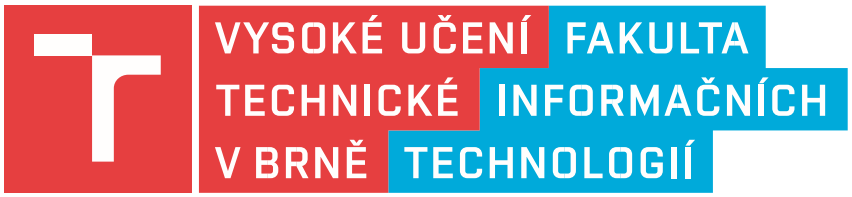



Projektová dokumentace

# 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 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 následujících podkapitolách.

## 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, přirovnávací a matematické operátory, desetinné a celé číslo, klíčová slova a další znaky, patřící do části letošního projektu IFJ 18. Hlavní funkcí scanneru v našem případně 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 ukládaní právě oněch přijatých symbolů. To v našem případně 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 případně na základě zvoleného zadání to bylo metodou binárního stromu.

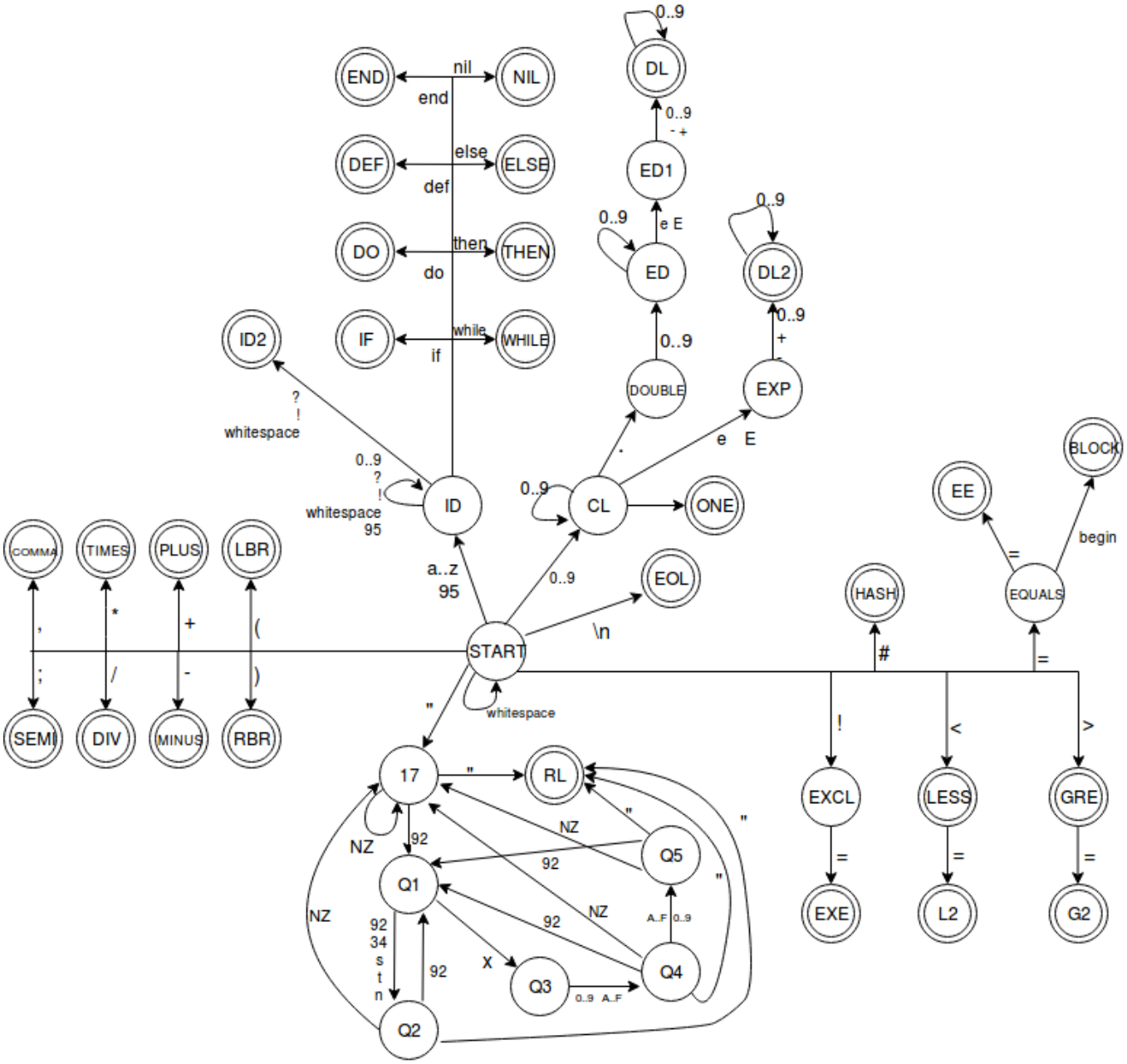
## 3.1 Tabulka symbolů – Binary tree

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

Pracovali jsme na souboru strings.c , jež slouží pro práci s řetězci dynamické délky. Nemůžeme s jistotou vědět, jak dlouhý řetězec můžeme očekávat, proto tento pomocný soubor alokuje případné místo pro příchozí řetězce, překopírovává jejich hodnoty a po provedené práci alokované místo opět uvolňuje pro pozdější využití.

# 6. Konečný automat lexikálního analyzátoru



# 7. LL Gramatika

1. <prog> -> <stat_list> EOF
2. <stat_list> -> <stat> EOL <stat_list>
3. <stat_list> -> ε
4 <stat> -> ε
5. <stat> -> def id ( <param_list> ) EOL <stat_list> end
6. <param_list> -> ε
7. <param_list> -> id <part>
8. <part> -> id <part>
9. <part> -> ε
10. <stat> -> exp
11. <stat> -> if exp then EOL <stat_list> else EOL <stat_list> end
12. <stat> -> while exp do EOL <stat_list> end
13. <stat> -> id <assignment>
14. <assignment> -> ε
15. <assignment> -> = <assigned>
16. <assigned> -> exp
17. <assigned> -> <f_call>
18. <f_call> -> id <param_group>
19. <param_group> -> <term_list>
20. <param_group> -> ( <term_list> )
21. <term_list> -> ε
22. <term_list> -> <term> <term_part>
23. <term_part> -> <term> <term_part>
24. <term_part> -> ε
25. <term> -> id
26. <term> -> int
27. <term> -> float
28. <term> -> string
29. <term> -> nil

# 8. LL Tabulka

| | def | id | if | while | else | end | = | ( | ) | , | EOL | EOF | int | float | string | nil | exp |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <prog> | 1 | 1 | 1 | 1 | | | | | | | 1 | 1 | | | | | 1 |
| <stat_list> | 2 | 2 | 2 | 2 | 3 | 3 | | | | | 2 | 3 | | | | | 2 |
| <stat> | 5 | 13 | 11 | 12 | | | | | | | 4 | | | | | | 10 |
| <f_call> | | 18 | | | | | | | | | | | | | | | |
| <param_group> | | 19 | | | | | | 20 | | | 19 | | 19 | 19 | 19 | 19 | |
| <param_list> | | 7 | | | | | | | 6 | | | | | | | | |
| <part> | | | | | | | | | 9 | 8 | | | | | | | |
| <assigned> | | 17 | | | | | | | | | | | | | | | 16 |
| <assignment> | | | | | | | 15 | | | | 14 | | | | | | |
| <term_list> | | 22 | | | | | | | 21 | | 21 | | 22 | 22 | 22 | 22 | |
| <term_part> | | | | | | | | | 24 | 23 | 24 | | | | | | |
| <term> | | 25 | | | | | | | | | | | 26 | 27 | 28 | 29 | |

# 9. Precedenční tabulka

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