

### Dokumentácia k projektu z predmetu IFJ a IAL

### Implementácia interpretu imperatívneho jazyka IFJ16

### Tým 016, varianta b/4/II

### 5. decembra 2016

Riešitelia:

Sámel Šimon, xsamel02 , 0%

Patrik Sztefek, xsztef02 , 25%  
Tomáš Szűcs, xszucs01 , 25%  
Marek Šipoš, xsipos03 , 25%  
Jakub Štol, xstolj00 , 25%

1. **Úvod**

V tejto dokumentácii je popísaný vývoj a implementácia interpreta imperatívneho jazyka IFJ16, ktorý je zjednodušenou podmnožinou jazyka Java SE 8. Celá dokumentácia je rozdelená do kapitol, v ktorých sú popísané jednotlivé časti interpreta. Od návrhu, implementácie až po samotnú prácu v tíme.

1. **Návrh interpretu**

Z dôvodu veľkosti a náročnosti projektu sme boli nútení rozdeliť na jednotlivé časti(spomenuté nižšie), aby sme si zachovali prehľadnosť . A hlavne zabráneniu zbytočnému chaosu. Preto sme projekt rozdelili do piatich hlavných častí:

* Lexikálna Analýza
* Syntaktická analýza

1. prediktívna
2. precedentná

* Sémantická analýza
* Generátor trojadresného kódu
* Interpret

1. **Implementácia**

V tejto kapitole sa popisuje implementácia jednotlivých častí interpretu.

* 1. **Lexikálny Analyzátor**

Lexikálny analyzátor je konečný automat s epsilon prechodmi(slúžia k čo najpresnejšiemu určeniu tokenu pomocou medzi stavov), ktorý načíta vstup zo zdrojového súboru. Následne rozlišuje od jednoduchých znakov až po kľúčové slová, ktoré prevádza na tokeny. Token je tvorený štruktúrou, v ktorej sa nachádza číslo riadku a typ.(obrázok konečného automatu viz. Príloha A)

* 1. **Syntaktický analyzátor**

Úlohou syntaktického analyzátoru je skontrolovať syntax a poskytnúť priebeh programom pre ostatné analýzy. Využíva sa kombinácia syntaktické analýzy zhora dole a zdola nahor. Pre výrazy sa využíva teda precedentná analýza (zdola nahor), pre zbytok analýza prediktívna(zhora dole).Využívame prediktívnu analýzu namiesto rekurzívnej, aj napriek tomu , že je doporučená rekurzívna. Je to z dôvodu, že umožňuje voľne meniť návrh gramatiky a LL tabuľky. Zároveň je rýchlejšou ale je o dosť obťažnejšou na implementáciu.

Použitá gramatika je LL(1) gramatika. Pretože však jazyk IFJ16 nejde popísať čistou LL(1) gramatikou, bolo pre popísanie jazyka použito drobných heuristík, ktoré mierne “ohýbajú” princíp syntaktickej analýzy.(Príloha B)

**4.Vývoj interpretu**

V tejto kapitole je popísane rozdelenie práce na interpretu, až po použité nástroje pri vývoji.

**4.1 Rozdelenie práce**

Prácu pred začatím vývoja interpretu rozdeľoval náš vedúci Patrik Sztefek. Rozdeľoval ju spôsobom, kto bol ako zručný v programovaní. Ale zároveň nám dal nám možnosť si vybrať, ktorú časť chceme riešiť. Aj keď sa môže zdať, že rozdelenie bodov neprináleží k odvedenej práce jednotlivých členov, každý sa snažil ako len mohol. Nižšie uvedené popisy slúžia pre predstavu, kto akú časť problematiky riešil.

**Patrik Sztefek** - Generátor, algoritmy, interpret, testovanie

**Tomáš Szűcs** – Lexikálny analyzátor, výpomoc, dokumentácia,

**Marek Šipoš** – Syntaktický analyzátor, Sémantický analyzátor, testovanie, Generátor  
**Jakub Štol** - Lexikálny analyzátor, vstavané funkcie, výpomoc

**4.2 Komunikácia a schôdze**

Už pred začatím vývoja sme vedeli, že nebude čas na pravidelne schôdze z dôvodu odlišnosti rozvrhu každého člena, tak sme sa stretávali len vo vážných prípadoch. Namiesto schôdze sme zvolili variantu Facebook-ovej skupiny a Facebook instant-messaging. Výhoda spočíva v tom, že vieme komunikovať z ktoréhokoľvek miesta. Zároveň podávať správu o pokroku na svojej práci.

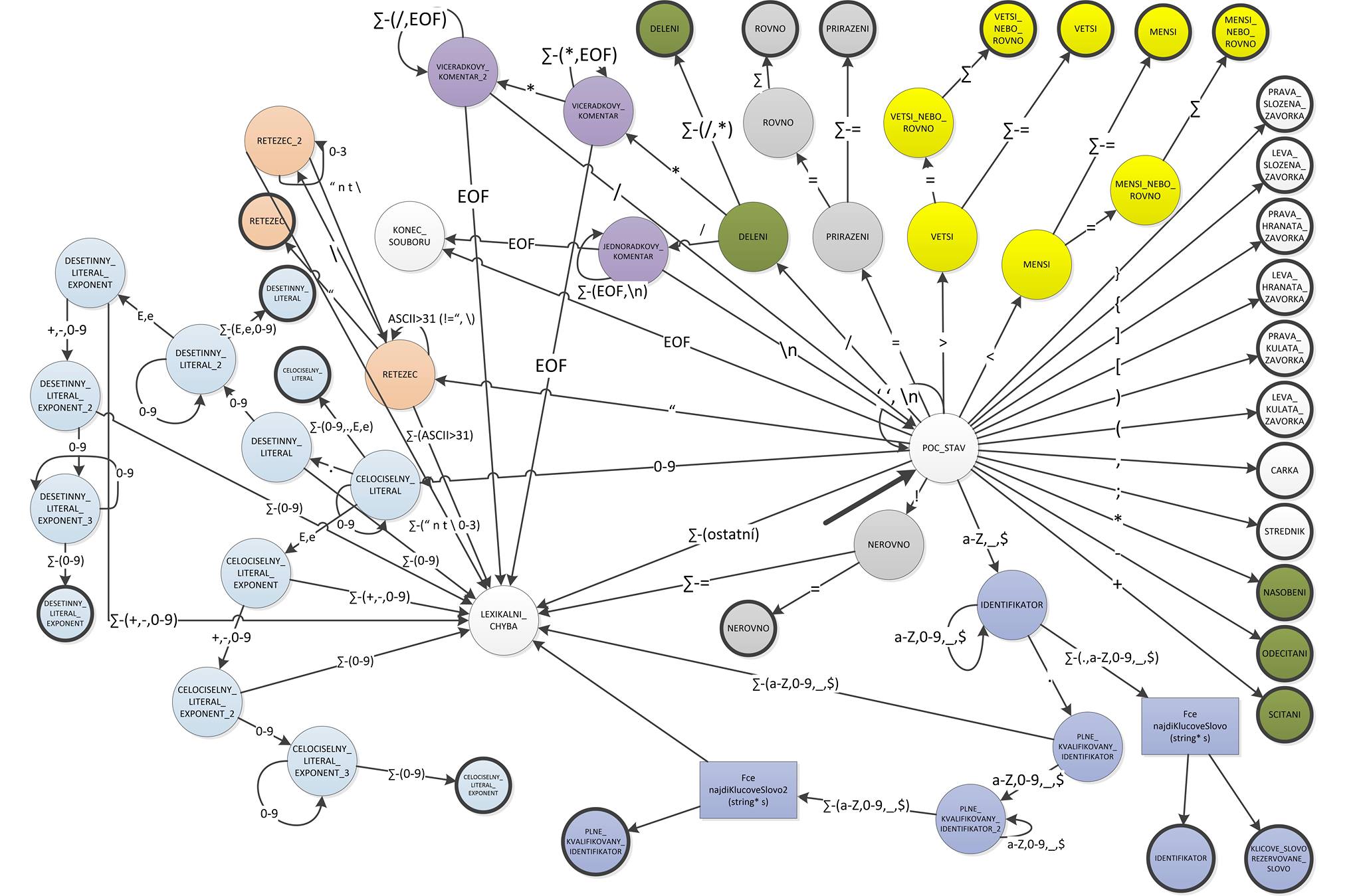
**4.3 Použité nástroje**

Na vývoj interpretu sa použil verzovaci systém **Git**. Ako úložný priestor pre náš projekt poslúžil **GitHub**. Z dôvodu dostupnosti na internete a zároveň má zabudované funkcie. Ktoré sprehľadňujú zmeny, a to v konkrétnom súbore na konkrétnom riadku. Aj z dôvodu poskytnutia voľnej licencie pre študentov, aj kvôli chuti vyskúšať moderné trendy. Ako ďalší nastroj sme použili **Google docs**. Najviac užitočný bol vo fázach vývoja, keď sme mali obrovské množstvo hlavičkových a zdrojových súborov. Zároveň slúžil ako poznámkový blok, či už pre brainstorming alebo popis jednotlivej časti interpretu.

**4.4 Metodika vývoja interpretu**

Ranné fázy vývoja by sa dali prirovnať k V-modelu, či už z pohľadu testovania alebo samotnej implementácie. Z dôvodu, že každá časť interpretu sa navrhla, implementovala následne testovala.

**Príloha A: Konečný automat Lexikálneho analyzátoru**

****

**Priloha B: LL gramatika a precedencna tabulka**

01 PROGRAM → TRIDA PROGRAM

02 PROGRAM → eof

03 TRIDA → class identifikator { SEZNAM-DEFINIC-STATIC }

04 SEZNAM-DEFINIC-STATIC → static DATOVY-TYP DEFINICE-STATIC SEZNAM-DEFINIC-STATIC

05 SEZNAM-DEFINIC-STATIC → ԑ

06 DEFINICE-STATIC → DEFINICE-FUNKCE

07 DEFINICE-STATIC → DEFINICE-PROMENNA ;

08 DEFINICE-PROMENNA → identifikator DEF-PROM-KONEC

09 DEF-PROM-KONEC → PRIRAZENI

10 DEF-PROM-KONEC → ԑ

11 DEFINICE-FUNKCE → f\_identifikator ( SEZNAM-PARAMETRU ) SLOZENY-PRIKAZ

12 SEZNAM-PARAMETRU → PARAMETR-PRVNI PARAMETR-DALSI

13 SEZNAM-PARAMETRU → ԑ

14 PARAMETR-PRVNI → primitivni\_typ identifikator

15 PARAMETR-DALSI → , primitivni\_typ identifikator PARAMETR-DALSI

16 PARAMETR-DALSI → ԑ

17 SEZNAM-VSTUPU → vyraz VSTUP-DALSI

18 SEZNAM-VSTUPU → identifikator VSTUP-DALSI

19 SEZNAM-VSTUPU → ԑ

20 VSTUP-DALSI → , VSTUP-KONEC

21 VSTUP-DALSI → ԑ

22 VSTUP-KONEC → vyraz VSTUP-DALSI

23 VSTUP-KONEC → identifikator VSTUP-DALSI

24 SLOZENY-PRIKAZ → { BLOK-PRIKAZU }

25 BLOK-PRIKAZU → PRIKAZ BLOK-PRIKAZU

26 BLOK-PRIKAZU → ԑ

27 PRIKAZ → primitivni\_typ DEFINICE-PROMENNA ;

28 PRIKAZ → vyraz ;

29 PRIKAZ → identifikator POUZITI ;

30 PRIKAZ → f\_identifikator VOLANI-FUNKCE ;

31 PRIKAZ → return NAVRAT-KONEC ;

32 PRIKAZ → if ( vyraz ) SLOZENY-PRIKAZ else SLOZENY-PRIKAZ

33 PRIKAZ → while ( vyraz ) SLOZENY-PRIKAZ

34 POUZITI → PRIRAZENI

35 POUZITI → ԑ

36 VOLANI-FUNKCE → ( SEZNAM-VSTUPU )

37 NAVRAT-KONEC → vyraz

38 NAVRAT-KONEC → identifikator

39 NAVRAT-KONEC → ԑ

40 PRIRAZENI → = PRAVA-STRANA

41 PRAVA-STRANA → vyraz

42 PRAVA-STRANA → identifikator

43 PRAVA-STRANA → f\_identifikator VOLANI-FUNKCE

44 DATOVY-TYP → void

45 DATOVY-TYP → primitivni\_typ

vyraz → Předá se precedenční synt. analýze - symboly: identifikator(proměnné) číslo řetězec ( ) aritmetické + relační operátory

## **Precedenčná tabuľka**

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

