Returning Rvalue Reference and Temporary Materialization

Asked 1 year, 11 months ago Modified 1 year, 11 months ago Viewed 106 times



Consider the following functions. I'd like answers for **C++17**.









```
MyClass&& func() {
   return MyClass{};
}

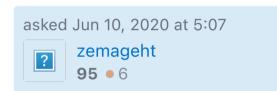
int main() {
   MyClass&& myRef = func();
}
```

Questions:

- 1. Is the expression func() an xvalue? Why?
- 2. Why is myRef a dangling reference? Or, more specifically, why is func() returning a dangling reference? Wouldn't returning rvalue reference cause temporary materialization, and extend the temporary object's lifetime?



Share Edit Follow Flag



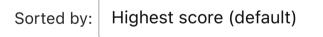


"Is the expression func() an xvalue? Why?" ... why wouldn't it be? - Nicol Bolas Jun 10, 2020 at 5:17

Add a comment

Start a bounty

1 Answer





func() is an xvalue because one of the rules of the language is that if a function is declared to have a return type of rvalue reference to object, then an expression consisting of calling that function is an xvalue. (C++17 expr.call/11).



Temporary materialization occurs any time a reference is bound to a prvalue.



The *result* of the function is <code>myRef</code> which is initialized by the prvalue <code>func()</code>. However if we consult the lifetime extension rules in class.temporary/6 it has:



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

So the temporary object materialized by func() is destroyed when the return statement completes, with no extension.

Share Edit Follow Flag

