

## **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

<b>Streszczenie pracy</b>	<b>3</b>
<b>Abstract</b>	<b>3</b>
<b>1 Wstęp</b>	<b>6</b>
<b>2 Pojęcia i technologie</b>	<b>7</b>
2.1 Zakres projektu . . . . .	7
2.1.1 Kompilator . . . . .	7
2.1.2 JavaScript . . . . .	7
2.1.3 Node.js . . . . .	7
2.1.4 .NET Core . . . . .	7
2.1.5 IL Assembler . . . . .	7
2.2 Technologie pokrewne . . . . .	7
2.2.1 ECMAScript . . . . .	7
2.2.2 ActionScript . . . . .	7
2.2.3 JScript . . . . .	7
2.2.4 TypeScript . . . . .	7
2.2.5 ?CoffeeScript . . . . .	8
2.2.6 Babel . . . . .	8
2.2.7 Deno . . . . .	8
2.2.8 asm.js . . . . .	8
2.2.9 Emscripten . . . . .	8
2.2.10 WebAssembly . . . . .	8
2.2.11 C# . . . . .	8
2.2.12 .NET Framework . . . . .	8
2.2.13 Mono . . . . .	8
2.2.14 DotGNU . . . . .	8
2.2.15 ?Roslyn . . . . .	8
<b>3 Projekt kompilatora</b>	<b>9</b>
3.1 Środowisko i narzędzia . . . . .	9
3.2 Analiza języka JavaScript i określenie zakresu implementacji . . .	9
3.3 Parser . . . . .	9
3.4 Struktura projektu . . . . .	9
<b>4 Implementacja aplikacji kompilatora</b>	<b>10</b>
4.1 Parser . . . . .	10
4.2 Analiza leksykalna . . . . .	10
4.3 Gramatyka . . . . .	10
4.4 Funkcjonalności . . . . .	10
4.5 Generowanie assemblera . . . . .	10
<b>5 Testy</b>	<b>10</b>
5.1 Proste operacje matematyczne . . . . .	10
5.2 Kolejność wykonywania działań . . . . .	10
5.3 Wyrażenia warunkowe . . . . .	10
5.4 Tablice . . . . .	10
5.5 Obiekty . . . . .	10

5.6	Klasy . . . . .	11
5.7	Funkcje . . . . .	11
5.8	Algorytm 1 . . . . .	11
5.8.1	Opracowanie pseudokodu algorytmu 1 . . . . .	11
5.8.2	Implementacja algorytmu 1 . . . . .	11
5.8.3	Testy algorytmu 1 . . . . .	11
5.9	Algorytm 2 . . . . .	11
5.9.1	Opracowanie pseudokodu algorytmu 2 . . . . .	11
5.9.2	Implementacja algorytmu 2 . . . . .	11
5.9.3	Testy algorytmu 2 . . . . .	11
<b>6</b>	<b>Podsumowanie</b>	<b>11</b>

# 1 Wstęp

W branży programistycznej istnieje bardzo dużo języków programowania oraz różnych środowisk uruchomieniowych dla nich przeznaczonych. Podczas tworzenia oprogramowania niezbędny jest dla programisty kompilator. Zamienia on kod programu napisanego w konkretnym języku programowania, na zrozumiałą dla środowiska uruchomieniowego ciąg instrukcji. Warto podkreślić, że języki programowania, które są proste do zrozumienia i opanowania dla programistów oraz pozwalają na pisanie programów, które można uruchomić na różnych urządzeniach i umożliwiają tworzenie bardzo zróżnicowanych rodzajów oprogramowania, są o wiele częściej używane niż inne. Powoduje to, że powstaje dużo różnych bibliotek i frameworków ułatwiających i automatyzujących tworzenie oprogramowania.

Jednym z popularnych języków wśród programistów jest język JavaScript, który może być uruchamiany w przeglądarkach internetowych lub w maszynie wirtualnej takiej jak Node.js. Innym z popularnych środowisk uruchomieniowych jest platforma .NET, dla której istnieje wiele kompilatorów różnych języków. Wykorzystanie istniejących bibliotek, frameworków czy modułów napisanych w języku JavaScript w niezmienionej formie na platformie .NET, umożliwi programistom na tworzenie bardziej uniwersalnego kodu.

Celem niniejszej pracy jest zaprojektowanie oraz implementacja kompilatora języka JavaScript na kod IL Assembler uruchamiany na platformie .NET. Następnie w celu sprawdzenia poprawności działania kompilatora, zostaną przeprowadzone testy przy pomocy prostych implementacji kodu JavaScript. Zostaną również zaprojektowane oraz zaimplementowane dwa bardziej skomplikowane testy, do porównania maszyn wirtualnych Node.js oraz .NET. Zostanie także porównany kod assemblera generowanego przez kompilator .Net Core języka C# z kodem generowanym przez implementowany kompilator.

Praca podzielona jest na cztery części. Pierwsza część opisuje pojęcia i technologie wykorzystywane do realizacji tego projektu oraz technologie pokrewne, które w pewnym stopniu realizują cel pracy lub realizują podobne założenia. Druga część poświęcona jest zaprojektowaniu tworzonego kompilatora. Kolejna część opisuje sposób implementacji tego kompilatora na podstawie zdefiniowanych założeń. Ostatnia część opisuje przeprowadzone testy realizowane w ramach pracy oraz przedstawia wyniki ich działania.

## 2 Pojęcia i technologie

### 2.1 Zakres projektu

Wstęp do opisu technologii które będą wykorzystywane w projekcie.

#### 2.1.1 Kompilator

Opis co to jest i do czego służy.

#### 2.1.2 JavaScript

Opis

#### 2.1.3 Node.js

Opis

#### 2.1.4 .NET Core

Opis

#### 2.1.5 IL Assembler

Opis

### 2.2 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.

#### 2.2.1 ECMAScript

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

#### 2.2.2 ActionScript

Obiektowy język oparty na ECMAScript używany w Adobe Flash/

#### 2.2.3 JScript

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

#### 2.2.4 TypeScript

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

### **2.2.5 ?CoffeeScript**

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

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

### **2.2.6 Babel**

Kompiluje przykładowo TypeScript do JavaScript lub JavaScript ES6 do ES5.

### **2.2.7 Deno**

Maszyna wirtualna dla języka JavaScript oraz TypeScript.

### **2.2.8 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.

### **2.2.9 Emscripten**

Kompilator kodu LLVM do JavaScript.

### **2.2.10 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).

### **2.2.11 C#**

Czy opisywać rodzinę .NET?

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

### **2.2.12 .NET Framework**

Platforma programistyczna.

### **2.2.13 Mono**

Implementacja open source platformy .NET Framework.

### **2.2.14 DotGNU**

Alternatywa dla Mono.

### **2.2.15 ?Roslyn**

.NET Compiler Platform

## 3 Projekt kompilatora

Opis

### 3.1 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

### 3.2 Analiza języka JavaScript i określenie zakresu implementacji

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

### 3.3 Parser

Używamy ANTLR z gotową gramatyką

Rozważane możliwości i wykonano przegląd narzędzi: 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.

po 2 zdania: 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](https://en.wikipedia.org/wiki/Comparison_of_parser_generators)

### 3.4 Struktura projektu

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

## **4 Implementacja aplikacji kompilatora**

### **4.1 Parser**

Sposób implementacji lub przygotowania i użycia gotowych narzędzi.

### **4.2 Analiza leksykalna**

Tekst

### **4.3 Gramatyka**

Tekst

### **4.4 Funkcjonalności**

Opis sposobu przetwarzania instrukcji

### **4.5 Generowanie assemblera**

Opisać jak wygenerować kod .NET

## **5 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 assemblerowego wygenerowanego za pośrednictwem języka C#.

### **5.1 Proste operacje matematyczne**

Test dodawania, odejmowania, mnożenia, dzielenia, przypisywania.

### **5.2 Kolejność wykonywania działań**

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

### **5.3 Wyrażenia warunkowe**

Test wyrażeń warunkowych. Kolejność wykonywania operacji and i or.

### **5.4 Tablice**

Test obsługi tablic jedno i wielowymiarowych.

### **5.5 Obiekty**

Test obsługi obiektów.



## **5.6 Klasy**

Jeśli będzie implementacja. Test działania obiektów klas.

## **5.7 Funkcje**

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

## **5.8 Algorytm 1**

### **5.8.1 Opracowanie pseudokodu algorytmu 1**

### **5.8.2 Implementacja algorytmu 1**

### **5.8.3 Testy algorytmu 1**

## **5.9 Algorytm 2**

### **5.9.1 Opracowanie pseudokodu algorytmu 2**

### **5.9.2 Implementacja algorytmu 2**

### **5.9.3 Testy algorytmu 2**

## **6 Podsumowanie**