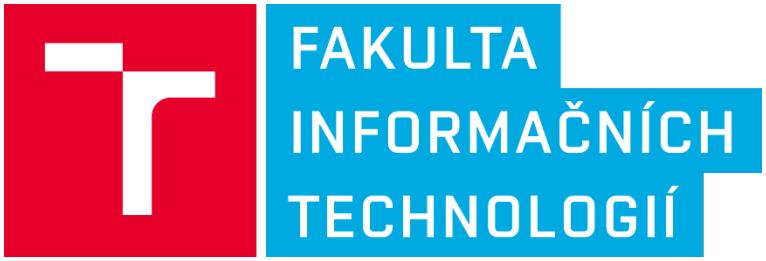
# VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
# Fakulta informačních technologií




## Dokumentace k projektu

Implementace překladače imperativního jazyka IFJ19

| Tým 095, varianta I | | | |
|---|---|---|---|
| Vedoucí | Dubec Matej | xdube00 | 25 % |
| | Spišák Samuel | xspisa02 | 25 % |
| | Zobaník Michal | xzoban01 | 25 % |
| | Malínek Libor | xmalin30 | 25 % |

# Obsah

# 1. Úvod

V této zprávě dokumentujeme náš návrh a implementaci překladače imperativního jazyka IFJ19 a postupu jakým jsme postupovali.

Dokumentace je rozdělená do kapitol, kde podrobněji rozebíráme práce v týmu, přístup k implementaci a způsoby řešení vzniklých problémů.

# 2. Týmová práce

## 2.1.Způsob práce

Na projektu jsme začali dělat v půlce října. Na prvním setkání jsme si rozdělili celkový projekt na dílčí části a následně jsme si je rozdělili. Na částech jsme většinou pracovali samostatně, avšak kontrola probíhala napříč celým týmem.

## 2.2.Komunikace

Komunikace mezi členy probíhala z větší části elektronicky, prostřednictvím komunikačního serveru Discord.

Osobní setkání proběhla průměrně jednou za 14 dní, kde jsem si řekli, jak postupovat dál. Domluva nebyla snadná, poněvadž každý z členů měl jinak uzpůsobený čas, kdy mohl. Po celý čas byla komunikace vedena v přátelském duchu.

## 2.3.Rozdělení práce

| | |
|---|---|
| Dubec Matej | Token, Stack, Scanner, Errors |
| Spišák Samuel | PSA |
| Zobaník Michal | Parser, Scanner, LL gramatika |
| Malínek Libor | Symtable, Dokumentace |

Každý člen se účastnil na tvorbě generátoru kódu.

## 2.4.Správa kódu

Pro správu kódu jsme nejdřív používali komunikační server Discord, kde jsem si mezi sebou posílali různé části kódu, případně i pomocné materiály. Ke konci jsme začali používat verzovací systém GitHub.

# 3. Implementace

## 3.1.Lexikální analyzátor

Je implementovaný v podobě konečného automatu, který načítá jednotlivé znaky ze zdrojového souboru. Z těchto znaků vytvoří token.

Jedná se o strukturu, která obsahuje pole znaků, informaci o délce pole a typ samotného tokenu. Konečný automat vrací lexikální chybu v případě, že se analyzátor nedostane do požadovaného stavu.

### 3.2.Syntaktický analyzátor

Je implementovaný pomocí metody rekurzivního sestupu. Při této metodě využíváme LL gramatiku. Syntaktický analyzátor volá funkci na načítání tokenu ze scanneru, podle jednotlivých pravidel můžeme určit, jestli je posloupnost tokenů syntakticky validní. Na analýzu výrazů se používá metoda precedenční syntaktické analýzy.

### 3.3.Precedenční syntaktická analýza

Jedná se o část syntaktické analýzy, které slouží na vyhodnocení výrazu. Když rekurzivní sestup narazí na výraz, vloží ho na zásobník.

## 4. Abstraktní datové typy a struktury

### 4.1. Tabulka symbolů

Je implementovaná pomocí abstraktní datové struktury binárního vyhledávacího stromu. Značná část implementace byla inspirována z předmětu IAL (Algoritmy).

Museli jsme vytvořit strukturu, které by umožnila co nejjednodušší vkládání prvků během načítání tokenů. Vytvořili jsme univerzální kořen, který obsahoval prvky podle potřeby buď na funkci nebo na proměnnou.

### 4.2.Zásobník

Je abstraktní datový typ, implementovaný jako jednosměrný svázaný lineární seznam. Přistupujeme k němu pomocí ukazatele na nejvrchnější prvek.

## 5. Závěr

Ukázalo se, že práce v týmu je velmi náročná, hlavně z pohledu komunikace, ale snažili jsme se, co možná nejvíce. Projekt nám dal mnoho nových vědomostí a poznatků, hlavně z praktického pohledu. Projekt jsme v rámci časové tísně jednotlivých členů bohužel nevypracovali na 100 %, což nám potvrdila dvě pokusná odevzdání.

# 6. Přílohy

## 6.1.LL gramatika

<prog> -> <func_def>
<prog> -> <stat_list> <func_def>
<func_def> -> DEF ID ( <arg_def> ) : EOL INDENT <func_stat_list> DEDENT <prog>
<func_def> -> eps
<func_stat> -> eps
<func_stat> -> <func_stat_list>
<func_stat_list> -> IF <sp_expr> : EOL INDENT <func_stat_list> DEDENT ELSE : EOL INDENT <func_stat_list> DEDENT <func_stat>
<func_stat_list> -> WHILE <sp_expr> : EOL INDENT <func_stat_list> DEDENT <func_stat>
<func_stat_list> -> PASS EOL <func_stat>
<func_stat_list> -> ID <stat_2> EOL <func_stat>
<func_stat_list> -> <sp_expr> EOL <func_stat>
<func_stat_list> -> RETURN <return_value> EOL <func_stat>
<stat> -> eps
<stat> -> <stat_list>
<stat_list> -> IF <sp_expr> : EOL INDENT <stat_list> DEDENT ELSE : EOL INDENT <stat_list> DEDENT <stat>
<stat_list> -> WHILE <sp_expr> : EOL INDENT <stat_list> DEDENT <stat>
<stat_list> -> PASS EOL <stat>
<stat_list> -> ID <stat_2> EOL <stat>
<stat_list> -> <sp_expr> EOL <stat>
<stat_2> -> = <asign>
<stat_2> -> ( <func_arg> )
<asign> -> <sp_expr>
<asign> -> ID <asign_value>
<asign_value> -> ( <func_arg> )
<asign_value> -> eps
<func_arg> -> <asign> <func_arg_next>
<func_arg> -> eps
<func_arg_next> -> , <asign> <func_arg_next>
<func_arg_next> -> eps
<value> -> STRING
<value> -> INTEGER
<value> -> DOUBLE
<value> -> NONE
<value> -> DOCSTRING
<arg_def> -> ID <arg_def_next>
<arg_def> -> eps
<arg_def_next> -> , ID <arg_def_next>
<arg_def_next> -> eps
<sp_expr> -> <value>
<sp_expr> -> <expr>
<return_value> -> <asign>
<return_value> -> eps

## 6.2. Tabulka LL gramatiky

| | DEF | ID | ( | ) | : | EOL | INDENT | DEDENT | IF | ELSE | WHILE | PASS | RETURN | = | , | STRING | INTEGER | DOUBLE | NONE | DOCSTR | <expr> | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <prog> | 1 | 2 | | | | | | | 2 | | 2 | 2 | | | | 2 | 2 | 2 | 2 | 2 | 2 | 1 |
| <func_def> | 3 | | | | | | | | | | | | | | | | | | | | | 4 |
| <stat_list> | | 18 | | | | | | | 15 | | 16 | 17 | | | | 19 | 19 | 19 | 19 | 19 | 19 | |
| <arg_def> | | 34 | | 35 | | | | | | | | | | | | | | | | | | |
| <func_stat_list> | | 10 | | | | | | | 7 | | 8 | 9 | 12 | | | 11 | 11 | 11 | 11 | 11 | 11 | |
| <func_stat> | | 6 | | | | | | 5 | 6 | | 6 | 6 | 6 | | | 6 | 6 | 6 | 6 | 6 | 6 | |
| <sp_expr> | | | | | | | | | | | | | | | | 38 | 38 | 38 | 38 | 38 | 39 | |
| <stat_2> | | | 21 | | | | | | | | | | | 20 | | | | | | | | |
| <return_value> | | 40 | | | | 41 | | | | | | | | | | 40 | 40 | 40 | 40 | 40 | 40 | |
| <stat> | 13 | 14 | | | | | | 13 | 14 | | 14 | 14 | | | | 14 | 14 | 14 | 14 | 14 | 14 | 13 |
| <asign> | | 23 | | | | | | | | | | | | | | 22 | 22 | 22 | 22 | 22 | 22 | |
| <func_arg> | | 26 | | 27 | | | | | | | | | | | | 26 | 26 | 26 | 26 | 16 | 26 | |
| <asign_value> | | | 24 | 25 | | 25 | | | | | | | | | 25 | | | | | | | |
| <func_arg_next> | | | | 29 | | | | | | | | | | | 28 | | | | | | | |
| <value> | | | | | | | | | | | | | | | | 30 | 31 | 32 | 33 | | | |
| <arg_def_next> | | | | 37 | | | | | | | | | | | 36 | | | | | | | |

## 6.3. Precedenční tabulka symbolů

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

## 6.4. Konečný automat