

Przemysław Gawlas

numer albumu: gp36035

kierunek studiów: Informatyka

specjalność: Inżynieria oprogramowania

forma studiów: studia niestacjonarne

PROJEKT I IMPLEMENTACJA KOMPILATORA JĘZYKA JAVASCRIPT NA PLATFORMĘ .NET.

DESIGN AND IMPLEMENTATION OF THE JAVASCRIPT COMPILER INTO THE .NET PLATFORM.

praca dyplomowa magisterska

napisana pod kierunkiem:

dr inż. Piotr Błaszyński

Katedra Inżynierii Oprogramowania i Cyberbezpieczeństwa

Data wydania tematu pracy: 05.10.2020

Data dopuszczenia pracy do egzaminu:

(uzupełnia pisemnie Dziekanat)

Szczecin, 2021

Oświadczenie autora pracy dyplomowej

Oświadczam, że praca dyplomowa magisterska pn. *Projekt i implementacja kompilatora języka JavaScript na platformę .NET.* napisana pod kierunkiem dr inż. Piotr Błaszyński jest w całości moim samodzielnym autorskim opracowaniem sporządzonym przy wykorzystaniu wykazanej w pracy literatury przedmiotu i materiałów źródłowych. Złożona w dziekanacie Wydziału Informatyki treść mojej pracy dyplomowej w formie elektronicznej jest zgodna z treścią w formie pisemnej.

Oświadczam ponadto, że złożona w dziekanacie praca dyplomowa ani jej fragmenty nie były wcześniej przedmiotem procedur procesu dyplomowania związanych z uzyskaniem tytułu zawodowego w uczelniach wyższych.

Podpis autora:

Szczecin, dnia:

Streszczenie

W tym miejscu trzeba napisać streszczenie pracy w języku polskim. Zawiera krótką charakterystykę dziedziny, przedmiotu i wyników zaprezentowanych w pracy. Maksymalnie 1/2 strony.

słowa kluczowe: np. informatyka, sterowanie, grafika komputerowa

Abstract

The abstract's purpose, which should not exceed 150 words, is to provide sufficient information to allow potential readers to decide on the thesis's relevance—a maximum of half the page.

keywords: e.g.: computer science, control, computer graphics

Spis treści

	Wstęp	7
1	Pojęcia i technologie	9
1.1	Zakres projektu	9
1.1.1	Kompilator	9
1.1.2	Maszyna wirtualna	10
1.1.3	JavaScript	10
1.1.4	Node.js	11
1.1.5	.NET Core	11
1.1.6	IL Assembler	12
1.2	Technologie pokrewne	12
1.2.1	ActionScript	12
1.2.2	JScript	13
1.2.3	TypeScript	13
1.2.4	?CoffeeScript	13
1.2.5	Babel	13
1.2.6	Deno	13
1.2.7	asm.js	13
1.2.8	Emscripten	13
1.2.9	WebAssembly	13
1.2.10	C#	13
1.2.11	.NET Framework	13
1.2.12	Mono	13
1.2.13	DotGNU	13
1.2.14	?Roslyn	14
2	Projekt kompilatora	15
2.1	Środowisko i narzędzia	15
2.2	Analiza języka JavaScript i określenie zakresu implementacji	15
2.2.1	Wyrażenia	15
2.2.2	Komentarze	15
2.2.3	Deklaracje zmiennych i stałych	16

2.2.4	Typy danych	16
2.2.5	Operacje arytmetyczne	16
2.2.6	Operacje warunkowe	16
2.2.7	Zakres implementacji projektu	16
2.3	Parser	17
2.4	Struktura projektu	17
3	Implementacja aplikacji kompilatora	19
3.1	Parser	19
3.2	Analiza leksykalna	19
3.3	Gramatyka	19
3.4	Funkcjonalności	19
3.5	Generowanie assemblera	19
4	Testy	21
4.1	Proste operacje matematyczne	21
4.2	Kolejność wykonywania działań	21
4.3	Wyrażenia warunkowe	21
4.4	Tablice	21
4.5	Obiekty	21
4.6	Klasy	21
4.7	Funkcje	21
4.8	Algorytm 1	22
4.8.1	Opracowanie pseudokodu algorytmu 1	22
4.8.2	Implementacja algorytmu 1	22
4.8.3	Testy algorytmu 1	22
4.9	Algorytm 2	22
4.9.1	Opracowanie pseudokodu algorytmu 2	22
4.9.2	Implementacja algorytmu 2	22
4.9.3	Testy algorytmu 2	22
	Podsumowanie	23
	Spis literatury	25
	Książki	25
	Artykuły	25
	Źródła internetowe i inne	25
A	Dodatek	27

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.

1. Pojęcia i technologie

1.1 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*. Następnie zostaną omówione technologie wiodące w projekcie takie jak *JavaScript*, *Node.js*, *.NET Core* oraz *IL Assembler*.

1.1.1 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

¹W. 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.

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

1.1.2 Maszyna wirtualna

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

1.1.3 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 programi-

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.

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

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

stycznych 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 zmiennych 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

1.1.4 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 głównie 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

1.1.5 .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.).

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

⁶Node.js. *About | Node.js*. 2017.

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

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

- CLS (Common Language Specification) - 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
- 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

1.1.6 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

1.2 Technologie pokrewne

Aktualnie istnieje wiele rozwiązań przetwarzających język JavaScript jak i narzędzi, dzięki którym można budować, jak i uruchamiać aplikacje dedykowane na platformę .NET. W tym rozdziale zostaną opisane niektóre z tych technologii.

1.2.1 ActionScript

Obiektowy język oparty na ECMAScript używany w Adobe Flash/

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

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

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

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

1.2.2 JScript

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

1.2.3 TypeScript

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

1.2.4 ?CoffeeScript

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

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

1.2.5 Babel

Kompiluje przykładowo TypeScript do JavaScript lub JavaScript ES6 do ES5.

1.2.6 Deno

Maszyna wirtualna dla języka JavaScript oraz TypeScript.

1.2.7 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.

1.2.8 Emscripten

Kompilator kodu LLVM do JavaScript.

1.2.9 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).

1.2.10 C#

Czy opisywać rodzinę .NET?

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

1.2.11 .NET Framework

Platforma programistyczna.

1.2.12 Mono

Implementacja open source platformy .NET Framework.

1.2.13 DotGNU

Alternatywa dla Mono.

1.2.14 ?Roslyn

.NET Compiler Platform

2. Projekt kompilatora

Opis

2.1 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)

2.2 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.

2.2.1 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 wyrażenia, 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).

2.2.2 Komentarze

Rozróżniane są dwa typy komentarzy:

1. Jednoliniowy - definiowany jest przy pomocy znaku “//” oraz umieszczany jest na końcu linii.

```
1 console.log(); // komentarz
```

Algorytm 2.1: Przykład komentarza jednoliniowego

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

```
1 console.log();
2 /*
3     komentarz na
4     wiele linii
5 */
```

Algorytm 2.2: Przykład komentarza wieloliniowego

2.2.3 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`.

```
1 var zmienna1;
2 let zmienna2;
3 const stała = true;
```

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

2.2.4 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.

2.2.5 Operacje arytmetyczne

2.2.6 Operacje warunkowe

2.2.7 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.
- Instrukcja `if` oraz pętle `while` oraz `for`
- 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.

2.3 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

2.4 Struktura projektu

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

3. Implementacja aplikacji kompilatora

3.1 Parser

Sposób implementacji lub przygotowania i użycia gotowych narzędzi.

3.2 Analiza leksykalna

Tekst

3.3 Gramatyka

Tekst

3.4 Funkcjonalności

Opis sposobu przetwarzania instrukcji

3.5 Generowanie assemblera

Opisać jak wygenerować kod .NET

4. 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#.

4.1 Proste operacje matematyczne

Test dodawania, odejmowania, mnożenia, dzielenia, przypisywania.

4.2 Kolejność wykonywania działań

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

4.3 Wyrażenia warunkowe

Test wyrażen warunkowych. Kolejność wykonywania operacji and i or.

4.4 Tablice

Test obsługi tablic jedno i wielowymiarowych.

4.5 Obiekty

Test obsługi obiektów.

4.6 Klasy

Jeśli będzie implementacja. Test działania obiektów klas.

4.7 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

4.8 Algorytm 1

4.8.1 Opracowanie pseudokodu algorytmu 1

4.8.2 Implementacja algorytmu 1

4.8.3 Testy algorytmu 1

4.9 Algorytm 2

4.9.1 Opracowanie pseudokodu algorytmu 2

4.9.2 Implementacja algorytmu 2

4.9.3 Testy algorytmu 2

Podsumowanie

Podsumowanie pracy powinno na maksymalnie dwóch stronach przedstawić główne wyniki pracy dyplomowej. Struktura zakończenia to:

1. Przypomnienie celu i hipotez
2. Co w pracy wykonano by cel osiągnąć (analiza, projekt, oprogramowanie, badania eksperymentalne)
3. Omówienie głównych wyników pracy
4. Jak wyniki wzbogacają dziedzinę
5. Zamknięcie np. poprzez wskazanie dalszych kierunków badań.

Spis literatury

Książki

- [1] *Engineering a Compiler*. Elsevier, 2012. ISBN: 9780120884780. DOI: 10.1016/C2009-0-27982-7. URL: <https://linkinghub.elsevier.com/retrieve/pii/C20090279827>.
- [2] Marijn Haverbeke. *Eloquent JavaScript: A Modern Introduction to Programming*. 2011. ISBN: 1593272820. DOI: 10.1190/1.9781560801597.

Artykuły

- [3] J.E. Smith i Ravi Nair. “The architecture of virtual machines”. W: *Computer* 38.5 (maj 2005), s. 32–38. ISSN: 0018-9162. DOI: 10.1109/MC.2005.173. URL: <http://ieeexplore.ieee.org/document/1430629/>.
- [4] Stefan Tilkov i Steve Vinoski. “Node.js: Using JavaScript to Build High-Performance Network Programs”. W: *IEEE Internet Computing* 14.6 (list. 2010), s. 80–83. ISSN: 1089-7801. DOI: 10.1109/MIC.2010.145. URL: <http://ieeexplore.ieee.org/document/5617064/>.

Źródła internetowe i inne

- [5] HOW TO ASP.NET. *.NET FCL (Framework Class Library)*. (dostępny Lipiec 6, 2020). URL: <https://www.howtoasp.net/net-fcl-framework-class-library/>.
- [6] MDN contributors. *About JavaScript*. Maj 2020 (dostępny Maj 28, 2020). URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript.
- [7] MDN contributors. *Common Type System & Common Language Specification*. Czer. 2016 (dostępny Lipiec 6, 2020). URL: <https://docs.microsoft.com/en-us/dotnet/standard/common-type-system>.
- [8] MDN contributors. *Common Language Runtime (CLR) overview*. Kw. 2019 (dostępny Lipiec 6, 2020). URL: <https://docs.microsoft.com/pl-pl/dotnet/standard/clr>.

- [9] Sameers Javed. *Introduction to IL Assembly Language*. Kw. 2003 (dostępny Lipiec 8, 2020). URL: <https://www.codeproject.com/Articles/3778/Introduction-to-IL-Assembly-Language>.
- [10] Paweł Łukasiewicz. *.NET Core vs .NET Framework*. (dostępny Lipiec 6, 2020). URL: <https://www.plukasiewicz.net/Artykuly/NetFrameworkVsNetCore>.
- [11] Node.js. *About | Node.js*. 2017.

A. Dodatek

W dodatkach mogą znaleźć się większe fragmenty kodów źródłowych, instrukcje działania programów, specyfikacje opcji, z którymi wywołuje się program, większe dane tabelaryczne, specyfikacje oprogramowania, dłuższe opisy teoretyczne, itp.

What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Why do we use it? It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

Where does it come from? Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

The standard chunk of Lorem Ipsum used since the 1500s is reproduced below for those interested. Sections 1.10.32 and

What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic

typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Why do we use it? It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

Where does it come from? Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, *consectetur*, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, *Łorem ipsum dolor sit amet..*", comes from a line in section 1.10.32.

The standard chunk of Lorem Ipsum used since the 1500s is reproduced below for those interested. Sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" are the only two sections of Lorem Ipsum that contain a sentence (or two) which are identical in the original and in the Lorem Ipsum version. What is Lorem Ipsum? Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Why do we use it? It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

Where does it come from? Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, *consectetur*, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45

BC. This book is a treatise on the theory of ethics, very popular during the Renaissance. The first line of Lorem Ipsum, Lorem ipsum dolor sit amet..", comes from a line in section 1.10.32.

The standard chunk of Lorem Ipsum used since the 1500s is reproduced below for those interested. Sections 1.10.32 an