Překladač jazyka Zyba

Staticky typovaný jazyk kompilovaný do PHP

Zyba language compiler

Language with static typing transpiled into PHP

Středoškolská odborná činnost, rok 2022
Richard Blažek
Gymnázium Brno, třída Kapitána Jaroše 14

Prohlášení

Prohlašuji, že jsem svou závěrečnou maturitní práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze závěrečné maturitní práce jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Brně dne 25. prosince 2021 .....

Poděkování

Tímto bych chtěl poděkovat Matěji Žáčkovi za odborné vedení práce.

Anotace

Práce se zabývá navržením jazyka Zyba a implementací překladače tohoto jazyka do PHP, což by mělo umožnit používání tohoto jazyka na všech webhostinzích, které podporují PHP skripty. Rovněž bude možné vyvíjet část projektu v Zybě a část v PHP.

Klíčová slova

programovací jazyk; překladač; transpiling; webová aplikace; PHP; Zyba

Annotation

The thesis is concerned with the design of the Zyba language and implementing its compiler. The compiler generates PHP code, making it possible for the language to be used on all webhostings where PHP is supported. Also, it will be possible to combine Zyba and PHP when developing web applications.

Keywords

programming lanugage; compiler; transpiling; web application; PHP; Zyba

Obsah

1 Úvod

Cílem této práce je navrhnout programovací jazyk Zyba, který by měl umožnit psaní přehlednějšího kódu než PHP, a překladač, který by přeložil kód ze Zyby do PHP, aby bylo možné programy v Zybě používat na všech na všech serverech s podporou PHP skriptů. Navrhl jsem Zybu jako staticky typovaný jazyk, protože překlad z jednoho dynamicky typovaného jazyka do druhého by umožnil provádět pouze minimální kontrolu při překladu a Zyba by představovala jen alternativní syntaxi pro PHP. Překladač jsem se rozhodl napsat v jazyce Haskell, protože umožňuje psát velmi stručné a přehledné programy. Haskell totiž vyžaduje, aby funkce byly referenčně transparentní (tzn. bez vedlejších efektů), a jeho typový systém umožňuje zápis rekurzivních typů, což se u překladače hodí například na zápis syntaktického stromu. Naopak není nutné používat oddělovače, struktura programu se vyjadřuje odřádkováním a odsazením, což dále přispívá k čitelnosti výsledných programů.

2 Návrh jazyka

2.1 Syntaxe

⟨*program*⟩ ::= ⟨*declaration*⟩*

⟨*declaration*⟩ ::= ⟨*name*⟩ ⟨*expression*⟩

⟨*expression*⟩ ::= ⟨*call-expression*⟩ (⟨*operator*⟩ ⟨*call-expression*⟩)*

⟨*call-expression*⟩ ::= ⟨*value-expression*⟩ (⟨*function-call*⟩ | ⟨*field-access*⟩ | ⟨*primitive-call*⟩)*

⟨*function-call*⟩ ::= **"["** ⟨*expression*⟩* **"]"**

⟨*field-access*⟩ ::= **":"** ⟨*name*⟩

⟨*primitive-call*⟩ ::= **"."** ⟨*name*⟩ **"["** ⟨*expression*⟩* **"]"**

⟨*value-expression*⟩ ::= ⟨*literal-int*⟩ | ⟨*literal-bool*⟩ | ⟨*literal-float*⟩ | ⟨*literal-string*⟩

⟨*value-expression*⟩ ::= **"("** ⟨*expression*⟩ **")"**

⟨*value-expression*⟩ ::= **"fun"** **"["** (⟨*name*⟩+ **":"** ⟨*expression*⟩)* **"]"** ⟨*block*⟩

⟨*block*⟩ ::= **"{"** ⟨*statement*⟩* **"}"**

⟨*statement*⟩ ::= ⟨*expression*⟩ | ⟨*assignment*⟩ | ⟨*if-statement*⟩ | ⟨*while-statement*⟩

⟨*assignment*⟩ ::= ⟨*name*⟩ **"="** ⟨*expression*⟩

⟨*if-statement*⟩ ::= ⟨*if-block*⟩ (**"else"** ⟨*if-block*⟩)* (**"else"** ⟨*block*⟩)*

⟨*if-block*⟩ ::= **"if"** ⟨*expression*⟩ ⟨*block*⟩

⟨*while-statement*⟩ ::= **"while"** ⟨*expression*⟩ ⟨*block*⟩

3 Závěr