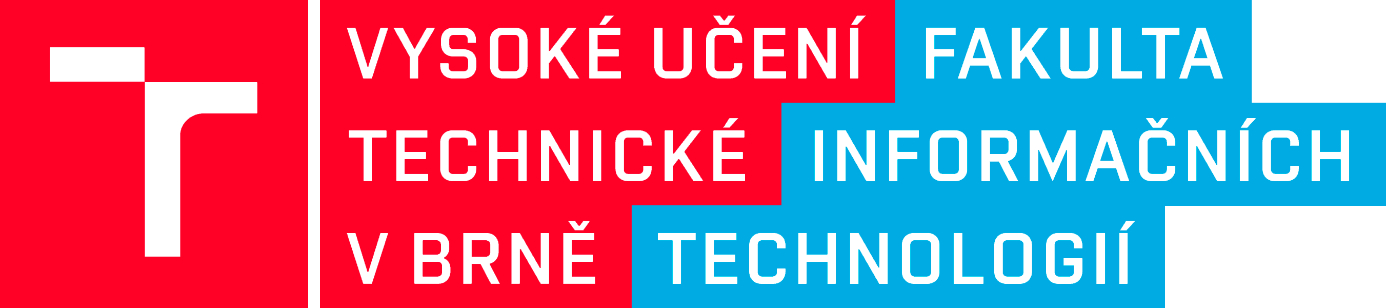
# **VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

# **FAKULTA INFORMAČNÍCH TECHNOLÓGIÍ**



Dokumentácia k projektu do predmetu IFJ a IAL

**Interpret jazyka IFJ16**

Tím 066, varianta a/1/II

Dátum TBD.12.2016

Vedúci tímu: Jozef Urbanovský (xurban66)

Členovia: Adrián Tomašov (xtomas32)

Roman Dobiáš (xdobia11)

Adam Šulc (xsulca00)

Kristián Barna (xbarna02)

**Obsah**

[1. Úvod 2](#_Toc468618733)

[2. Cyklus vývoja projektu 2](#_Toc468618734)

[2.1. Návrh a použitá metodika 2](#_Toc468618735)

[2.2. Spôsob práce v tíme 2](#_Toc468618736)

[2.2.1. Zdieľanie a verziovanie 2](#_Toc468618737)

[2.2.2. Schôdzky tímu a komunikácia 3](#_Toc468618738)

[2.2.3. Rozdelenie práce 3](#_Toc468618739)

[3. Implementácia častí 4](#_Toc468618740)

[3.1. Lexikálna analýza 4](#_Toc468618741)

[3.2. Syntaktická analýza 4](#_Toc468618742)

[3.3. Sémantická analýza 4](#_Toc468618743)

[3.4. Interpret 4](#_Toc468618744)

[4. Implementácia algoritmov 5](#_Toc468618745)

[4.1. Knuth-Morris-Pratt algoritmus 5](#_Toc468618746)

[4.2. Quick-sort algoritmus 5](#_Toc468618747)

[4.3. Tabuľka s rozptýlenými položkami 5](#_Toc468618748)

[5. Testovanie 6](#_Toc468618749)

[6. Metriky kódu 6](#_Toc468618750)

[7. Prílohy 7](#_Toc468618751)

[7.1. Konečný automat lexikálnej analýzy 7](#_Toc468618752)

[7.2. LL gramatika 8](#_Toc468618753)

[7.3. Precedenčná tabuľka 9](#_Toc468618754)

# **Úvod**

Táto dokumentácia popisuje návrh a implementáciu interpretu k jazyku IFJ16, ktorý je podmnožinou jazyka Java SE 8. V dokumentácii sa nachádza spôsob riešenia lexikálnej analýzy, syntaktickej analýzy, sémantickej analýzy a intepretu samotného. Zvolili sme si variantu a/1/II, ktorá zahŕňa implementáciu vstavanej funkcie find pomocou Knuth-Morris-Prattovho algoritmu, implementácia vstavanej funkcie sort pomocou Quick sortu a implementáciu tabuľky symbolov cez abstraktnú dátovú štruktúru tabuľky s rozptýlenými položkami.

TODO ešte niečo

# **Cyklus vývoja projektu**

## **Návrh a použitá metodika**

Pri vývoji projektu sme si bližšie neurčili akým spôsobom budeme vyvíjať interpret IFJ16, ale dalo by sa to prirovnať k agilnej metodológii. Náš tím, zostavený už v na konci 2. semestra mal za cieľ nazbierať nutné informácie o tomto projekte už pred jeho zadaním. Po zadaní projektu sme si rozdelili prácu na jednotlivé celky, ktoré mali dopredu dohodnuté rozhranie a mohli byť tým pádom vyvíjané nezávisle na sebe. Po tom ako boli vyvinuté prototypy, resp. funkčné časti sme ich zintegrovali a poriadne otestovali. V návrhu sme vývoj projektu rozdelili na tieto časti, ktorých implementácia bude detailne popísaná v ďalších sekciách.

* Lexikálna analýza
* Syntaktická analýza
* Precedenčná analýza a sémantické kontroly
* Vstavané funkcie
* Tabuľka symbolov a dátové štruktúry
* Interpret

## **Spôsob práce v tíme**

### **Zdieľanie a verziovanie**

Nakoľko je na tomto projekte nutná spolupráca 4-5 študentov rozhodli sme sa pristúpiť k verziovaciemu systému *Git*, ktorý nám umožnil pracovať súčasne na projekte, vracať sa k predchádzajúcim verziám, vytvárať zálohy a zjednodušil následnej aj integráciu samotnú. Používali sme webovú službu *Github*, na ktorej ako študenti môžeme mať priváte repozitáre a patrí k najspoľahlivejším. V repozitári sme neuchovávali len zdrojové kódy, ale aj TODO, ktoré bolo upravované každý týždeň a prípadne na každej schôdzke. Následne sa v repozitári nachádza aj dokumentácia a všetky potrebné prílohy k projektu.

### **Schôdzky tímu a komunikácia**

Na začiatku semestru po zadaní projektu sme sa stretávali približne každé 2 týždne aby sme prediskutovali momentálnu situáciu a ako pokračovať ďalej. Na prvej schôdzke sme zhodnotili schopnosti nášho tímu a podľa toho rozdelili úlohy, kto bude pracovať na akej časti. Dohodli sme si isté konvencie ako budeme postupovať, aké rozhranie budeme používať a podobné. Schôdzky boli približne každé 2 týždne s tým, že sme neustále komunikovali v spoločnom chate na Facebooku a uskutočňovali videokonferencie, na ktorým sem sa radili ako postupovať ďalej, ak osobná schôdzka nebola možná. Členovia tímu, ktorí vyvíjali na sebe závislé časti sa stretávali a kontaktovali častejšie ako zbytok tímu.

### **Rozdelenie práce**

**Jozef Urbanovský**: vedúci tímu, testovanie, dokumentácia, interpret, vstavané funkcie

**Roman Dobiáš**: lexikálny analyzátor, syntaktický analyzátor, sémantický analyzátor, testovanie

**Adrián Tomašov**: interpret, testovanie, vstavané funkcie, git, integrácia

**Adam Šulc**: syntaktický analyzátor, sémantický analyzátor, precedenčná tabuľka

**Kristián Barna**: tabuľka symbolov, radiaci algoritmus Quick-sort, Knuth-Morris-Pratt

# **Implementácia častí**

## **Lexikálna analýza**

Lexikálny analyzátor, scanner, je prvá časť projektu, ktorá pracuje ako jediná so zdrojovým textom napísaným v jazyku *IFJ16*. Jeho úlohou je transformovať zdrojový text na tokeny. Celý scanner je implementovaný ako veľký končený automat (viď. príloha 5.1.), ktorý spracováva vstup znak po znaku a na základe stavu v ktorom skončil, určí typ tokenu, či zahlási chybu. Pokiaľ scanner nájde lexikálnu chybu vo vstupnom súbore podľa návratovej hodnoty funkcie, ktorá spracováva lexém vieme zistiť o akú chybu sa jednalo. Náš scanner implementuje funkciu *getToken(),* ktorá po volaní syntaktickým analyzátorom vracia práve jeden načítaný lexém v podobe globálnej štruktúry t\_token. Token obsahuje typ a hodnotu lexému, riadok a pozíciu na riadku, kde sa lexém nachádza. Tokeny sú interne uložené v jednosmerne viazanom zozname, pre jednoduchšiu manipuláciu a možnosti implementovať funkciu *ungetToken().*

## **Syntaktická analýza**

Syntaktický analyzátor je jadrom celého projektu a riadi všetky ostatné časti. Pri konštrukcii syntaktickej analýzy pre kontext jazyka založeného na LL gramatike (viď. príloha 5.2.), sme využili metódu rekurzívneho zostupu. Pri top-down parseri sa hľadá najľavejšia derivácia vstupného programu, ktorý simuluje vytváranie derivačného stromu, ktorý fyzicky nie je zostrojený. Vstupné tokeny od scannera sú spracovávané pomocou pravidiel LL gramatiky. Spracovanie výrazov syntaktickej analýzy je implementované precedenčnou syntaktickou analýzou, ktorá je realizovaná podľa precedenčnej tabuľky (viď. príloha 5.3.). Pri spracovávaní výrazov využívame prevod do postfixovej notácie (poľskej notácie), kvôli následnému zjednodušeniu ich vyhodnocovania.

TODO

## **Sémantická analýza**

Sémantická analýza je implementovaná ako priama súčasť syntaktického parsera. Popri kontrolovaní syntaktickej korektnosti, parser spolupracuje s tabuľkou symbolov a overuje, či jednotlivé konštrukcie sú aj typovo kompatibilné, prípadne konvertovateľné. Samotné implicitné konverzie hodnôt premenných sú realizované priamo v operáciach interprétu.

## **Interpret**

TODO

# **Implementácia algoritmov**

## **Knuth-Morris-Pratt algoritmus**

Algoritmus na vyhľadávanie podreťazca umožňujúci pri nezhode znakov pokračovať v hľadaní od počiatočnej zhody

TODO

## **Quick-sort algoritmus**

Quick-sort je veľmi rýchly a rekurzívne jednoducho implementovateľný radiaci algoritmus založený na princípe „divide and conquer“ s najhoršou možnou časovou zložitosťou a s očakávanou zložitosťou . Prvým krokom pri implementácii tohto algoritmu je zvolenie si ľubovoľného prvu v poli, pivotu. V našom prípade sa ako pivot vyberie pseudomedián, tj.

Algoritmus Quick-sort následne prechádza a preusporiadáva pole tak, aby na jednej strane boli prvky väčšie než pivot, na druhej strane prvky menšie než pivot a pivot samotný bol umiestnený približne v strede medzi týmito časťami.

## **Tabuľka s rozptýlenými položkami**

Na tabuľku pre uchovávanie symbolov sme podľa zadania využili tabuľku s rozptýlenými položkami (TRP). Výhodou TRP je rýchlosť hľadania symbolov v nej uložených. Jej priemerná doba vyhľadávania je a v najhoršom prípade až . Tabuľka pozostáva z poľa ukazateľov na jednosmerne viazaný zoznam synoným, kde každá položka zoznamu obsahuje symbol a ukazateľ na nasledujúcu položku zoznamu. V TRP hľadáme symbol pomocou takzvanej hashovacej funkcie, ktorá na základe hľadaného reťazca vypočíta index do poľa ukazateľov na zoznam, v ktorom by sa hľadaný symbol mal nachádzať. Následne sa prechádza zoznamom, pokiaľ nie je prvok nájdený, prípadne kým sa nedostaneme na koniec zoznamu.

# **Testovanie**

Nakoľko sme si projekt rozdelili na niekoľko nezávislých častí bolo nutné otestovať ich funkčnosť jednotkovými testami. Samotné jednotkové testy boli najväčšou výzvou, nakoľko sme museli simulovať vstupné, prípadne výstupné dáta na overenie správnosti danej časti. Testovanie nášho interpretu po integrácií prebiehalo na základe automatizovaného bashového skriptu, ktorý mal za úlohu overiť správnosť projektu. Testy boli rozdelené do 3 kategórií, správne, nesprávne a runtime testy. Správne testy obsahovali zdrojový kód, vstup a výstup, ktorý bol následne skontrolovaný, či sa správne interpretoval, vyhodnotil a vrátil správny návratový kód. Nesprávne a runtime testy obsahovali v hlavičke zdrojového kódu typ chyby a jej návratový kód, ktorý očakávame. Následne skript vyextrahoval dáta z hlavičky a po vykonaní daného testu skontroloval jeho výstup a či vrátil očakávaný návratový kód. V prípade chybne vyhodnoteného testu bol oboznámený člen, ktorý mal daný celok na starosť, aby danú chybu opravil. Taktiež sme využili šancu pokusných odovzdaní, ktoré nám pomohli verifikovať náš interpret. Na pokusných odovzdaniach sme obstáli s 93% a xx% respektívne.

* Počet správnych testov: 49
* Počet nesprávnych testov: 85
* Počet runtime testov: 26

# **Metriky kódu**

* Počet súborov: 22
* Počet riadkov: 6430

# **Prílohy**

## **Konečný automat lexikálnej analýzy**

## **LL gramatika**

<source-program> → <class-definition-list>

<class-definition-list> → <class-definition> <class-definition-list>

<class-definition-list> → ε

<class-definition> → class simple-identifier { <class-body> }

<class-body> → ε

<class-body> → <definition> <class-body>

<definition> → static <type-specifier> simple-identifier <more-definition>

<more-definition> → ( <function-parameters-list> ) { <function-body> }

<more-definition> → ;

<more-definition> → = <expr> ;

<function-body> → <statement> <function-body>

<function-body> → <local-definition> <function-body>

<function-body> → ε

<local-definition> → <type-specifier> simple-identifier <variable-initialization> ;

<variable-initialization> → = <more-next>

<variable-initialization> → ε

<parameter-definition> → <type-specifier> simple-identifier

<function-parameters-list> → <parameter-definition> <more-function-parameters>

<function-parameters-list> → ε

<more-function-parameters> → , <parameter-definition> <more-function-parameters>

<more-function-parameters> → ε

<argument-definition> → <term>

<function-arguments-list> → <argument-definition> <more-function-arguments>

<function-arguments-list> → ε

<more-function-arguments> → , <argument-definition> <more-function-arguments>

<more-function-arguments> → ε

<statement> → <compound-statement>

<statement> → <assign-statement>

<statement> → <selection-statement>

<statement> → <iteration-statement>

<statement> → <jump-statement>

<block-items-list> → <statement> <block-items-list>

<block-items-list> → ε

<compound-statement> → { <block-items-list> }

<assign-statement> → <identifier> <next> ;

<selection-statement> → if ( <expr> ) <compound-statement> else <compound-statement>

<iteration-statement> → while ( <expr> ) <compound-statement>

<jump-statement> → return <expr> ;

<next> → ( <function-arguments-list> )

<next> → = <more-next>

<more-next> → <identifier> ( <function-arguments-list> )

<more-next> → <expr>

<identifier> → simple-identifier

<identifier> → fully-qualified-identifier

<term> → simple-identifier

<term> → fully-qualified-identifier

<term> → decimal-constant

<term> → floating-point-constant

<type-specifier> → void

<type-specifier> → double

<type-specifier> → int

<type-specifier> → String

<expr> → ε

<expr> → expression

## **Precedenčná tabuľka**

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