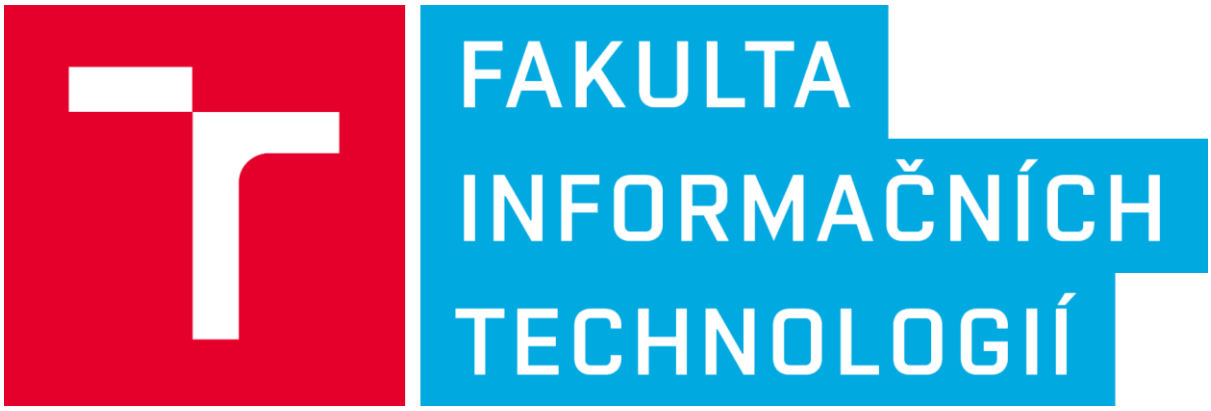
# Vysoké učení technické v Brně
## Fakulta informačních technologií




## Dokumentace k projektu do předmětů IFJ a IAL

Implementace překladače imperativního jazyka IFJ18.

Tým 27, varianta I

5. prosince 2018


| | | |
|---|---|---|
| Aleš Tetur (vedoucí) | (xtetur01), | 25% |
| Šimon Matyáš | (xmatya11), | 25% |
| Jan Tilgner | (xtilgn01), | 25% |
| Jan Bíl | (xbilja00), | 25% |

# Obsah

# 1 Úvod

Tato dokumentace popisuje implementaci překladače imperativního jazyka IFJ18. V naší variantě číslo I. je zadáno, že tabulka symbolů je implementována pomocí binárního vyhledávacího stromu. Projekt jsme rozdělili na tři části:

1. Lexikální analýza (scanner)
2. Syntaktická a sémantická analýza
3. Generování cílového kódu

Každou částí se zabývá jedna kapitola této dokumentace. V dokumentaci jsou rovněž obsaženy všechny přílohy: LL gramatika, LL tabulka, precedenční tabulka a konečný automat pro lexikální analýzu.

# 2 Překladač

Překladač je softwarový prostředek používaný pro překlad víceúrovňových jazyků do strojových kódů či jiných konkrétních jazyků. Překladač nebo také kompilátor z obecného hlediska je stroj, nebo spíše program, provádějící překlad ze vstupního jazyka do jazyka výstupního. Z matematického hlediska je kompilátor funkce, jež mapuje jeden nebo více kódů podle parametrů pro překlad na kód výstupního jazyka.

# 3 Lexikální analyzátor

Cílem lexikálního analyzátoru je čtení vstupních znaků, nebo také lexémů, jejich zpracování a předání v podobě tokenu parseru, který jej dále zpracuje. Další funkcí scanneru je také odhalení chyb v syntaxi vstupního programu. Tato kontrola probíhá během běhu scanneru. Token je implementován pomocí struktury Token_t, která obsahuje proměnné s: typem tokenu, hodnotou tokenu (pokud token hodnotu má). Při implementaci scanneru jsme vycházeli z návrhu konečného automatu, viz. příloha 8.4. Po správné naskenování tokenu, předá tuto proměnnou parseru. Poté scanner čeká, než je opět volán pro další token.

# 4 Syntaktická a sémantická analýza

## 4.1 Syntaktická analýza (bez zpracování výrazů)

Vstupem části překladače, která vykonává syntaktickou analýzu (parser), je posloupnost tokenů, které jsou získány pomocí lexikální analýzy (scanneru). Parser si bere tyto tokeny jeden po druhém a v případě, že posloupnost je v souladu s pravidly vyplívajícími z vyrobené LL gramatiky, tak byla tvorba derivačního stromu nasimulována úspěšně. Pokud přijde token, který neodpovídá posloupnosti dané LL gramatikou, znamená to, že simulace derivačního stromu selhala a analýza se ukončí a vrací příslušnou chybu. Zároveň se v průběhu ukládají nově definované proměnné do globální tabulky symbolů.

### 4.2 Syntaktická analýza (zpracování výrazů)

Zpracování výrazů se provádí pomocí funkce expression, která se volá v případě, že parser rozpozná volání funkce nebo výraz. V případě volání funkce expression kontroluje, zda je zadán správný počet parametrů a v kladném případě pomocí rámců předává parametry volané funkci. Zároveň se provádí i sémantická kontrola kontrolou předchozí definice případných proměnných v tabulce symbolů. Pokud je nalezen výraz, je pomocí precedenční tabulky převeden na postfixový tvar. V průběhu převodu je kontrolováno, jestli jsou všechny proměnné výrazu již definovány. Následně jsou v postfixovém výrazu provedeny případné typové převody a výraz je převeden na kód v jazyce IFJcode2018.

## 5 Generování cílového kódu

### 5.1 Vnitřní funkce

Vestavěné funkce **ord,substr,chr** a **len** jsou generované pomocí dvou funkcí. Funkce končící *_generate* vygenerují samostatnou funkci, a funkce končící *_call* vygenerují kód kde se pomocí zásobníkových příkazů nahraji parametry a funkce zavolá.

Funkce **print** si hlídá, aby parametry byly oddělené čárkou a pokud příkaz začíná závorkou tak musí závorkou i končit. Hned jak narazí na tisknutelný parametr, tak přidá příslušnou instrukci do seznamu instrukcí. Součásti funkce print je také funkce, která převádí řetězce znaků do vhodné formy pro IFJcode18.

**Inputi**,**Inputf**,**Inputs** zavolají příkaz READ a výsledek nahrají do výsledné proměnné.

### 5.2 Cílový kód

Cílový kód je generován v průběhu procházení celého programu zároveň se syntaktickou a sémantickou analýzou. Jednotlivé příslušné instrukce se generují do jednosměrného seznamu instrukcí, který je po projití celého kódu vypsán na standartní výstup.

# 6 Práce v týmu

## 6.1 Komunikace

Pro komunikaci jsme používali Facebook včetně sdílení souborů při tom jsme zjistili nevýhody takového sdílení a při dalších týmových projektech nejspíše využijeme GitHub. Dále jsme se v průběhu semestru několikrát sešli ve školní knihovně a domlouvali se na rozdělení práce.

## 6.2 Rozdělení

Bíl Jan:

- Tvorba Syntaktického analyzátoru
- Tvorba Sémantického analyzátoru
- Generování kódu
- Návrh LL gramatiky

Matyáš Šimon:

- Generovaní kódu vestavěných funkcí

Tetur Aleš:

- Lexikální analyzátor

Tilgner Jan:

- Tvorba Syntaktického analyzátoru
- Tvorba Sémantického analyzátoru
- Generování kódu
- Návrh LL gramatiky

# 7 Závěr

Jednalo se asi o nejrozsáhlejší projekt, na kterém jsme zatím pracovali, navíc pro nás bylo nové pracovat v týmu. Projekt byl přínosný, nabrali jsme hodně zkušeností ohledně práce v týmu což se nám určitě bude hodit v budoucí kariéře.

# 8 Přílohy

## 8.1 Precedenční tabulka

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

## 8.2 LL Tabulka

| | def | token_id | token_functon_id | if | while | eol | epsilon | token_, |
|---|---|---|---|---|---|---|---|---|
| prog | 1 | 3 | 3 | 4 | 5 | 2 | | |
| define_function | 6 | | | | | | | |
| statement | | 8 | 8 | 9 | 10 | 7 | | |
| call_func | | 11 | 11 | | | | | |
| param_list | | | | | | | | |
| next_param | | | | | | | | |
| def_param | | 12 | | | | | | |
| next_def_param | | | | | | | 13 | 14 |
| param | | | | | | | | |
| statement_list | | 16 | 16 | 17 | 18 | 15 | | |

## 8.3 LL Gramatika

| | | | |
|---|---|---|---|
| **1** | <prog> | **>** | <define-function> EOL <prog> |
| **2** | <prog> | **>** | EOL <prog> |
| **3** | <prog> | **>** | <statement> EOL <prog> |
| **4** | <define_function> | **>** | token_def token_id token_( <def_param_list> token_) EOL <statement_list>  token_end |
| **5** | <statement> | **>** | token_id token_= <expression> |
| **6** | <statement> | **>** | token_if  <expression> token_then EOL <statement_list> token_else EOL <statement_list> token_end |
| **7** | <statement> | **>** | token_while  <expression> token_do EOL <statement_list> token_end |
| **8** | <statement> | **>** | token_id token_=  <call_func> |
| **9** | <statement> | **>** | <expression> |
| **10** | <statement> | **>** | EOL |
| **11** | <statement> | **>** | <call_func> |
| **12** | <call_func> | **>** | token_id token_(  <param-list>  token_) |
| **13** | <call_func> | **>** | token_id  <param-list> |
| **14** | <param_list> | **>** | <param> <next_param> |
| **15** | <next_param> | **>** | token_, <param> <next_param> |
| **16** | <next_param> | **>** | epsilon |
| **17** | <def_param> | **>** | token_id |
| **18** | <def_param> | **>** | epsilon |
| **19** | <def_param_list> | **>** | <def_param> <next_def_param> |
| **20** | <next_def_param> | **>** | token_, token_id <next_def_param> |
| **21** | <next_def_param> | **>** | epsilon |
| **22** | <param> | **>** | epsilon |
| **23** | <param> | **>** | <expression> |
| **24** | <statement_list> | **>** | <statement> <statement_list> |
| **25** | <statement_list> | **>** | epsilon |

## 8.4 Konečny automat lexikalniho analyzatoru