
# Dokumentace projektu z předmětů IFJ a IAL

## Implementace překladače imperativního jazyka IFJ17

Tým 125, varianta I
6. prosince 2017

Členové týmu:
**Pavel Parma** - xparma02 25%
Vojtěch Bargl - xbargl01 25%
Jakub Čábera - xcaber00 25%
Tomáš Vondráček - xvondr23 25%

Rozšíření:
SCOPE, GLOBAL, BASE, FUNEXP, IFTHEN, UNARY, BOOLOP

# 1. Řešení projektu

## 1.1 Lexikální analýza

Lexikální analýza produkuje stream tokenů (jsou analyzovány na požádání). Pro zjednodušení práce byla rezervovaná slova analyzována jako identifikátory a až následně ověřována vůči slovníku rezervovaných slov, který svou implementací poskytuje konečný automat. Pro konverzi čísel nebyla využita žádná nativní funkce, ale byla vytvořena vlastní implementace.

## 1.2 Syntaktická analýza

Syntaktická analýza produkuje AST, jehož uzle jsou kontrolovány v průběhu vytváření. Parser je typu LL(1) jako **Recursive descent parser**. Protože nebyla definována složitá pravidla pro výrazy, byl implementován algoritmus **Precedence climbing method**, který byl vhodnější pro integraci do rekurzivního parseru. Precedence operátorů je tak definován přímo na úrovni přechodů.

## 1.3 Sémantická analýza

Sémantická analýza vyváří AST uzle a zároveň uplatňuje sémantické kontroly. Pro zvýšení rychlosti byl dodržen démetřin zákon. Místo hlubší analýzy níže se tak potřebné informace propagují směrem nahoru při vytváření.

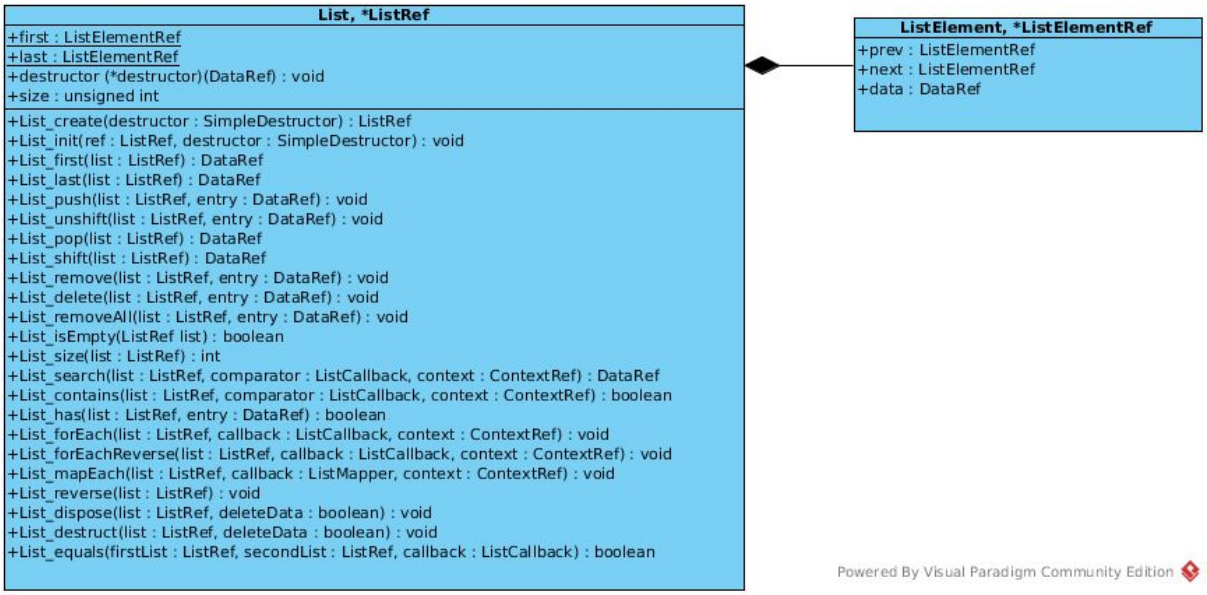
## 1.4 Generování

Generování tří-adresného kódu probíhá z vytvořeného AST. Pro tří-adresný jsou vytvořené 2 API, které zprostředkovávají kontrolu nejpoužívanějších instrukcí a operandů.
Celé generování probíhá do dvou listů, které jdou měnit. Tyto listy představují inicializaci programu a samotný kód programu. Pro zajištění stínění proměnných byla využita simulace proměnných ve scopu, které upravují skutečné jméno použité v interpretru.

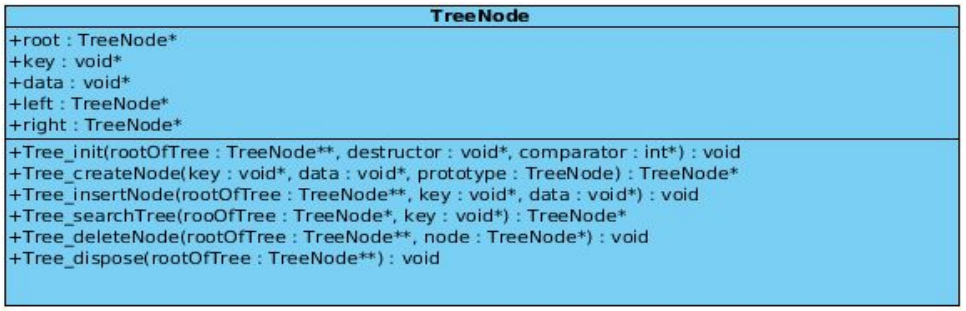
# 2. Techniky a alogritmy

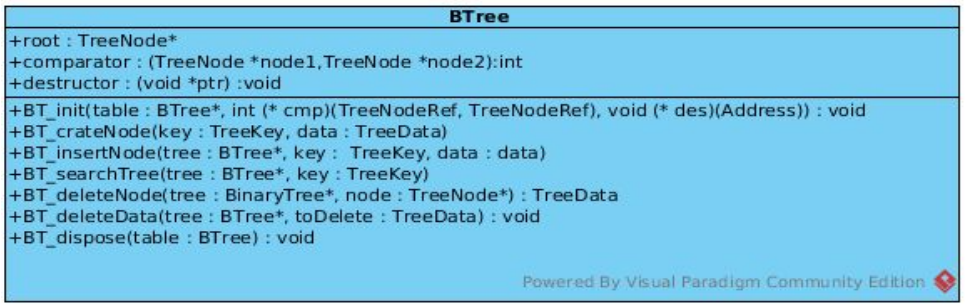
Ve strukturách byla použita genericita (bez kontroly typovosti) s motivací variability využití.

## 2.1 Double linked list



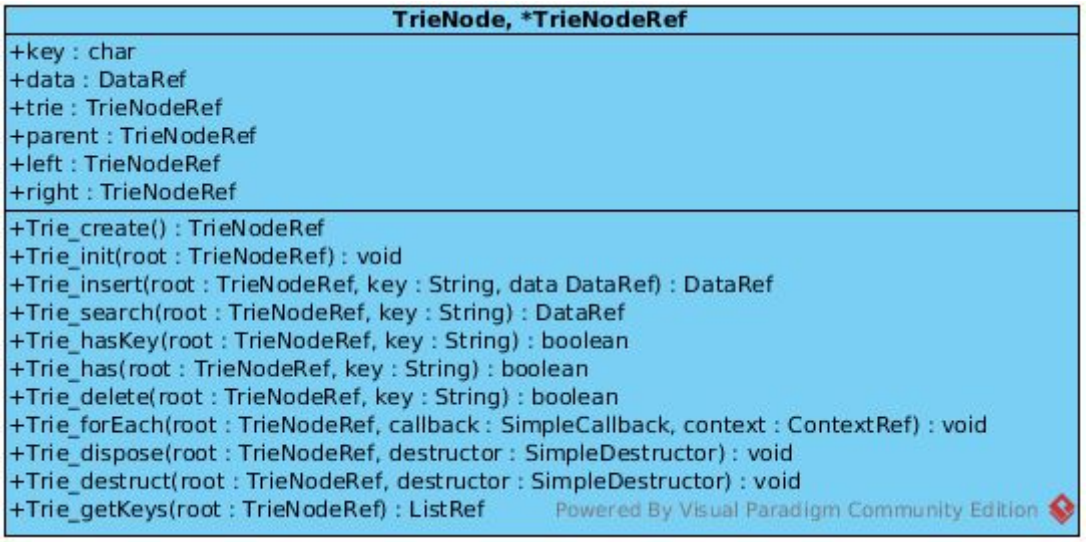

## 2.2 Binary tree



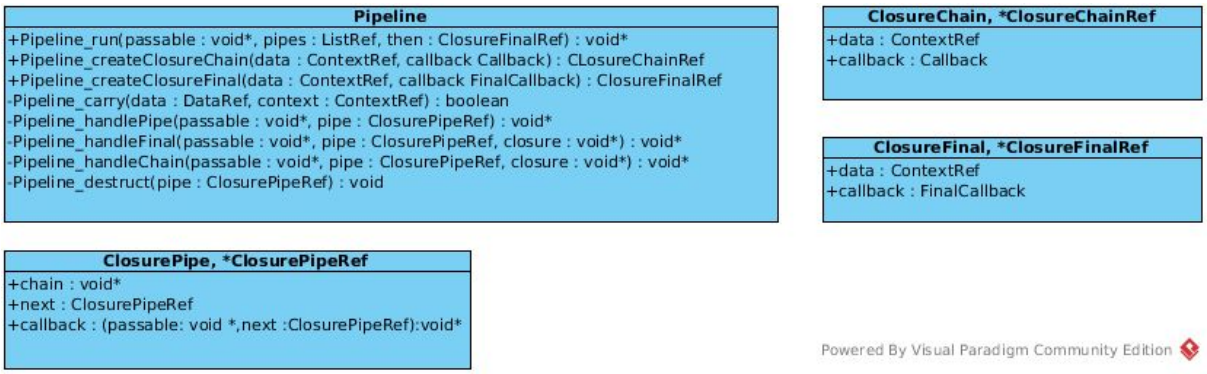
## 2.3 Trie

Trie (digital tree) je typ vyhledávacího stromu. Složitost vyhledávání je stejná jako u binárního stromu s textovými klíči **O(m*n)** (typická ale **O(m*log(n))**), kde **m** je délka klíče a **n** je počet prvků, ale podle testování je rychlejší porovnávat číselné klíče a vytvořit hierarchii stromů. Zde je **n** rovno velikosti ukládané abecedy (záleží na kontextu využívání). Běžná trie je implementována polem, takže má zaručenou složitost **O(m),** kde **m** je délka textu a velikost pole je udána velikostí abecedy. Jelikož jsme ale nechtěli omezovat velikost abecedy a zbytečně plýtvat pamětí, rozhodli jsme se použít návrh hierarchie stromů, která si dokázala na testovaných datech zachovat rychlost a několikanásobně snížit paměťovou náročnost (v porovnání s implementací s polem). Díky rozložení znaků v hierarchii stromů nad testovanými daty je tato implementace 2-3x rychlejší jak binární strom nad stejnými daty a 2x paměťově náročnější (ale 6x méně oproti implementaci polem).




## 2.4 Pipeline

Implementace middleware mechanizmu. Každá činnost může nechat pracovat tu další a následně pracovat s výsledkem, přičemž poslední finální činnost může pouze něco provádět. Jelikož **c** jazyk neposkytuje **closures**, bylo potřeba poradit si skrze struktury, které je reprezentují. Při procházení má každá činnost k dispozici **pipe**, která obsahuje odkaz na další činnost. Tím je zajištěn průchod (v podstatě přeskakování ze struktury na strukturu) se zachováním jednoznačné signatury a umožněna rekurzivní definice signatury (činnost, která přebírá činnosti, která přebírá činnost, která přebírá činnost, …).




# 3. Práce v týmu

## 3.1 Rozdělení práce

**Pavel Parma** - xparma02
- Project leader
- Plánování a organizace
- Kontrola kvality a progresu
- implementace

**Vojtěch Bargl** - xbargl01
- Návrh a architektura
- implementace
- Kontrola kvality

**Jakub Čábera** - xcaber00
- Testování a testovací technologie
- Správa prostředí a technologie pro vývoj
- implementace

**Tomáš Vondráček** - xvondr23
- implementace
- Dokumentace

## 3.2 Workflow

Pro vývoj byla zvolena agilní metodika **Kanban**, skrze issue tracking system **Youtrack**.
Pro verzování kódu se používal **git** se vzdáleným repozitářem na **gitlab.com**
Pro kontrolu kvality se používaly nástroje **Upsrouce** (code review) a **gitlab CI** (testing)
Pro komunikace se používal nástroj **slack**.
Pro implementaci se používalo IDE **CLion** a **docker** pro sjednocení platformy.

# 4. Pravidla LL gramatiky

<program> => <program_definiton> SCOPE <statement_list> END SCOPE <empty_lines>
<program_definition> => ε
<program_definition> => <function_declaration> <function_definition> <variable_definition>
<empty_lines>  <program_definiton>
<emty_lines> => EOL <emty_lines>
<emty_lines> => ε
<statement_list> => <statement> <statement_list>
<statement_list> => ε
<function_declaration> => ε
<function_declaration> => DECLARE FUNCTION ID ( <parameter_list> ) AS TYPE EOL
<function_definition> => ε
<function_definition> => FUNCTION ID ( <parameter_list> ) AS TYPE EOL <statement_list> END FUNCTION EOL
<variable_definition> => ε
<shared> => ε
<shared> => SHARED
<variable_definition> => DIM <shared> ID AS TYPE <initialization> EOL
<initialization> => ε
<initialization> => = <expression>
<parameter_list> => ε
<parameter_list_optional> => ε
<parameter_list_optional> => , <parameter>
<parameter_list> => <parameter> <parameter_list_optional>
<paremter> => ID AS TYPE
<statement> => EOL
<statement> => INPUT ID EOL
<statement> => PRINT <print_argument> <print_argument_list>  EOL
<statement> => IF <expression> THEN EOL <statement_list> <else_if> <else> END IF EOL
<statement> => <variable>
<statement> => RETURN <expression>
<statement> => SCOPE <statement_list> END SCOPE EOL
<statement> => ID = <expression> EOL
<statement> => STATIC ID AS TYPE <initialization EOL
<statemen> => DO WHILE <expression> EOL <statement_list> LOOP EOL
<else_if> => ELSEIF <expression> THEN EOL <statement_list> <else_if>
<else_if> => ε
<else> => ε
<else> => ELSE EOL <statement_list>
<print_argument_list> => ε
<print_argument_list> => <print_argument> <print_argument_list>
<print_argument> => <expression> ;
<expression> => <not_expression> <expression_list>
<expression_list> => ε
<expression_list> => AND <expression>
<expression_list> => OR <expression>
<not_expression> => NOT <relational_expression>
<not_expression> => <relational_expression>
<relational_expression> => <additive_expression> <relational_expression_list>

<relationa_expression_list> => ε
<relational_expression_list> => > <relational_expression>
<relational_expression_list> => < <relational_expression>
<relational_expression_list> => >= <relational_expression>
<relational_expression_list> => <= <relational_expression>
<relational_expression_list> => = <relational_expression>
<relational_expression_list> => <> <relational_expression>
<additive_expression> => <decimal_division_expression> <additive_expression_list>
<additive_expression_list> => ε
<additive_expressiont_list> => + <additive_expression>
<additive_expressiont_list> => - <additive_expression>
<decimal_division_expression> => <multiplication_expression> <decimal_division_expression_list>
<decimal_division_expression_list> => ε
<decimal_division_expression_list> => \ <decimal_division_expression>
<multiplication_expression> => <unary_atomic_expression> <multiplication_expression_list>
<multiplication_expression_list> => ε
<multiplication_expression_list> => * <multiplication_expression>
<multiplication_expression_list> => / <multiplication_expression>
<unary_not_expression> => <atomic_expression>
<unary_not_expression> => - <atomic_expression>
<atomic_expression> => INT
<atomic_expression> => STRING
<atomic_expression> => ID
<atomic_expression> => float
<atomic_expression> => ( <expression> )
<atomict_expression> => id <function_call_expression>
<function_call_expression> = ε
<function_call_expression> = ( <argument_list> )
<argument_list> => ε
<argument_list> => <expression> <argument_list_optional>
<argument_list_optional> => ε
<argument_list_optional> => , <expression>

# 5. Konečný automat