

# TKOM – Język do wizualizacji

## Dokumentacja Końcowa

### 1. Opis

Celem projektu jest stworzenie języka, pozwalającego na wizualizację obiektów. Język ten ma mieć możliwość definiowania własnych figur przy pomocy funkcji oraz rysowania podstawowych kształtów

### 2. Funkcjonalność

2.1 Definiowanie funkcji

2.2 Instrukcje warunkowe

2.3 Pętle

2.4 Rysowanie podstawowych figur geometrycznych (odcinek, kwadrat, koło, punkt)

### 3. Specyfikacja

#### 3.1 Środowisko programistyczne

Projekt zostanie wykonany w języku C++. Wykorzystana zostanie biblioteka Qt do rysowania obiektów. Biblioteka cxxopts może zostać wykorzystana do pobierania argumentów z linii poleceń. Biblioteka catch2 może zostać wykorzystana do pisania testów jednostkowych. Aby skompilować program trzeba będzie użyć polecenia qmake, a następnie poleceń make.

#### 3.2 Interpreter

Interaktywny z nieskończoną pętlą, tzw. Read Evaluate Print Loop. Wczytane zostanie polecenie ze standardowego wejścia, wykonane, a wynik zostanie umieszczony na rysunku.

#### 3.3 Typy

3.3.1 Integer

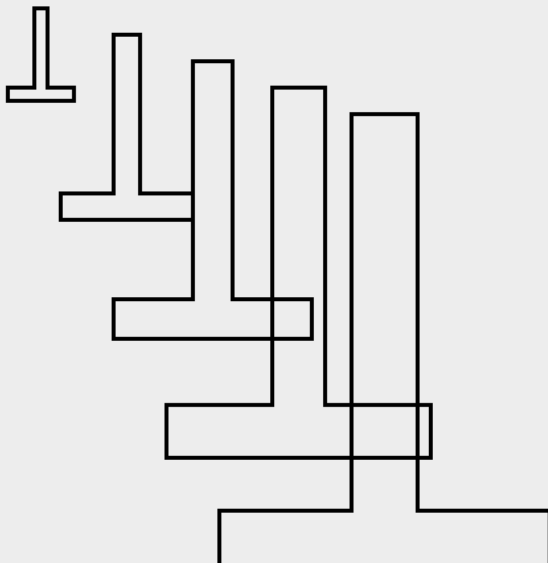
3.3.2 func

3.3.3 void

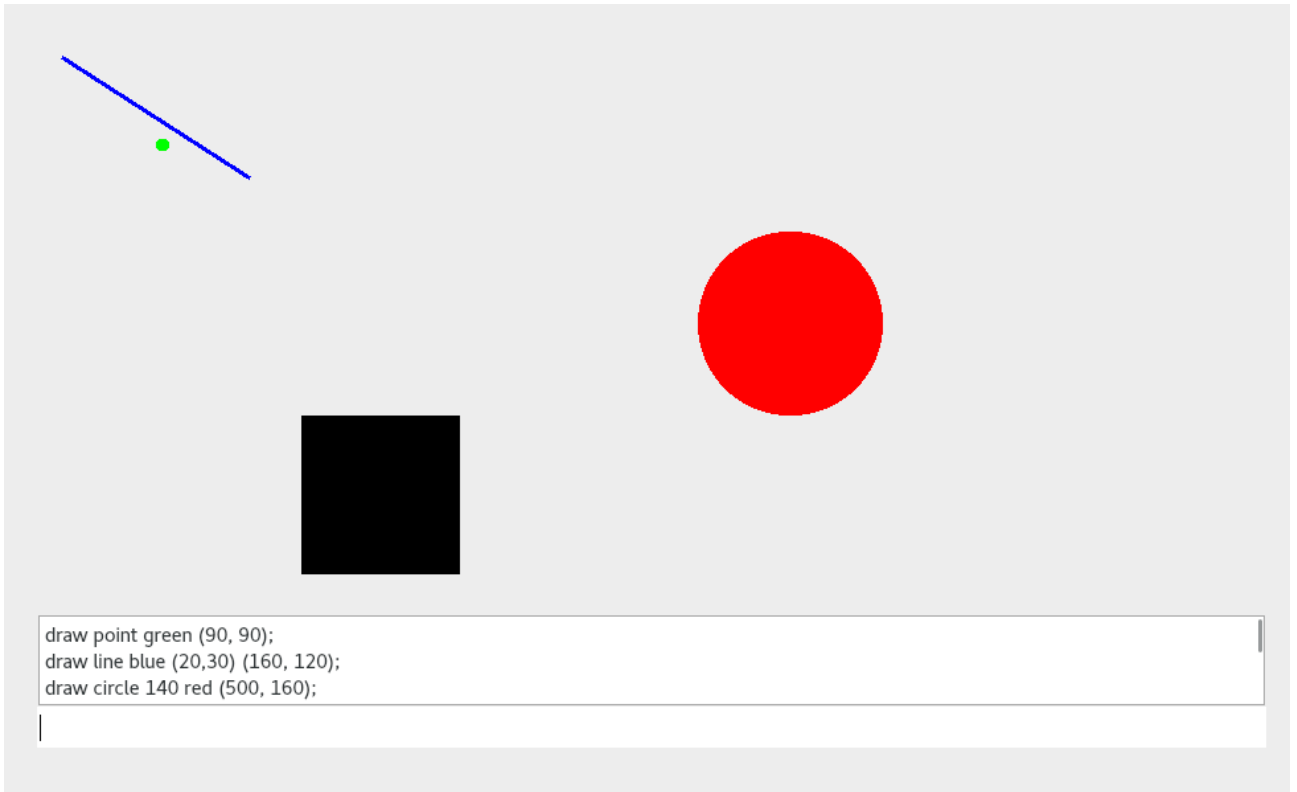
## 4. Przykłady:



```
draw circle 120 red (600, 150);  
draw circle 90 green (400, 300);  
draw circle 60 blue (150,200);
```



```
when ( x>0) { draw line black (60*x, 20*x) (60*x, 80*x);draw line black (60*x, 80*x) (40*x, 80*x) ; draw line black (40*x, 80*x) (40*x,  
90*x) ; draw line black (40*x, 90*x) (90*x, 90*x) ; draw line black (90*x, 90*x) (90*x, 80*x) ; draw line black (90*x, 80*x) (70*x, 80*x)  
; draw line black (70*x, 80*x) (70*x, 20*x) ; draw line black (70*x, 20*x) (60*x, 20*x); x = x - 1;}
```



## 5. Gramatyka:

Opis gramatyki w notacji EBNF. Symbol startowy to input text.

```
statement = (assign_statement | loop_statement | if_statement | function_literal | function_apply |
draw | clear_statement | exit_statement) ;
assign_statement = identifier, '=', add_expr, ';';
if_statement = 'if', condition, operation_block ;
loop_statement = 'when', condition, operation_block ;
function_literal = 'func', identifier, '(', [identifier_list], ')' block ;
function_apply = identifier, '(' [parameter_list] ')', ';';
clear_statement = 'clear', ';';
exit_statement = 'exit', ';';
return_statement = 'return', expr, ';';
block = '{', {statement}, [return statement] '}' ;
operation_block = '{', {statement}, '}' ;
alpha = ? A-Z or a-z ? ;
non-zero digit = ? 1-9 ? ;
digit = '0' | non-zero digit ;
alphanumeric = alpha | digit ;
natural_number = non-zero digit, {digit} ;
integer = '0' | ['-', natural_number] ;
identifier = (alpha | '_'), {alphanumeric | '_'} ;
mul_operator = '*' | '/' ;
add_operator = '+' | '-' ;
relation_operator = '<' | '<=' | '>' | '>=' ;
eq_operator = '==' | '!=' ;
and_operator = '&&' ;
or_operator = '||' ;
condition = '(', or_expr ')' ;
position = '(', add_expr, ',', add_expr, ')' ;
atom = integer | function_apply | function_literal | identifier ;
predefined_figure = quadrangle | point | circle | line ;
circle = 'circle', size, color, position ;
quadrangle = 'quadrangle', size, color, position ;
point = 'point', color, position ;
line = 'line', color, position, position ;
draw = 'draw', predefined_figure, ';';
size = integer ;
color = 'black' | 'red' | 'blue' | 'green' ;
expr = (add_expr | or_expr) ;
add_expr = mul_expr {add_operator, mul_expr} ;
mul_expr = atom {mul_operator, atom} ;
or_expr = and_expr {or_operator and_expr} ;
and_expr = eq_expr {and_operator eq_expr} ;
eq_expr = rel_expr {eq_operator rel_expr} ;
rel_expr = atom {rel_operator atom} ;
parameter_list = expr {'', expr} ;
identifier_list = identifier {'', identifier} ;
```

## 6. Testy

Testy przeprowadzone zostały dla Lexera, Parsera oraz Generacji drzewa AST.  
Są to testy jednostkowe podzielone na moduły.

### 6.1. Testy Lexera:

- 6.1.1. IDENTIFIERS
- 6.1.2. INTEGERS
- 6.1.3. FUNCTIONS AND BOOLEANS
- 6.1.4. MATHEMATICA OPERATIONS
- 6.1.5. GEOMETRIC PRIMITIVES
- 6.1.6. COLORS
- 6.1.7. OTHERS

### 6.2. Testy Parsera:

- 6.2.1. INTEGERS
- 6.2.2. POSITIONS
- 6.2.3. COLORS
- 6.2.4. IDENTIFIERS
- 6.2.5. ASSIGN STATEMENTS
- 6.2.6. FUNCTION APPLIES
- 6.2.7. CLEAR STATEMENT
- 6.2.8. EXIT STATEMENT
- 6.2.9. DRAW FIGURE

### 6.3. Testy AST

- 6.3.1. AST CLEAR
- 6.3.2. AST EXIT
- 6.3.3. AST LINE
- 6.3.4. AST CIRCLE
- 6.3.5. AST QUADRANGLE
- 6.3.6. AST POINT
- 6.3.7. AST ASSIGN
- 6.3.8. AST IF STATEMENT
- 6.3.9. AST LOOP STATEMENT
- 6.3.10. AST FUNCTION LITERAL
- 6.3.11. AST FUNCTION APPLY

Wszystkie testy wykonują się poprawnie:

*All tests passed (154 assertions in 21 test cases)*