Kategorie wartości wyrażeń

dr inż. Ireneusz Szcześniak

jesień 2017

1 Wprowadzenie

Różne operatory oczekują różnych kategorii wartości. Na przykład, operator wyłuskania oczekuje r-wartości, a wynikiem jego opracowania jest l-wartość. Na przykład:

```
int main()
{
  int &l = *(int *)0;
}
```

W języku C++, kategorie l-wartość i r-wartość nie definiuje się w odniesieniu do operatora przypisania. L-wartość nie zawsze może być lewej stronie operatora przypisania, na przykład:

```
int main()
{
  const int i = 1;
  &i; // jest to l-wartość, bo możemy pobrać adres
  // i = 2; // błąd!
}
```

Po lewej stronie przypisania nie musi być l-wartości, może być r-wartość, na rzecz której może być wykonany operator przypisania zdefiniowany przez programistę. Na przykład:

```
int main()
{
  struct A
  {
    void
    operator = (int i)
    {
    }
  };

  A() = 1;
}
```

2 Kategorie wartości obiektów tymczasowych

Obiekt tymczasowy może być tworzony, kiedy jest potrzebny jako argument wywołania funkcji. Wyrażenie, które jest argumentem wywołania fukcji i które jest miejscem tworzenia obiektu, jest r-wartością, ale ten sam obiekt jest wskazywany przez referencję, która jest parametrem funkcji, i która to referencja jest l-wartością (bo ma nazwę). Na przykład:

```
#include <iostream>

struct A
{
  A()
```

```
  {
    std::cout << "ctor:␣" << this << std::endl;
  }
};

void
foo(const A &a)
{
  std::cout << "foo:␣" << &a << std::endl;
}

int main()
{
  foo(A());
}
```

Podobnie jest w przypadku obiektów tymczasowych, które są rzucane jako wyjątek. Obiekt rzucany może być użyty w wyrażeniu kategorii r-wartość, a potem ten sam obiekt może być użytu w wyrażeniu kategorii l-wartość. Na przykład:

```
#include <iostream>

int main()
{
  struct A
  {
    A()
    {
      std::cout << "ctor:␣" << this << std::endl;
    }
  };

  try
    {
      throw A();
    }
  catch (A &a)
    {
      std::cout << "catch:␣" << &a << std::endl;
    }
}
```

Koniecznie `catch (A &a)` musi mieć referencję, bo inaczej otrzymamy kopię oryginalnego obiektu.

Co ciekawe, wyrażenie (czyli "A()"), które było argumentem instrukcji throw było r-wartością, a typem parametru bloku `catch` jest `A &a` niestałego typu referencyjnego. Zgodnie z zasadą C++98 r-wartość może być tylko wskazywana przez referencję stałą. Czyli powinno być `catch(const A &a)`, ale `catch(A &a)` też przechodzi, co uznaję za nieścisłość.

Niestety, bloki try i catch muszą mieć `{}`, nawet jeżeli zawierają pojedynczą instrukcję.

Można rzucić wyjątek dowolnej klasy. Ale czy można rzucić wartość typu bool, albo int, czy może nawet void?

3 Konwersja z l-wartości na r-wartość

Standard C++03 (czyli C++98 z poprawkami i małymi dodatkami) mówi:

> An lvalue of a non-function, non-array type T can be converted to an rvalue.

Operator + dla typów podstawowych, takich jak int, wymaga r-wartości jako swoich operandów. Zgodnie z zasadą wyżej, l-wartość może być poddana konwersji niejawnej do r-wartości, np.:

```
int main()
{
  int x = 1, y = 2;
  x + y;
}
```

Nie ma niejawnej konwersji z r-wartości do l-wartości.

4 L-wartości mogą być typów niepełnych

Wyrażenia kategorii l-wartość mogą być typów niepełnych, czyli nie w pełni zdefiniowanych. Wyrażenia kategorii r-wartość muszą być typów pełnych. Na przykład:

```
class B;

B &
boo()
{
}

int main()
{
  B &br = boo();
  // B(); - błąd, bo r-wartość
}
```

5 Kategorie wartości wyrażeń z funkcjami

Nazwa funkcji jest l-wartością, bo można pobrać jej adres `&boo`.

Wywołanie funkcji jest r-wartością, jeżeli funkcja zwraca typ niereferencyjny, na przykład:

```
int
roo()
{
}

int main()
{
   // &roo(); - błąd, bo r-wartość.
   // int &r = roo(); - błąd, bo r-wartość.
}
```

Wywołanie funkcji jest l-wartością, jeżeli funkcja zwraca typ referencyjny, na przykład:

```
int &
loo()
{
}

int main()
{
  &loo(); // OK, bo l-wartość.
  int &l = loo(); // OK, bo l-wartość.
}
```