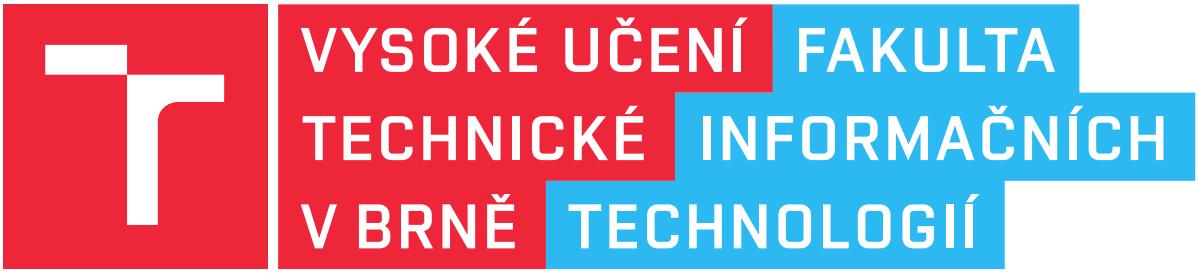
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



Dokumentácia k projektu do predmetu IFJ a IAL

Tým 51, varianta I

Denis Leitner (xleitn02) -vedúci - 30%
Maroš Holko (xholko01) - 30%
Vlastimil Rádsetoulal (xradse00) -30%
Michal Štábel (xstabe00) -10 %

1 Úvod

Táto dokumentácia sa zaoberá implementáciou prekladača imperatívneho jazyka IFJ17 do cieľového jazyka IFJcode17. Náš návrh sa skladá z troch hlavných častí: lexikálny analyzátor, syntaktický analyzátor a časti, ktorá sa zaoberá precedenčnou analýzou. Každá z týchto častí bude detailnejšie popísaná v nasledujúcej kapitole. Vybrali sme si variantu I, v ktorej sme mali za úlohu implementovať tabuľku symbolov pomocou binárneho vyhľadávacieho stromu. V dokumentácii sa ďalej zaoberáme prácou v našom tíme. Na záver skonštatujeme náš pohľad na tento projekt, čo nám projekt dal a čo nám vzal.

2 Implementácia

2.1 Lexikálny analyzátor

Lexikálny analyzátor pracuje na základe konečného automatu. Jeho hlavnou úlohou je čítať vstupný text, rozpoznávať lexémy a previesť ich na príslušné tokeny [1]. Token sme implementovali ako dátovú štruktúru, ktorá obsahuje typ tokenu a ak to typ vyžaduje tak aj jeho hodnotu. Príslušný token vyberá na základe konečného automatu, ktorého graf je na obrázku [1] v kapitole prílohy. V prípade nerozoznania lexémy automatom, nastáva lexikálna chyba a prekladač ukončí svoju činnosť.

Jadrom lexikálneho analyzátora je funkcia `get_token()`, ktorá vykonáva vyššie popísanú činnosť a je volaná syntaktickým analyzátorom. Nakoľko nezáleží na veľkosti písmen u identifikátorov a kľúčových slov v jazyku IFJ17, lexikálny analyzátor prevedie ich písmená na malé. Jeho ďalšou funkciou je formátovanie reťazcového literálu zadaného vo vstupnom súbore na tvar požadovaný inštrukciami jazyka IFJcode17. To zahŕňa prevod znakov s ASCII hodnotou 0-32, 35 a 92.

2.2 Syntaktický analyzátor

Syntaktický analyzátor tvorí hlavnú časť nášho prekladača. Jeho úlohou je kontrolovať či na základe prichádzajúcich tokenov je možné zostrojiť syntakticky správny program. Táto kontrola sa vykonáva na základe LL pravidiel [2], ktoré sú v prílohách. Na implementáciu syntaktickej analýzy sme si zvolili metódu vo forme rekurzívneho zostupu. Zdalo sa nám to jednoduchšie a nemuseli sme implementovať zásobník.

Popri kontrole syntaxe sa zároveň vykonávajú sémantické kontroly, akými sú: kontrola počtu a poradia parametrov funkcie, či je funkcia definovaná/deklarovaná, či funkcia vracia správny návratový typ a ďalej kontrola typov na ľavej a pravej strane priradenia. V tejto časti sa taktiež vykonáva časť generovania cieľového kódu. Na toto sme si vytvorili lineárny zoznam inštrukcií. Každá nová inštrukia sa pridá na koniec zoznamu. Ak bol celý vstupný súbor zapísaný správne lexikálne aj syntakticky, prešiel sémantickými kontrolami a nenastala žiadna interná chyba prekladača, všetky inštrukcie z lineárneho zoznamu sa vypíšu na štandardný výstup. Ak syntaktický analyzátor narazí na začiatok výrazu, predá riadenie precedenčnej analýze.

2.3 Precedenčná analýza

Precedenčná analýza pracuje metódou zdola hore a vykonáva spracovávanie výrazov podľa precedenčnej tabuľky [3]. Symbol `E` značí v tabuľke chybu a symbol `$` značí korektné ukončenie výpočtu

výrazu. Zároveň tu prebieha typová kontrola a prípadné pretypovanie. Výrazy sa počítajú generovaním do inštrukcií jazyka IFJcode17. Výsledok výrazu sa nachádza v premennej 'lf@$result'. Precedenčná syntaktická analýza využíva algoritmus, ktorý potrebuje k svojej činnosti zásobník, ktorý je implementovaný ako obojsmerne viazaný zoznam. Po spracovaní výrazu sa riadenie predáva naspäť syntaktickému analýzatoru. Predáva sa mu takisto aj typ vyhodnoteného výrazu('i', 'd', 's', 'b') a posledný načítaný token, ktorý už nepatrí do výrazu.

2.4 Tabuľka symbolov

Tabuľku symbolov sme implementovali pomocou binárneho vyhľadávacieho stromu. Máme 2 druhy tabuliek: lokálnu a globálnu. Lokálna tabuľka symbolov slúži na ukladanie premennných. Ukladá sa tu názov a typ premennej. Každá funkcia má svoju vlastnú lokálnu tabuľku symbolov a takisto aj hlavné telo programu má svoju lokálnu tabuľku.

Do globálnej tabuľky symbolov sa ukladajú funkcie a informácie o nich akými sú: názov funkcie, či bola deklarovaná/definovaná, jej návratový typ, počet parametrov, ich názvy a typy a ukazateľ do lokálnej tabuľky symbolov.

Na prácu s obidvomi tabuľkami používame funkcie:

- **Vyhľadávanie -** na vyhľadanie položky v tabuľke používame rekurzívne vyhľadávanie v binárnom vyhľadávacom strome, vyučované v predmete IAL [2].
- **Pridanie -** položka sa vloží ako list na vhodné miesto v strome, tak aby boli zachované vlastnosti BVS.
- **Zmazanie -** zmazanie prebieha na konci programu, tak že sa zmaže celý BVS.

3 Práca v tíme

3.1 Stretnutia

Stretnutia prebiehali každý týždeň v pondelok o 16:00. Fundamentálna časť našich stretnutí pozostávala z riešenia problémov spojených s projektom a štúdiom na tejto škole. Na prvom stretnutí, nám išlo hlavne o to, aby sme sa navzájom zoznámili a prešli si celé zadanie. Na ďalších stretnutiach sme urobili dekompozíciu projektu a rozdelenie jednotlivých častí medzi členov tímu, podľa individuálnych zručností. Ak niektorý z členov týmu narazil pri implementácii svojej časti na nejaký problém a nevedel ako ďalej, na stretnutí sme dali hlavy dokopy a spoločne sme to vyriešili.

3.2 Rozdelenie práce

- Denis Leitner: LL gramatika, syntaktická a sémenatická analýza, generovanie programu, tabuľka symbolov, testovanie, dokumentácia
- Maroš Holko: LL gramatika, precedenčná analýza, typová kontrola vo výrazoch, precedenčná tabuľka, generovanie vstavaných funkcií, testovanie, dokumentácia
- Vlastimil Rádsetoulal: Konečný automat, lexikálna analýza, testovanie, dokumentácia
- Michal Štábel: transformácia konečného automatu do pdf, testovanie, dokumentácia

3.3 Metriky kódu

- Počet riadkov : 6575
- Počet súborov : 15 + Makefile

4 Záver

Napriek tomu, že sme do projektu vložili veľa času, ktorý sme mohli stráviť pri príjemnejších aktivitách, neľutujeme to. Projekt nám priniesol veľa programátorských skúseností, ale hlavne nám priniesol nový pohľad na to, koľko práce ide do vývoja takého prekladača. Dokážeme viac oceniť prácu programátorov, ktorí vyvýjajú prekladače ako napríklad GCC, ktoré su oveľa komplexnejšie a náročnejšie na vývoj ako ten náš.

Naučili sme sa pracovať v malom tíme a efektívne rozdeliť prácu. Overili sme si, že termíny dokončenia práce je nutné dodržiavať aj keď ide len o interné termíny, pretože častokrát práca jedného člena závisela od splnenia práce ďalších členov. Do budúcna by sme mohli zlepšiť prácu s verzovacími systémami, keďže ich efektívne používanie by nám ušetrilo veľa námahy a času. Tento projekt bol tak ako iné projekty na tejto škole časovo náročný, čo sa odzrkadlilo pri prvom pokusnom odovzdaní, kde sme ešte nemali úplne zakomponovaný lexikálny analyzátor. Napriek týmto problémom sa nám podarilo projekt dokončiť a funguje podľa našich predstáv. Bohužiaľ na rozšírenia a optimalizácie nám už nezostal žiaden čas.

5 Referencie

Literatúra

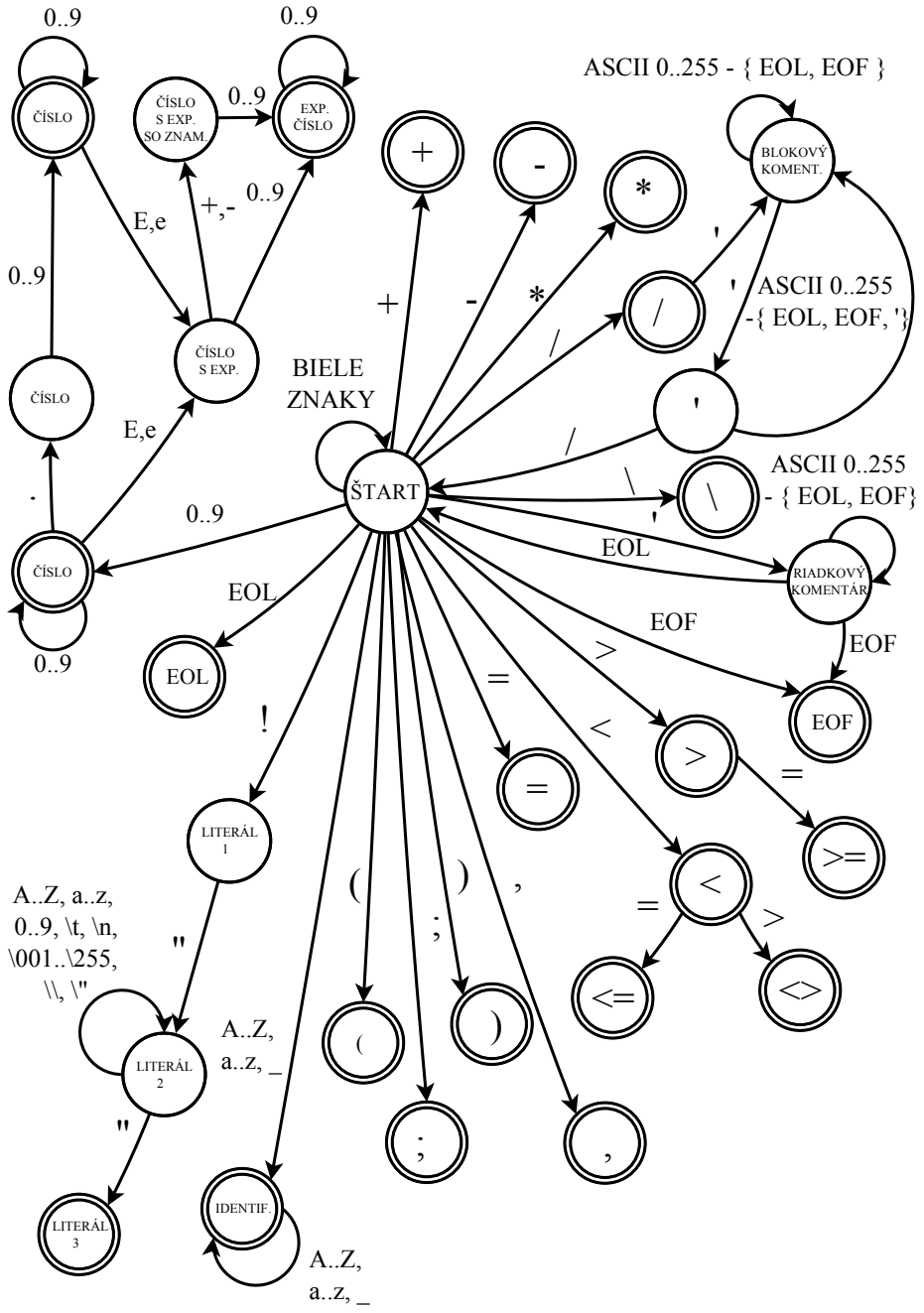
[1] Meduna, A.; Lukáš, R.: *Slidy z predmetu IFJ*.

[2] Prof. Ing.Jan M Honzík, C.: *Algoritmy IAL, Studijní opora*.

6 Prílohy

(A) Konečný automat

Obr. 1: Konečný automat



(B) LL gramatika

1. <Program> → <FceDec> Scope EOL <StatList> End Scope EOL EOF
2. <FceDec> → <Declaration> EOL <FceDec>
3. <FceDec> → <Definition> EOL <FceDec>
4. <FceDec> → ε
5. <Declaration> → Declare Function ID (<ItemList>) As <Type>
6. <Definition> → Function ID (<ItemList>) As <Type> EOL <StatList> End Function
7. <VarDec> → DIM ID As <Type> <VarDef>
8. <Assign> → ID = <RS>
9. <VarDef> → = **EXPR**
10. <VarDef> → ε
11. <ExprList> → **EXPR** ; ExprList
12. <ExprList> → ε
13. <ItemList> → ε
14. <ItemList> → ID As <Type> <Item2List>
15. <Item2List> → ε
16. <Item2List> → , ID As <Type> <Item2List>
17. <ParList> → ε
18. <ParList> → <ParPar> <Par2List>
19. <Par2List> → ε
20. <Par2List> → , <ParPar> <Par2List>
21. <RS> → **EXPR**
22. <RS> → ID (<ParList>)
23. <Stat> → Input ID
24. <Stat> → Print **EXPR** ; <ExprList>
25. <Stat> → If **EXPR** Then EOL <StatList> Else EOL <StatList> End If
26. <Stat> → Do While **EXPR** EOL <StatList> Loop
27. <Stat> → Return **EXPR**

28. <Stat> → <Assign>
29. <Stat> → <VarDec>
30. <StatList> → ε
31. <StatList> → <Stat> EOL <StatList>
32. <Type> → Integer
33. <Type> → Double
34. <Type> → String
35. <ParPar> → ID
36. <ParPar> → Integer
37. <ParPar> → Double
38. <ParPar> → String

(C) LL tabuľka

Obr. 2: LL tabuľka

LL tabuľka	Scope	Declare	Function	End	DIM	ID	(	)	,	Input	Print	If	Do	Return	Else	Loop	Integer	Double	String	EOL	Int	Str	Doub	=
<Program>	1	1	1																					
<FceDec>	4	2	3																					
<Declaration>		5																						
<Definition>			6																					
<VarDec>					7																			
<Assign>						8																		
<VarDef>																				10				9
<ExprList>						11	11													12	11	11	11	
<ItemList>						14		13																
<Item2List>								15	16															
<ParList>						18		17																
<Par2List>		-						19	20															
<RS>						22/21	21														21	21	21	
<Stat>					29	28				23	24	25	26	27										
<StatList>				30	31	31				31	31	31	31	31	30	30								
<Type>																	32	33	34					
<ParPar>						35											36	37	38					

(D) Precedenčná tabuľka

Obr. 3: Precedenčná tabuľka

	*	+	-	/	\	<	>	=	<=	>=	<>	(	)	id	int	doub	str	bool	$
*	>	>	>	>	>	>	>	>	>	>	>	<	>	<	<	<	<	E	>
+	<	>	>	<	<	>	>	>	>	>	>	<	>	<	<	<	<	E	>
-	<	>	>	<	<	>	>	>	>	>	>	<	>	<	<	<	<	E	>
/	>	>	>	>	>	>	>	>	>	>	>	<	>	<	<	<	<	E	>
\	<	>	>	<	>	>	>	>	>	>	>	<	>	<	<	<	<	E	>
<	<	<	<	<	<	E	E	E	E	E	E	<	>	<	<	<	<	<	>
>	<	<	<	<	<	E	E	E	E	E	E	<	>	<	<	<	<	<	>
=	<	<	<	<	<	E	E	E	E	E	E	<	>	<	<	<	<	<	>
<=	<	<	<	<	<	E	E	E	E	E	E	<	>	<	<	<	<	<	>
>=	<	<	<	<	<	E	E	E	E	E	E	<	>	<	<	<	<	<	>
<>	<	<	<	<	<	E	E	E	E	E	E	<	>	<	<	<	<	<	>
(	<	<	<	<	<	<	<	<	<	<	<	<	=	<	<	<	<	<	E
)	>	>	>	>	>	>	>	>	>	>	>	E	>	E	E	E	E	E	>
id	>	>	>	>	>	>	>	>	>	>	>	E	>	E	E	E	E	E	>
int	>	>	>	>	>	>	>	>	>	>	>	E	>	E	E	E	E	E	>
doub	>	>	>	>	>	>	>	>	>	>	>	E	>	E	E	E	E	E	>
str	>	>	>	>	>	>	>	>	>	>	>	E	>	E	E	E	E	E	>
bool	E	E	E	E	E	>	>	>	>	>	>	E	>	E	E	E	E	E	>
$	<	<	<	<	<	<	<	<	<	<	<	<	E	<	<	<	<	E	$

Pravidlá pre precedenčnú analýzu:

$N \rightarrow N * N$
$N \rightarrow N + N$
$N \rightarrow N - N$
$N \rightarrow N/N$
$N \rightarrow N \backslash N$
$N \rightarrow N < N$
$N \rightarrow N > N$
$N \rightarrow N = N$
$N \rightarrow N <= N$
$N \rightarrow N >= N$
$N \rightarrow N <> N$
$N \rightarrow int$
$N \rightarrow double$
$N \rightarrow string$
$N \rightarrow (N)$