TKOM - dokumentacja wstępna

Daniel Lipniacki 304067

Opis projektu

Projekt ma na celu wykonanie interpretera prostego języka. Język ma być podobny do Pythona, wyposażony w typ Object, numeric i string. Do instancji obiektów powinno być możliwe dynamiczne dodawanie atrybutów. Można będzie odpytywać o istnienie atrybutu o danej nazwie. Język będzie posiadał zmienne z zasięgiem, pętle i instrukcje warunkową.

Zakładana funkcjonalność

- Odczytanie, sprawdzenie i wykonanie skryptów zapisanych w plikach tekstowych
- Raportowanie błędów wykonania i błędów w plikach
- Typowanie dynamiczne
- Możliwość definiowania typów:
  - numeric
  - string
  - Object
- Wyrażeń matematycznych uwzględniajace priorytet operatorów ((),*,/,+,-)
- Zmienne maja swój zakres
- Istnieje możliwość tworzenia komenatrzy
- Wyrażeń logicznych uwzględniajace priorytet operatorów ((),==,||,&&, and, or,!=)
- Zapewnienie możliwość użycia instrukcji warunkowej i pentli `for` i `while`
- Funkcja `print`, wypisująca informacje podane przez użytkownika
- Możliwość definiowania własnych funkcji oraz ich późniejszego wywoływania w skryptach
- Pliki powinny być kodowane w UTF-8
- Biblioteka sztandarowa złożona z:
  - `print`
  - `input`

    - `inputNumber`
    - `numeric`
    - `string`
    - `Object`
    - `exit`
- Każdy wyrażenie będzie zakończony `;`
- Możliwość odpytania o posiadanie danej wartośći przy pomocy `has`
- Program rozpoczyna się od funkcji `main`

Próbki języka

```
## inclizacja zmiennych
numeric1 = 1 ;
numeric2 = 2.3;
str = "to jest napis";
toJestObiekt = Object();

toJestObiekt.nowaWlasciwosc = 1;
# to spowoduje runtime error jak nie ma nowaWlasciwosc
print(toJestObiekt.innaWlasciowos);

#pytanie o to czy taki atrybut istnieje
if toJestObiekt has "nowaWalscioos"  {

}
# 1 - jest takiej właściwości
# 0 - nie ma

# dodawanie funkcji jako atrybutu
printOjb() {

};

toJestObiekt.funkcja = printObj

obiekt = Object();
obiekt.cos = "fa";

# pętla for, iterowanie się po atrybutach obiektu
for pole in obiekt {

}

# intrukcja warunkowa
if cos.wlasciwosc == 11 {
    cos.nowaWlasnosc = "FAFFAA";
} else {
```

```
    cos.innaWlasciowas = "fafafaf";
}

# deklaracja funkcji
nazwaFunkcji(argumenty) {

}
nazwaFunkcji(1, 2, 3, object);

## pentla while
i = 0;
while (i < 10) {
    print(i);
    i = i + 1;
}

## STD
# wypisywanie na ekran
print("tkom");
# pobieranie inputu
string = input();
number = inputNumber();
```

```
==================
Hello world
==================

main () {
    print("Hello World")
}

================
skrypt 1
================


Descript(p) {
    print(person.name, "has")
    for c in p {
        print(c)
    }
}

main() {
    person.Descript = Descript;

    person.Descript(person);
}
```

```
================
skrypt 2
================


main () {
    name = input()
    lastName = input()

    if name == "Daniel" {
        print("Hello Daniel")
    } else {
        print("Hello", name)
    }
}
```

Gramatyka

```
program = { functionDefinition };
functionDefinition = identifier parameters block ";" ;
parameters = "(" [ identifier { "," identifier } ] ")"  ;

ifStatement = "if" "(" condition ")" block [ "else" "if" block ] ["else"
bolck] ;
forLoop = "for" identifier "in" identifier  block ;
while = while "(" condition ")" block ;
returnStatement = "return" [ expresion ] ";" ;
comment = "#" ... ;

block = "{" { statement } "}";

argumentList = { argument "," } ;
argument = expresion ;

statement =  expresion ";" | ifStatement |  forLoop  | whileLoop |
returnStatement  ";" ;

expresion = assignmentExpression ;
assignmentExpression = conditionalExpression { assignmentOperator
expresion } ;

conditionalExpression = andExpression { orOperator andExpression }  ;
andExpression = relationalExpression { andOperator relationalExpression
} ;
relationalExpression = additiveExpression {relationOperator
additiveExpression } ;
additiveExpression = multiplicativeExpression { additionOperator
multiplicativeExpression } ;
```

```
multiplicativeExpression = unaryExpression { multiplicationOperator
unaryExpression } ;
unaryExpression = [notOperator] primaryExpression ;
primaryExpression = (literal | identifier, { ".", identifier } {"("
argumentList ")"} {"has" identifier } ) | "(" expresion ")" ;

notOperator = "!" ;
assignmentOperator = "=" | "+=" | "-=" | "*=" | "/=" ;
additionOperator = "+" | "-" ;
multiplicationOperator = "*" | "/" | "%" ;
orOperator = "or" | "||" ;
andOperator = "and" | "&&" ;
equalOperator = "==" | "!="  ;
relationOperator = "<" | ">" | "<=" | ">=" ;

identifier = letter { digit | letter };

literal = numberLiteral | stringLiteral ;
numberLiteral = nonzeroDigit { digit } [ "." { digit } ] ;
stringLiteral = '"' ? all characters ?'"' | "'" ? all characters ?"'"
;

letter = "a".."z" | "A".."Z" ;
digit = "0".."9";
nonzeroDigit = "1" .. "9";
```

Przewidywane tokeny

```
"while", "if", "{", "}", "(", ")", ".", " " ", " ' ", "else", "<" | ">"
| "<=" | ">=", "==" | "!=", "and" | "&&", "or" | "||" , "*" | "/" | "%",
"+" | "-", "=" | "+=" | "-=" | "*=" | "/=" , "!"
```

Opis modułów

Program będzie złożony z modułów, odpowiedzialny za poszczególne etapy procesu interpretacji:

Moduł Źródła/Pliku

Moduł będzie odpowiedzialny za otwarcie pliku i zarządzanie nim. Będzie umożliwiał czytanie znak po znaku z źródła. Będzie udostępniać odpowiedzi interface (trait). Zostanie zaimplementowane dwa podstowe rodzaje źródła, jedno działające na pliku, drugie działające na STDIN. Powstanie również źródło dla testów.

```
interface {
    get_next_char() char
    get_pos() uint
}
```

Moduł Lexera

Moduł będzie odpowiedzialny za analizę leksykalną, czyli rozbicie znaków na tokeny. Lexer będzie pobierał kolejne znaki ze źródła. W momencie skomponowania tokenu, token będzie zwracany do klienta. Tokeny będą zwracane jeden po drugim na wywołanie metody `get_next_token` . W raz z modułem będzie dostępny enum dostępnych tokenów. Moduł będzie miał połączenie z modułem obłsugi błędów

```
interface {
    get_next_token() Token
    get_current_token() Token
    has_next_token() bool
}
```

Moduł Parsera

Moduł będzie odpowiedzialny za analizę składniową. Będzie pracował wraz z modułem lexera i pobierał z niego tokeny. Jego zadanie to będzie sprawdzanie czy tokeny łączą się w poprawną gramatykę i łączenie ich w drzewo składniowe. W razie problemów będzie go raportował do modułu obłogi błędów,

Moduł analizatora semantycznego

Moduł będzie sprawdzał czy otrzymane drzewo składniowe jest poprawne. Będzie sprawdzał:

- Poprawność używanych identyfikatorów,
- Brak nadpisywania funkcji,
- Zgodność ilości parametrów wywołań funkcyjnych
- Zgodność operacji arytmetycznych i logicznych,

Moduł wykonania

Moduł wykonania będzie odpowiedzialny ze wykonanie dostanego drzewa składniowego w odpowiedni sposób, i zakomunikowanie użytkownikowi wyników wykonania.

Moduł Raportowania błędów

Będzie odpowiedzialny za raportowanie błędów dla użytkownika. Przewiduje trzy rodzaje błędów:

- Runtime
- Składni
- Semantyczny
- Leksykalny

Każdy będzie miał swój własny typ i interfejs.

Biblioteka Standardowa

- `print` - wypisywanie tekstu podanego przez użytkownika na ekran
- `input` - pobieranie danych od użytkownika - pobiera stringa
- `inputNumber` - pobieranie danych od użytkownika - pobiera stringa
- `numeric` - zmienia napis na numeric wartość, dla obiektów jest runtime error
- `string` - zmienia liczbę w napis, dla obiektów wypisuje jsona ich atrybutów
- `Object` - stworzenie nowego pustego obiektu
- `exit` - kończy działanie interpretera

Opis techniczny

Projekt zostanie napisy w języku `Rust`.

Biblioteki

- **Clap** - zostanie użyta do parsowanie command line arguments
- **UTF8-Reader** - biblioteka wspomagająca czytanie plików UTF-8

Testowanie

Testowanie zostanie zrealizowane używając wbudowanych funkcjonalności w Rust i Cargo. Jeżeli okaże się ze jest to nie wystarczające zostanie użyta prawdomównie ta **biblioteka**

Testy będą jednostkowe poszczególnych komponentów, jak i "integracyjne", wykonania danego kodu przez intepreter.

Opis działania

Program będzie aplikacją konsolową, której będzie można przekazać informacje jak ścieżkę do pliku i dodatkowe flagi jako argumenty. Wyniki jak i dodatkowe informacji z każdego etapu będą wyświetlane. Błędy wykonania, leksykalne i składniowe będą raportowane do użytkownika.

```
$ ./intepreter -i plik.pys
```