Vysoké učení technické v Brne

Fakulta informačních techologií

**Dokumentácia**

Implementácia prekladača imperatívneho jazyka IFJ22

**Richard Harman 25%**

Marek Špirka 25%

Jasmína Csalová 25% **30.11.2022** Terézia Hundáková 25%

Obsah

[Úvod 2](#_Toc121053222)

[Práca v tíme 3](#_Toc121053223)

[Rozdelenie práce 3](#_Toc121053224)

[Postup práce 3](#_Toc121053225)

[Implementácia interpreta jazyka IFJ22 4](#_Toc121053226)

[Lexikálna analýza 4](#_Toc121053227)

[Syntaktická analýza 4](#_Toc121053228)

[Sémantická analýza 4](#_Toc121053229)

[Interpret 5](#_Toc121053230)

[Prílohy 5](#_Toc121053231)

[Diagram konečného automatu lexikálnej analýzy 5](#_Toc121053232)

[LL-gramatika 6](#_Toc121053233)

[Gramatika výrazov 7](#_Toc121053234)

[Precedenčná tabuľka 8](#_Toc121053235)

# Úvod

Dokumentácia popisuje našu implementáciu interpreta jazyka IFJ22, ktorý je podmnožinou jazyka PHP. Implementovali sme to v niekoľkých častiach a to lexikálna analýza, syntaktická analýza, sémantická analýza a interpret. Jednotlivé časti popisujeme v prislúchajúcich kapitolách.

# Práca v tíme

## Rozdelenie práce

Richard Harman – Lexikálna analýza, Sémantická analýza, Generátor kódu

Marek Špirka – Generátor kódu

Jasmína Csalová – Syntaktická analýza, Dokumentácia

Terézia Hundáková – Syntaktická analýza, Sémantická analýza

## Postup práce

Nakoľko projekty takéhoto rozsahu sú časovo aj technicky náročnejšie, dohodli sme sa na priebežnom a systémovom vykonávaní našich povinností. Ako komunikačné kanály sme využívali Messenger a v niektorých prípadoch Discord. Dohodli sme sa na verzovacom systéme Git, ktorý sme používali prostredníctvom GitKraken. Mali sme niekoľko osobných tímových konzultácií, a aj vždy sme komunikovali prostredníctvom komunikačných kanálov, kde sme riešili vzniknuté naliehavé záležitosti.

# Implementácia interpreta jazyka IFJ22

## Lexikálna analýza

Lexikálna analýza je jedna z prvých častí interpreta, ktorá je implementovaná na základe lexikálnych pravidiel jazyka a na základe nami navrhnutého konečného stavového automatu. Ten prečíta znak zo súboru, vyhodnotí ho a určí nasledujúci krok.

Jej hlavné úlohy sú čítanie znakov zo vstupu a ich preklad na tokeny, odstránenie komentárov a bielych znakov ktoré sa v kóde nachádzajú a nájdenie lexikálnych chýb.

Lexikálne pravidlá pomáhajú rozdeliť jednotlivé postupnosti znakov na lexémy – menšie lexikálne časti. Rozpoznané lexémy vstupujú do nášho programu cez štruktúru – token. Táto štruktúra obsahuje informácie o dátovom type tokenu a o jeho dátach.

Jednotlivé tokeny odovzdávame do syntaktickej analýzy v prípade lexikálnej bezchybnosti. V prípade, ak by bola lexikálna bezchybnosť porušená, znamená to, že s daným tokenom pokračovať nemôžeme.

Implementované v

## Syntaktická analýza

Naša syntaktická analýza je implementovaná rekurzívne podľa pravidiel LL gramatiky a aplikuje sa na každý prijatý token až po úspešnej lexikálnej analýze. V prípade, ak narazíme na výraz, spúšťame precedenčnú analýzu formou modulu *expr*. Vykoná kontrolu výrazu a daný výraz vyhodnotí podľa jeho dátových typov a uloží si potrebný výsledný dátový typ z výrazu. Syntaktiku spúšťame v *main* pomocou prologu.

Implementované v

## Sémantická analýza

Vykonáva sa po úspešnej syntaktickej analýze. Spúšťa sa v *main* pomocou antilogu. Vzostupným spôsobom sa vykonávajú potrebné sémantické kontroly a naplňujeme zásobník. Kontrolujú sa volania nedefinovaných funkcií, použitia neexistujúcich premenných a tiež volanie funkcie so zlým počtom parametrov.

Implementované v

## Interpret

# Prílohy

## Diagram konečného automatu lexikálnej analýzy



## LL-gramatika

1. <program> -> <?php EOL declare(strict\_types=1); < body >
2. < body > -> < stmt > < body >
3. < body > -> <end>
4. <end> -> ?> EOF
5. <end> -> EOF
6. < func > -> function func\_ID (<args>) <ret\_type> { <stmt\_list> }
7. < stmt \_list > -> < stmt > <stmt\_list >
8. <stmt\_list > -> ε
9. < stmt > -> < func >
10. < stmt > -> [expresion];
11. < stmt > -> $var\_ID = [expresion];
12. < stmt > -> while ([expresion]) { <stmt\_list> }
13. < stmt > -> if ([expresion]) { stmt\_list } <else\_stmt>
14. < stmt > -> return [expresion];
15. < args > -> <data\_type> $var\_ID <arg\_def> <arg\_list>
16. < args > -> ε
17. < arg\_list > -> , <data\_type> $var\_ID> <arg\_list>
18. <arg\_list > -> ε
19. < ret\_type > -> : <data\_type>
20. < ret\_type > -> ε
21. < else\_stmt > -> else { <stmt\_list > }
22. < else\_stmt > -> ε
23. < data\_type > -> null
24. < data\_type > -> int
25. < data\_type > -> float
26. < data\_type > -> string
27. < arg\_def > -> = literal
28. < arg\_def > -> ε

## Gramatika výrazov

EXPR → EXPR + EXPR

EXPR → EXPR \* EXPR

EXPR → EXPR – EXPR

EXPR → EXPR / EXPR

EXPR → EXPR . EXPR

EXPR → EXPR < EXPR

EXPR → EXPR > EXPR

EXPR → EXPR <= EXPR

EXPR → EXPR >= EXPR

EXPR → EXPR === EXPR

EXPR → EXPR !== EXPR

EXPR → ! EXPR

EXPR → ( EXPR )

EXPR → id ( )

EXPR → id ( EXPR )

EXPR → id ( EXPR , EXPR )

EXPR → id ( EXPR , EXPR , EXPR )

EXPR → literal

## Precedenčná tabuľka

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | value | + - | \* / | < | > | <= | >= | === | !== | ( | ) | . | END |
| value | X | < | < | < | < | < | < | < | < | < | < | < | < |
| + - | > | > | < | > | > | > | > | > | > | < | > | > | < |
| \* / | > | > | > | > | > | > | > | > | > | < | > | > | < |
| < | > | < | < | > | > | > | > | > | > | < | > | < | < |
| > | > | < | < | > | > | > | > | > | > | < | > | < | < |
| <= | > | < | < | > | > | > | > | > | > | < | > | < | < |
| >= | > | < | < | > | > | > | > | > | > | < | > | < | < |
| === | > | < | < | > | > | > | > | > | > | < | > | < | < |
| !== | > | < | < | > | > | > | > | > | > | < | > | < | < |
| ( | > | < | < | < | < | < | < | < | < | < | = | < | < |
| ) | > | > | > | > | > | > | > | > | > |  | > | > | < |
| . | > | < | < | > | > | > | > | > | > | < | > | > | < |
| END | > | > | > | > | > | > | > | > | > | > | > | > | > |