JPP - deklaracja języka Risotto

Michał Napiórkowski

10 maja 2023

1 Opis języka

Risotto jest językiem imperatywnym ze statycznym typowaniem.

1.1 Funkcje

Program w Risotto jest listą funkcji. Punktem wejścia do programu jest bezparametrowa funkcja `void main()`. Domyślnie argumenty do funkcji są przekazywane przez wartość. Aby przekazać przez referencję, należy poprzedzić typ parametru w definicji funkcji słowem kluczowym `ref`. Ostatnią instrukcją każdej funkcji musi być `return` lub `turnback` (zobacz niżej). Instrukcje te nie mogą wystąpić wcześniej niż jako ostatnie w funkcji. Risotto nie umożliwia zagnieżdżania funkcji.

1.2 Instrukcja turnback

Alternatywą dla kończenia funkcji instrukcją `return` jest zastosowanie instrukcji `turnback`. Powoduje ona ponowne wykonanie całej funkcji, tyle że w odwrotnej kolejności. Dotyczy to jedynie instrukcji najwyższego poziomu wewnątrz danej funkcji - bloki wewnątrz pętli i instrukcji warunkowych nie są odwracane. Składnia `turnback EXPR` umożliwia funkcji zwrócenie wyrażenia, przy czym wyrażenie jest ewaluowane po powrocie do początku funkcji. Przykład ilustrujący działanie tej instrukcji znajduje się poniżej.

1.3 Typy i zmienne

W języku Risotto dostępne są typy: `int`, `bool`, `string` oraz typ `void` służący do definiowania procedur. Zmienne globalne deklaruje się na tym samym poziomie co funkcje, wówczas typ zmiennej należy poprzedzić słowem kluczowym `global`. Zmienne lokalne deklaruje się porzedzając typ słowem kluczowym `let`.

1.4 Pętle i instrukcje warunkowe

Pętla ma składnię `while: _ endwhile`. W pętlach obsługiwane są instrukcje `break` oraz `continue`. Instrukcja warunkowa ma składnię `if: _ elif: _ else: _ endif`.

1.5 Pozostałe

Dostępne są wbudowane instrukcje `print` oraz `println` do wypisywania wartości wyrażeń na standardowe wyjście. Funkcje oraz zmienne globalne mogą być deklarowane w dowolnej kolejności i są ewaluowane leniwie.
Risotto obsługuje dwa rodzaje błędów - błędy typowania zgłaszane podczas statycznego typowania oraz błędy czasu wykonania (np. dzielenie przez zero).

2 Tabelka cech

2.1 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)
- 08 (zmienne read-only i pętla for)

2.2 Na 20 punktów

+ 09 (przesłanianie i statyczne wiązanie - zmienne lokalne i globalne)
+ 10 (obsługa błędów wykonania)
+ 11 (funkcje zwracające wartość)

2.3 Na 30 punktów

+ 12 (4) (statyczne typowanie)
- 13 (2) (funkcje zagnieżdżone ze statycznym wiązaniem)
- 14 (1/2) (rekordy/listy/tablice/tablice wielowymiarowe)
- 15 (2) (krotki z przypisaniem)
+ 16 (1) (break, continue)
- 17 (4) (funkcje wyższego rzędu, anonimowe, domknięcia)
- 18 (3) (generatory)

2.4 Dodatkowo

+ 19 (3) (instrukcja turnback)
+ 20 (2) (funkcje i zmienne globalne mogą być deklarowane w dowolnej kolejności)

Razem: 30

3 Gramatyka

```
layout ":" ;
layout stop "return", "turnback", "endwhile", "elif", "else", "endif" ;

-- programs ------------------------------------------------

entrypoints     Progr ;
Program.        Progr ::= [TopDef] ;

FnDef.          TopDef ::= TType Ident "(" [Param] ")" ":" Block Ret ;
GlobVar.        TopDef ::= "global" TType [Item] ;
terminator      nonempty TopDef ";" ;

ValParam.       Param ::= TType Ident ;
RefParam.       Param ::= "ref" TType Ident ;
separator       Param "," ;

Return.         Ret ::= "return" Expr ;
```

```
VReturn.        Ret ::= "return" ;
Turnback.       Ret ::= "turnback" Expr ;
VTurnback.      Ret ::= "turnback" ;

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

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

SExpr.          Stmt ::= Expr ;

SDecl.          Stmt ::= "let" TType [Item] ;
NoInit.         Item ::= Ident ;
Init.           Item ::= Ident "=" Expr ;
separator       nonempty Item "," ;

SAss.           Stmt ::= Ident "=" Expr ;
SIncr.          Stmt ::= Ident "++" ;
SDecr.          Stmt ::= Ident "--" ;

SIf.            Stmt ::= "if" "(" Expr ")" ":" Block [Elif] "endif" ;
SIfElse.        Stmt ::= "if" "(" Expr ")" ":" Block [Elif] "else" ":" Block "endif"
SElif.          Elif ::= "elif" "(" Expr ")" ":" Block ;
separator       Elif "" ;

SWhile.         Stmt ::= "while" "(" Expr ")" ":" Block "endwhile" ;

SBreak.         Stmt ::= "break" ;
SContinue.      Stmt ::= "continue" ;

SPrint.         Stmt ::= "print" Expr ;
SPrintLn.       Stmt ::= "println" Expr ;

-- types ---------------------------------------------------

TInt.           TType ::= "int" ;
TBool.          TType ::= "bool" ;
TString.        TType ::= "string" ;
TVoid.          TType ::= "void" ;

-- expressions ---------------------------------------------

EVar.           Expr6 ::= Ident ;
ELitInt.        Expr6 ::= Integer ;
ELitTrue.       Expr6 ::= "true" ;
ELitFalse.      Expr6 ::= "false" ;
EString.        Expr6 ::= String ;
EApp.           Expr6 ::= Ident "(" [Expr] ")" ;

ENeg.           Expr5 ::= "-" Expr6 ;
ENot.           Expr5 ::= "not" Expr6 ;

EMul.           Expr4 ::= Expr4 MulOp Expr5 ;

EAdd.           Expr3 ::= Expr3 AddOp Expr4 ;
```

```
ERel.           Expr2 ::= Expr2 RelOp Expr3 ;

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

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

separator       Expr "," ;
coercions       Expr 6 ;

-- operators ----------------------------------------------

OPlus.          AddOp ::= "+" ;
OMinus.         AddOp ::= "-" ;

OTimes.         MulOp ::= "*" ;
ODiv.           MulOp ::= "/" ;
OMod.           MulOp ::= "%" ;

OLt.            RelOp ::= "<" ;
OLeq.           RelOp ::= "<=" ;
OGt.            RelOp ::= ">" ;
OGeq.           RelOp ::= ">=" ;
OEq.            RelOp ::= "==" ;
ONeq.           RelOp ::= "!=" ;

-- comments ------------------------------------------------

comment         "//" ;
comment         "/*" "*/" ;
```

4 Przykłady

```
int sumEven(int x):
    let int i = 0;
    let int sum = 0;
    while (true):
        i++;
        if (i == (x + 1)):
            break;
        elif (i % 2 == 1):
            continue;
        else:
            sum = sum + i;
        endif;
    endwhile;
    return sum;

global int x = 500;

void globVar():
    x = x + 1;
    println x; // 501
    return;

void locVar():
```

```
    let int x = 10;
    globVar();
    println x; // 10
    return;

string risotto(ref string s):
    s = s + "o";
    s = s + "t";
    turnback "Ris" + s;

int fibo(int n):
    let int res = 1;
    if (n > 2):
        res = fibo(n - 1) + fibo(n - 2);
    endif;
    return res;

void main():

    // Example 1.
    println sumEven(10); // 30

    // Example 2.
    locVar();
    println x; // 501

    // Example 3.
    let string s;
    println risotto(s); // "Risotto"
    println s; // "otto"

    // Example 4.
    println fibo(12);

    return;
```