Opis

Opisywany język jest oparty na C / Latte.

Bloki pętli, instrukcji warunkowej if oraz funkcji muszą być otoczone klamrami { } ze względu na czytelność.

Krotki są oznaczane za pomocą nawiasów ostrokątnych <> dla wizualnego odróżnienia od innych konstrukcji.

Tablice są deklarowane bez podawania rozmiaru.

Tabela cech

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 / in/out)
+ 08 (zmienne read-only i pętla for)
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/listy/tablice/tablice wielowymiarowe)
+ 15 (2) (krotki z przypisaniem)
+ 16 (1) (break, continue)
17 (4) (funkcje wyższego rzędu, anonimowe, domknięcia)
18 (3) (generatory)

Razem: 24 / 30

Propozycje uciekawienia języka (hipotetyczne deklaracje, do omówienia na zajęciach):

1. wg opisu zadania punkty 07 i 08 są zamienne do punktacji 15pkt. Chciałbym zadeklarować punkt 07 jeśli zaimplementowanie obu przynosi dodatkowe punkty.

2. dodanie słabo typowanych zmiennych, na przykład wint, który w zależności od instrukcji może być traktowany jako inny typ.
   Własności:
   - wint x jest traktowane jak wartość liczbowa;
   - jeśli wint jest użyty w wyrażeniu, które powinno ewaluować się do wartości typu boolean, to wint x == true wtw, gdy x != 0;
   - - jeśli wint jest użyty w wyrażeniu, które powinno ewaluować się do wartości typu string, to wint x = (string) x;

   (nie jestem pewien tego pomysłu ze względu na fakt, że nie planowałem uwzględniać statycznego typowania w programie. Z drugiej strony jakieś rozpoznawanie typów i tak tam przecież będzie.)

3. Zakładam, że funkcja print() wliczona w wymogi 15pkt to prosta funkcja wypisująca pojedynczą zmienną i nic więcej.
   Chciałbym spróbować zaimplementować wersję tej funkcji opartą na pochodzącym z C printf, postaci: print :: [Int] -> [String] -> [Boolean] -> String -> String

Gramatyka

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

entrypoints     Program ;

Program.        Program ::= [TopDef] ;

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

separator       nonempty TopDef "" ;

Arg.            Arg ::= Type Ident;

separator       Arg "," ;

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

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

separator       Stmt "" ;

Empty           Stmt ::= ";" ;

BStmt.          Stmt ::= Block ;

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

NoInit.         Item ::= Ident ;

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 ")" Block  ;
```

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

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

For.           Stmt ::= “for” “(” Ident “:=” Expr “to” Expr “)” Block;

SExp.          Stmt ::= Expr  ";" ;

BreakExp.      Stmt ::= “break” “;” ;

ContExp.       Stmt ::= “continue” “;” ;

Print          Stmt ::= “print” “(” Expr “)”;

Printf         Stmt ::= “printf” “(“ Expr Expr Expr String “)”

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

Int.           Type ::= "int" ;

Str.           Type ::= "string" ;

Bool.          Type ::= "boolean" ;

Void.          Type ::= "void" ;

Wint.          Type ::= “wint”;

Tuple.         Type ::= “tuple” “<” [Type] “>”;

Array.         Type ::= Type “[]”;

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

separator      Type "," ;

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

ETuple.        Expr7 ::= “<” [Expr] “>”;

EArr.          Expr7 ::= “[“ [Expr] “]”;

EVar.          Expr6 ::= Ident ;

ELitInt.       Expr6 ::= Integer ;

ELitTrue.      Expr6 ::= "true" ;
```

```
ELitFalse.      Expr6 ::= "false" ;

ELitWInt.       Expr6 ::= Integer;

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 ::= "!=" ;

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

comment         "#" ;

comment         "//" ;

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

Przykłady

Poniżej kilka przykładów na odbiegające od standardowych konstrukcje.

wint vs int

Poprawny program:

```
wint condition = 2 + 2;
if (condition) {
     print (“If is true”);
}
```

Niepoprawny program (niepoprawny typ)**:**

```
int condition = 2 + 2;
if (condition) {
     print (“If is true”);
}
```

Niepoprawny program (brak klamer)**:**

```
wint condition = 2 + 2;
if (condition)
     print (“If is true”);
```

```
wint x = 42;
string s = "Number is ";
print (s + x);
```

Wynik: "Number is 42"

```
wint x = 0;
if (x) {return "True";} else {return "False";}
```

Wynik: "False"

```
wint x = -1;
wint y =  1;
int  z =  1;
```

Wówczas:

x + y == 0 == false;
x + z == 0 == false;
z + x == 0 jest wartością liczbową, nie logiczną, dlatego w ewaluacji wartości logicznej powoduje błąd;

Krotki

```
tuple<int, string> example_tpl = <42, “AlaMaKota”>;
```

Tablice

```
int[] example_arr = [1, 2, 3, 4];
```

Printf

```
printf([42], ["Hello World"], [true, false], "a is %b, your number
is %d, your message is %s and b == %b");
```

Zwróci:
"a is true, your number is 42, your message is Hello World and b == false"

Niepoprawny program (brak argumentów odpowiedniego typu)**:**

```
printf([], [“some_string”], [], “Number = %d”);
```

for

```
for (int i := 0 to 3) {
     print(i);
}
```

Wypisze: 0, 1, 2, 3

Niepoprawny program (zmienna i nie może być modyfikowana)**:**

```
for (int i := 0 to 3) {
     i = i * 2;
     print(i);
}
```

Niepoprawny program (zmienna i istnieje tylko wewnątrz pętli)**:**

```
for (int i := 0 to 3) {}
print(i);
```