Referencje

dr inż. Ireneusz Szcześniak

jesień 2017 roku

Referencje

- *Referencja* jest aliasem (inną nazwą) obiektu.
- Składnia referencji jest taka sama, jak nazwy obiektu.
- Referencja musi być zainicjalizowana, nie ma referencji "pustych", jak puste wskaźniki.
- Referencji nie można zmienić, żeby wskazać inny obiekt, tak jak wskaźnik.
- Można tworzyć pary (`std::pair`) i krotki (`std::tuple`) referencji, ale już nie kontenery czy tablice.
- Referencja nie zawsze ma miejsce w pamięci, bo może być wyoptymalizowana w czasie kompilacji.
- Główne zastosowanie:
  - przekazywanie argumentów wywołania funkcji przez referencję,
  - zwracanie wartości funkcji przez referencję,
  - pola składowe klasy.

Typy referencji

- `T &` - referencja typu l-wartość: służy do wskazania obiektu, który możemy modyfikować, ale nie przenosić, zakładając, że ten obiekt będzie później używany,
- `const T &` - referencja stała typu l-wartość: służy do wskazania obiektu, którego nie możemy modyfikować ani przenosić,
- `T &&` - referencja typu r-wartość: służy do wskazania obiektu, który możemy modifikować i przenosić, zakładając, że ten obiekt wkrótce zostanie zniszczony.

Nazwy skrócone:

- l-referencja to referencja typu l-wartość,
- r-referencja to referencja typy r-wartość.

Terminy l-wartość i r-wartość w nazwie typu

- Terminy l-wartość i r-wartość określają kategorię wartości wyrażenia:
  - wyrażenie `"1"` ma kategorię l-wartość,
  - wyrażenie `1` ma kategorię r-wartość.
- Ale słowa te mogą też definiować typ referencji:
  - `int &x;` - wyrażenie `x` ma typ "referencja typu l-wartość" i kategorię l-wartość,
  - `int &&x;` - wyrażenie `x` ma typ "referencja typu r-wartość" i kategorię l-wartość.

TERAZ ROZUMIEM: Even if the variable's type is rvalue reference, the expression consisting of its name is an lvalue expression.

L-referencja

L-referencja wskazuje l-wartość, a nie może wskazać r-wartości.

L-referencję definiujemy z użyciem `&`: `T &`

Jeżeli mamy funkcję `int &foo()`, to wyrażenie `foo()` jest l-wartością, dlatego tak można zainicjalizować l-referencję:
`int &z = foo();`

L-referencja - przykłady

```
int x = 1;
int y = 2;

// BŁĄD: brak inicjalizacji.
int &a;

// OK, raz na zawsze zainicjalizowane.
int &z = x;

// Wskaźnik na referencję to wskaźnik na obiekt.
assert(&z == &x);

// To nie jest reinicjalizacja!
z = y;
```

Stała l-referencja do r-wartości

To nie jest r-referencja!

Zasada wprowadzona w C++98, która obowiązuje do dzisiaj:

- stała l-referencja może wskazywać l-wartość lub r-wartość.

Po angielsku: const lvalue reference to rvalue

Kwalifikator `const` odnosi się do wskazywanego obiektu (to obiekt jest stały), a nie referencja, bo referencji i tak nie można zmienić.

Stała l-referencja do r-wartości - przykłady

- `int &a = 1;` nie będzie się kompilować, bo 1 jest r-wartością i chcemy ją wskazać niestałą l-referencją,
- `const int &b = 1;` będzie się kompilować, bo r-wartość wskazujemy stałą l-referencją,
- `int foo();` - deklaracja funkcji
- `const int &c = foo();` - OK!
- `const int *p = &foo();` - błąd!
- `const int *p = &c;` - OK!

R-referencja

R-referencja wskazuje r-wartość, a nie może wskazać l-wartości.

R-referencję definiujemy z użyciem &&: `T &&`

To nowość wprowadzona w C++11, żeby umożliwić:

- mechanizm przenoszenia obiektów,
- doskonałe przekazywanie argumentów funkcji.

R-referencja - przykłady

```
int x = 1;

// BŁĄD: brak inicjalizacji.
int &&a;

// BŁĄD: nie może wskazać l-wartości.
int &&z = x;

// OK: wskazujemy r-wartość.
int &&z = 1;
```

R-referencja do l-wartości

Możemy uzyskać r-referencję do l-wartości z użyciem operatora `static_cast<T &&>(expr)` lub funkcji `std::move(expr)`, gdzie `expr` może być l-wartością albo r-wartością. Funkcja `std::move` jest szablonowa i kompilator sam wnioskuje typ wyrażenia, którego nie trzeba już podawać, jak dla `static_cast`. Na przykład:

```
#include <utility>

class A {};

int main()
{
  A a;
  A &&r1 = static_cast<A &&>(a);
  A &&r2 = std::move(a);
}
```

Funkcji `std::move(x)` będziemy używać, aby umożliwić przenoszenie obiektu `x`, dla którego domyślnie przenoszenie nie jest stosowane, ponieważ wyrażenie `x` jest l-wartością.

Przeciążanie funkcji a referencje

Funkcję można przeciążyć różnymi typami referencyjnymi:

1. `void foo(T &);`
2. `void foo(const T &);`
3. `void foo(T &&);`

Dla wywołania funkcji `foo(expr)`, kompilator wybierze:

- przeciążenie nr 1, jeżeli `expr` jest l-wartością typu niestałego,
- przeciążenie nr 2, jeżeli `expr` jest l-wartością typu stałego,
- przeciążenie nr 3, jeżeli `expr` jest r-wartością.

Stała l-referencja (użyta w przeciążeniu nr 2) może wskazać l-wartość typu niestałego lub r-wartość, więc jeżeli nie ma przeciążenia nr 1 lub 3, kompilator wybierze przeciążenie nr 2.

Wiązanie obiektu tymczasowego przez referencję

Jeżeli obiekt tymczasowy jest wskazywany przez referencję, to jest on niszczony wtedy, kiedy referencja wychodzi poza zakres (ang. scope). Inaczej obiekt byłby niszczony po opracowaniu wyrażenia.

```
// Obiekt tymczasowy T() jest niszczony
// wtedy, kiedy t1 wychodzi poza zakres.
const T &t1 = T();

// To samo, ale dla r-referencji.
T &&t2 = T();
```

Nazywa się to wiązaniem obiektu tymczasowego przez referencję.

Podsumowanie

- Referencja to inna nazwa obiektu.
- Referencję trzeba zainicjalizować.
- Wartości referencji (wskazywanego obiektu) nie można zmienić.
- Podstawowe typy referencji:
  - l-referencja,
  - r-referencja.
- Niestała l-referencja może wskazać l-wartość, ale nie r-wartość.
- Stała l-referencja może wskazywać l-wartość lub r-wartość.
- R-referencja może wskazywać r-wartość, ale nie l-wartość.
- Stała r-referencja nie ma sensu.
- Referencje wiążą obiekty tymczasowe.

Dziękuję za uwagę.