

Wolke

Opis języka

Wolke jest imperatywnym językiem programowania zaimplementowanym w Haskellu. Bazuje na języku Latte z layoutem oraz kilkoma wstawkami syntaktycznymi zapożyczonymi z Pythona.

Obsługiwane są trzy typy: Int, String, Boolean. Język umożliwia przesłanianie, statyczne typowanie, statycznie wiązanie.

Programy składają się ze zmiennych globalnych oraz funkcji. Główną funkcją oraz punktem wejściowym każdego programu jest funkcja main. Przekazywanie argumentów do funkcji domyślnie odbywa się przez wartość. Aby przekazywać argument przez referencję, należy dodać znak '&' po typie zmiennej.

Podobnie jak w Pythonie, Wolke nie ma operatorów inkrementacji, dekrementacji.

Dodatkowo Wolke posiada wyrażenia kontrolujące przepływ pętli programu - break oraz continue.

Wypisywanie odbywa się poprzez instrukcję print. Sprawdzić dobrą ewaluację wyrażenia można poprzez instrukcję assert. Operatory and, or, not - jak w Pythonie. Nagłówki funkcji są postaci `def funkcja(arg1 | typ, arg2 | typ) -> typ`.

Przykładowe programy w języku Wolke

```
# (three types)
# (literals, arithmetic, comparison)
# (variables, assignment)

def main() -> Void:
    Int x = (2 + 3 * 4 / 5) % 6
    Int y = (6 + 5 * 4 / 3) % 2
    Boolean z = x >= y
    assert(z)
```

```
# (print, assert)

def main() -> Void:
    String x = "aaa"
    String y = "bbb"

    print(x + y)
    assert(x + y == "aaabbb")

    Int a = 5
    print(a)
    print(a > 6)
```

```
# (while, if)

def main() -> Void:
    Int x = 10

    while x > 0:
        if x % 2 == 1:
            print(x)
        x = x - 1
```

```
# (functions returning value)

def add_or_sub(x | Int, y | Int, add | Boolean) -> Int:
    Int res = 0
    if add:
        res = x + y
    else:
        res = x - y
    return res

def main() -> Void:
    print(add_or_sub(6, 5, False))
```

```
# (functions, recursion)

def fib(n | Int) -> Int:
    if n == 0 or n == 1:
        return n
    return fib(n - 1) + fib(n - 2)

def nothing() -> Void:
    pass

def main() -> Void:
    nothing()
    Int res = fib(15)
    assert(res == 610)
    print(res)
```

```
/*
    (pass by value, pass by reference)
*/

def incVal(x | Int) -> Int:
    return x + 1

def incRef(y | Int&) -> Void:
    y = y + 1

def main() -> Void:
    Int x = 6
    print(incVal(x))
    incRef(x)
    assert(x == 7)
```

```
# (variable shadowing, static binding)
# (global, local variables)
```

```
String s = "global"
```

```
def check() -> Void:
    assert(s == "global")
```

```
def main() -> Void:
    assert(s == "global")
```

```
    String s = "local"
```

```
    assert(s == "local")
```

```
    print(s)
```

```
# (break, continue)
```

```
def main() -> Void:
```

```
    Int x = 1
```

```
    Int i = 0
```

```
    while True:
```

```
        x = x * 2
```

```
        i = i + 1
```

```
        if i % 2 == 1:
```

```
            continue
```

```
        print(x)
```

```
        if x > 200:
```

```
            break
```

Tabela 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: 25