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. *z użyciem wbudowanych funkcji*
- 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()*
- Wartości logiczne sa reprezentowane przez liczby gdzie 0=fałsz, !0 = prawda. Każdy wektor jest ewaluowany do prawdy.
- Język wspierać ma operację *print(...)* przyjmująca oprócz typów numerycznych stałe tekstowe *[w cudzysłowach]* istniejące wyłącznie na potrzeby *print*.

1.2 Funkcje Biblioteczne

- fun **product3**(vec1, vec2) - *iloczyn wektorowy wektorów wymiaru 3*
- fun **product2**(vec1, vec2) - *iloczyn wektorowy wektorów wymiaru 2*
- fun **scalar3**(vec1, vec2) - *iloczyn skalarny wektorów wymiaru 3*
- fun **scalar2**(vec1, vec2) - *iloczyn skalarny wektorów wymiaru 2*

2 Opis Struktury Języka

TODO

3 Założenia Implementacyjne

Projekt miałby zostać zaimplementowany w języku *C++* z użyciem biblioteki *boost* do testów jednostkowych *boost::unit_test* oraz parsowania argumentów *boost::program_options*. Całość korzystać ma z narzędzia *CMake* do zarządzania procesem budowania.

Program miałby składać się z następujących modułów :

- analizator leksykalny
- analizator składniowy
- interpreter

3.1 Produkt Końcowy

Finalny program ma być konsolowa aplikacja uruchamianą wraz z parametrem reprezentującym ścieżkę do pliku do interpretacji oraz poziomem wypisywanych informacji. 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: leksykalne, składniowe, semantyczne lub wynik wykonania skryptu.

4 Przykłady

Przykład 1

```
fun licz(a) {
    a = a * 2;
    if (a < 10) {
        fun(a);
    }
    return a;
}
```

```
fun main() {
    if(11){
        print(fun(2));
    }

}
```

Przykład 2

```
fun printAndRet3(a) {
   print(a);
   return 3;
}

fun main() {
   var a = 0;
   if(a == 0) {
      print("a=";
      print(a);
   }

   print(vec(1,2));
   print(vec(1,2,3));

   var b = a;
   while(b < 10) {
      b = b + 1;
   }

   print("b= ", b);

   fun(1);
   fun(fun(2));
}
```