Interpreter - część 1.

Gramatyka

Jest to lekko zmodyfikowana gramatyka języka Latte. Wszystkie wprowadzone przeze mnie zmiany mają krótki komentarz tłumaczący je.

```
-- programs ------------------------------------------------

entrypoints Program ;

Program.   Program ::= [TopDef] ;

FnDef.     TopDef ::= Type Ident "(" [Arg] ")" Block ;

separator nonempty TopDef "" ;

Arg.       Arg ::= Type Ident;

-- Przekazywanie przez zmienną
RefArg.    Arg ::= Type "&" Ident ;

separator  Arg "," ;

-- statements ----------------------------------------------

Block.     Block ::= "{" [Stmt] "}" ;

separator  Stmt "" ;

Empty.     Stmt ::= ";" ;

BStmt.     Stmt ::= Block ;

-- Zagnieżdzona definicje
FnStmt.    Stmt ::= TopDef ;

Decl.      Stmt ::= Type [Item] ";" ;

NoInit.    Item ::= Ident ;

Init.      Item ::= Ident "=" Expr ;

separator nonempty Item "," ;

Ass.       Stmt ::= Ident "=" Expr  ";" ;

-- Przypisanie wartości do tablicy lub krotki pod konkretnym indeksem
ArrAss.    Stmt ::= Ident "[" Integer "]" "=" Expr ";" ;
```

```
-- Pythonowe przypisanie do krotki
TupleAss.  Stmt ::= "(" [Ident] ")" "=" Expr ";" ;

separator nonempty Ident "," ;

Incr.      Stmt ::= Ident "++"  ";" ;

Decr.      Stmt ::= Ident "--"  ";" ;

-- Procedura print
Print.     Stmt ::= "print" Expr ";" ;

Ret.       Stmt ::= "return" Expr ";" ;

VRet.      Stmt ::= "return" ";" ;

Cond.      Stmt ::= "if" "(" Expr ")" Stmt  ;

CondElse.  Stmt ::= "if" "(" Expr ")" Stmt "else" Stmt  ;

While.     Stmt ::= "while" "(" Expr ")" Stmt ;

-- Zmiana wielkości tablicy. Przy skracaniu tracona jest odpowiednia ilość
elementów na końcu tablicy. Przy rozszerzaniu dodane elementy odpowiadają 0 dla
danego typu (0 dla int, flase dla bool, "" dla string itp.)
Resize.    Stmt ::= "resize" Ident Integer ;

-- Zmiana wielkości tablicy. Przy skracaniu tracona jest odpowiednia ilość
elementów na końcu tablicy. Przy rozszerzaniu dodane elementy odpowiadają 0 dla
danego typu (0 dla int, flase dla bool, "" dla string itp.)
ResizeId.  Stmt ::= "resize" Ident Expr ;

-- Break
Break.     Stmt ::= "break" ";" ;

-- Continue
Continue.  Stmt ::= "continue" ";" ;

SExp.      Stmt ::= Expr  ";" ;

-- Types ---------------------------------------------------

Int.       Type ::= "int" ;

Str.       Type ::= "string" ;

Bool.      Type ::= "bool" ;

Void.      Type ::= "void" ;

-- Typ tablicy
Arr.       Type ::= Type "[]" ;

- Typ krotki
```

```
Tuple.     Type ::= "Tuple" "(" [Type] ")" ;

internal   Fun. Type ::= Type "(" [Type] ")" ;

separator  Type "," ;

-- Expressions ------------------------------------------

EVar.      Expr6 ::= Ident ;

-- Odczytanie wartości zmiennej pod danym indeksem tablicy, lub danego elementu
krotki
EArr.      Expr6 ::= Ident "[" Integer "]" ;

-- Ilość elementów w tablicy
ESize.     Expr6 ::= "size" Ident ;

-- Sposób na stworzenie nowej tablicy o danym rozmiarze. Wstawione elementy
odpowiadają 0 dla danego typu (0 dla int, flase dla bool, "" dla string itp.)
ENewArr.   Expr6 ::= "new" "[" Integer "]" ;

ELitInt.   Expr6 ::= Integer ;

ELitTrue.  Expr6 ::= "true" ;

ELitFalse. Expr6 ::= "false" ;

-- Stworzenie nowej krotki
ETuple.    Expr6 ::= "(" [Expr] ")" ;

EApp.      Expr6 ::= Ident "(" [Expr] ")" ;

EString.   Expr6 ::= String ;

Neg.       Expr5 ::= "-" Expr6 ;

Not.       Expr5 ::= "!" Expr6 ;

EMul.      Expr4 ::= Expr4 MulOp Expr5 ;

EAdd.      Expr3 ::= Expr3 AddOp Expr4 ;

ERel.      Expr2 ::= Expr2 RelOp Expr3 ;

EAnd.      Expr1 ::= Expr2 "&&" Expr1 ;

EOr.       Expr ::= Expr1 "||" Expr ;

coercions  Expr 6 ;

separator  Expr "," ;

-- operators -------------------------------------------
```

```
Plus.      AddOp ::= "+" ;

Minus.     AddOp ::= "-" ;

Times.     MulOp ::= "*" ;

Div.       MulOp ::= "/" ;

Mod.       MulOp ::= "%" ;

LTH.       RelOp ::= "<" ;

LE.        RelOp ::= "<=" ;

GTH.       RelOp ::= ">" ;

GE.        RelOp ::= ">=" ;

EQU.       RelOp ::= "==" ;

NE.        RelOp ::= "!=" ;
```

Przykłady

Przekazywanie przez zmienną i wartość

```
int x = 5, y = 10;

void f (int wartosc, int & zmienna) {
    wartosc = 50;
    zmienna = 100;
    print wartosc;
    print zmienna;
}

int main () {
    print "Przed f:"
    print x;
    print y;

    print "f:"
    f(x, y)

    print "Po f:"
    print x;
    print y;
}
```

Wynikiem powinno być:

```
Przed f:
5
10
f:
50
100
Po f:
5
100
```

Użycie tablicy

```
int main () {
    int[] t = new int[5];

    t[1] = 10;

    print t;
    print (size t);

    int[] t2 = t;
    resize t2 10;
    t[4] = -1;

    print t;
    print (size t)
    print t2;
    print (size t2)
}
```

Wynikiem powinno być:

```
[0, 10, 0, 0, 0]
5
[0, 10, 0, 0, -1]
5
[0, 10, 0, 0, 0, 0, 0, 0, 0, 0]
10
```

Użycie krotki

```
int main () {
    Tuple(int, string) t = (3, "str");
    print t;

    t[0] = 30;
```

```
    (x, y) = t;
    x = x + 10;

    print x;
    print y;
    print t[0]
}
```

Wynikiem powinno być:

```
(3, "str")
40
"str"
30
```

Przesłanianie identyfikatorów ze statycznym wiązaniem

```
int x = 5;

int f () {
    print x;
}

int main () {
    int x = 10;

    print x;
    print f();
}
```

Wynikiem powinno być:

```
10
5
```

Break i continue

```
int main () {
    int i = 0;
    while (true) {
        i++;

        if (i == 5) {
            continue;
```

```
        }

        if (i == 10) {
            break;
        }

        print i;
    }
}
```

Wynikiem powinno być:

```
1
2
3
4
6
7
8
9
```

Opis języka

Język, który będę chciał zaimplementować będzie lekko zmodyfikowanym jezykiem Latte. Główne różnice między nim są następujące:

- Możliwość użycia & aby przekazać do funkcji argument przez zmienną (używa się w taki sam sposób jak przekazywania przez referencję w C++)
- Dodane tablice indeksowane int, przykład użycia poniżej
- Dodane krotki, przykład użycia poniżej
- Dodana procedura print
- Dodane break, continue
- Zmieniona nazwa typu boolean na bool

Poza tym zmiany zadeklarowane w tabelce poniżej.

Tablice

Tablice są typu `Type[]` i używa się ich podobnie jak w C/C++. Nową tablicę o danej ilości elementów tworzy się używając konstrukcji `new [Integer]`. Tablica niezainicjowana zwraca błąd przy próbie jej modyfikacji, lub odczytu. Język udostępnia `size Ident` zwracający długość tablicy oraz `resize Ident Integer` pozwalający na zmianę jej rozmiatu. W przypadku zmniejszenia rozmiaru, z końca tablicy usuwana jest odpowiednia ilość elementów. Za to w przypadku zwiększenia rozmiaru, na końcu tablicy dodawana jest odpowiednia ilość domyślnych wartości typu. Domyślna wartość typu to:

- int: 0
- bool: false
- string: ""

- krotka: krotka domyślnych wartości typów przechowywanych w tej krotce
- tablica: pusta, niezainicjowana tablica

Krotki

Krotki działają podobnie to tablic. Ich typ to `Tuple([Type])`, a stworzyć ją można pisząc `([Expr])`. Umożliwiają one przypisanie jak w Pythonie (np. `(x, y) = (false, 10);`) oraz można odczytać i modyfikować wartości ich elementów za pomocą operatora `[]`, tak jak w tablicach.

Tabelka funkcjonalności

Na 15 punktów
01 (trzy typy)
02 (literały, arytmetyka, porównania)
03 (zmienne, przypisanie)
04 (print)
05 (while, if)
06 (funkcje lub procedury, rekurencja)
07 (przez zmienną / przez wartość)
Na 20 punktów
09 (przesłanianie i statyczne wiązanie)
10 (obsługa błędów wykonania)
11 (funkcje zwracające wartość)
Na 30 punktów
12 (4) (statyczne typowanie)
13 (2) (funkcje zagnieżdżone ze statycznym wiązaniem)
14 (1) (rekordy/tablice/listy)
15 (2) (krotki z przypisaniem)
16 (1) (break, continue)

Razem: 30