# Projekt i implementacja kompilatora języka JavaScript na platformę .NET.

Przemysław Gawlas

Opiekun pracy: dr inż. Piotr Błaszyński

# Cel pracy

Celem pracy jest implementacja aplikacji wykonującej kompilację języka JavaScript do kodu pośredniego wykonywalnego na platformie .NET oraz porównania zaimplementowanych w języku JavaScript testowych algorytmów uruchamianych na platformie Node.js oraz .NET.

# Zakres pracy

- Opis pojęć i technologii wykorzystanych w projekcie oraz nawiązujących do tematu pracy.
- Analiza języka JavaScript
- Określenie zakresu implementacji
- Implementacja aplikacji kompilatora
- Wybór oraz implementacja algorytmów testujących
- Przeprowadzenie testów i zebranie wyników
- Podsumowanie wyników

# Wykorzystane narzędzia

## Implementacja kompilatora:

- Platforma *.NET core 3.1*
  - Język C#
- ANTLR 4.8

## Implementacja kodu testującego:

- Platforma *Node.js 10.15*
  - Język JavaScript
- Platforma *.NET Framework v4.0*
  - Język C#
  - Język JScript

# Wykorzystane narzędzia cd.

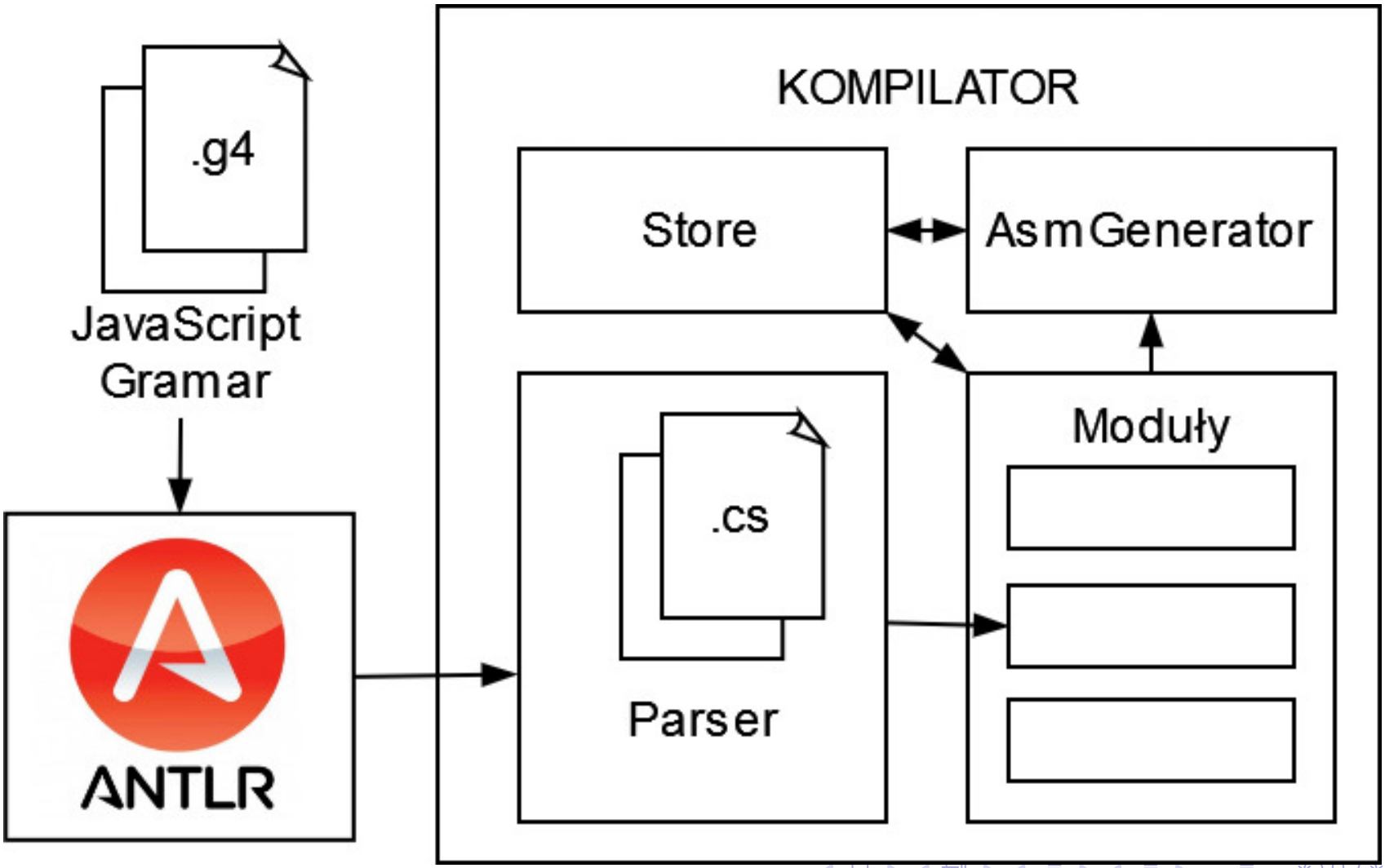
## Narzędzia testujące:

- Dekompilacja kodu:
  - Platforma *.NET Framework v4.0*
  - JetBrains dotPeek 2020.2.1
- Pomiar czasu i wielkości plików wykonywalnych:
  - PowerShell 7.1.1
- Pomiar zużycia pamięci:
  - JetBrains dotMemory 2020.2.1

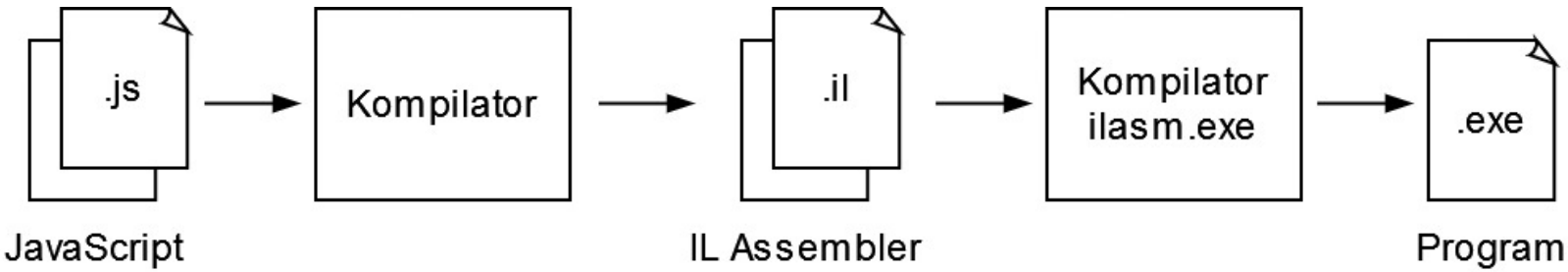
## Narzędzia pomocnicze:

- Skrypty pomocnicze
  - PowerShell 7.1.1

# Konstrukcja kompilatora



# Sposób działania



# Przygotowanie testów

## Zakres testów:

- Testy dla poszczególnych modułów
- Implementacja oraz przygotowanie algorytmów odnalezionych w Internecie
  - Algorytm sortowania bąbelkowego
  - Algorytm liniowego przeszukiwania

## Przygotowanie implementacji algorytmów:

- Niewielka modyfikacja implementacji.
- Przygotowanie odpowiedników w języku:
  - JScript
  - C#

# Kryteria testów

## Testy modułów

- Porównanie wyniku wykonania programów
- Porównanie generowanego kodu assemblera

## Testy algorytmów

- Pomiar czasu wykonywania programów
- Pomiar zużycia pamięci
- Pomiar wielkości plików programów

# Wyniki testów algorytmów

## Pomiar czasu wykonywania programów

### Algorytm sortowania

| Ilość elementów | C# | JavaScript | JScript |
|---|---|---|---|
| Oryginalne (7) | 01.0280485 | 01.0211699 | 01.0249265 |
| 1000 | 01.0267120 | 01.0209711 | 01.0281652 |
| 5000 | 01.0231339 | 01.0286513 | 11.1320074 |
| 10000 | 01.0331534 | 01.0466795 | 47.3928596 |

### Algorytm przeszukiwania

| Ilość elementów | C# | JavaScript | JScript |
|---|---|---|---|
| Oryginalne(13) | 01.0156735 | 01.0262087 | 01.0222548 |
| 1000 | 01.0101172 | 01.0150919 | 01.0134737 |
| 5000 | 01.0183013 | 01.0163112 | 01.0146355 |
| 10000 | 01.0167586 | 01.0116767 | 01.0203511 |

# Wyniki testów algorytmów

## Pomiar zużycia pamięci

### Algorytm sortowania

| Kompilator | Zużycie pamięci |
|---|---|
| C# | 11,26 MB |
| JavaScript | 11,25 MB |
| JScript | 28,72 MB |

### Algorytm przeszukiwania

| Kompilator | Zużycie pamięci |
|---|---|
| C# | 10,52 MB |
| JavaScript | 10,53 MB |
| JScript | 22,36 MB |

# Wyniki testów algorytmów

## Pomiar wielkości plików programów

### Algorytm sortowania

| Ilość elementów | C# | JavaScript | JScript |
|---|---|---|---|
| Oryginalne (7) | 4,00 kB | 3,00 kB | 7,00 kB |
| 1000 | 15,00 kB | 16,50 kB | 23,00 kB |
| 5000 | 57,50 kB | 71,00 kB | 89,50 kB |
| 10000 | 111,00 kB | 139,50 kB | 172,00 kB |

### Algorytm przeszukiwania

| Ilość elementów | C# | JavaScript | JScript |
|---|---|---|---|
| Oryginalne(13) | 4,00 kB | 2,50 kB | 6,00 kB |
| 1000 | 14,50 kB | 16,00 kB | 22,00 kB |
| 5000 | 57,50 kB | 71,00 kB | 88,50 kB |
| 10000 | 111,00 kB | 139,00 kB | 171,50 kB |

# Podsumowanie

- Przy analizie wyników poszczególnych testów funkcjonalności zostały wykazane niewielkie różnice przy wyświetlaniu elementów na konsoli.
- Porównanie generowanego kodu assemblera wykazało słabą optymalizacją stworzonego kompilatora.
- Testów algorytmów pokazują porównywalne wyniki dla kodu kompilowanego z języka C# oraz lepsze wyniki w porównaniu do kodu JScript.

# Kierunki dalszych prac

## Przykłady:

- Implementacja pełnej funkcjonalności języka JavaScript
- Zastosowanie pełnej gamy instrukcji asemblerowych
- Testy na bardziej złożonych algorytmach lub bibliotekach
- Optymalizacja generowanego kodu assemblera
- Dodanie mechanizmów zrównoleglania kodu

# Dziękuję za uwagę