a algoritmy a různé odchylky od před-nášené látky či tradičních přístupů. Nezapomínejte také citovat literaturu a uvádět refe-rence na čerpané zdroje včetně správné citace převzatých částí (obrázky, magické konstanty,vzorce). Nepopisujte záležitosti obecně známé či přednášené na naší fakultě.

•Rozdělení práce mezi členy týmu (uveďte kdo a jak se podílel na jednotlivých částechprojektu; povinně zdůvodněte odchylky od rovnoměrného rozdělení bodů).

• překážkách, se kterými jste se při řešení setkali; problémech, které jste řešilia jak jste je řešili; atd.)V rámci dokumentace bude rovněž vzat v úvahu stav kódu jako jeho čitelnost, srozumitel-nost a dostatečné, ale nikoli přehnané komentáře.

PROJEKT Z PREDMETOV IFJ A IAL

IMPLEMENTÁCIA PREKLADAČA IMPERATÍVNEHO JAZYKA IFJ19

TÍM 36, VARIANTA 02

Vedúci: Žitňanský Adam, xzitna02 (%)

Otčenáš Matej, xotcen01 (%)

Dúdik Samuel, xdudik01 (%)

Žitňanská Lívia, xzitna03 (%)

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:

Kapitola popisuje návrh a implementáciu riešenia, časti prekladača a predávanie informácií medzi nimi.

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** - jeho reprezentácia konečným automatom sa nachádza nižšie v dokumentácii (viz obr.č.1). Scanner využíva zásobník a knižnicu **string_lib**, ktorá slúži na prácu s reťazcami. Hlavné telo Scanneru pozostáva zo switchu, ktorý získa zo vstupného súboru token a zisťuje počet odsadení, či token patrí medzi operátory, operandy, identifikátory, kľúčové slová alebo čísla. Na základe tohto vyhodnotenia vykonáva príslušné príkazy pre každý stav a posunie sa na ďalší token, pre ktorý celý proces zopakuje. Každý spracovaný token je odoslaný na syntaktickú analýzu do Parseru.

2.2. Syntaktická analýza:

Syntaktická analýza sa riadi LL gramatikou a je reprezentovaná súbormi **parser.s** a **expr_parser.c**. Po inicializácii Parseru a tabuľky symbolov, Parser postupne žiada tokeny zo Scannera a zostavuje „strom programu“ a pravidlami LL gramatiky (príloha č.2) ho zredukuje a vyhodnotí. 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“. Na vyhodnocovanie výrazov pre Parser slúži Expr_parser, ktorý je Parserom zavolaný v prípade, že sa za premennou nevyskytuje ľavá zátvorka. Parser obsahuje makrá slúžiace na sprehľadnenie kódu.

2.3. Sémantická analýza:

V štruktúre Parseru sú uložené tabulky symbolov – lokálna, globálna

3. Implementácia

3.1. Tabulka symbolov – máme dve

Implementácia tabuľky symbolov sa nachádza v súbore **symtable.c**. Ako alokovanú 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. Tabulky máme dve – na lokálne a globálne premenné.

3.2. Generovanie kódu

3.3. Zásobník

Dátovú štruktúru zásobník sme použili pri lexikálnej analýze na kontrolu počtu odsadení a jeho zmeny, aby sme správne generovali tokeny pre indent a dedent. Touto funkciou dopĺňa zásobník Scanner. Ďalej sme sa rohodli použiť túto dátovú štruktúru 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 skúsenosti s prácou v ňom. Preferovali sme osobné stretnutia a skupinovú prácu - všetci členovia tímu sa stretli a na mieste sa rozdelili úlohy.

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.

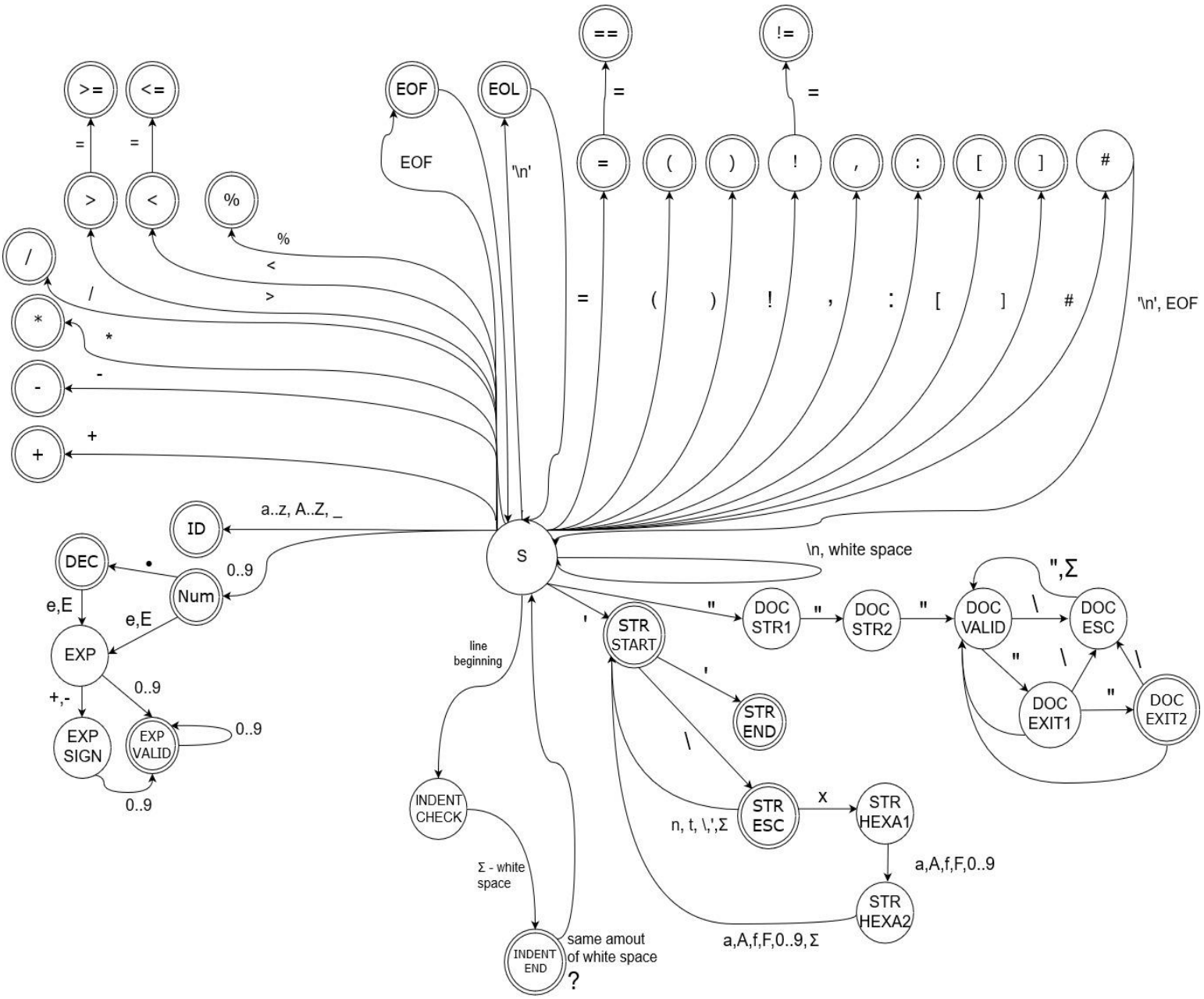
5. Záver

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	RETURN	OPERATOR	INT	DOUBLE	STRING	NONE	INPUTS	INPUTF	INPUTI	LEN	SUBSTZ	ORD	CHR	$
<statement>	1	2	3	4	5		6	7	8	9													
<statement_inside>				10	11		12	14	15	13													
<end>	16	17																					
<params>							18																19
<next_params>						20																	21
<expression_start>											24												22, 23
<value>							25, 30				26	26	27	28	29	31	32	33	34	35	36	37	
<arg>																							38, 39

6.4. Precedentná tabuľka

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