Objektinis Programavimas

L-reikšmės, r-reikšmės jų nuorodos

Turinys

L-values ir r-values (1)

- C++ kalboje kiekviena išraiška yra arba **l-reikšmė** (l-value) arba **r-reikšmė** (r-value).
- **l-value**: objektas ar funkcija turinti dedikuotą atminties adresą, kurį pasiekiame per operatorių &;
  - išteklių negalima pakartotinai naudoti (perduoti)
- **r-value**: visa kas yra ne `l-value`: skaičiai (literals) (10), laikinos reikšmės (x + 1) ir anoniminiai obj. (Vector{5}).
  - negalime naudoti operatoriaus &; negali būti kairėje "**operator=**" pusėje ir turi išraiškos scopą.

L-reikšmės ir r-reikšmės (2)

```
#include<iostream>

int main() {
    int x{10};              // l-value ar r-value yra x?
    int a = x++;            // l-value ar r-value yra x++?
    int b = ++x;            // l-value ar r-value yra ++x?
    int y{new int[10]}; // l-value ar r-value yra y?
    int c = y[0];           // l-value ar r-value yra y[0]?
}
```

— C: abi išraiškos x++ ir ++x yra r-value.

— C++: ++x yra l-value, tačiau x++ yra r-value. **Įsitikinkite**!

Non-const l-value nuorodos (&)

— Iki C++11 turėjome tik vieno tipo (l-value) nuorodas.

— **Nekonstantinės l-value nuorodos** inicializuojamos tik non-const l-values tipo reikšmėmis.

```
int x = 10;           // x yra non-const l-value
const int y = 20;     // yra yra const l-value
int &ref = x;         // OK, `ref` yra `x` alias
int &wrongRef1;       // negalima, nuoroda turi būti inicializuota!
int &wrongRef2 = y;   // negalima!, y yra const l-value
int &wrongRef3 = 10; // negalima!, 10 yra r-value
```

— Naudojamos pass-by-reference semantikai.

Const **l-value nuorodos (&)**

— **Konstantinės l-value nuorodos** inicializuojamos: non-const l-value, const l-values ar r-values reikšmėmis:

```
int x = 10;               // x yra non-const l-value
const int y = 20;         // yra yra const l-value
const int &ref  = x;      // OK, x yra non-const l-value
const int &ref2 = y;      // OK, y yra const l-value
const int &ref3 = 10;     // OK, 10 yra r-value
```

R-value nuorodos (&&) (1)

— **r-value nuorodos** (&&) inicializuojamos **tik** r-value reikšmėmis ir prailgina šių (r-value) objektų gyvavimą.

```
int x = 10;
int &lref = x;   // l-value nuoroda inicializuota l-reikšme x
int &&rref = 10; // r-value nuoroda inicializuota r-reikšme 10
```

— const l-value nuorodos irgi tą atlieka, tačiau r-value nuorodos leidžia keisti r-value reikšmes:

```
rref = 20; // r-value nuorada su vardu yra lvalue!;
void printElem(const std::vector<int>& v); // input: lvalue arba rvalue, bet negali modifikuot
void printElem(std::vector<int>&& v);      // Tik rvalue, bet gali modifikuoti
```

R-value nuorodos (&&) (2)

```cpp
#include<iostream>

bool checkIt(const int&) { return false; } // false = 0
bool checkIt(int&&) { return true; }       // true  = 1

int main() {
    int x = 5;
    std::cout << checkIt(x) << std::endl;  // Ką grąžins?
    std::cout << checkIt(5) << std::endl;  // Ką grąžins?
}
```

— Persidengiančiose funkcijose jei input yra r-value, tuomet pirmenybė teikiama versijai su r-value nuoroda, o ne su const l-value.

Klausimai?