

const reference to a temporary object becomes broken after function scope (life time)

Ask Question

Asked 8 years, 8 months ago   Modified 7 years, 9 months ago   Viewed 2k times

While asking [this question](#), I learned const reference to a temporary object is valid in C++:

```
int main ()
{
    int a = 21;
    int b = 21;

    //error: invalid initialization of non-const reference
    //int      sum = a + b;e [...]

    //OK
    int const & sum = a + b;

    return sum;
}
```

But in the following example, the const reference `refnop` refers to a destroyed temporary object. I wonder why?

```
#include <string>
#include <map>

struct A
{
    // data
    std::map <std::string, std::string>  m;
    // functions  用户没有提供没有构造函数
    const A& nothing() 成员函数 nothing() const { return *this;  }
    void init() 成员函数init() { m["aa"] = "bb";  }
    bool operator!=( A const& a) const { return a.m != m;  }
    // 不等于号
};

int main()
{
    A a;
    a.init();

    A const& ref = A(a);
    A const& refnop = A(a).nothing();

    int ret = 0;
    if (a != ref)      ret += 2;
    if (a != refnop)  ret += 4;

    return ret;
}
```

Tested using GCC 4.1.2 and MSVC 2010, it returns 4;

```
$> g++ -g refnop.cpp
$> ./a.out ; echo $?
```

The difference between `ref` and `refnop` is the call to `nothing()` which does really nothing. It seems after this call, the temporary object is destroyed!

My question:

**Why in the case of `refnop`, the life time of the temporary object is not the same as its const reference?**

`c++` `reference` `scope` `temporary-objects` `const-reference`

Share Edit Follow Flag

edited May 23, 2017 at 10:33

asked Jun 21, 2013 at 11:28

Community Bot

1 1

oHo

44.8k 26 149 187

Caution: using g++ versions 4.4 and 4.6, this snippet returns 0... – [oHo](#) Jun 13, 2014 at 11:48

Add a comment

Start a bounty

## 2 Answers

The lifetime-extension of a temporary object can be performed **only once**, when the temporary object gets bound to the **first reference**. After that, the knowledge that the reference refers to a temporary object is gone, so further lifetime extensions are not possible.

The case that is puzzling you

```
A const& refnop = A(a).nothing();
```

is similar to this case:

```
A const& foo(A const& bar)
{
    return bar;
}
//...
A const& broken = foo(A());
```

In both cases, **the temporary gets bound to the function argument** (the implicit `this` for `nothing()`, `bar` for `foo()`) and gets its lifetime 'extended' to the lifetime of the function argument. I put 'extended' in quotes, because the natural lifetime of the temporary is already longer, so no actual extension takes place.

Because the lifetime extension property is non-transitive, returning a reference (that happens to refer to a temporary object) will not further extend the lifetime of the temporary object, with as result that **both `refnop` and `broken` end up referring to objects that no longer exist**.

Share Edit Follow Flag

answered Jun 21, 2013 at 12:09

Bart van Ingen Schenau

14.9k 4 30 41

While other typing rules in C++ are so strict to protect against error, the fact that the C++ standard so secretly allows using a temporary object to initialize a const reference and then allows the lifetime of the temporary object to expire if the reference is passed too many times is very frustrating. Certainly can't trust keeping references to const as class members. – [Jason](#) Jun 24, 2020 at 3:41

Add a comment

My original example is complex.

Therefore I post here a simpler example and I provide the corresponding [ISO C++ standard](#) paragraph.

This simpler example is also available on [coliru.stacked-crooked.com/](#)

```
#include <iostream>

struct A
{
    A(int i) { std::cout<<"Cstr "<<i<<"\n"; p = new int(i); }
    ~A() { std::cout<<"Dstr "<<*p<<"\n"; delete p; }

    const A& this() const { return *this; }

    int *p;
};

const A& constref( const A& a )
{
    return a;
}

int main()
{
    const A& a4 = A(4);
    const A& a5 = A(5).this();
    const A& a6 = constref( A(6) );

    std::cout << "a4 = "<< *a4.p <<"\n";
    std::cout << "a5 = "<< *a5.p <<"\n";
    std::cout << "a6 = "<< *a6.p <<"\n";
}
```

The output using command line `g++-4.8 -std=c++11 -O2 -Wall -pedantic -pthread main.cpp && ./a.out`:

```
Cstr 4
Cstr 5
Dstr 5
Cstr 6
Dstr 6
a4 = 4
a5 = 0
a6 = 0
Dstr 4
```

As you can see, the temporary objects referenced by `a5` and `a6` are destructed at the end of functions `this` and `constref` respectively.

This is an extract of *§12.2 Temporary objects*, where the bold part applies in this case:

The second context is when a reference is bound to a temporary. The temporary to which the reference is bound or the temporary that is the complete object of a subobject to which the reference is bound persists for the lifetime of the reference except:

- A temporary bound to a reference member in a constructor's ctor-initializer (12.6.2) persists until the constructor exits.
- A temporary bound to a reference parameter in a function call (5.2.2) persists until the completion of the full-expression containing the call.

- The lifetime of a temporary bound to the returned value in a function return statement (6.6.3) is not extended; the temporary is destroyed at the end of the full-expression in the return statement.**

- A temporary bound to a reference in a *new-initializer* (5.3.4) persists until the completion of the full-expression containing the *new-initializer*.

This is a more complete example:

```
#include <iostream>

struct A
{
    A() { std::cout<<"Cstr 9\n"; p = new int(v = 9); }
    A(int i) { std::cout<<"Cstr "<<i<<"\n"; p = new int(v = i); }
    A(const A&o) { std::cout<<"Copy "<<o.v<<"\n"; p = new int(v = 10+o.v); }
    ~A() { std::cout<<"Del "<<*p<<"\n"; *p = 88; delete p; }

    const A& this() const { return *this; }

    int *p;
    int v;
};

const A& constref( const A& a )
{
    return a;
}

std::ostream& operator<<( std::ostream& os, const A& a )
{
    os <<"{ *p="<< *a.p <<" , v="<< a.v <<" }\n";
    return os;
}

int main()
{
    std::cout << "----const A a1 = A(1)" <<"\n";
    const A a1 = A(1);
    std::cout << "----const A a2 = A(2).this()" <<"\n";
    const A a2 = A(2).this();
    std::cout << "----const A a3 = constref( A(3) )" <<"\n";
    const A a3 = constref( A(3) );
    std::cout << "----const A& a4 = A(4)" <<"\n";
    const A& a4 = A(4);
    std::cout << "----const A& a5 = A(5).this()" <<"\n";
    const A& a5 = A(5).this();
    std::cout << "----const A& a6 = constref( A(6) )" <<"\n";
    const A& a6 = constref( A(6) );
    Del 6 6
    a1 = { *p=1 , v=1 }
    a2 = { *p=12 , v=12 }
    a3 = { *p=13 , v=13 }
    a4 = { *p=4 , v=4 }
    a5 = { *p=0 , v=5 }
    a6 = { *p=0 , v=6 }
    Del 4 4
    Del 13 13
    Del 12 12
    Del 1 1
```

And the corresponding output using same `g++` command line:

```
----const A a1 = A(1)
Cstr 1
----const A a2 = A(2).this()
Cstr 2
Copy 2
Del 2 2
----const A a3 = constref( A(3) )
Cstr 3
Copy 3
Del 3 3
----const A& a4 = A(4)
Cstr 4
----const A& a5 = A(5).this()
Cstr 5
Del 5 5
----const A& a6 = constref( A(6) )
Cstr 6
Del 6 6
a1 = { *p=1 , v=1 }
a2 = { *p=12 , v=12 }
a3 = { *p=13 , v=13 }
a4 = { *p=4 , v=4 }
a5 = { *p=0 , v=5 }
a6 = { *p=0 , v=6 }
Del 4 4
Del 13 13
Del 12 12
Del 1 1
```

Share Edit Follow Flag

edited May 23, 2017 at 10:28

answered Jun 13, 2014 at 16:34

Community Bot

1 1

oHo

44.8k 26 149 187

Add a comment

The Overflow Blog

Welcoming the new crew of Stack Overflow podcast hosts

Rewriting Bash scripts in Go using black box testing

Featured on Meta

Stack Exchange Q&A access will not be restricted in Russia

Planned maintenance scheduled for Friday, March 18th, 00:30-2:00 UTC...

Announcing an A/B test for a Trending sort option

Improving the first-time asker experience - What was asking your first...

## Linked

- 1
- returning reference to temporary object
- 2
- C++11 move to local const reference: scope
- 17
- Temporary lifetime extension
- 9
- ISO C++ standard draft
- 8
- Is reference binding to a temporary of a temporary undefined behavior?
- 3
- c++: const reference to an rvalue

## Related

- 241
- How come a non-const reference cannot bind to a temporary object?
- 6
- Difference between reference and const reference as function parameter?
- 188
- Does a const reference class member prolong the life of a temporary?
- 23
- why copy constructor is called when passing temporary by const reference?
- 9
- What exactly happens when returning const reference to a local object?
- 6
- Is the lifetime of a C++ temporary object created in ?: expression extended by binding it to a local const reference?
- 0
- Prolonging life of a temporary object using const reference

## Hot Network Questions

- Why are there so few democratic governments with multiple rulers?
- Sort a Python list of strings where each item is made with letters and numbers
- Did Shackleton's crew all "miss out" on WW1?
- Why does Samarium have such a large c-axis lattice constant? (>26 Å) Are there significantly larger lattice constants of the elements?
- How can I calculate the average gas cost per a transaction on RSK?
- Open Source License restrictions and recent sanctions against Russia
- How can I change binary numbers from from top to bottom?
- Why do quantum computers have more qubits than classical computers have bits?
- What is meant by "without resorting to the sexton's spade that buried Jacob Marley" in A Christmas Carol?
- Any name for this special function?
- Speak the password and enter, if you dare
- Concrete vectors spaces without an obvious basis or many "obvious" bases?
- Count the Liberties
- Is there a way to add INCLUDE for a UNIQUE NONCLUSTERED index?
- What's the origin of the Mimic?
- Why are square roots of lines in the complex plane hyperbolas?
- my new convection/convention oven will not bake anything correctly
- Upgrade to 5.47.x fails with error: Exception: "API error: DB Error: constraint violation on ReportTemplate.create"
- Restart potential energy scan in Gaussian with additional scan points
- If an employer owns any work you produce whilst employed, do you have to be unemployed before you can begin a startup?
- On what grounds did Vladimir Putin invoke Article 51 of the UN Charter for self defence while going into Ukraine?
- Conceptual reason why the sign of a permutation is well-defined?
- Does every monoidal category admit a braiding?
- Does my PhD dissertation need a copyright notice?

Question feed