

Projektová dokumentace
Implementace překladače imperativního jazyka IFJ 18
Tým 123
Varianta I

Martina Tučková	(xtucko00)	25 %
Martina Jendrálová	(xjendr03)	25 %
Marek Šťastný	(xstast33)	25%
Martin Janda	(xjanda27)	25 %

Obsah

1 Úvod a vysvětlení k jazyku IFJ 18	2
2 Implementace a části našeho řešení	2
2.1 – Lexikální analyzátor neboli scanner	
2.2 – Syntaktická a sémantická analýza	
3 Použité algoritmy a speciální datové struktury	
3.1 – Binární strom, implementován rekurzí	3
3.2 – Dynamické řetězce	3
3.3 – Alokace zásobníků	3

1 Úvod a zadání našeho projektu

Tato dokumentace slouží k popisu implementace překladače imperativního jazyka IFJ 18. Naším cílem bylo v jazyce C napsat program, jež načítá zdrojový kód. Tento zdrojový kód je zadán v jazyce IFJ 18, jež je zjednodušenou podmnožinou jazyka Ruby 2.0 a přeloží jej do výsledného jazyka IFJcode 18. Program dále vrací návratovou hodnotu dle situace, kde 0 značí, že překlad proběhl v pořádku a vrací jinou návratovou hodnotu v případě některé chyby.

2 Implementace a návrh našeho projektu

Náš projekt je sestaven z většího počtu částí, jejich jednotlivá implementace je popsána v následující kapitole.

2.1 Lexikální analýza

Jako jednu z prvních částí jsme začali pracovat na lexikálním analyzátoru, neboli scanneru. Jeho hlavním úkolem je načítat ze vstupu posloupnosti příchozích znaků (lexémů) a dále je zpracovávat. Takto zpracované lexémy, dále tokeny, jsou v další části předávány syntaktické analýze. Mezi tokeny, jež rozlišujeme, patří EOL, EOF, identifikátory, přírovnávací a matematické operátory, desetinné a celé číslo, klíčová slova a další znaky, patřící do části letošního projektu IFJ 18. Hlavní funkcí scanneru v našem případě je funkce getToken.

Analyzátor byl implementován na základě modelu konečného automatu, jež je graficky přidán na konec této dokumentace.

2.2 Syntaktická a sémantická analýza

Syntaktický analyzátor, neboli parser, je nejdůležitější částí celého programu. Spouští funkce scanneru a načítá posloupnost příchozích tokenů, jež si dál zpracovává a případně žádá scanner o další token. Takovýto syntaktický analyzátor se řídí LL – gramatikou zakreslené v LL tabulce a v našem případě je použita metoda procházení rekurzivním sestupem seshora dolů. LL gramatika i LL tabulky jsou dále také přiloženy v grafickém zpracování.

Společně se syntaktickou analýzou se provádí i sémantická analýza. Ta kontroluje přijaté symboly ze syntaktické analýzy a přiřazuje jim správný význam. Sémantická kontrola také hlídá, aby nedošlo k implicitnímu deklarování již vytvořené proměnné a pracuje s precedenční gramatikou a precedenční tabulkou, kde jsou jednotlivé symboly znázorněny dle jejich priorit. Se sémantickou analýzou je úzce spjatá podoba ukládání právě oněch přijatých symbolů. To v našem případě bylo implementováno rekurzivní metodou binárního stromu. Ona precedenční tabulka je také dále přiložena v grafické podobě.

3 Použité algoritmy a speciální struktury

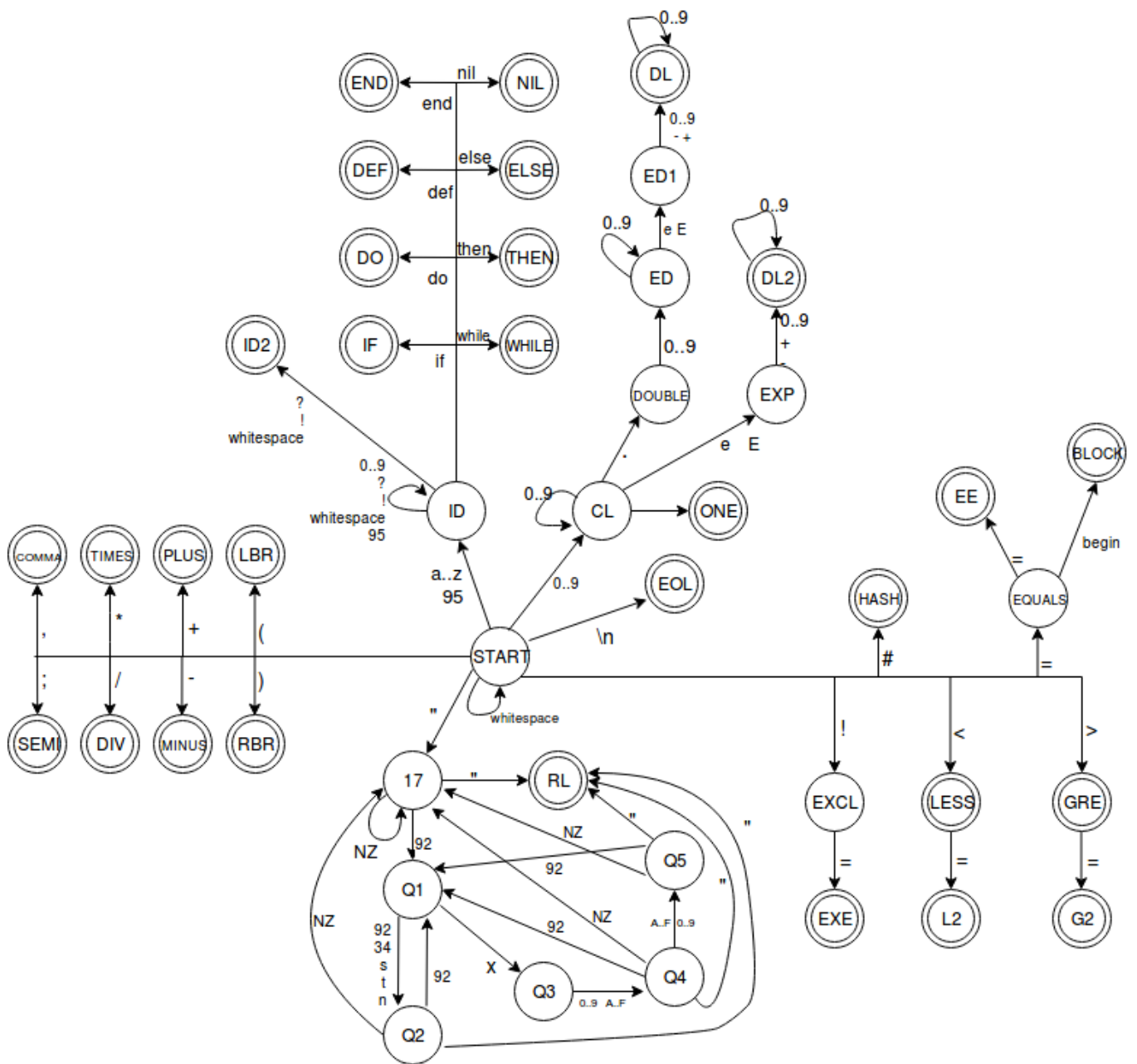
Během projektu jsme použili i techniky, jež jsme se naučili v předmětu IAL – algoritmy. Například tabulku symbolů jsme mohli implementovat pomocí jednoho ze dvou možných algoritmů. V našem případě na základě zvoleného zadání to bylo metodou binárního stromu.

3.1 Tabulka symbolů – Binary tree

Tabulku symbolů jsme implementovali v souboru symtable. My provedli její implementaci metodou binárního stromu. Binární strom je datová struktura pro ukládání a vyhledávání dat. V našem případě obsahuje řadu funkcí pro rychlejší vyhledávání a práci s hodnotami.

3.2 Dynamické řetězce

Konečný automat lexikálního analyzátoru



Precedenční tabulka

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