TKOM

Projekt wstępny

„Język „Python” z typem walutowym” Maciej Paszylka

Opis funkcjonalny

Język będzie umożliwiał definiowanie oraz przetwarzanie zmiennych liczbowych zawierających informację o przypisanych im walutach. Implementacja ma pozwolić na konwersję oraz obliczanie wyników operacji na różnorodnych walutach bez potrzeby ich uprzedniego ujednolicania. Język będzie umożliwiał użytkownikowi definiowanie własnych funkcji oraz przyjmowanych i zwracanych przez nie wartości.

Szczegóły dotyczące języka

Język posiada zaimplementowane następujące typy:

Typy danych:

- typ liczbowy:

    a. z podaniem waluty

```
PLN cash = 5.0;
```

    b. bez podawania waluty

```
var i = 5.0;
```

Analizowany kod pozwala na definiowanie funkcji. Każda z definiowanych funkcji musi zwracać wartość za pomocą funkcji return. Jeśli funkcja zawiera pętlę while lub instrukcję warunkową if to w przypadku, gdy ich bloki zwracają

jakieś wartości to konieczna jest zgodność z typem wartości zwracanej przez return. Zmienne są przekazywane do funkcji poprzez wartość. W język wbudowana jest funkcja print() która umożliwia wypisanie zmiennych, liczb zmiennoprzecinkowych oraz walut. Główną walutą jest EUR. W dodatkowym pliku w utils znajdują się przeliczniki, które później umożliwiają konwersję na inne waluty. Podczas deklaracji zmiennej walutowej możemy do niej przypisać inta, doubla lub inną walutę.

Nie jest dozwolone przypisanie do zmiennej wartości innego typu niż ten podany przy deklaracji.

Dostępne elementy:

- Pętla while
- Instrukcja warunkowa If – else

- Operatory : `+, -, *, /, &, |, ==, !=, >, <, >=, <=, ()`

Przykłady

1. Deklaracja zmiennych:

```
PLN złote = 1.0 ;
USD dolary = 2.0 ;
var k = 0.0;
```

2. Dodawanie, odejmowanie (dopuszczalne tylko pomiędzy zmiennymi tego samego typu : waluta/waluta, liczba/liczba):

```
AUD suma = 0.0 ; suma = złote + dolary;
```

Wartość sumy zostanie podana w walucie podanej przy deklaracji.

3. Mnożenie i dzielenie – dopuszczalne jedynie w wariantach (waluta*liczba, waluta/liczba, liczba*liczba, liczba/liczba)

```
var iloczyn = dolary * k;
```

4. Operatory >, <, >=, <=, ==, != mogą być używane na zmiennych tego samego typu jednak waluty po obu stronach porównania nie muszą być takie same.

```
złote >= dolary;
```

5. Definicja funkcji

```
def USD savingsInUsdCurrency(PLN currency1, EUR
currency2, USD currency3)
{
USD savings = 0 ;
savings = currency1 + currency2  ;
savings = savings + currency3 ;
return savings;
}
PLN złote = 200.0 ;
EUR euro = 10.0 ;
USD dolary = 342.0 ;
CZK mySavings = savingsInUsdCurrency(złote, euro, dolary)
;
print(mySavings) ;

EUR mySavings2 =  mySavings * 2;
print(mySavings2);

def var fib(var n){
if (n == 1 | n == 2)
{


    return 1 ;
```

```
    }
    else
    {
        return  fib(n - 1) + fib(n - 2);
    }
    return 0;
    }
    var sum = 0 ;
   var iter = 0 ;
    var param = 1;
  while ( iter < 30 )
        {
          sum = sum + fib(param);
            iter = iter + 1;
        }
    print(sum);
```

Wywołanie programu

Program wywołujemy z terminala za pomocą komendy:

python3 __main__.py <ścieżka_do_pliku_z_kodem>

Testy odbywały się na wersji Python3.8.5

Opis gramatyki

KONWENCJE LEKSYKALNE

```
char = letter | digit | specialSymbol | " " specialSymbol = '_'
text = {char}
name = {letter | digit | specialSymbol };
digit = '0' | nonZeroDigit ; nonZeroDigit = '1' | ... | '9' ;
number = '0.', {digit} | nonZeroDigit, {digit}, '.', {digit} ;
letter = 'a' | ... | 'z' | 'A' | ... | 'Z' ;

currencyID =  'EUR' | 'USD' | 'CHF' | 'JPY' | 'GBP);
```

SKŁADNIA

```
Program = ProgStatement, {ProgStatement} ;

ProgStatement = FunDef | Statement ;

Statement = ConditionalStatement | SimpleStatement ;

ConditionalStatement = IfStatement | WhileStatement ;

SimpleStatement = (Assignment | FunCall | PrintCall | ReturnStatement | VarDecl), ';' ;

FunDef = 'def', name, '(', [ParamList], ')', Block ;

VarDecl = 'var', name, ['=', Expression] ;

CurrencyDecl = currencyId, name, [` = ` , Expression];

IfStatement = IfBlock, [ElseBlock] ;

WhileStatement = 'while', '(', Expression, ')', '{', Statement, {Statement}, '}' ;

Assignment = name, '=', Expression ;

FunCall = name, '(', [ArgList], ')' ;

PrintCall = 'print', '(', (string | OppExpr), ')' ;

ParamList = Param, {',', Param} ;

Param = 'var', name ;

OrExpr = AndExpr, {'|', AndExpr};

AndExpr = Condition, {'&', Condition} ;

Condition = Expression, {('==' | '!=' | '>' | '>=' | '<' | '<='), Expression} ;

Expression = MultiplicativeExpr, {('+' | '-'), MultiplicativeExpr } ;

MultiplicativeExpr = OppExpr {('*' | '/'), OppExpr};

OppExpr = FunCall | name | number | currency | `(', Expression, `)' ;

IfBlock = 'if', '(', OrExpr , ')', Block ;

ElseBlock = 'else', Block ;

Block = '{'{Statement}, [ReturnStatement] '}' ;

ArgList = Arg, {',', Arg} ;

Arg = Expression;

ReturnStatement = 'return', [Expression], ';';
```