JPP Interpreter Decleration

Resul Hangeldiyev (rh402185)

April 2022

1 Grammar

Below presented grammar of the planned programming language in LBNF:

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

entrypoints Program ;

Program.   Program ::= [TopDef] ;

FnDef.    TopDef ::= Ident ":" Block ;

separator nonempty TopDef "" ;

Glob.      TopDef ::= [Item] ";" ;

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

-- Below rule might need fix for end of the block
-- (like DEDENT token in Python grammar).
Block.     Block ::= "\n\t" [Stmt] ;

separator  Stmt "" ;

Empty.     Stmt ::= ";" ;

BStmt.     Stmt ::= Block ;

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

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

separator nonempty Item "," ;
```

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

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

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

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

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

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

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

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

ConstFor.  Stmt ::= "for" "(" Ident "=" Expr "to" Expr "):" Stmt;

SExp.      Stmt ::= Expr  ";" ;

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

internal     Int.       Type ::= "int" ;

internal     Str.       Type ::= "string" ;

internal     Bool.      Type ::= "bool" ;

internal     Void.      Type ::= "void" ;

internal     FuncType.  Type ::= "("[Type] "->" Type ")" ;

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

EVar.      Expr6 ::= Ident ;

ELitInt.   Expr6 ::= Integer ;

ELitTrue.  Expr6 ::= "true" ;

ELitFalse. Expr6 ::= "false" ;

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

EString.   Expr6 ::= String ;
```

```
EList.     Expr6 ::= "[" "]";

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 ::= "!=" ;

----------------------- comments ----------------------------
```

```
comment    "#" ;

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

2 Examples

Hello world:

```
main:
    out("hello world!");
```

Even numbers up to 10:

```
main:
    i = 0;
    while (i < 10):
        if (i % 2 == 0):
            out(i);
        i++;
    out(i);
```

Factorial:

```
main:
    out(fact(7));
    out(factr(7));

# iterative
fact(n):
    r = 1;
    for (i = 2; to n):
        r *= i;
    return r;

# recursive
factr(n):
    if (n < 2):
        return n;
    return n * factr(n - 1);
```

3 Formal description

[Name here] - is dynamically typed (might add support for static type system if enough time) interpreted imperative programming language which has both

Python and C++ syntactic flavor. Blocks look like as if in Python without curly brackets, meanwhile each instruction is terminated by semicolon like in C/C++. Unlike in Python brackets are required for *if, while* and *for* instructions and *for* looks more like in C/C++.

I may change to ordinary C/C++ curly bracket syntax if it turns out to be too cumbersome to deal with indentations. Thereof, in the examples above only blocks with indentations would be replaced with curly brackets. In that case, syntax will more resemble C++.

In the long term, language is aimed to be used in coding competitions/olympiads where typing less in short period of time matters.

4 Functionality table

Below listed all required functionalities, where ~~strikethroughed~~ indicates features not planned to implement in the first stage. Planned functionalities can be extended/removed as per the progress made during the implementation phase.

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 zmienna / przez wartość / in/out)
- 08 (zmienne read-only i petla for)

Na 20 punktów

- 09 (przesłanianie i statyczne wiazanie)
- 10 (obsługa błedów wykonania)
- 11 (funkcje zwracajace wartość)

Na 30 punktów

- ~~12 (4) (statyczne typowanie)~~
- 13 (2) (funkcje zagnieżdżone ze statycznym wiazaniem)
- 14 (1~~/2~~) (rekordy/listy/tablice/tablice wielowymiarowe)

- ~~15 (2) (krotki z przypisaniem)~~
- 16 (1) (break, continue)
- 17 (4) (funkcje wyższego rzedu, anonimowe, domkniecia)
- ~~18 (3) (generatory)~~

Razem: 28