JPP - Deklaracja języka

Anna Jabłonowska

24 kwietnia 2022

1 Opis języka

Język imperatywny wzorowany na języku Latte.

Język posiada trzy typy (int, string, bool). Występuje inicjalizacja domyślna zmiennych (int - 0, string - "", bool - false). Na typach dostępne są standardowe porównania (>=, <=, ==, !=, <, >). Na typie bool można wykonywać operacje "&&", "||" oraz "!". Na typie string można wykonywać operację konkatenacji ("+"). Na typie int można wykonywać wszystkie standardowe operacje (+, -, *, / - dzielenie całkowite, % - reszta z dzielenia). Zmienne typu int nie mają ograniczenia wartości.

Język wymaga korzystania z funkcji zwracających wartość - nie posiada możliwości tworzenia procedur. Dozwolone jest nieczytanie wartości zwracanych przez funkcję, jednak każda tworzona funkcja musi zwracać pewną wartość. Funkcje i zmienne są widoczne od momentu ich zdefiniowania. Dozwolona jest rekurencja bezpośrednia w funkcjach. Argumenty przekazywane do funkcji są przez wartość lub metodą in/out. Kolejność wyliczania obowiązuje od lewej do prawej. Wyliczanie wartości w wyrażeniach logicznych jest krótkie.

Wykonywanie rozpoczyna się od wywołania funkcji main, która jest wymagana. Typ wartości zwracanej przez tę funkcję jest typu int. Funkcja main nie przyjmuje argumentów.

Możliwe jest tworzenie zmiennych globalnych. Zmienne globalne muszą być inicjalizowane od razu przy definicji. Obowiązuje przesłanianie zmiennych i statyczne wiązania.

Instrukcja print pozwala na wypisywanie wartości na wyjście. Dozwolone jest wypisywanie wartości wszystkich istniejących typów.

Konstrukcje while, if oraz else są zapożyczone z języka Latte.

Błędy wykonania powodują przerwanie działania programu i wypisanie komunikatu o błędzie.

2 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 / 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 (1/2) (rekordy/listy/tablice/tablice wielowymiarowe)
15 (2) (krotki z przypisaniem)
16 (1) (break, continue)
17 (4) (funkcje wyższego rzędu, anonimowe, domknięcia)
18 (3) (generatory)

Razem: 20 punktów

3 Gramatyka mojego języka w notacji LBNF

Część gramatyki wzorowana na gramatyce języka Latte.

```
-- Programy (programs) ------------------------------------------------
entrypoints Program ;
Program.  Program ::= [Global][TopDef] ;
GlobalDef. Global ::= Type Ident "=" Expr ";" ;
separator Global "" ;
FnDef.     TopDef ::= Type Ident "(" [Arg] ")" Block ;
separator nonempty TopDef "" ;
```

```
Arg.    Arg ::= Type Ident ;
InArg.    Arg ::= "in" Type Ident ;
OutArg.   Arg ::= "out" Type Ident ;
separator Arg "," ;

-- Instrukcje (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 ";" ;
Cond.     Stmt ::= "if" "(" Expr ")" Stmt ;
CondElse. Stmt ::= "if" "(" Expr ")" Stmt "else" Stmt ;
While.    Stmt ::= "while" "(" Expr ")" Stmt ;
SExp.     Stmt ::= Expr ";" ;
Print.    Stmt ::= "print" Expr ";" ;

-- Typy (types) -----------------------------------------------------
Int.      Type ::= "int" ;
Str.      Type ::= "string" ;
Bool.     Type ::= "bool" ;
internal  Fun. Type ::= Type "(" [Type] ")" ;
separator Type "," ;

-- Wyrazenia (expressions) ------------------------------------------------
EVar.     Expr6 ::= Ident ;
ELitInt.  Expr6 ::= Integer ;
ELitTrue. Expr6 ::= "true" ;
ELitFalse. Expr6 ::= "false" ;
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 "," ;

-- Operatory (operators) --------------------------------------------------
Plus.     AddOp ::= "+" ;
```

```
Minus.    AddOp ::= "-" ;
Times.    MulOp ::= "*" ;
Div.      MulOp ::= "/" ;
Mod.      MulOp ::= "%" ;
LTH.       RelOp ::= "<" ;
LE.       RelOp ::= "<=" ;
GTH.       RelOp ::= ">" ;
GE.       RelOp ::= ">=" ;
EQU.       RelOp ::= "==" ;
NE.       RelOp ::= "!=" ;

-- Komentarze (comments) ------------------------------------------------
comment   "//" ;
comment   "/*" "*/" ;
```

4 Przykładowe programy

4.1 Funkcja Hello world

```
// Funkcja Hello world
string s = "Hello world";

int main(){
  print s;
  return 0;
}
```

4.2 Funkcja wypisująca liczby parzyste do 10

```
// Funkcja wypisujaca liczby parzyste do 10
int main(){
  int i = 0;

  while(i <= 10){
    if(i % 2 == 0){
      print i;
    }
    i++;
  }

  return 0;
}
```

4.3 Funkcja licząca k-tą liczbę Fibonacciego z wykorzystaniem rekurencji

```
// Funkcja liczaca k-ta liczbe Fibonacciego z wykorzystaniem rekurencji
int fibonacci(in int k, out int f){
  if(k == 0 || k == 1){
    f = k;
  } else{
    int a, b;
    fibonacci(k - 1, a);
    fibonacci(k - 2, b);

    f = a + b;
  }
  return 0;
}

int main(){
  int k = 5;
  int f;

  print k;
  fibonacci(k, f);
  print f;

  return 0;
}
```