# Techniki Kompilacji - Projekt

Interpreter prostego języka z typem wektorowym

Stawczyk Przemysław 293153

## 1 Opis Projektu

Projekt zakłada wykonanie interpretera prostego języka z obsługą wbudowanego typu *wektora* o *dim = {1,2}*. Język ten ma obsługiwać : zmienne z zasięgiem, instrukcje wykonywane na żądanie *funkcje*, instrukcje wykonywane nie zawsze *warunkowe*, wyrażenia matematyczne oraz operatory wraz z priorytetami.

### 1.1 Ogólne założenia

- Wartości liczbowe są reprezentowane przez liczby całkowite.
- Obsługiwane są typy :
  - Typ liczbowy *skalar*.
  - Typ wektorowy *vec* o *dim = {1,2}*
    np. *vec(1,2)*, *vec(4,5,6)*.
- Na typach liczbowych można definiować wyrażenia arytmetyczne za pomocą operatorów: *+ - * / ( )* z uwzględnieniem ich priorytetu i łączności.
- Wykonywanie operacji na wektorach : iloczynu skalarnego oraz wektorowego.
- Na typach *vec* i liczbowych można użyć operatorów przyrównania *== !=* oraz łączyć w wyrażenia za pomocą *|| && ( )*.
- Można definiować instrukcje warunkowe za pomocą konstrukcji *if()* oraz *else*.
- Można tworzyć pętle korzystając z konstrukcji *while()*.
- Można definiować funkcje z użycie słowa kluczowego *fun*.
- Zmienne są przekazywane do funkcji przez referencje.
- Program zaczyna wykonanie od bezparametrycznej funkcji *main()*

## 2 Opis Struktury Języka

TODO

## 3 Założenia Implementacyjne

Projekt miałby zostać zaimplementowany w języku *C++* z użyciem biblioteki *boost* do testów jednostkowych oraz narzędzia *CMake* do zarządzania procesem budowania.

Program miałby składać się z następujących modułów :

- modułu dostarczającego strumień kodu
- analizatora leksykalnego
- analizatora składniowego
- interpreter

Program jest aplikacja konsolową, uruchamianą wraz z parametrem reprezentującym ścieżkę do pliku ze skryptem do interpretacji. Wynik poszczególnych etapów analizy pliku oraz samego wyniku interpretacji końcowej i wykonania będzie wyświetlany na standardowym wyjściu. W zależności od ogólnego wyniku analizy, na standardowe wyjście mogą być zgłaszane: błędy leksykalny, błędy składniowe, błędy semantyczne lub wynik wykonania skryptu (wraz z możliwymi błędami wykonania)

## 4 Przykłady

```
function fun(a) {
  a = a * 2;
  if (a < 8) {
    fun(a);
  }
  print("a ", a);
  return a;
}

function main() {
  print(fun(2));
}
```