Interpreter prostego języka
dokumentacja wstępna

Mariusz Zakrzewski

2 kwietnia 2017

1 Temat i zakres projektu

Tematem projektu jest interpeter prostego języka programowania, umożliwiającego operacje na dodatkowym typie danych.

Język będzie umożliwiał podstawowe operacje na typach danych: liczba całkowita, liczba rzeczywista, napis, zmienna boolowska oraz na dodatkowym typie danych reprezentującym poruszający się obiekt, wraz z jego położeniem, prędkością, przyspieszeniem w przestrzeni dwu-wymiarowej. Umożliwiać będzie definiowanie własnych funkcji, intrukcji rozgałęziających, pętli, wypisywania na standardowe wyjście.

2 Wymagania funkcjonalne

1. odczytywanie oraz wykonywanie programów zapisanych w plikach tekstowych
2. zgłaszanie błędów kompilacji
3. możliwość definiowania własnych funkcji
4. program dba o poprawną kolejność operatorów na danych
5. język umożliwia pisanie instrukcji rozgałęziających
6. język umożliwia pisanie pętli

7. wyświetlanie komunikatów na standardowe wyjście termianala

8. program dba także o zasięg zmiennych - możliwe jest użycie 2 identycznych nazw w 2 różnych funkcjach - brak konfliktu, możliwe wykorzystanie funkcji rekurencyjnych

3 Wymagania niefunkcjonalne

1. komunikaty o błędach są czytelne

2. czas interpretacji i wykonania jest rozsądnie krótki

4 Sposób obsługi programu i opis realizacji projektu

Program zostanie napisany oraz testowany pod systemem Linux w języku C++(standard C++11) jako aplikacja konsolowa.
Do zbudowania projektu zostanie użyte narzędzie make lub scons.

Struktura projektu będzie obejmowała:

- moduł lexera(analizatora leksykalnego) odpowiadający za wytworzenie strumienia tokenów z pliku tekstowego

- moduł parsera(analizatora składniowego) odpowiadający za utworzenie drzewa AST(Abstract Syntax Tree)

- moduł analizatora semantycznego, odpowiadającego za sprawdzenie powstałego drzewa ast i ewentualne generowanie błędów

- tablica symboli - klasy pomocnicze reprezentująca zmienne i ich zasięg

- obsługa błędów - klasy umożliwiające obsługę błędów - wypisanie komunikatu, ewentualną korektę dla procesu wykrycia i wyświetlenia dalszych potencjalnych błędów

5 Przykładowe użycie programu i składnia

```
def main() {
    // komentarz
    /* kom
    entarz
    */

    var myInt: int = 6;
    var myString: string = "hello";
    var myDouble: doube = 3.14;
    var myBoolean: boolean = true;

    //((polozenieX,polozenieY),(predkoscX,predkoscY),(przyspieszenieX,przyspieszenieY))
    var myMoveable: moveable = ((1,2),(3,4),(5,6));
    print(myMoveable.posX); // 1
    print(myMoveable.speedX); // 3
    print(myMoveable.accX); // 5
    var myPosition: position = (2,3);
    print(myPosition.x); // 2
    print(myPosition.y); // 3

    // obiekt singleton globalny dla calego programu, steruje 'czasem' w ruchu wszystkich obiektow
    // zmieniajac wartosc globalTime poruszamy wszystkimi zdefiniowanymi instancjami typu moveable
     // (takze zmieniamy predkosci obiektow o niezerowym przyspieszeniu)
    moveable.globalTime = 0;

    var myMoveable2: moveable = ((0,0),(1,0),(0,0));
    globalTime += 1;
    print(myMoveable2.posX); // 1
    print(myMoveable2.posY); // 0

    var myMoveable3: moveable = ((0,0),(1,0),(0,0));
    var myMoveable4: moveable = ((10,0),(-1,0),(0,0)) ;

    moveable.setOnCollision(myMoveable3, myMoveable4, onCollisionFun);

    globalTime += 4;
    globalTime += 1;
    // zostanie zawolana funkcja onCollisionFun(myMoveable3, myMoveable4, (5,0), 6)

    // dodatkowe mozliwosci typu danych moveable, pozwalajace na rozsadne
     // zarzadzanie tymi obiektami
    myMoveable.id = 12;
    myMoveable.name = "nazwa konkretnego obiektu";

    var myBoolean1: boolean = false;
    if(myBoolean1 && 2 < 4){
        print("true");
    }else{
        print("false");
    }

    var iter: int = 0;
     while(iter < 10){
          iter += 1;
          print(iter + " ");
     }

    var a: int = 2;
    var b: int = 3;
    var c: int = 4;
    print(a + b * c); // 14
    print((a + b) * c); // 24

    var myStr: string = "hello";
    var myStr2: string = " world";
    var myStr3: string = myStr + myStr2; // "hello world"
    var myStr4: string = myStr + a; // "hello2"

    /* ==== obsluga bledow === */
    myStr = 3; // type mismatch
    myStr = myStr * 3; // undefined operand '*' for string and int
    if(false {  // ')' expected
        print("abc");
    }
```

```
    var myInt5 = 8; // type not specified
    myMoveable.nmae = "abc"; // 'nmae' field not undefined
    // program bez funkcji 'main': function 'main' not defined

    var d: int
    d = sayHello(); // incompatible types: void cannot be converted to int
}

def sayHello() {
    print("hello world");
    // nic nie zwraca
}

def mullBy2(myInt: int): int {
    return myInt * 2
}

def onCollisionFun(mov1: moveable, mov2: moveable, pos: position, time: int) {
    print("zderzenie obiketu: " + mov1.name + ", z obiektem: "
        + mov2.name + ", w miejscu:" + pos + ", w czasie: " + time)
}
```

6 Gramatyka

6.1 Lista tokenów

def "= , . : ( ) { } while int boolean string double moveable position posX posY speedX speedY accX accY var true false * / + - % ! [ ] <><= >= == != && || return null

6.2 Gramatyka

```
function_definition
    : "def" identifier "(" [ formal_arglist ] ")" [":" type ] "{" statement_list "}"

formal_arglist
    : identifier ":" type [ "," formal_arglist ]

statement_list
    : statement
    | (statement statement_list)

statement
    : compound_statement
    | if_statement
    | while_statement
    | expression ";"
    | "return"
    | "return" expression;
    | "var" identifier ":" type "=" expression ";"
    | "var" identifier ":" type ";"

compound_statement:
    : ("{" "}")
    | ("{" statement_list "}")

if_statement
    : "if" "(" expression ")" statement [ "else" statement]

while_statement
    : "while" "(" expression ")" statement

expression
    : numeric_constant
    | string_constant
    | "(" "(" expression "," expression ")" "," "(" expression "," expression ")" "," "(" expression "," expressior
```

```
    | "(" expression "," expression ")"
    | "null"
    | identifier
    | identifier "(" actual_arglist ")"
    | identifier "." identifier
    | expression binary_operator expression
    | unary_operator expression
    | "(" expression ")"
    | identifier assignment_operator expression
    | identifier "." identifier assignment_operator expression

actual_arglist
    : expression [ "," actual_arglist]

binary_operator
    : "+" | "-" | "*" | "/" | "%" | "<" | "<=" | "==" | "=>" | ">"

unary_operator
    : "!" | "-" | "&"

assignment_operator
    : "+="| "*=" | "-=" | "/=" | "="

numeric_constant
    : (1-9)[0-9]* [ "." (0-9)*]

string_constant
    : (a-Z | "_")*
```