БУДУЩЕЕ
В НАШИХ
РУКАХ

Lifetime extension

Елена Степанова
21.11.2023

Что такое lifetime extension и временные объекты

Когда lifetime extension работает

Когда lifetime extension не работает

Избитый пример

```
struct Data
{
   // ...
   std::vector<int> const& items();
};

Data getData();

for (auto const& item : getData().items())
{
   // ...
}
```

Что такое lifetime extension? (ISO\IEC 14882 пар. 6.7.7.6)

5 There are three contexts in which temporaries are destroyed at a different point than the end of the full-expression. The first context is when a default constructor is called to initialize an element of an array with no corresponding initializer (9.4). The second context is when a copy constructor is called to copy an element of an array while the entire array is copied (7.5.5.2, 11.4.4.2). In either case, if the constructor has one or more default arguments, the destruction of every temporary created in a default argument is sequenced before the construction of the next array element, if any.

Что такое lifetime extension? (ISO\IEC 14882 пар. 6.7.7.6)

6 The third context is when a reference is bound to a temporary object.[36] The temporary object to which the reference is bound or the temporary object that is the complete object of a subobject to which the reference is bound persists for the lifetime of the reference if the glvalue to which the reference is bound was obtained through one of the following:

(6.1) — a temporary materialization conversion (7.3.4),

(6.2) — **(** ***expression*** **)**, where expression is one of these expressions,

(6.3) — subscripting (7.6.1.1) of an array operand, where that operand is one of these expressions,

(6.4) — a class member access (7.6.1.4) using the . operator where the left operand is one of these expressions and the right operand designates a non-static data member of non-reference type,

(6.5) — a pointer-to-member operation (7.6.4) using the .* operator where the left operand is one of these expressions and the right operand is a pointer to data member of non-reference type,

(6.6) — a

(6.6.1) — **const_cast** (7.6.1.10),

(6.6.2) — **static_cast** (7.6.1.8),

(6.6.3) — **dynamic_cast** (7.6.1.6), or

(6.6.4) — **reinterpret_cast** (7.6.1.9)

converting, without a user-defined conversion, a glvalue operand that is one of these expressions to a glvalue that refers to the object designated by the operand, or to its complete object or a subobject thereof,

(6.7) — a conditional expression (7.6.16) that is a glvalue where the second or third operand is one of these expressions, or

(6.8) — a comma expression (7.6.20) that is a glvalue where the right operand is one of these expressions.

Что такое lifetime extension?

- Если во время инициализации или копирования массивов возникают временные объекты, время их жизни продляется до конца инициализации элемента массива

- Когда есть ссылка на временный объект по T const& или T&&, время жизни временного объекта продляется на время жизни ссылки

Что такое временный объект?

6.7.7 Temporary objects [class.temporary]

1 Temporary objects are created

(1.1) — when a prvalue is converted to an xvalue (7.3.4)

(1.2) — when needed by the implementation to pass or return an object of trivially-copyable type (see below), and

(1.3) — when throwing an exception (14.2). [*Note*: The lifetime of exception objects is described in 14.2. — *end note*]

Что такое временный объект?

2 The materialization of a temporary object is generally delayed as long as possible in order to avoid creating unnecessary temporary objects. [*Note*: Temporary objects are materialized:

(2.1) — when binding a reference to a prvalue (9.4.3, 7.6.1.3, 7.6.1.6, 7.6.1.8, 7.6.1.10, 7.6.3),

(2.2) — when performing member access on a class prvalue (7.6.1.4, 7.6.4),

(2.3) — when performing an array-to-pointer conversion or subscripting on an array prvalue (7.3.2, 7.6.1.1),

(2.4) — when initializing an object of type **std::initializer_list<T>** from a *braced-init-list* (9.4.4),

(2.5) — for certain unevaluated operands (7.6.1.7, 7.6.2.4), and

(2.6) — when a prvalue that has type other than cu void appears as a discarded-value expression (7.2).

Что такое временный объект?

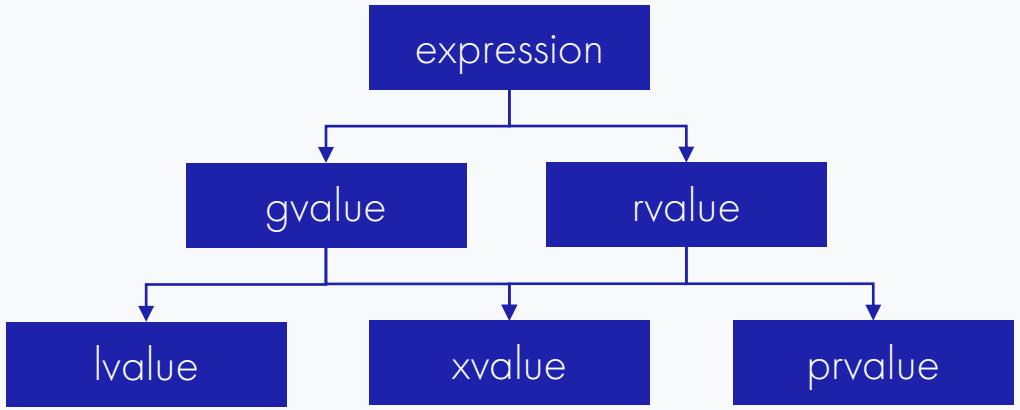
6.7.7 Temporary objects [class.temporary]

1 Temporary objects are created

(1.1) — when a prvalue is converted to an xvalue (7.3.4)

(1.2) — when needed by the implementation to pass or return an object of trivially-copyable type (see below), and

(1.3) — when throwing an exception (14.2). [*Note*: The lifetime of exception objects is described in 14.2. — *end note*]



Про категории выражений:

- Стандарт – 8.2.1 Value category
- "New" Value Terminology (B.Stroustrup)

Что такое временный объект?

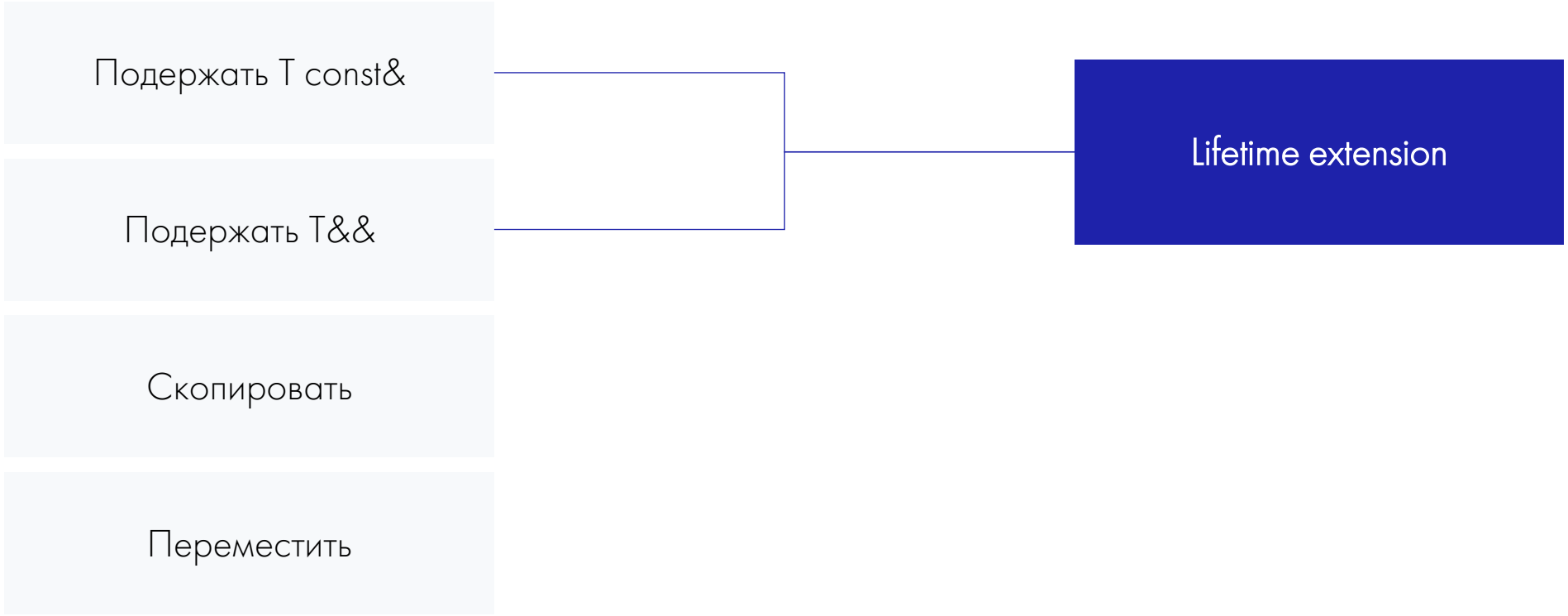
Объект, который должен уничтожиться где-то недалеко в поле зрения

Временный объект живет до конца определения значения выражения

4 When an implementation introduces a temporary object of a class that has a non-trivial constructor (11.4.4.1, 11.4.4.2), it shall ensure that a constructor is called for the temporary object. Similarly, the destructor shall be called for a temporary with a non-trivial destructor (11.4.6). Temporary objects are destroyed as the last step in evaluating the full-expression (6.9.1) that (lexically) contains the point where they were created. This is true even if that evaluation ends in throwing an exception. The value computations and side effects of destroying a temporary object are associated only with the full-expression, not with any specific subexpression.

(Чаще всего – до ";")

Что можно сделать с временным объектом?



Note: Copy elision может выглядеть похоже на обращение по ссылке, но это не lifetime extension

Note 2: Перемещение может выглядеть похоже на обращение по ссылке, но это не lifetime extension

Что такое lifetime extension и временные объекты

Когда lifetime extension работает

Когда lifetime extension не работает

Для T const& или T&&

```
class Data {};

auto const& data1 = Data(); // lifetime extension
auto&& data2 = Data(); // lifetime extension

struct S
{
   Data d;
};

auto const& data3 = S().d; // lifetime extension
```

Когда T const& или T&& инициализируется результатом выражения (например, вызова функции), возвращающего:

- временный объект T, или
- объект T внутри временного объекта (например, структуры, содержащей T)

Что такое lifetime extension и временные объекты

Когда lifetime extension работает

Когда lifetime extension не работает

При не прямом доступе к вложенному объекту

```
struct S
{
   Data d;
   Data const& getData() const { return d; }
};

auto const& data1 = S().d; // lifetime extension
auto const& data2 = S().getData(); // possibly dangling
reference
```

TIP: Инициализируя ссылку другой ссылкой, убедитесь, что берете её не из временного объекта

Copy elision – это другое

```
Data getData()
{
   Data d;
   return d;
}

auto data1 = getData(); // Data, NRVO
```

Copy elision – это другое

```
Data getData()
{
   Data d;
   return std::move(d);
}

auto data1 = getData(); // Move ctor, no NRVO
```

TIP: std::move ломает NRVO!

Copy elision – это другое

```
Data getData()
{
   Data d;
   return d;
}

auto data1 = getData(); // Data, NRVO
decltype(auto) data2 = getData(); // Data, NRVO
decltype(getData()) data3 = getData(); // Data, NRVO
decltype(std::declval<Data>()) data4 = getData(); //
Data&&, lifetime extension
```

TIP: std::move ломает NRVO!

TIP 2: любой decay ломает lifetime extension

Copy elision – это другое

```
Data getData()
{
   Data d;
   return d;
}

template <class T>
void f(T par) {}

f(data3); // Data, copy
```

TIP: std::move ломает NRVO!

TIP 2: любой decay ломает lifetime extension, в т.ч. выведение шаблонных типов

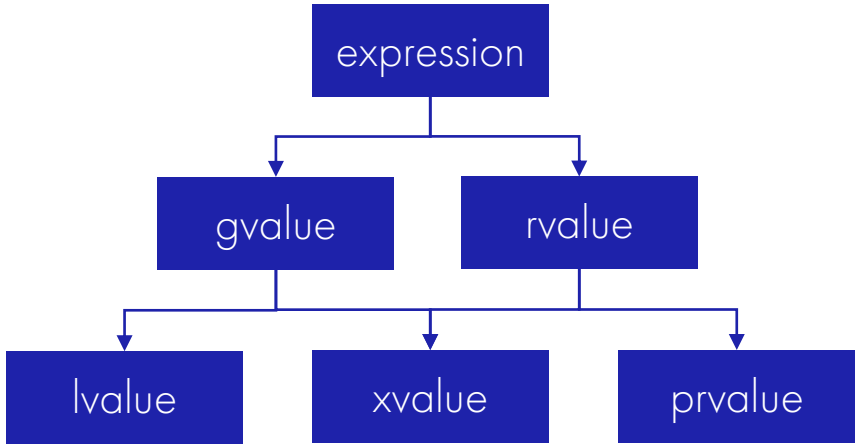
В range based for по временному объекту

```
struct Data
{
   // ...
   std::vector<int> const& items();
};

Data getData();

for (auto const& item : getData().items()) { /* ... */ }

// Workaround since C++20
for (Data d = getData(); auto const& item d.items())
{ /* ... */ }
```

В C++23 обещали починить.

Для xvalue

```
Data getData()
{
   Data data;
   return data;
}

Data const& data1 = getData(); // lifetime extension
// dangling reference
Data const& data2 = std::move(getData());
// dangling reference
Data&& data3 = std::move(getData());
```



Для xvalue

```
17  Data getData()
18  {
19      Data data;
20      return data;
21  }
22
23  int main()
24  {
25      auto const& data = getData();
26      auto const& data1 = std::move(getData());
27      auto&& data2 = std::move(getData());
28  }
29
```

Output of x86-64 gcc 13.2 (Compiler #1)

A ▾ | Wrap lines | Select all

```
Compiler returned: 0
```

Output of x86-64 clang (trunk) (Compiler #2)

A ▾ | Wrap lines | Select all

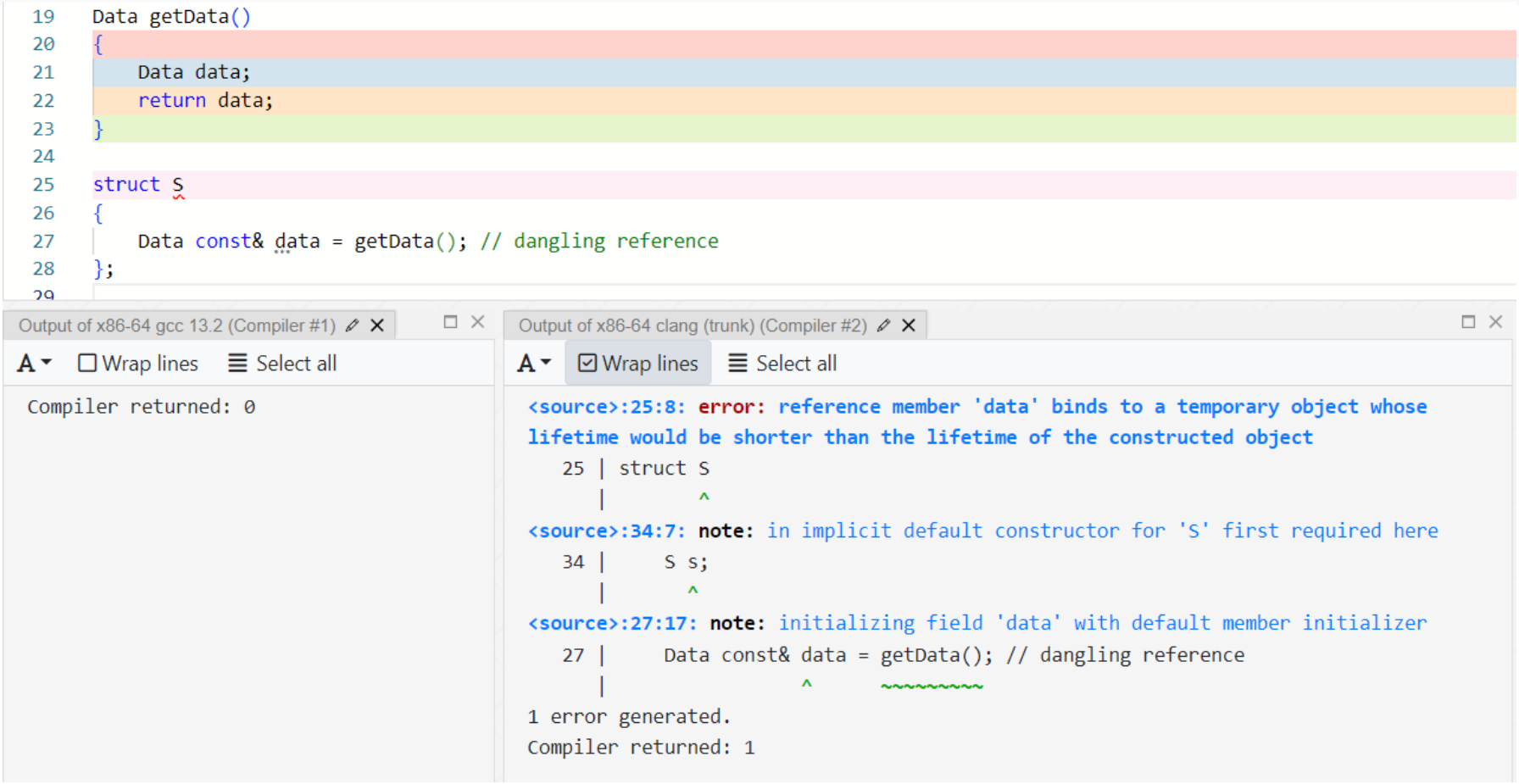
```
<source>:26:35: warning: temporary bound to local reference 'data1'
will be destroyed at the end of the full-expression [-Wdangling]
   26 |     auto const& data1 = std::move(getData());
      |                                   ^~~~~~~~~
<source>:27:30: warning: temporary bound to local reference 'data2'
will be destroyed at the end of the full-expression [-Wdangling]
   27 |     auto&& data2 = std::move(getData());
      |                              ^~~~~~~~~
```

godbolt.org/z/P1qvn3aYe

При хранении ссылки на временный объект полем класса

```
Data getData()
{
   Data data;
   return data;
}

struct S
{
   Data const& data = getData(); // dangling reference
};
```

При хранении ссылки на временный объект полем класса



godbolt.org/z/hb9zMnMb5

С тернарным оператором никаких гарантий

```
Data getData()
{
   Data data;
   return data;
}

Data&& refData = getData();
Data const& crefData = getData();

decltype(auto) data1 = condition ? crefData : crefData;
decltype(auto) data2 = condition ? refData : refData;
// probably copy
decltype(auto) data3 = condition ? refData : crefData;
```

Не всегда работает

- При использовании не полиморфных преобразований типов
- Для RAII объектов (зависит от уровня оптимизации)
- При использовании std::reference_wrapper
- При использовании std::ranges
- При передаче ссылок на временные объекты в operator new
- Для ссылки на элемент временного массива (зависит от компилятора)
- ...

Рекомендации

- T const& и T&& уместны на коротких дистанциях
  - Долго хранить временный объект – это 🏳🏳🏳
- Осторожно использовать ссылки на ссылки
- Избегать:
  - Хранить ссылки полем класса
    - Умные указатели всегда безопаснее
    - ...и даже глупые могут быть безопаснее
  - Хранить ссылки в контейнерах
- Осторожно итерироваться по временным объектам

БУДУЩЕЕ
В НАШИХ
РУКАХ