DOKUMENTÁCIA K PROJEKTU Z PREDMETOV IFJ A IAL

IMPLEMENTÁCIA PREKLADAČA IMPERATÍVNEHO JAZYKA IFJ19

TÍM 36, VARIANTA 02

Vedúci: Žitňanský Adam, xzitna02 (30%)

Otčenáš Matej, xotcen01 (30%)

Dúdik Samuel, xdudik01 (30%)

Žitňanská Lívia, xzitna03 (10%)

Obsah:

1. Úvod:

Dokumentácia zaznamenáva postup pri riešení projektu z predmetov IFJ a IAL, ktorého úlohou je načítať zdrojový kód IFJ19 a generovať výsledný medzikód v IFJcode19 , kde jazyk IFJ19 je podmnožina jazyka Python 3^2 . Zameriava sa na návrh riešenia, implementáciu, prácu v tíme a zhrnutie. Okrem toho obsahuje návrh konečného automatu a tabuľky ako prílohy.

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 (viď príloha 7.1.). Skener číta vstupný súbor po znakoch a tvorí z nich internú reprezentáciu lexémov takzvané tokeny. Hlavnou funkciou tohto modulu je funkcia *read_token()*, ktorá obsahuje implementáciu konečného automatu v nekonečnom cykle while, čo je bežná implementačná metóda, a vracia prečítaný token. Konečný automat je rozšírený o kontrolu odsadení, na čo využíva abstraktný dátový typ zásobník, a knižnicu pre dynamický reťazec v súboroch **string_lib.c** a **string_lib.h**, ktorá slúži na prácu s reťazcami, hlavne pri názvoch premenných a funkcií.

2.2. Syntaktická analýza:

Syntaktická analýza sa riadi pravidlami LL gramatiky (viď. príloha 7.2.) a je implementovaná odporučenou metódou rekurzívneho zostupu. Keďže sa jedná o syntaxou riadený preklad, Parser najprv inicializuje vo funkcii *start_compiler()* všetky potrebné moduly prekladača a následne sa začne sa Syntaktická analýza volaním funkcie *statement()* - Parser žiada Skener o tokeny, aplikuje jednotlivé pravidlá gramatiky a volá funkcie generátoru kódu. Pre sprehľadnenie kódu sú v Parseri niektoré funkcie rutiny implementované pomocou makier. Syntaktická analýza taktiež spolupracuje s Výrazovým parserom – v prípade, e narazí na výraz, zavolá jeho funkciu *expression().* Implementácia Syntaktickej analýzy sa nachádza v súbore **parser.c** s rovnomenným hlavičkovým súborom.

2.2.1. Vyhodnocovanie výrazov

Spracovanie výrazov je implementované Precedentným Shift-Reduce parserom riadeným precedentnou tabuľkou (uloženou v matici 16x16 – viď. príloha.č.4), ktorý dohliada na správne poradie generovania zásobníkových inštrukcií na výpočet výrazov

a taktiež zabezpečuje syntaktickú kontrolu výrazov. Implementovaný je v súbore **expr_parser.c.**

2.3. Sémantická analýza:

Sémantická analýza je úzko zviazaná so Syntaktickou analýzou a s generovaním kódu. Jej neoddeliteľnou súčasťou je tabuľka symbolov **symtable.c** . Keďže je IFJ19 dynamicky typovaný jazyk, chyby sémantickej analýzy sa vyhodnocujú nielen pri preklade, ale aj za behu, čo sme riešili volaním funkcií pre typovú kontrolu v medzikóde.

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 konštantu pre 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

Po inicializácii Generátoru (implementovaný v súboroch **generator.c** a **generator.h**) sa vygeneruje hlavička výstupného medzikódu, ktorá pozostáva zo vstavaných funkcií a funkcii zabezpečujúcich sémantické kontroly za behu. Ďalej generátor poskytuje implementáciu funkcii pre generovanie konštrukcii jazyka IFJ19, ktoré sú postupne volané Parserom počas syntaktickej analýzy. Medzikód je počas prekladu ukladaný do dynamického bufferu a v prípade, že preklad sa skončil bez chýb je následne vypísaný na štandardný výstup.

3.3. Zásobník

Abstraktný dátový typ zásobník (implementovaný v súboroch **stack.c** a **stack.h**) v projekte slúži viacerým účelom – z tohto dôvodu bolo nutné, aby podporoval viacero dátových typov. Zásobník položiek typu *integer* sa využíva pri Lexikálnej analýze na udržiavanie správneho počtu medzier odsadenia pre správne generovanie tokenov indent a dedent. Ďalej sa využíva v Precedenčnej analýze, tentokrát s položkami dátového typu symbol. Posledný nemenej významný účel má zásobník pri generovaní kódu a to konkrétne pre správne použitie náveští pri zanorených podmienkach a cykloch.

4. Práca v tíme:

4.1. Komunikácia:

Ako komunikačný kanál sme zvolili Discord. Preferovali sme osobné stretnutia a skupinovú prácu - členovia tímu sa stretli, na mieste sa rozdelili úlohy a všetci 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 sa mohli konzultovať návrhy a prípadne si objasniť spôsob riešenia.

4.2. Verzovací systém:

Vzhľadom na predošlé skúsenosti sme sa rozhodli projekt verzovať pomocou systému Git. Repozitár hostovaný na platforme Github bol založený začiatkom septembra a každý člen tímu postupne pridával svoju prácu a prípadné úpravy do vetiev, ktoré sa implementácii jednotlivých častí 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, počas osobných stretnutí sa konzultovali idei, podávali návrhy na riešenie a rady. Rozdiely v hodnotí sú v množstve práce vykonanej jednotlivými členmi.

5. Záver

Projekt hodnotíme veľmi kladne. Keďže náš tím bol rovnako zložený aj na predmete IVS, bolo jednoduchšie nadviazať na predošlý systém práce, využiť nadobudnuté skúsenosti a vylepšiť predošlé nedostatky. Prípadné problémy a nejasnosti zadania sme riešili osobne vzájomnou diskusiou, hľadali v materiáloch z prednášok, prípadne sme nahliadli do fóra k projektu. Pracovať sme začali dostatočne skoro, takže sme stihli projekt dokončiť do pokusného odovzdania a vďaka nemu a priebežnému 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.

6. LITERATÚRA:

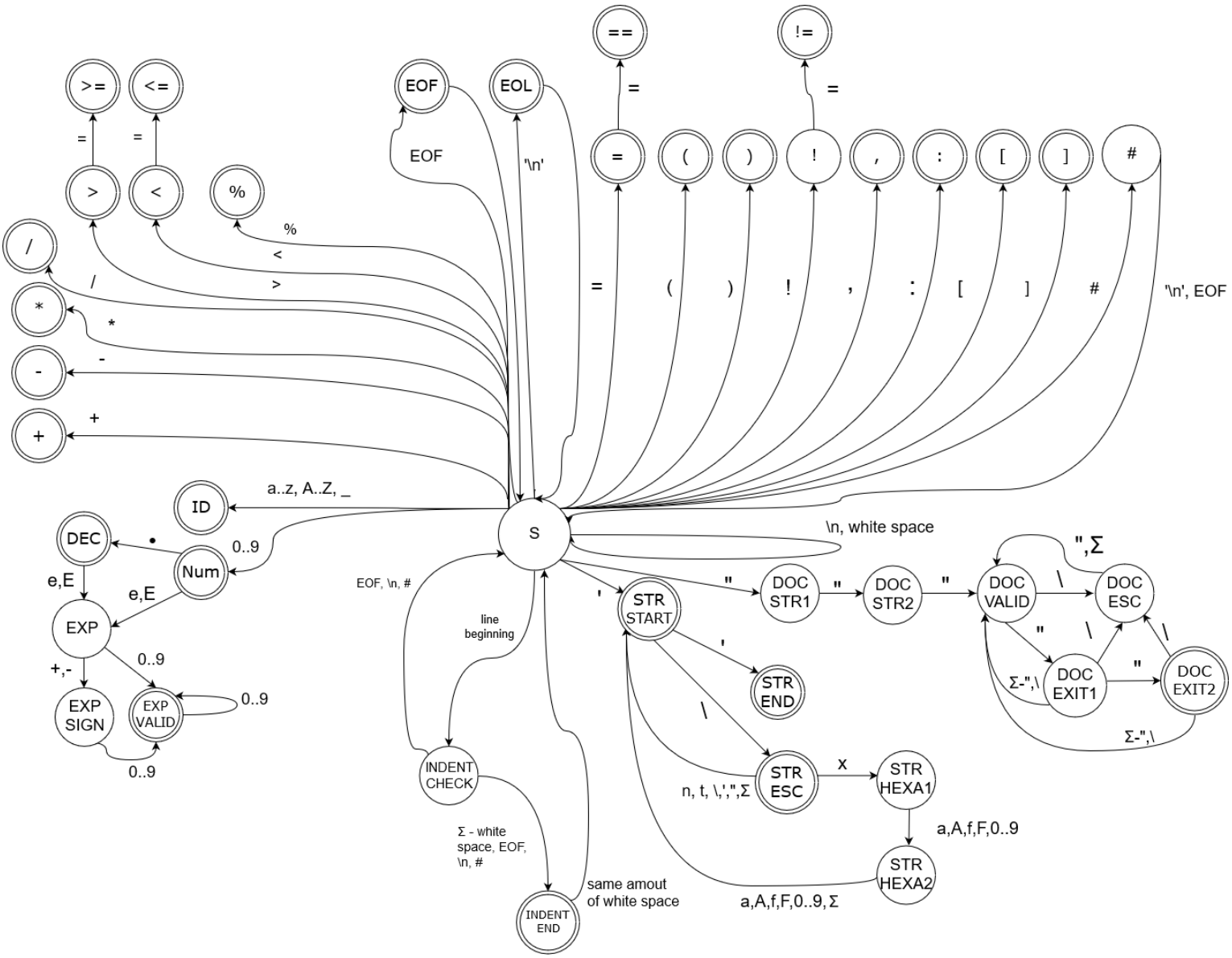
[1] Materiály k predmetu IFJ

[2] Študijná opora predmetu IAL

[3] Wikipédia: Shift-reduce parser
(odkaz: https://en.wikipedia.org/wiki/Shift-reduce_parser)

7. Prílohy:

7.1. Konečný automat lexikálneho analyzátoru



7.2. 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>: EOL INDENT <statement_inside> EOL DEDENT ELSE: EOL INDENT <statement_inside> <end> DEDENT <statement>

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

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

7. <statement> -> PASS <statement>

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

9. <statement> -> <expression_start> <statement>

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

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

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

13. <statement_inside> -> RETURN <expression> <end> <statement_inside>

14. <statement_inside> -> PASS <end> <statement_inside>

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

16. <statement_inside> -> <expression_start> <end> <statement_inside>

17. <end> -> EOF

18. <end> -> EOL

19. <end> -> eps

20. <params> -> ID <next_params>

21. <params> -> eps

22. <next_params> -> , ID <next_params>

23. <next_params> -> eps

24. <expression_start> -> <value>

25. <expression_start> -> <function_call>

26. <value> -> ID <expression>

27. <value> -> INT <expression>

28. <value> -> DOUBLE <expression>

29. <value> -> STRING <expression>
30. <value> -> NONE <expression>
31. <function_call> -> ID (<arg>)
32. <function_call> -> INPUTS()
33. <function_call> -> INPTUF()
34. <function_call> -> INPUTI()
35. <function_call> -> LEN (<arg>)
36. <function_call> -> SUBSTR (<arg>)
37. <function_call> -> ORD (<arg>)
38. <function_call> -> CHR (<arg>)
39. <arg> -> eps
40. <arg> -> <value> <arg>

7.3. LL tabuľka

	EOF	EOL	DEF	IF	WHILE	,	ID	PASS	PRINT	RETURN	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												16
<end>	17	18																				19
<params>							20															21
<next_params>						22																23
<expression_start>																						24,25
<value>							26				27	28	29	30								
<function_call>							31								32	33	34	35	36	37	38	
<arg>																						39,4

7.4. Precedentná tabuľka

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