Streszczenie pracy

Celem niniejszej pracy dyplomowej... W skrócie: Stworzyć kompilator który zamieni JavaScript na IL Assembler i będzie można go uruchomić na .NET. Następnie porównanie działania skryptów uruchamianych na maszynach Node.js oraz .NET. Można jeszcze porównać program napisany w C# z programem napisanym w js.

Abstract

The aim of this diploma thesis was...

Spis treści

Streszczenie pracy **3**

Abstract **3**

1 Wstęp **6**

2 Pojęcia i technologie **6**
2.1 Kompilator 6
2.2 JavaScript 6
2.3 Node.js 6
2.4 .NET Core 6
2.5 IL Assembler 6

3 Technologie pokrewne **6**
3.1 ECMAScript 6
3.2 ActionScript 6
3.3 JScript 6
3.4 TypeScript 7
3.5 ?CoffeeScript 7
3.6 Babel 7
3.7 Deno 7
3.8 asm.js 7
3.9 Emscripten 7
3.10 WebAssembly 7
3.11 C# 7
3.12 .NET Framework 7
3.13 Mono 7
3.14 DotGNU 7

4 Projekt kompilatora **8**
4.1 Środowisko i narzędzia 8
4.2 Analiza języka JavaScript i określenie zakresu implementacji . . . 8
4.3 Parser 8
4.4 Struktura projektu 8

5 Implementacja aplikacji kompilatora **8**
5.1 Parser 8
5.2 Analiza leksykalna 9
5.3 Gramatyka 9
5.4 Funkcjonalności 9
5.5 Generowanie assemblera 9

6 Testy **9**
6.1 Proste operacje matematyczne 9
6.2 Kolejność wykonywania działań 9
6.3 Wyrażenia warunkowe 9
6.4 Tablice 9
6.5 Obiekty 9
6.6 Klasy 9

6.7 Funkcje 10
6.8 Algorytm 1 10
6.8.1 Opracowanie pseudokodu algorytmu 1 10
6.8.2 Implementacja algorytmu 1 10
6.8.3 Testy algorytmu 1 10
6.9 Algorytm 2 10
6.9.1 Opracowanie pseudokodu algorytmu 2 10
6.9.2 Implementacja algorytmu 2 10
6.9.3 Testy algorytmu 2 10

7 Podsumowanie **10**

1 Wstęp

Motywacja, założenia i cele pracy. Nie więcej niż jedna strona.

2 Pojęcia i technologie

Wstęp do opisu technologii które będą wykorzystywane w projekcie.

2.1 Kompilator

Opis co to jest i do czego służy.

2.2 JavaScript

Opis

2.3 Node.js

Opis

2.4 .NET Core

Opis

2.5 IL Assembler

Opis

3 Technologie pokrewne

W tej sekcji będą przedstawione technologie pokrewne. Najpierw technologie konkurencyjne, następnie istniejące rozwiązania a na końcu rozwiązania podobne/nawiązujące w jakiś sposób do tematu.

3.1 ECMAScript

W sumie jest to standard języka na podstawie którego definiowana jest składnia JavaScript. Czy warto o tym wspominać?

3.2 ActionScript

Obiektowy język oparty na ECMAScript używany w Adobe Flash/

3.3 JScript

Jest to implementacja JavaScript przez Microsoft która jest uruchomiana w środowisku .NET. Istnieją pewne różnice w porównaniu do JavaScript.

3.4 TypeScript

Język programowania stworzony przez Microsoft. Uruchamiany na Node.js lub Deno. Może być przekompilowany do js es5 przez Babel.

3.5 ?CoffeeScript

Język programowania kompilowany do JavaScript. Nie wiem czy warto wspominać.

Inne języki kompilowane do JavaScript: Roy, Kaffeine, Clojure, Opal

3.6 Babel

Kompiluje przykładowo TypeScript do JavaScript lub JavaScript ES6 do ES5.

3.7 Deno

Maszyna wirtualna dla języka JavaScript oraz TypeScript.

3.8 asm.js

Jeśli dobrze zrozumiałem jest to okrojony JavaScript który tak samo może być uruchamiany w przeglądarkach czy Node.js/Deno. Wykorzystywany przy kompilacji kodu c++ do uruchamiania na tych maszynach.

3.9 Emscripten

Kompilator kodu LLVM do JavaScript.

3.10 WebAssembly

Język niskopoziomowy, który działa z szybkością zbliżoną do rozwiązań natywnych i pozwala na kompilację kodu napisanego w C/C++ do kodu binarnego działającego w przeglądarce internetowej. (Pomija JavaScript).

3.11 C#

Czy opisywać rodzinę .NET?

Na maszynę .NET istnieją implementacje języków takich jak Python, Java, C++ i inne.

3.12 .NET Framework

Platforma programistyczna.

3.13 Mono

Implementacja open source platformy .NET Framework.

3.14 DotGNU

Alternatywa dla Mono.

4 Projekt kompilatora

Opis

4.1 Środowisko i narzędzia

Opis sprzętu na którym będzie wszystko uruchomiane. Do implementacji będą wykorzystane:

- C#
- Visual Studio Code
- WSL (Ubuntu 20.04 LTS)
- JavaScript
- Node.js

4.2 Analiza języka JavaScript i określenie zakresu implementacji

Opis składni JavaScript. Implementacja JavaScript w standardzie ES5. Musi być Turing-complete. Dodatkowo implementacja funkcji.

4.3 Parser

Opcje: 1. Napiszę własną implementację, która może być mało czytelna i mieć dużo błędów. (Baza: http://informatyka.wroc.pl/node/391) 2. Użyję ANTLR i napiszę własną gramatykę. 3. Użyję ANTLR i użyję gotowe gramatyki.

Gotowe narzędzia:

- LEX & YYAC
- ANTLR
- Coco/R
- ?gppg & gplex
- Owl (https://github.com/ianh/owl)

i więcej... https://en.wikipedia.org/wiki/Comparison_of_parser_generators

4.4 Struktura projektu

Diagramy i opisy. Jak będzie wyglądał ten rozdział zależy jak wyjdzie implementacja.

5 Implementacja aplikacji kompilatora

5.1 Parser

Sposób implementacji lub przygotowania i użycia gotowych narzędzi.

5.2 Analiza leksykalna

Tekst

5.3 Gramatyka

Tekst

5.4 Funkcjonalności

Opis sposobu przetwarzania instrukcji

5.5 Generowanie assemblera

Tekst

6 Testy

Opis zakresu testów i jak będą przebiegać. Każdy z poniższych testów będzie sprawdzany pod kątem poprawności wykonywania działań, czasu wykonywania w porównaniu do wykonania kodu na Node.js (dla bardziej złożonych testów z czasem będzie więcej), zajętość pamięci (również tylko przy tych których to ma sens) oraz porównaniu kodu z asemblerowego wygenerowanego za pośrednictwem języka C#.

6.1 Proste operacje matematyczne

Test dodawania, odejmowania, mnożenia, dzielenia, przypisywania.

6.2 Kolejność wykonywania działań

Test na bardziej złożonych wyrażeniach. Sprawdzenie poprawności działania nawiasów oraz kolejności wykonywania działań.

6.3 Wyrażenia warunkowe

Test wyrażeń warunkowych. Kolejność wykonywania operacji and i or.

6.4 Tablice

Test obsługi tablic jedno i wielowymiarowych.

6.5 Obiekty

Test obsługi obiektów.

6.6 Klasy

Jeśli będzie implementacja. Test działania obiektów klas.

6.7 Funkcje

Test działania funkcji. 1. Funkcja "void"bez parametrów. 2. Funkcja "voidź parametrami. 3. Funkcja zwracająca różne typy (proste, tablice, obiekty) bez parametrów. 4. Funkcje zwracająca różne typy z parametrami. 5. inne

6.8 Algorytm 1

6.8.1 Opracowanie pseudokodu algorytmu 1

6.8.2 Implementacja algorytmu 1

6.8.3 Testy algorytmu 1

6.9 Algorytm 2

6.9.1 Opracowanie pseudokodu algorytmu 2

6.9.2 Implementacja algorytmu 2

6.9.3 Testy algorytmu 2

7 Podsumowanie