Temat projektu

Język do opisu figur geometrycznych i ich właściwości.Podstawowe typy figur geometrycznych (trójkąt, prostokąt, romb,trapez,koło itd.) są wbudowanymi typami języka. Każdy typ posiada wbudowane metody służące do wyznaczania charakterystycznych dla niego wielkości, np. obwód, pole powierzchni, wysokość, średnica itp.Kolekcja figur tworzy scenę wyświetlaną na ekranie.

Wariant: `Język statycznie typowany z silnym typowaniem`

Język programowania: `Python`

Opis funkcjonalności

Język do opisu figur geometrycznych i ich właściwości umożliwia opisanie i obliczanie charakterystycznych wielkości dla różnych typów figur geometrycznych.

Każdy typ figury geometrycznej jest reprezentowany przez swoje właściwości. Metody wbudowane umożliwiają wyznaczenie charakterystycznych dla danej figury wielkości. Wyniki obliczeń można wyświetlić na ekranie w formie tekstowej.

Kolekcja figur geometrycznych tworzy scenę, na której użytkownik może umieszczać figury i dokonywać na nich różnych operacji, takich jak skalowanie i przenoszenie

Figury geometryczne i ich charakterystyczne wielkości:

- Koło:
    - Promień (r) - odległość od środka koła do dowolnego punktu na jego obwodzie.
    - Średnica (d) - dwukrotność promienia.
- Kwadrat:
    - Bok (a) - długość każdej z czterech równych krawędzi.
    - Przekątna (diagonal) - odległość między dwoma przeciwległymi wierzchołkami.
    - Promień okręgu wpisanego (r) i promień okręgu opisanego (R)
- Prostokąt:
    - Bok krótszy (a) i bok dłuższy (b) - długości dwóch przeciwległych krawędzi.
    - Przekątna (d) - odległość między dwoma przeciwległymi wierzchołkami.
    - Promień okręgu opisanego (R)
- Trójkąt:
    - Bok (a), bok (b) i bok (c) - długości trzech krawędzi.
    - Wysokość (h) - pionowa odległość między jednym z wierzchołków a przeciwległą krawędzią.
    - Promień okręgu wpisanego (r) i promień okręgu opisanego (R)
- Romb:

    - Bok (a) - długość każdej z czterech równych krawędzi.
    - Przekątna (e) i przekątna (f) - długości dwóch przeciwległych przekątnych.
    - Kąt między przekątnymi (α) - kąt pomiędzy przekątnymi, mierzony w stopniach.
    - Promień okręgu wpisanego (r)

- Trapez:

    - Bok (c) i bok (d) - długości dwóch równoległych krawędzi.
    - Wysokość (h) - pionowa odległość między dwoma równoległymi krawędziami.
    - Podstawa mniejsza (a) i podstawa większa (b) - długości dwóch pozostałych krawędzi.
    - Kąt wewnętrzne przy podstawie (α) i (β)
    - Promień okręgu wpisanego (r) i promień okręgu opisanego (R)

- Wielokąt formeny:

    - Bok (a) - długość każdej z krawędzi.
    - Liczba boków (n) - liczba krawędzi w wielokącie.
    - Kąt wewnętrzny (α) - kąt pomiędzy dwoma sąsiednimi krawędziami
    - Promień okręgu wpisanego (r) i promień okręgu opisanego (R)

- Shape, jest to figura nadrzędna, która będzie zawierała metody, które można wyznaczyć i wykonać na wszystkich figurach geometrycznych:

    - Pole powierzchni (area)
    - Obwód (perimeter)
    - Przesuń o wektor (move)

Kolekcja figur służąca do wyświetlania:

- Płótno (canvas)
    - dodawanie elementu do kolekcji (push)
    - usuwanie ostatnio dodanego elementu z kolekcji (pop)
    - wyświetlanie kolekcji (display)

Założenia dotyczące programu:

- Każda instrukcja musi być zakończona znakiem `;`
- Program musi zawierać funkcję `main()`, która jest funkcją startową
- Program składa się z bloków funkcji, które zawarte są między znakami `{`, `}`

Wymagania funkcjonalne

1. Typowanie statyczne, silne typowanie, Mutowalność
2. Funkcję mogę być wywoływane rekursywnie
3. Zmienne widoczne są jedynie w blokach, poza nimi już nie
4. Rzutowanie wartości liczbowych, ucięcie cyfr po przecinku
5. W wywołaniach funkcji typy, przekazujemy przez referencję

Wymaganie niefunkcjonalne

1. Język powinien działać na różnych platformach i systemach operacyjnych takich jak Windows, Linux i macOS
2. Język powinien być dobrze udokumentowany, wszystkie jego funkcjonaliści i sposób korzystania będzie opisany w dokumentacji
3. Język działa w sposób deterministyczny, zawsze zwraca te same wyniki
4. Zapewniamy bezpieczeństwo na poziomie, czytania kodu znak po znaku, ustawiając odpowiednie limity na zmiennych

Semantyka

1. Typy danych:

- proste
  - `int` - typ liczby całkowita,
  - `dec` - typ zmienno przecinkowy (liczba dziesiętna - decimal)
  - `bool` - wartość logiczna (prawda/fałsz)
- złożone
  - `String` - ciąg znaków
  - `Shape` - figura, jest to typ nadrzędny, dla pozostałych figur
  - `Circle(x, y, value)` - koło
    - pierwsze dwa argumenty, to współrzędne punktu początkowego, mogą być typu int lub flaot
    - jako argumenty podajemy promień koła, promień może być typu int lub float z wartością dodatnią.
  - `Square(x, y, a)` - kwadrat
    - pierwsze dwa argumenty, to współrzędne punktu początkowego, mogą być typu int lub flaot
    - jako argumenty podajemy bok kwadratu, bok może być typu int lub float z wartością dodatnią.
  - `Rectangle(x, y, a, b)` - prostokąt
    - pierwsze dwa argumenty, to współrzędne punktu początkowego, mogą być typu int lub flaot
    - jako argumenty podajemy boki kwadratu, boki mogą być typu int lub float z wartością dodatnią
  - `Triangle(x, y, a, b, alfa)` - trójkąt
    - pierwsze dwa argumenty, to współrzędne punktu początkowego, mogą być typu int lub flaot
    - jako argumenty podajemy dwa boki trójkąta i kąt pomiędzy nimi, boki mogą być typu int lub float z wartością dodatnią, natomiast kąt jest typu int z zakresu od 0 do 180.
  - `Rhomb(x, y, a, b, alfa)` - romb
    - pierwsze dwa argumenty, to współrzędne punktu początkowego, mogą być typu int lub flaot
    - jako argumenty podajemy bok rombu i kąt pomiędzy nimi, bok może być typu int lub float z wartością dodatnią, natomiast kąt jest typu int z zakresu od 0 do 180.
  - `Trapeze(x, y, a, b, alfa, beta)` - trapez

        - pierwsze dwa argumenty, to współrzędne punktu początkowego, mogą być typu int lub flaot
        - jako argumenty podajemy dwie podstawy trapezu i kąty przy dłuższej podstawie, boki mogą być typu int lub float z wartością dodatnią, natomiast kąty są typu int z zakresu od 0 do 90.
    - `Polygon(x, y, a, n)` - wielokąt foremny
        - pierwsze dwa argumenty, to współrzędne punktu początkowego, mogą być typu int lub flaot
        - jako argumenty podajemy bok i ilość boków, bok może być typu int lub float z wartością dodatnią, natomiast ilość boków może być typu int o wartości co najmniej 3.
    - `Canvas` - kolekcja figur
        - do kolekcji możemy dodawać figury `push(shape)`
        - usuwanie elementu z kolekcji `pop()`
        - wyświetlanie kolekcji `display()`

2. operatory arytmetyczne:

- `+` - dodawanie
- `-` - odejmowanie
- `*` - mnożenie
- `/` - dzielenie

3. operatory logiczne:

- `and` - koniunkcja
- `or` - alternatywa
- `not` - negacja

4. operatory porównania:

- `>` - większy
- `>=` - większy równy
- `<` - mniejszy
- `<=` - mniejszy równy
- `==` - równy
- `!=` - nierówny

5. instrukcja warunkowa:

- `if`:

```
if (warunek){

} else if (warunek){

} else {

};
```

6. pętle

- `while` - pętla warunkowe

```
while (warunek) {

};
```

- `for` - pętla iterująca po elementach kolekcji

```
Canvas canvas = Canvas();
for ( Shape shape : canvas){

};
```

7. Funkcje. Jeżeli funkcja zwraca wartość musi rozpoczynać się od typu, który zwraca, natomiast, jeżeli funkcja nic nie zwraca musi on zaczynać się od słowa `def`

```
def Rectangle nazwa( int a, dec b){
    Circle circle = Circle(a);
    print(circle.area());
    return Rectangle(a, b);
};

def nazwa(Rectangle r){
    print(r.area());
};
```

8. Komentarze, umożliwiamy komentarze w jednej linii

```
# to jest komentarz
```

9. Deklaracja zmiennych, umożliwiamy przypisywanie nowej wartości do zmiennych

```
bool isSquare = False;
bool isCircle = True;
bool isRect = isSquare;
isCircle = isRect;
```

10. instrukcja print, służąca do wypisywania tekstu na ekranie

```

```

```
Triangle t = Triangle(3, 4, 55);
print(t.area());
```

Tokeny

```
ADD = "+"
SUBTRACT = "-"
MULTIPLY = "*"
DIVIDE = "/"

ASSIGN = "="

AND = "and"
OR = "or"
NOT = "not"

EQUAL = "=="
NOT_EQUAL = "!="
GREATER = ">"
LESS = "<"
GREATER_EQUAL = ">="
LESS_EQUAL = "<="

COMMENT = "#"

INTEGER = "int"
DECIMAL = "dec"
BOOL = "bool"

BOOL_TRUE = "True"
BOOL_FALSE = "False"

STRING = "String"
STRING_QUOTE = "\""

SHAPE = "Shape"
CIRCLE = "Circle"
SQUARE = "Square"
RECTANGLE = "Rectangle"
TRIANGLE = "Triangle"
RHOMB = "Rhomb"
TRAPEZE = "Trapeze"
POLYGON = "Polygon"
CANVAS = "Canvas"

SEMICOLON = ";"
COLON = ":"
COMMA = ","
DOT = "."
```

```
FUNCTION = "def"
RETURN = "return"

START_CURLY = "{"
STOP_CURLY = "}"
START_ROUND = "("
STOP_ROUND = ")"
START_SQUARE = "["
STOP_SQUARE = "]"

IF = "if"
ELSE = "else"

WHILE = "while"
FOR = "for"

UNDEFINED = "undefined"
IDENTIFIER = "identifier"
EOF = "eof"
```

Gramatyka

```
program             = {fun_declaration};
argument_dec        = type, identifier;
argument_list       = argument_dec, {',', argument_dec};
fun_declaration     = "def", [type], identifier,
                    '(', [argument_list], ')', block;

function_call       = identifier, '(', [expression,
                    {',', expression}], ')';
method_call         = expression, '.', function_call;
cast                = '(', ("int" | "dec"), ')', factor;

return_statement    = "return", expression, ';';
if_statement        = "if", "(", expression, ")", block,
                    ["else", block] ;
while_statement     = "while", '(', logical_expression, ')', block;
iterate_statement   = "for", '(', type, identifier, ':',
                    expression, ')', block;
declaration         = type, identifier, ['=', expression], ';';
assignment          = identifier, '=', expression, ';';
block               = '{',  {statement}, '}';
statement           = assignment | if_statement | while_statement
                    | iterate_statement | declaration
                    | identifier | sub_expression | logical_expression
                    | function_call | method_call | '(', statement, ')',
';';

logical_expression  = or_expression | "not" logical_expression;
```

```
or_expression       = and_expression, {or_operator, and_expression};
and_expression      = relative_expression,
                    {and_operator, relative_expression};
relative_expression = sub_expression, {relative_operator, sub_expression};
sub_expression      = mul_expression, {subtract_operator, mul_expression};
mul_expression      = factor, {multiply_operator, factor};
factor              = ["-" | "not"], number | identifier | string
                    | function_call | method_call
                    | cast | '(', logical_expression, ')';

subtract_operator   = '+' | '-';
multiply_operator   = '*' | '/';
relative_operator   = '>' | '>=' | '<' | '<=' | '==' | '!=';
or_operator         = 'or';
and_operator        = 'and';
access_operator     = '.';

type                = simple_type | complex_type;
simple_type         = "int" | "dec" | "bool";
complex_type        = "Shape" | "Circle" | "Square"
                    | "Rectangle" | "Triangle"| "Rhomb"
                    | "Trapeze" | "Polygon" | "Canvas";

identifier          = letter, {letter | digit};

string              = '"', {chars}, '"';
comment             = '#', {chars}, '\n' | '\r' | '\n\r' | '\n\n';
chars               = letter | digit;

number              = integer | decimal;
integer             = zero | (not_zero_digit, {digit});
decimal             = integer, '.', digit, {digit};
bool                = "True" | "False";

digit               = zero | not_zero_digit;
not_zero_digit      = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
zero                = '0';
letter              = #'[a-z]' | #'[A-Z]' (*inne znaki specjalne*);
```

Obsługa błędów:

- Napotkanie błędu powoduje wyświetlenie odpowiedniego komunikatu użytkownikowi.

Komunikat składa się z numera wiersza lini oraz numera kolumny w którym dany błąd wystąpił, następnie wyświetlana jest treść komunikatu.

Napotkanie błędu nie powoduje zakończenia pracy programu, dopiero napotkanie błędu krytycznego, kończy działanie problemu.

Błędy podzielone są na trzy kategorie, w zależności od tego w którym miejscu wystąpią, wyróżniamy:

- błędny leksykalne

Kod:

```
def main(){
    int x = 123123123123;
}
```

Komunikat błędu:

```
Error [2, 13]: Type 'int' value out of range.
```

Kod:

```
def main(){
    int x = 123$123;
}
```

Komunikat błędu:

```
Error [2, 16]: Invalid character '$'.
```

- błędy składniowe

Kod:

```
def main() {
   printf("HI")
}
```

Komunikat błędu:

```
Error [2, 16]: Invalid token, expected ';' before '}'.
```

Kod:

```
def main() {
   int x 0;
}
```

Komunikat błędu:

```
Error [2, 9]: Missing ';' before '0'.
```

Kod:

```
def main() {
   int x = 1 + (2  2);
}
```

Komunikat błędu:

```
Error [2, 19]: Missing operator before '2'.
```

- błędy semantyczne

Kod:

```
int a = 1;
bool b = False;
int c = a + b;
```

Komunikat błędu:

```
Error [3, 11]: Invalid conversion from 'bool' to 'int'.
```

Kod:

```
int a = 1;
bool b = 0;
int c = a / b;
```

Komunikat błędu:

```
Error [3, 11]: Division by zero.
```

Kod:

```
def pow(int a){
    return a * a;
}
def main(){
    pow(True);
}
```

Komunikat błędu:

```
Error [5, 0]: Invalid 1. argument conversion from 'bool' to 'int'.
```

Sposób Uruchomiania

Program można uruchomiać za pomocą programu napisanego w języku python podając odpowiednie do argumenty przy jego wywołaniu.

Do poprawnego działania wymagany jest zainstalowany `Python` w wersji `3.9.0`, a także zainstalowane wymagane biblioteki znajdujące się w pliku `requirements.txt`.

Sposób instalacji: `pip3 install -r requirements.txt`.

Przykładowy sposób uruchomienia:

```
$ ./run.py code.txt
```

Wynik programu:

- wyświetlenie się nowego okna, na którym narysowane są figury
- wypisanie tekstu w konsoli

Testowanie

Projekt zawiera testy jednostkowe oraz testy integracyjne, sprawdzające poprawność działającego kodu jak i programu.

Wykorzystywana biblioteka: `pytest`

Testy podzielone sa na kilka kategorii:

- działanie analizatora leksykalnego, czy podany kod został przetworzony, na poprawną listę tokenów, oczywiście będą sprawdzane też przykłady wymienione w sekcji `Obsługa błędów`:

Kod:

```
int a = 10 + 5 * 2;
```

Przykładowy Wynik:

```
<int, type>
<a, identifier>
<=, symbol>
<10, integer>
<+, symbol>
<5, integer>
<* , symbol>
<2, integer>
<;, symbol>
```

Kod:

```
def main(){
    int x = 123$123;
}
```

Przykładowy Wynik:

```
Error [2, 16]: Invalid character '$'.
```

- działanie analizatora składniowego, czy listę tokenów został przetworzony na poprawne drzewo składniowe, oczywiście będą sprawdzane też przykłady wymienione w sekcji `Obsługa błędów`
- Test programu zawierającego błąd składniowy

```
def main() {
    int x = 5;
    int y = 10;
    if (x < y {
        print("x is less than y");
    }
}
```

Oczekiwany wynik:

```
Error [4, 14]: Invalid token, expected ')' before '{'.
```

- działanie analizatora semantycznego, czy program poprawnie działa, czy rzucane są odpowiednie wyjątki, oczywiście będą sprawdzane też przykłady wymienione w sekcji `Obsługa błędów`

```
def divide(a: int, b: int) {
    return a / b;
}
```

Oczekiwany wynik:

```
Error [4, 14]: Invalid return type, expected non-return type
```

- Do testów zostaną załączone oczywiście wszystkie przykładowe fragmenty kodu podane w sekcji `przykłady wykorzystania języka`, a także te z sekcji `Obsługa błędów`

Biblioteki

- `matplotlib` - biblioteka służąca do wyświetlania figur geometrycznych i nie tylko na ekranie
- `pytest` - biblioteka służąca do pisania testów w języku python
- `typing` - biblioteka umożliwiająca definiowanie typów zmiennych i argumentów funkcji w celu poprawy czytelności i jakości kodu
- `math` - biblioteka zawierająca funkcje matematyczne
- `enum` - biblioteka umożliwiająca definiowanie i wykorzystywanie typów wyliczeniowych

Sposób realizacji

- Analizator leksykalny - Odpowiedzialny za przerobienie kodu źródłowego na sekwencję tokenów
- Analizator składniowy - Sprawdza, czy sekwencja tokenów odpowiada zdefiniowanemu w gramatyce języka programowania i tworzy z nich drzewo składniowe.
- Analizator semantyczny - Wykonuje analizę semantyczną na drzewie składniowym, sprawdzając poprawność wykorzystania zmiennych, typów danych i wyrażeń.

przykłady wykorzystania języka

- definiowanie typów

```
def main() {
    int a = 2;
    int b = a;
    dec c = 2.95;
    bool d = False;
```

```
    String string = "Hello World!";
    dec negative = -1.5;

    int x = a * (b + (int) c);
}
```

- wyznaczanie pola, obwodu, przekątnej, przesuwanie o wektor figury

```
def main() {
    Triangle t = Triangle(0, 0, 3, 4, 55);
    print(t.area());

    Rectangle r = Rectangle(0, 0, 5);
    print(r.perimeter());

    Square s = Square(0, 0, 5);
    print(s.diagonal());

    s.move(2, 3);
}
```

- dodawanie/usuwanie figur do/z kolekcji i wyświetlanie ich

```
def main(){
    Canvas c = Canvas();
    Circle circle = Circle(0, 0, 6);
    Polygon p = Polygon(10, 20, 5, 6);
    Rhomb r = Rhomb(6, 20, 4, 60);
    Trapeze t = Trapeze(8, 9, 6, 8, 60, 90);
    c.push(circle);
    c.push(p);
    c.push(r);
    c.push(t);
    c.pop();
    c.display();
}
```

- pętle, instrukcje warunkowe i iterowanie po kolekcji

```
def main(){
    Rectangle r = Rectangle(0, 0, 5);
    dec r_per = r.perimeter();

    Square s = Square(0, 0, 5);
    dec s_per = s.perimeter();

    dec suma = 0;
```

```
    if ( s.area() != 2.5 or r_per == 2) {
        print(s.diagonal());
    }

    if (r_per > s_per) {
        print("Prostokąt większy");
    } else {
        suma = r_per + s_per;
        print(suma);
    }

    int i = 0;
    Canvas c = Canvas();
    while ( i <= 20 ) {
        c.add(Cricle(i, i, i));
        i = i - 1;
    }

    for ( Shape shape : c) {
        shape.move(3, 10);
    }

    c.display();
}
```

- definiowanie funkcji i rekursywne wywołanie

```
def Square gasket(int x, int y, dec dim, Canvas c){
    if ( dim < 8) {
        return Square(x, y, dim);
    } else {
        dec = new_dim = dim / 2;
        gasket(x, y, new_dim);
        gasket(x + new_dim, y, new_dim);
        gasket(x + new_dim, y + new_dim, new_dim);
    }

}

def Triangle getTriangle(int x, int y, int height, int width){
    return Triangle(x, y, width / 2, height, 90);
}

def printInformation(Shape shape){
    # pole
    print(shape.area());
    # obwód
    print(shape.perimeter());
}

def main(){
```

```
    Canvas c = Canvas();
    c.add(gasket(0,0, 248));
    c.display();
    Triangle t = getTriangle(0, 10, 20, 30);

    printInformation(t);
}
```

Autor

Mateusz Brzozowski, 310608