Interpreter języka obiektowego - dokumentacja wstępna

Tomasz Potomski 298898

24 stycznia 2022

1 Wstęp

Celem projektu jest stworzenie prostego języka skryptowego inspirowanego językiem Python. Język posiadać będzie typ wbudowany int, oraz string używany jako stała tekstowa w wywoływaniach funkcji print. Oprócz tego język umożliwiać będzie tworzenie klas posiadających atrybuty typu int oraz metody. Wymagane jest, aby każdy program miał funkcje main, która jest funkcją początkową, a wszystkie klasy były zdefiniowane nad nią.

2 Opis funkcjonalności

2.1 Typy wbudowane

Sposób działania zmiennych inspirowany jest językiem Python - przy inicjacji zmiennych nie deklarujemy typu zmiennej. W każdej chwili możemy przypisać wartość innego typu.

2.1.1 int

```
variable = 1;
```

2.1.2 string

Typ string występować będzie tylko jako stała tekstowa w wywołaniach funkcji print

```
print("some text");
```

2.2 Operatory

- operator przypisania "="

```
a = 1;
```

2.2.1 Arytmetyczne

- dodawanie "+"

```
a = 1;
b = 2;
c = a + b;
d = 1 + 2;
e = a + 5;
```

- odejmowanie -"

```
a = 1;
b = 2;
c = b - a;
d = 8 - 2;
e = a - 2;
```

- mnożenie "*"

```
a = 1;
b = 2;
c = b * a;
d = 8 * 2;
e = a * 5;
```

- dzielenie "/"

```
a = 1;
b = 2;
c = b / a;
d = 8 / 2;
e = a / 5;
```

Niedozwoloną operacją jest dzielenie przez 0. Interpreter zwróci wyjątek jeśli w programie będzie takowe dzielenie.

```
a = 5 / 0;
```

2.3 inne porównania

- operatory równości "==", "!="

```
a == b;
a != b;
```

- operator większości ">"

```
a > b;
```

- operator mniejszości "<"

```
a < b;
```

- operator większy-równy ">="

```
a >= b;
```

- operator mniejszy-równy "<="

```
a <= b;
```

2.4 operatory logiczne

- and

```
a>b and b<c;
```

- or

```
a>b and b<c;
```

- operator negacji

```
not a;
!a;
```

2.5 Funkcje wbudowane

- print - wypisywanie na wyjście

```
a = 1
print(a)
#output: 1

print("some text")
#output: some text
```

2.6 Instrukcje sterujące

- instrukcja warunkowa

```
if(a<b)
{
    #do something
}
```

- pętla

```
while(i<a)
{
    #do something
}
```

Możliwe jest korzystanie z instrukcji **continue** oraz **break** w bloku pętli **while**. Gdy istrukcje te są używane poza pętlą nie wywołają efektu.

2.7 klasa

Język umożliwia tworzenie klas wraz z atrybutami i metodami Tworzenie klasy:

```
class ClassName
{
}
```

Można zdefiniować atrybuty wewnątrz definicji danej klasy:

```
class ClassName
{
    x = 1;
}
```

Można również zdefiniować metody wewnątrz definicji danej klasy. Można odwoływać się do atrybutów klasy za pomocą słowa kluczowego śelf":

```
class ClassName
{
    count = 1;
    def counter()
    {
        print(self.count);
        self.count = self.count + 1;
    }
}
```

Można również określić argumenty który przyjmuje metoda i odwoływać się do nich wewnątrz metody:

```
class ClassName
{
    def printer(message)
    {
        print(message);
    }
}
```

Tworzenie obiektu i odwoływanie się do metod danej klasy odbywa się w następujący sposób:

```
startValue = 5;
classObject = ClassName();
classObject.counter();
```

Wszystkie atrybuty danej klasy są prywatne, więc jeśli chcemy się do nich dostać potrzebujemy funkcji settera.

2.8 Nawiasy i kolejność wykonywania działać

Operacje logiczne i arytmetyczne będą miały zachowaną standardową kolejność wykonywania działań. Będzie można zmienić kolejność wykonywania działań za pomocą nawiasów "("i ")".

3 Formalny opis gramatyki

```
start                    : {classDef} functionDef
classDef                 : class id classDefinitionBlock
classDefinitionBlock     : leftBracket {varAssigment} {functionDef} rightBracket
varAssigment             : id assign intLiteral
functionDef              : def id parameters block
parameters               : leftParen [id {comma id}] rightParen
block                    : leftBracket {statement} rightBracket
statement                : ifStatement
                         | whileStatement
                         | returnStatement semicolon
                         | idAssigmentOrFuncCall semicolon
                         | continue semicolon
                         | break semicolon

ifStatement              : if leftParen condition  rightBracket block {elif block} [else block]
condition                : expression
whileStatement           : while leftParen condition rightParen block
returnStatement          : return expression
idAssigmentOrFuncCall    : id [dot id]
                         | assignment
                         | funcCall
assignment               : id [dot id] assign expression
funcCall                 : id [dot id] arguments
arguments                : leftParen [expression {comma expression}] rightParen
expression               : andExpression {or andExpression}
andExpression            : logicNegationExpression {and logicNegationExpression}
logicNegationExpression  : {negation relationExpression}
relationCondition        : addSubExpression [(greater|greaterEqual|less|lessEqual|equal|notequal) addSubExpressi
addSubExpression         : multiplicationExpression {(add|minus) multiplicationExpression}
multiplicationExpression : unaryExpression {(mult | div) unaryExpression}
unaryExpression          : {minus primaryExpression}
primaryExpression        : id
                         | funcCall
                         | stringLiteral
                         | intLiteral
                         | leftParen expression rightParen

variable                 : [id dot] id
id                       : ["_"] letter {digit | letter}
letter                   : "a"..."z"
                         | "A"..."Z"
digit                    : "0"..."9"

literal                  : intLiteral
                         | stringLiteral


class                    : "class"
def                      : "def"
if                       : "if"
```

```
elif                    : "elif"
else                    : "else"
leftParen               : "("
rightParen              : ")"
leftBracket             : "{"
rightBracket            : "}"
comma                   : ","
or                      : "or"
and                     : "and"
equal                   : "=="
notequal                : "!="
greater                 : ">"
greaterEqual            : ">="
less                    : "<"
lessEqual               : "<="
negation                : "!"
                        | "not"
dot                     : "."
add                     : "+"
minus                   : "-"
mult                    : "*"
div                     : "/"
semicolon               : ";"
continue                : "continue"
break                   : "break"

intLiteral              :(0|[1-9][0-9]*)

String                      :\"([^\n\"]|\\.)*\"
```

4 Analiza wymagań

4.1 Wymagania funkcjonalne

- Program poprawnie odczytuje, parsuje i analizuje plik źródłowy ze skryptem, wykonuje wszystkie instrukcje, a w razie potrzeby zwraca potrzeby zwraca wykryte błędy
- Program umożliwia tworzenie własnych klas wraz z atrybutami i metodami i używania ich w skrypcie
- Program umożliwia wykonywanie operacji arytmetycznych, logicznych i wypisywanie ich na standardowe wyjście

5 Wymagania niefunkcjonalne

- Program jest intuicyjny w użyciu - w razie błędnego użycia program powinien wyświetlać informację o będzie i przyjmowanych argumentach
- Komunikaty o błędach powinny jasno wskazywać miejsce popełnionego błędu

6 Opis projektu

6.1 Stos technologiczny i uruchamianie

Projekt będzie pisany w języku c++ z myślą o kompilacji i uruchamianiu na systemie linuxowym. Do kompilacji będzie wykorzystywany kompilator clang and program cmake. Projekt testowany jest na systemie Pop OS 21.04
Projekt będzie aplikacją konsolową, uruchamianą z argumentem będącym ścieżką do pliku źródłowego skryptu. W celu zapisania wyników potrzebne będzie przekierowanie wyjścia do pliku.
Kompilacje można wykonać za pomocą komendy

cmake

w folderze projektu. Uruchamianie interpretera wykonujemy za pomocą komendy

./interpreter <file_name>

będąc w folderze z plikiem wynikowym kompilacji.

6.2 Moduły

Projekt będzie podzielony na moduły 5 główne moduły odpowiedzialne za proces translacji.

1. Lexer - moduł odpowiedzialny będzie za analizę leksykalną danych wejściowych tj. będzie rozbijał dane wejściowe skryptu na tokeny, z których korzystać będzie Parser.

2. Parser - moduł odpowiedzialny za analizę składniową. Będzie korzystał z Tokenów utworzonych za pomocą Lexera. Moduł ten będzie sprawdzał poprawność ułożenia tokenów z gramatyką języka. Struktury gramatyczne które Parser zaakceptuje będą tworzyć drzewo składniowe.

3. Moduł obsługi błędów - moduł ten będzie współpracował ze wszystkimi powyższymi modułami i będzie zapewniał obsługę i prezentację błędów.

4. Moduł odpowiedzialny za wykonanie instrukcji zawartych w drzewie powstałym w Parserze. Moduł ten zrealizowany będzie za pomocą wzorca projektowego wizytator.

Główną częścią projektu będą klasy związane z drzewem składniowym. Drzewo to będzie złożone z obiektów reprezentujących określone symbole i przechowujące odpowiednie wartości z pliku źródłowego. Klasą bazową dla wszystkich elementów w drzewie będzie ***Node***

7 Przykłady poprawnych skryptów

7.1 Przykład 1

```
class Counter
{
    count = 0;

    def execute()
    {
        self.count = self.count + 1;
        return self.count;
    }
}
def main()
{
    a = 0;
    counter = Counter();
    while(a<100)
    {
        b = counter.execute();
        print(b);
        a = a + 1;
    }
}
```

7.2 Przykład 2

```
class Fibonacci
{
    a = 1;
    b = 1;

    def CalculateNextFibonacciNumber()
    {
        c = b;
        self.b = self.a + self.b;
        self.a = c;
```

```
        return self.b;
    }
}

def main()
{
    maxValue = 100000;
    fibonacci = Fibonacci();
    fibonacciNumber = 0;
    while(fibonacciNumber < maxValue)
    {
        fibonacciNumber = fibonacci.CalculateNextFibonacciNumber();
    }
    print("First fibonacci number higher than 100000 is ");
    print(fibonacciNumber);
}
```

8 Raport z testów

8.1 Testy podstawowych funkcjonalności

- \textbf{Nazwa pliku:} test-zmienne-i-wyrazenia-1.txt

```
def main()
{
    x = 1;
    y = 1;
    print(x);
    print("\n");
    print(y);
    print("\n");
    z = x + y;
    print(z);
    print("\n");
    print(x + y);
}
```

Rezultat:

1
1
22

- \textbf{Nazwa pliku:} test-zmienne-i-wyrazenia-2.txt

```
def main()
{
    x = 2;
    y = 2;

    z = (x + y)*x;
    print(z);
    print("\n");
    print(x+y*x);
}
```

Rezultat:

8
6

- **Nazwa pliku:** test-zmienne-i-wyrazenia-3.txt

```
def main()
{
    x=0;
    while(1 == 1)
    {
        x = x+1;
        if(x == 5)
        {
            break;
        }
    }
    y=0;
    while(y<=5)
    {
        y = y+1;
    }
    print(x);
    print("\n");
    print(y);
}
```

Rezultat:
5
6

- **Nazwa pliku:** test-zmienne-i-wyrazenia-4.txt

```
def main()
{
    x = 0;
    if(x == 0 or x/0)
    {
        print("dzielenie przez zero nie jest sprawdzane \n");
    }
    if(x < 0 and x/0)
    {
        print("dzielenie przez zero nie jest sprawdzane \n");
    }
    else
    {
        print("jestem w elsie \n");
    }
    a=0;
    while(a<60 and a<30 or a == 30)
    {
            a = a+1;
    }
    print(a);
}
```

Rezultat:
dzielenie przez zero nie jest sprawdzane
jestem w elsie
31
Komentarz: jeśli w warunku AND pierwszy warunek nie jest spełniony, to drugi nie jest sprawdzany. Podobnie w warunku OR gdzie jeśli pierwszy jest prawdziwy to drugi nie jest sprawdzany.

8.2 Testy klas

- **Nazwa pliku:** test-klasy-1.txt

```
class Fibonacci
{
    a = 1;
    b = 1;

    def CalculateNextFibonacciNumber()
    {
        c = self.b;
        self.b = self.a + self.b;
        self.a = c;
        return self.b;
    }

}

def main()
{
    fibonacci = Fibonacci();
    maxValue = 100;
    fibonacciNumber = 0;
    fibonacciNumber = fibonacci.CalculateNextFibonacciNumber();
    while(fibonacciNumber < maxValue)
    {
        fibonacciNumber = fibonacci.CalculateNextFibonacciNumber();
        if(fibonacciNumber < 20)
        {
                continue;
        }
        print(fibonacciNumber);
        print("\n");
    }
}
```

Rezulat:
21
34
55
89
144

- **Nazwa pliku: test-klasy-2.txt**

```
class Max
{
    def FindMax(x,y)
    {
            if(x>y)
            {
                    return x;
            }
            else
            {
                    return y;
            }
```

```
    }
}

def main()
{
    a = Max();
    print(a.FindMax(10, 20));
    print("\n");
    print(a.FindMax(20, 10));
}
```

Rezultat:
20
20

8.3 Testy wykrywania błędów

- **Nazwa pliku:** test-blad-parsera.txt

```
def main
{
    a = 5;
    y = 2
}
```

Rezultat:
Unexpected token: LeftBracket (Line: 2, Pos: 0)
expected: LeftParen
Termination...

- **Nazwa pliku:** test-brak-maina.txt

```
def cos()
{
    x= 0;
    y=0;
}
```

Rezultat:
There is no main function
Termination...

- **Nazwa pliku:** test-niezadeklarowana-zmienna.txt

```
def main()
{
    x = 2;
    y = x + z;
}
```

Rezultat:
No variable named z at line 3
Termination...

- **Nazwa pliku:** zła-liczba-argumentow.txt

```
class Min{
    def FindMin(x,y)
    {
        if(x>y)
        {
            return y;

        }
        else{
            return x;
        }
    }
}

def main()
{
    min = Min();
    min.FindMin(1);
}
```

Rezultat:
bad number of arguments: expected: 2 but given 1 at line 17
Termination...

8.4 Złożony test

Nazwa pliku: test.txt

```
class Fibonacci
{
    a = 1;
    b = 1;

    def CalculateNextFibonacciNumber()
    {
        c = self.b;
        self.b = self.a + self.b;
        self.a = c;
        return self.b;
    }
    def Max(x,y)
    {
            if(x>y)
            {
                    return x;
            }
            else
            {
                    return y;
            }
    }
    def CountTo20()
    {
            x=0;
            while(1==1)
            {
                    x=x+1;
```

```
                    if(x==20)
                    {
                            return x;
                    }

            }
    }
}

def main()
{
    print("test \n");
    fibonacci = Fibonacci();
    x = fibonacci.CountTo20();
    print(x);
    a = 1;
    while(a<60 and a<30 or a == 30)
    {
            a = a+1;
            if(a == 25)
            {
                    break;
            }
    }
    print(a);
    print("\n");
    maxValue = 1000;
    fibonacciNumber = 0;
    fibonacciNumber = fibonacci.CalculateNextFibonacciNumber();

    while(fibonacciNumber < maxValue)
    {
        fibonacciNumber = fibonacci.CalculateNextFibonacciNumber();
        if(fibonacciNumber < 20)
        {
                continue;
        }
        print(fibonacciNumber);
        print("\n");
    }
}
```

Rezultat:

test
2025
21
34
55
89
144
233
377
610
987
1597