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

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 zrozumiały 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 uruchomiany 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

Projekt będzie realizował zaprojektowanie i implementację kompilatora języka JavaScript na platformę uruchomieniową .NET core. W tym rozdziale zostaną opisane pojęcia oraz technologie, które będą wykorzystywane w realizacji tego projektu. Na początku opisane będą takie pojęcia jak *kompilator* oraz *maszyna wirtualna*. Następie zostaną omówione technologie wiodące w projekcie takie jak *JavaScript*, *Node.js*, *.NET Core* oraz *IL Assembler*.

2.1.1 Kompilator

W dzisiejszych czasach niezbędnym narzędziem programisty jest kompilator. Jest to narzędzie, którego zadaniem jest tłumaczenie programu napisanego przez programistę, na program który będzie można uruchomić na konkretnym środowisku uruchomieniowym.

Mówiąc ściślej kompilator to program napisany w języku implementacyjnym, który odczytuje język źródłowy i tłumaczy go na język wynikowy. Proces zamiany kodu źródłowego na wynikowy nazywany jest **kompilacją**. Kodem wynikowym procesu kompilacji może być od razu kod maszynowy, który interpretowany jest bezpośrednio przez procesor lub maszynę wirtualną, albo do kodu pośredniego, który też może zostać skompilowany przez inny kompilator.

Kompilatory mogą być napisane w dowolnym języku programowania. Istnieje kilka specjalnie zaprojektowanych do tego zadania języków, takie jak *Pascal* czy *Algol 68*. Nie mniej jednak, wybór języka do implementacji kompilatora przez twórcę, powinien się opierać na założeniu, że powinien on zminimalizować wysiłek implementacyjny i zmaksymalizować jakość kompilatora.

Język źródłowy który przetwarzany jest przez kompilator prawie zawsze jest oparty na wcześniej zdefiniowanej gramatyce. Dzięki temu program kompilatora potrafi odróżnić od siebie kolejne instrukcje i zamienić je na równoważne ciągi instrukcji w języku docelowym.[1]

Podobnym w działaniu jest **interpreter**, który tak jak kompilator, jest pisany w jednym implementacyjnym oraz odczytuje język kodu źródłowego, ale nie produkuje kodu wynikowego, tylko odczytany kod jest od razu wykonywany. Niektóre języki przyjmują schematy zawierające wykorzystanie kompilatora oraz interpretera w procesie wytwarzania oprogramowania. Jednym z przykładów jest język **Java**, który kompilowany jest do postaci nazywanej *bytecode*, a następnie interpretowany jest przez maszynę wirtualną Java (Java Virtual Machine, JVM).[2]

2.1.2 Maszyna wirtualna

Wirtualizacja stała się ważnym narzędziem w projektowaniu systemów komputerowych, a maszyny wirtualne używane są w wielu obszarach informatycznych,

[1] W. M. Mckeeman. „Compiler Construction". W: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 1974, s. 1–36. ISBN: 9783540069584. DOI: 10.1007/978-3-662-21549-4_1. URL: http://link.springer.com/10.1007/978-3-662-21549-4_1, s. 1–4.

[2] *Engineering a Compiler*. Elsevier, 2012. ISBN: 9780120884780. DOI: 10.1016/C2009-0-27982-7. URL: https://linkinghub.elsevier.com/retrieve/pii/C20090279827, s. 3, 4.

od systemów operacyjnych po architektury procesorów języków programowania.

Dla programistów oraz użytkowników wirtualizacja likwiduje tradycyjne interfejsy oraz ograniczenia zasobów związanych z różnymi urządzeniami. Maszyny wirtualne zwiększają interoperacyjność oprogramowania oraz wszechstronność platformy, dlatego też często się je wykorzystuje.

Maszyna wirtualna to nic innego jak program uruchamiany na prawdziwej maszynie, który potrafi obsługiwać pożądaną architekturę. W ten sposób można obejść rzeczywistą kompatybilność maszyny i ograniczenia zasobów sprzętowych. Pozwala to, między innymi na równoczesne tworzenie oprogramowania dla wielu platform, bez konieczności stosowania bezpośrednio interfejsów rzeczywistej maszyny, a jedynie wykorzystanie tych udostępnianych przez maszynę wirtualną.[3]

2.1.3 JavaScript

Jest to skryptowy język programowania, dzięki którego można realizować aplikacje w paradygmacie imperatywnym, obiektowym oraz funkcyjnym. Najczęściej jest wykorzystywany w stronach internetowych, gdzie kolejne instrukcje wykonywane są przez przeglądarkę sieci Web, ale również zyskuje popularność w innych środowiskach.[4]

JavaScript został wdrożony w roku 1995 roku, jako sposób dodawania programów do stron internetowych. Jako pierwszą przeglądarką obsługującą JavaScript to Netscape Navigator. Następnie inne, głównie graficzne przeglądarki wprowadzały możliwość uruchamiania kodu napisanego w JavaScript. Umożliwiło to tworzenie nowoczesnych stron internetowych z którymi można było bezpośrednio współpracować, bez konieczności ponownego pobierania strony po każdej wykonanej akcji.

W momencie kiedy zaczęto używać JavaScript poza Netscape, został stworzony dokument standaryzujący, który opisuje sposób działania języka. Utworzono go, aby wszystkie nowo tworzone oprogramowanie mające wykorzystywać JavaScript, faktycznie używały tego samego języka. Dokument ten nazywany jest standardem **ECMAScript**, który został nazwany po organizacji Ecma International, twórców tego dokumentu.

JavaScript jest językiem bardzo elastycznym, przez co ma też swoje wady i zalety. Przez swoją elastyczność pozwala na wykorzystywanie wielu technik i praktyk programistycznych które mogą być niemożliwe w innych językach.

Jako, że jest to język skryptowy, to tak jak podobne tego typu języki posiada dynamiczne typowanie zmiennych. Oznacza to, że każda ze zmiennej jest definiowana poprzez słowo kluczowe `var`, a w nowszej wersji można to zrobić już przy pomocy dwóch różnych słów `const` oraz `let`. Kolejną z podstawowych rzeczy w JavaScript są funkcje. Dzięki nim można pisać programy we wspomnianych wcześniej paradygmatach. Pozwalają one nie tylko na wydzielenie kodu na mniejsze części ale również na definiowanie bardziej złożonych struktur czy klas.[5]

---

[3] J.E. Smith i Ravi Nair. „The architecture of virtual machines". W: *Computer* 38.5 (2005), s. 32–38. ISSN: 0018-9162. DOI: 10.1109/MC.2005.173. URL: http://ieeexplore.ieee.org/document/1430629/.

[4] MDN contributors. *About JavaScript*. 2020 (dostępny Maj 28, 2020). URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.

[5] Marijn Haverbeke. *Eloquent JavaScript: A Modern Introduction to Programming*. 2011. ISBN: 1593272820. DOI: 10.1190/1.9781560801597.

2.1.4 Node.js

Jest to asynchroniczne środowisko uruchomieniowe dla języka JavaScript. Node.js został zaprojektowany do tworzenia skalowalnych aplikacji sieciowych. Pozwala na jednoczesne przetwarzanie wielu połączeń. Przy każdym połączeniu następuje wywołanie zwrotne, a w przypadku jeśli nie będzie żadnej pracy do wykonania, Node.js przejdzie w tryb uśpienia.[6]

Środowisko Node.js oparte jest na implementacji silnika "V8" stworzonego przez Google. Zaimplementowany głownie jest w języku C i C++, koncentrując się na wydajności i niskim zużyciu pamięci. Różnica polega na tym, że silnik "V8" obsługuje głównie JavaScript w przeglądarkach internetowych, a Node.js został stworzony z myślą o obsłudze długotrwałych procesów serwerowych.

W celu obsługi jednoczesnego wykonywania logiki biznesowej, Node.js opiera się na asynchronicznym modelu zdarzeń wejścia i wyjścia, w przeciwieństwie do większości innych współczesnych środowisk, które oparte są na wielowątkowości. Model zdarzeń jest obsługiwany na poziomie języka, a jest to możliwe ponieważ JavaScript obsługuje wywołania zwrotne zdarzeń oraz funkcjonalny charakter JavaScript sprawia, że niezwykle łatwo jest tworzyć anonimowe obiekty funkcji, które można zarejestrować jako programy obsługi zdarzeń.[7]

2.1.5 .NET Core

Jest to platforma programistyczna ogólnego zastosowania z otwartymi kodami źródłowymi. Pozwala na tworzenie aplikacji dla systemów Windows, macOS oraz Linux przy wykorzystaniu różnych języków programowania.[8]

Architektura środowiska .NET składa się z takich komponentów jak:

- CTS (Common Type System) - opisuje wszystkie wspierane przez platformę typy. Definiuje zasady korzystania z danych typów, dostarcza zorientowany obiektowo model dla różnych języków implementowanych w .NET oraz zapewnia bibliotekę zawierającą prymitywne typy danych (takich jak `Boolean`, `char` itp.).

- CLS (Common Language Specyfication) - definiuje w jaki sposób mają być definiowane obiekty i funkcje, w języku przeznaczonym na platformę .NET. CLS jest podzbiorem CTS, co oznacza, że wszystkie opisane zasady CTS dotyczą również CLS.[9]

- FCL (Framework Class Library) - jest to standardowa biblioteka zawierająca podstawę implementacji klas, interfejsów, typów wartości czy usług, które wykorzystywane są do tworzenia aplikacji.[10]

[6] Node.js. *About — Node.js*. 2017.

[7] Stefan Tilkov i Steve Vinoski. „Node.js: Using JavaScript to Build High-Performance Network Programs". W: *IEEE Internet Computing* 14.6 (2010), s. 80–83. ISSN: 1089-7801. DOI: 10.1109/MIC.2010.145. URL: http://ieeexplore.ieee.org/document/5617064/.

[8] Paweł Łukasiewicz. *.NET Core vs .NET Framework*. (dostępny Lipiec 6, 2020). URL: https://www.plukasiewicz.net/Artykuly/NetFrameworkVsNetCore.

[9] MDN contributors. *Common Type System & Common Language Specification*. 2016 (dostępny Lipiec 6, 2020). URL: https://docs.microsoft.com/en-us/dotnet/standard/common-type-system.

[10] HOW TO ASP.NET. *.NET FCL (Framework Class Library)*. (dostępny Lipiec 6, 2020). URL: https://www.howtoasp.net/net-fcl-framework-class-library/.

- CLR (Common Language Runtime) - jest to środowisko uruchomieniowe, które uruchamia kod i zapewnia usługi ułatwiające proces programowania. Środowisko wykonawcze automatycznie obsługuje układ obiektów i zarządza referencjami no nich, zwalniając je w przypadku kiedy już nie są używane.[11]

2.1.6 IL Assembler

Każdy z kompilatorów przeznaczonych na platformę .NET, bez względu na wybrany język, kompiluje kod do postaci pośredniej, jakim jest kod IL.

Wykonywalny kod IL jest w formacie binarnym i nie jest czytelny dla człowieka. Oczywiście jak inne wykonywalne kody binarne, mogą zostać przedstawione w postaci assemblera, tak i kod IL może zostać zaprezentowany w postaci IL Assemblera. Zestaw instrukcji jest taki jak w przypadku tradycyjnego assemblera. Przykładowo, aby dodać dwie liczby należy użyć instrukcji `add`, a w przypadku odejmowania, należy użyć instrukcji `sub`.

Środowisko uruchomieniowe .NET nie potrafi jednak odczytywać bezpośrednio IL Assemblera. Aby kod napisany w IL Assemblerze można było uruchomić, trzeba skompilować go do postaci binarnej IL.[12]

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 uruchomiana 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.

[11] MDN contributors. *Common Language Runtime (CLR) overview*. 2019 (dostępny Lipiec 6, 2020). URL: `https://docs.microsoft.com/pl-pl/dotnet/standard/clr`.

[12] Sameers Javed. *Introduction to IL Assembly Language*. 2003 (dostępny Lipiec 8, 2020). URL: `https://www.codeproject.com/Articles/3778/Introduction-to-IL-Assembly-Language`.

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
- mono-devel (ilasm, ildasm)

3.2 Analiza języka JavaScript i określenie zakresu implementacji

W tym rozdziale zawarty zostanie zakres implementacji oraz opis poszczególnych elementów języka JavaScript. W projekcie zostanie zaimplementowana jedynie część standardu ECMAScript, a niektóre mechanizmy zostaną uproszczone.

3.2.1 Wyrażenia

Składnia języka JavaScript zapożycza wiele rozwiązań użytych w Javie, jednak na konstrukcję miły też wpływ takie języki jak: Awk, Perl i Python. W języku JavaScript instrukcję nazywane są wyrażeniami, które rozdzielane są znakiem średniaka. Znaki białe takie jak spacja, tabulator czy znak końca linii nie mają wpływu na sposób działania kolejnych elementów wyrażeń, stanowią jedynie sposób ich oddzielenia. W kodzie źródłowym JavaScript rozróżnialna jest wielkość liter oraz wspierany jest standard znaków Unicode. ECMAScript definiuje również zestaw słów kluczowych i literałów oraz zasady automatycznego umieszczania średników ASI (Automatic semicolon insertion).

3.2.2 Komentarze

Rozróżniane są dwa typy komentarzy:

1. Jednoliniowy - definiowany jest przy pomocy znaku „\\” oraz umieszczany jest na końcu linii.

```
console.log(); \\ komentarz
```

2. Wieloliniowy - zawarty jest pomiędzy dwoma elementami „/*” oraz „*/”

```
console.log();
/*
  komentarz na
  wiele linii
*/
```

3.2.3 Deklaracje zmiennych i stałych

Zmienne deklaruje się przy pomocy słów kluczowych `var`, `let` oraz `const`. Deklaracja przy pomocy `var` jest podstawowym sposobem tworzenia zmiennych w JavaScript. Zasięg takiej zmiennej nie może być ograniczony przez blok w którym jest zawarta, przez co może powodować błędy przy pisaniu kodu. W celu uściślenia zasięgu i przeznaczenia zmiennych powstały dwa inne sposoby deklaracji `let` oraz `const`. Oba te rodzaje deklaracji powodują, że zakres dostępności zmiennej jest ograniczony do bloku w którym została zadeklarowana. Różnicą między tymi dwoma deklaracjami jest taka, że przy pomocy `const` definiujemy stałą która musi być od razu zadeklarowana, a `let` działa podobnie jak `var`.

```
var zmienna1;
let zmienna2;
const stala = true;
```

Przy deklaracji zmiennych przy użyciu `var` lub `let`, których nie przypiszemy żadnej wartości to przyjmują one wartość `undefined`

3.2.4 Typy danych

W najnowszym standardzie ECMAScript zdefiniowanych jest siedem typów danych:

1. `Boolean` - może przybierać dwie wartości `true` lub `false`.
2. `null` - słowo kluczowe oznaczające wartość zerową.
3. `undefined` - wartość nieokreślona.
4. `Number` - tym przeznaczony dla literałów całkowitych jak i zmiennoprzecinkowych.
5. `String` - typ przeznaczony dla literałów łańcuchowych reprezentujących zero lub więcej pojedynczych znaków ujętych w podwójny lub pojedynczy cudzysłów.
6. `Symbol` - wprowadzony w ECMAScript 6 typ danych, który pozwala na tworzenie unikalnych i nie zmiennych wartości.
7. `Object` - typ złożony do którego zaliczają się funkcje, tablice, słowniki oraz instancje klas.

3.2.5 Operacje arytmetyczne

3.2.6 Operacje warunkowe

3.2.7 Zakres implementacji projektu

W niniejszym projekcie zostaną zaimplementowane następujące elementy:

- Komentarze jednoliniowe oraz wieloliniowe.
- Proste typy danych: `Boolean`, `Number`, `String`.
- Tworzenie zmiennych typu `var`.

- Uproszczona implementacja funkcji `console.log()`.
- Konwersja typów danych.
- Operacje matematyczne takie jak dodawanie, odejmowanie, mnożenie oraz dzielenie.
- Wyrażenia warunkowe takie jak sprawdzenie: równości, nierówności, większości lub mniejszości.
- Typ `Object` pod postacią tablicy elementów oraz słownika danych.
- Deklaracja oraz wywoływanie funkcji: bez parametrów oraz zwracanej wartości, bez parametrów oraz z zwracaną wartością, z parametrami bez zwracanej wartości oraz z parametrami z zwracaną wartością.

3.3 Parser

Używamy ANTLR z własną gramatyką ale posiłkując się gotowcem. Wykorzystanie gotowej gramatyki powoduje wygenerowanie tak dużych plików, że próba zrozumienia co jest do czego wymaga poświęcenia dużego wysiłku. Większość tych rzeczy i tak by nie została wykorzystana.

Rozważane możliwości i wykonano przegląd narzędzi:

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

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 asemblerowego 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 ”voidź parametrami. 3. Funkcja zwracająca różne typy (proste, tablice, obiekty) bez parametrów. 4. Funkcje zwracająca różne typy z parametrami. 5. inne

Bibliografia

ASP.NET, HOW TO. *.NET FCL (Framework Class Library)*. (dostępny Lipiec 6, 2020). URL: https://www.howtoasp.net/net-fcl-framework-class-library/.

contributors, MDN. *About JavaScript*. 2020 (dostępny Maj 28, 2020). URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.

— *Common Language Runtime (CLR) overview*. 2019 (dostępny Lipiec 6, 2020). URL: https://docs.microsoft.com/pl-pl/dotnet/standard/clr.

— *Common Type System & Common Language Specification*. 2016 (dostępny Lipiec 6, 2020). URL: https://docs.microsoft.com/en-us/dotnet/standard/common-type-system.

Engineering a Compiler. Elsevier, 2012. ISBN: 9780120884780. DOI: 10.1016/C2009-0-27982-7. URL: https://linkinghub.elsevier.com/retrieve/pii/C20090279827.

Haverbeke, Marijn. *Eloquent JavaScript: A Modern Introduction to Programming*. 2011. ISBN: 1593272820. DOI: 10.1190/1.9781560801597.

Javed, Sameers. *Introduction to IL Assembly Language*. 2003 (dostępny Lipiec 8, 2020). URL: https://www.codeproject.com/Articles/3778/Introduction-to-IL-Assembly-Language.

Mckeeman, W. M. „Compiler Construction". W: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 1974, s. 1–36. ISBN: 9783540069584. DOI: 10.1007/978-3-662-21549-4_1. URL: http://link.springer.com/10.1007/978-3-662-21549-4_1.

Node.js. *About — Node.js*. 2017.

Smith, J.E. i Ravi Nair. „The architecture of virtual machines". W: *Computer* 38.5 (2005), s. 32–38. ISSN: 0018-9162. DOI: 10.1109/MC.2005.173. URL: http://ieeexplore.ieee.org/document/1430629/.

Tilkov, Stefan i Steve Vinoski. „Node.js: Using JavaScript to Build High-Performance Network Programs". W: *IEEE Internet Computing* 14.6 (2010), s. 80–83. ISSN: 1089-7801. DOI: 10.1109/MIC.2010.145. URL: http://ieeexplore.ieee.org/document/5617064/.

Łukasiewicz, Paweł. *.NET Core vs .NET Framework*. (dostępny Lipiec 6, 2020). URL: https://www.plukasiewicz.net/Artykuly/NetFrameworkVsNetCore.