

[TKOM] Dokumentacja

Szymon Kisiel

Temat projektu

Interpreter języka z pozycją geograficzną jako niestandardowy typ danych.

Elementy języka

- Typ całkowitoliczbowy **int**
- Typ tekstowy **string**
- Niestandardowe typy: **geo** (pozycja geograficzna w postaci szerokości i długości geograficznej) i **geocoord** (współrzędna geograficzna w stopniach, minutach i sekundach)
 - Symbole $^{\circ}$, $'$, $''$ będą używane jako odpowiednio: stopnie, minuty, sekundy
 - Przykładowa współrzędna geograficzna: $55^{\circ} 45' 30''$
 - Przykładowa pozycja geograficzna: $55^{\circ} 45' 30''$ N $37^{\circ} 37' 45''$ E
- Obsługa zmiennych (statyczne, mutowalne, typowanie silne)
- Instrukcja warunkowa **if**, **elsif**, **else**
- Instrukcja pętli **while**
- Możliwość wołania i definiowania funkcji
- Komentarze
 - liniowy: **# ...**
 - blokowy: **/* ... */**

Przykłady użycia języka

Operacje matematyczne

```
int x = 2*5+2;
int y = 2*(5+2);
y = y - x + 3;
print(y);
```

Niestandardowy typ

```
geo warszawa = 52^13'5''N 21^30''E;
geo gdansk = 54^20'51''N 18^38'43''E;
geo diff;

if (warszawa > gdansk) {
    diff = warszawa - gdansk;
}

else {
    diff = gdansk - warszawa;
}
```

Instrukcje warunkowe i pętle

```
int i = 0;
int j = 10;
while (i < 10) {
    while (j > 0)
        if (i == 5) {
            if (j == 5) {
                print("a");
            }
            else
                print("b");
        }
        elseif (i == 6) {
            print("c");
        }
}
```

Funkcje

```
string test(string s1, string s2) {
    return s1 + " " + s2;
}
string x = test("abc", "def");
```

Funkcje rekurencyjne

```
int silnia(int a) {
    return a*silnia(a-1);
}
```

Opis gramatyki

program = {statement | function} ;

function = type , id , "(" , [parameters] , ")" , "{" , {statement} , "}" ;

statement = if_statement | while_statement | simple_statement | "{" , {statement} , "}" ;

if_statement = "if" , "(" , expression , ")" , statement
 , {"elseif" , "(" , expression , ")" , statement}
 , ["else" , statement] ;

```

while_statement = "while" , "(" , expression , ")" , statement ;

simple_statement = (var_declaration | assignment | function_call | return_statement) , ";"
;

var_declaration = type , id , "=", expression ;

assignment      = id , "=", expression ;

function_call   = id , "(" , [arguments] , ")" ;

return_statement = "return" , [expression] , ";" ;

parameters = parameter , {" , " , parameter} ;

parameter = type , id ;

arguments = argument , {" , " , argument} ;

argument = expression ;

expression      = add_expression , {comp_operator, add_expression} ;
add_expression = mult_expression , {add_operator, mult_expression} ;
mult_expression = factor , { mult_operator , factor} ;
factor          = ["-"] , (integer | float | geo | string | id | function_call
                        | "(" , expression , ")") ;

id = letter , {letter | digit | "_"} ;

type = "void" | "int" | "float" | "string" | "geo" | "geocoord" ;

comp_operator = "<" | ">" | "<=" | ">=" | "==" | "!=" ;
add_operator  = "+" | "-" | "or" ;
mult_operator = "*" | "/" | "and" ;

geo_pos = geo_coord , ("N" | "S") , geo_coord , ("W" | "E") ;
geo_coord = integer , "°" , [integer , "'"] , [integer , '"']
            | [integer , "°"] , integer , "'" , [integer , '"']
            | [integer , "°"] , [integer , "'"] , integer , '"';

comment = "/*" , {character} , "*/"
         | "#", {character}, newline ;

string = "'" , {character} , "'" ;
character = letter | digit | ... ;

```

```
letter    = "A" | "B" | "C" | ...
```

```
float      = integer , "." , digit, {digit}
```

```
integer    = ["-"] , (zero | digit_not_zero , {digit}) ;
```

```
digit      = zero | digit_not_zero ;
```

```
zero       = "0" ;
```

```
digit_not_zero = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
```

Komponenty

Lekser generuje kolejne tokeny na podstawie pliku wejściowego.

Parser na podstawie kolejnych tokenów z leksera tworzy drzewo składniowe.

Wykonanie programu polega na wywołaniu metody execute na drzewie składniowym.

Wbudowana funkcja print pisze wartość argumentu do strumienia cout.

Obsługa błędów

- Błędy leksera
 - Brak zakończenia typu tekstowego
 - Brak zakończenia komentarza
 - Przekroczenie rozmiaru int
 - Nieznany token
 - Błędy parsera
 - Błędy składniowe
 - Błędy wykonania
 - Użycie niezadeklarowanej zmiennej
 - Użycie niezadeklarowanej funkcji
 - Ponowna deklaracja zmiennej
 - Ponowna deklaracja funkcji
 - Niepoprawna ilość argumentów przekazana do funkcji
 - Niepoprawne operacje matematyczne/logiczne (np. dodawanie typów tekstowych)
- Niezaimplementowane:
- Sprawdzenie typu wartości zwracanej w funkcji
 - Sprawdzenie typów wartości argumentów
 - Sprawdzenie typu wartości przypisywanej do zmiennej

Niezaimplementowane funkcjonalności

- Parsowanie typów geograficznych i operacje na nich
- Parsowanie deklaracji zmiennej na poziomie „program”
- Negacja factor
- Testy jednostkowe