Interpreter prostego języka

- Dokumentacja wstępna

Autor: Aleksander Zamojski

# Opis projektu

Projekt ma na celu wykonanie interpretatora prostego języka. Język ma być wyposażony w zmienne z zasięgiem, dwie podstawowe konstrukcje sterujące (pętla oraz instrukcja warunkowa), możliwość definiowania funkcji oraz wbudowany typ wektorowy (2-, 3-wymiarowy). Dodatkowo język powinien obsługiwać wyrażenia matematyczne uwzględniając priorytet operatorów.

# Zakładana funkcjonalność

* Odczytanie, parsowanie i analiza skryptów zapisanych w plikach tekstowych
* Kontrola poprawności wprowadzonych danych oraz zgłaszanie błędów wykrytych podczas kolejnych etapów analizy plików
* Wykonywanie poprawnie zapisanych instrukcji, nie produkujących błędów, z plikach wejściowych
* Możliwość definiowania typów:
  + number (liczba całkowita)
  + vec, (2,3-wymiarowe wektory)
  + string (typ znakowy istniejący tylko w funkcji print)
* Wykonanie wyrażeń matematycznych uwzględniając priorytet operatorów ((),\*,/,+,-)
* Wykonanie wyrażeń logicznych uwzględniając priorytet operatorów ((),==,||,&&)
* Wykonywanie operacji na wektorach (dodawanie, odejmowanie, iloczyn skalarny, iloczyn wektorowy)
* Możliwość używania instrukcji warunkowych oraz pętli
* Funkcja print, wypisująca informacje podane przez użytkownika
* Możliwość definiowania własnych funkcji oraz ich późniejszego wywoływania w skryptach
* Użycie typizacji silnej i dynamicznej

# Biblioteka standardowa

Wstępnie przewidywana biblioteka funkcji wbudowanych:

* Podstawowe
  + print (…)  
    Wypisuje zawartość na standardowe wyjście.
* Operacje na wektorach
  + convertFrom2dTo3d(vec, vec)  
    Zmienia wektor dwuwymiarowy na trójwymiarowy.
  + convertFrom3dTo32(vec, vec)  
    Zmienia wektor trójwymiarowy na dwuwymiarowy.
  + crossProduct2(vec, vec)  
    Wykonuje iloczyn wektorowy na dwóch dwuwymiarowych wektorach.
  + crossProduct3(vec, vec)  
    Wykonuje iloczyn wektorowy na dwóch trójwymiarowych wektorach.
  + scalarProduct2(vec, vec)  
    Wykonuje iloczyn skalarny na dwóch dwuwymiarowych wektorach.
  + scalarProduct3(vec, vec)  
    Wykonuje iloczyn skalarny na dwóch trójwymiarowych wektorach

# Gramatyka

**program** = { functionDef } ;

**functionDef** = "function" identifier parameters statementBlock ;

**parameters** = "(" [ identifier { "," identifier } ] ")" ;

**statementBlock** = "{" { initStatement | assignStatement | returnStatement |

ifStatement | whileStatement | functionCall ";" | "continue" ";" |

"break" ";" | printStatement | statementBlock } "}" ;

**returnStatement** = "return" logicExpr ";" ;

**initStatement** = "var" identifier [ "=" logicExpr ] ";" ;

**assignStatement**​ = variable "=" logicExpr ";" ;

**ifStatement** = "if" "(" logicExpr ")" statementBlock [ "else" statementBlock ] ;

**whileStatement** = "while" "(" logicExpr ")" statementBlock ;

**functionCall** = identifier arguments ;

**arguments**​ = "(" [ logicExpr { "," logicExpr } ] ")" ;

**printStatement** = "print" "(" (stringLiteral | logicExpr) {"," (stringLiteral | logicExpr)} ")" ";"

**logicExpr**​ = andExpr { orOp andExpr } ;

**andExpr**​ = relationalExpr { andOp relationalExpr } ;

**relationalExpr** ​= baseLogicExpr [ relationOp baseLogicExpr ] ;

**baseLogicExpr**​ = [ unaryLogicOp ] mathExpr ;

**mathExpr** = multiplicativeExpr { additiveOp multiplicativeExpr } ;

**multiplicativeExpr** ​= baseMathExpr { multiplicativeOp baseMathExpr } ;

**baseMathExpr** ​= [unaryMathOp ] (value | parentLogicExpr) ;

**parentLogicExpr**​ = "(" logicExpr ")" ;

**value** = numberLiteral |vectorLiteral | variable | functionCall ;

**unaryMathOp**​ = "-" ;

**unaryLogicOp**​ = "!" ;

**additiveOp** ​= "+" | "­" ;

**multiplicativeOp** ​= "\*" | "/" | "%" ;

**orOp** ​= "||" ;

**andOp**​ = "&&" ;

**relationOp** ​= "==" | "!=" | "<" | ">" | "<=" | ">=" ;

**variable**​ = identifier [ index ] ;

**index** = "[" numberLiteral "]" ;

**stringLiteral** = '"' { allCharacters – '"' } '"' ;

**vectorLiteral** = "vec" "(" numberLiteral "," numberLiteral ["," numberLiteral] ")" ;

**numberLiteral** = digit { digit } ;

**identifier** = letter { letter | digit | specialElement } ;

**specialElement** = "\_" | “@" ;

**letter** = "a".."z" | "A".."Z" ;

**digit** = "0".."9";

**allCharacters** = ? all visible characters ? ;

# Informacje techniczne

## Środowisko

Projekt zostanie zaimplementowany w języku C++, wykorzystując bibliotekę do testów jednostkowych: "Catch". Całość będzie budowana za pomocą "CMake".

## Obsługa programu

Program będzie prostą aplikacja konsolową, uruchamianą wraz z parametrem reprezentującym ścieżkę do pliku ze skryptem do interpretacji, oraz ewentualnymi flagami (np. uruchomienie dokładniejszego trybu zgłaszania błędów).  
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 czasu wykonania). Jako że jest to aplikacja konsolowa, nie przewiduję zapisywania wyników do pliku (można to zrobić przekierowując wyjście bezpośrednio do pliku).

# Przykłady

|  |  |
| --- | --- |
| function fun1( variable ) {  if ( variable > 10) {  return 1;  }  while ( variable > 0 ) {  print ( variable );  variable = variable – 1;  }  return 0;  } | function main() {  var a = 1;  var b = a\*2 + 1;  var v1 = vec(1,2);  var v2 = vec(1,2,3);  print ("vector v1: ", v1);  fun1( b );  return 0;  } |