Języki i paradygmaty programowania, Interpreter języka imperatywnego

Rafał Klimek, rk418291, 10 kwietnia 2022

Tabelka punktów
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: 27

Zmienne

W języku są 4 typy zmiennych: `int`, `bool`, `string`, `tuple`.

`int`: liczby całkowite odpowiadające Haskellowemu Int. Pozwala na przypisanie literału całkowitoliczbowego i na operacje arytmetyczne: `+`, `-`, `*`, `/` (działa jak Haskellowy `` `div` ``), `%` (operacja modulo). Dostępne operacje porównania, zwracające typ `bool`: `<`, `<=`, `>`, `>=`, `==`, `!=`.

`bool`: wartości logiczne. Literały `_T` oraz `_F`. Pozwala na operacje logiczne: `||` (or), `&&` (and), `!` (unarny not, jak w C).

`string`: napisy. W języku nie ma list, zatem string nie jest reprezentowany jako lista znaków. Jest niemutowalny. Do zmiennej typu `string` można przypisać nową wartość, nie można jednak np. zmienić jednego znaku.

`tuple`: krotki. Zbiór dowolnych typów krotkowych, z dowolnymi zagnieżdżeniami. Jedyną operacją na krotce, jaką da się wykonać, jest jej odpakowanie do zmiennych przez przypisanie, np.:

```
var x := 0;
var y := 0;
var tup := (5, 6);
(x, y) := tup;
> > > x = 5
> > > y = 6
```

Można też użyć znaku _ aby zignorować część odpakowanych wartości:

```
var x := 0;
var y := 0;
var tup := (5, 6);
(_, y) := tup;
> > > x = 0
> > > y = 6
```

Deklaracja zmiennych przebiega przez użycie słowa kluczowego `var`. Użycie go determinuje typ dla zmiennej. Możliwe jest przysłanianie nazw zmiennych.

Język obsługuje zmienne read-only. Deklaruje się je słowem kluczowym `const`. Nie pozwalają na operację przypisania, poza momentem deklaracji.

Indeks inkrementowany w instrukcji `for` jest zmienną read-only.

Zmienne deklarowane poza blokami funkcji są zmiennymi globalnymi.

Funkcje

Funkcje zwracają wartość. Nie ma procedur. Deklarację funkcji zaczynamy od słowa kluczowego `fun`. Każda funkcja ma jeden określony typ wyniku. Próba zwrócenia np. w jednym przypadku boola, a w innym inta, skończy się niepowodzeniem przy statycznym sprawdzaniu typów.

Przykład:

```
fun a(int arg1, string arg2, (int, int) arg3) {
 return arg1;
};
```

Ta funkcja zwraca typ `int`.

Argumenty do funkcji przekazywane są przez kopię.

Zwracanie przez słowo `return`.

Pętle

Pętla `for`.

Użycie podobne do składni Pascalowej:

```
for i := pocz to kon {
 print i;
};
```

`i` w tym przykładzie jest zmienną read-only, autoinkrementowaną. Dozwolone są tylko pętle for działające na indeksach rosnących o 1 z każdą iteracją, aż do wartości podanej po słowie `to`. Tak jak w opisie zadania: wewnątrz pętli nie można zmienić wartości zmiennej sterującej, wartość `kon` liczona tylko raz - przed wejściem do pętli.

Próba wykonania:

```
for i:= 5 to 2 {
 print i;
};
```

powinna spowodować nieskończoną pętlę.

Pętla `while`.
Standardowa składnia:

```
while (wyrażenie logiczne) {
 dosth;
};
```

W obu rodzajach pętli obsługiwane są instrukcje `continue` i `break`.

Instrukcje warunkowe
Instrukcja `if else` oraz `if`.

Blok `if`:

```
if (wyrażenie logiczne) {
 dosth;
}
```

Blok `elif`:

```
elif (wyrażenie logiczne) {
 dosth;
}
```

Blok `else`:

```
else {
 dosth;
}
```

Każda sekwencja `if` zaczyna się od bloku `if`, następnie znajduje się pewna, również zerowa liczba bloków `elif`, a następnie opcjonalny blok `else`. Dzięki temu gramatyka dla instrukcji warunkowych jest jednoznaczna.

Błędy wykonania

Dzielenie przez 0.
Użycie instrukcji `continue` lub `break` poza pętlą.
Użycie instrukcji `return` poza ciałem funkcji.

Komentarze

```
-Jestem komentarzem jednolinijkowym
 {# Jestem komentarzem wielolinijkowym -}
```

Gramatyka

Literałami dla typu `bool` są _T i _F. Literałami dla typu `string` są dowolne ciągi znaków znajdujące się między znakami cudzysłowa (np. ”tekst”). Identyfikatorami zmiennych są dowolne ciągi znaków (innych niż białe znaki) zaczynające się od litery alfabetu angielskiego.

Literały dla typu `bool` nazwijmy *Bool*, dla typu `int` *Int*, dla typu `string`

String.

Literały definiujące typy krotek:

$Tuple ::= "(", (Bool|Int|String|Tuple), \{",", (Bool|Int|String|Tuple)\}, ")"$

Wyrażenia, składnia bazowana na Latte:

$Expr6 ::= Ident|Int|Bool|(Ident, " ("[Expr\{", ", Expr\}]")")|String|Tuple|("(", Expr, ")")$, gdzie $Ident$ to dozwolone identyfikatory zmiennych i funkcji, ciągi znaków zaczynające się od litery alfabetu angielskiego.

$Expr5 ::= ("-", Expr6)|("!", Expr6)|Expr6$

$Expr4 ::= (Expr4, MulOp, Expr5)|Expr5$

$Expr3 ::= (Expr3, AddOp, Expr4)|Expr4$

$Expr2 ::= (Expr2, RelOp, Expr3)|Expr3$

$Expr1 ::= (Expr2, " \&\&", Expr1)|Expr2$

$Expr ::= (Expr1, " ||", Expr)|Expr1$

$MulOp ::= " *"|"/"|"\%"$

$AddOp ::= " +"|"-"$

$RelOp ::="<" |" <=" |" >" |" >=" |"! =" |" ==$

Instrukcje:

entrypoint: $Prog$

$Prog ::= \{Stmt|("fun", Ident, " (", FunArgs, ")", Block, ";")\}$

$Stmt ::= (("var", Ident, ":=", Expr)$
$|("const", Ident, ":=", Expr)|"skip"|(Ident, ":=", Expr)$
$|("while", " (", Expr")", Block)|("for", Expr, " to", Expr, Block)$
$|(IfS, \{";", ElifS\}, [";", ElseS])$
$|("return", Expr)|"continue"|"break"|("print", Expr)), ";"$

$FunArgs ::= [FunArg, \{", ", FunArg\}]$

$FunArg ::= Type, Ident$

$Type ::= " string", " int", " bool", TupleType$

$TupleType ::= " (", [Type, \{", ", Type\}], ")"$

$Block ::= " \{"Stmt, \{";", Stmt\}, " \}"$

$IfS ::= " if", " (", Expr, ")"Block$

$ElifS ::= " elif", " (", Expr, ")"Block$

$ElseS ::= " else", Block$

Komentarze:
comment″ − −″
comment″{#″″−}″