ANSI Doc No: X3J16/93-0098
ISO Doc No: WG21/N0305
Date: July 8, 1993
Project: Prog. Language C++
Reply To: Dag Brück
dag@control.lth.se

Proposal for Lifetime of Temporaries

Dag M. Brück
Department of Automatic Control
Lund Institute of Technology
Box 118, S-221 00 Lund, Sweden

1. Abstract

The paper proposes a resolution to the lifetime of temporaries in C++. The main rule is that temporaries are destroyed at the end of a full expression.

The paper also contains a review of the alternatives discussed at meetings and the mail reflectors recently, and a few examples.

2. Proposal

The proposal consists of two rules that cover all temporaries, and one rule about function parameters.

R1. Temporaries are destroyed at the end of the full expression in which they are created.

R2. Temporaries, like all C++ objects, are destroyed in reverse order of construction.

R3. The lifetime of a parameter is that of the body of its function, cf. Section 5.2.2 of the Working Paper.

The term *full expression* is defined in Section 3.6 of [ISO, 1990] and [ANSI, 1989] as follows:

> A full expression is an expression that is not part of another expression. Each of the following is a full expression: an initializer; the expression in an expression statement; the controlling expression of a selection statement (`if` or `switch`); the controlling expression of a `while` or `do` statement; each of the three (optional) expressions of a `for` statement; the optional expression in a `return` statement. The end of a full expression is a sequence point.

See also Section 3.3 of [ISO, 1990] for a discussion of sequence points in expressions, especially the operators `()`, `&&`, `||` and `?:`.

Also note that a reference may be bound to a temporary, in which case the temporary has the lifetime of the enclosing scope, see Section 8.4.3 of the Working Paper.

3. Discussion

The lifetime of temporaries is one of the most important issues of the standardization committee. It is undefined in the base documents, but the committee has indicated that any resolution is better than the current situation.

The lifetime of a temporary is generally categorized as "short" if there is a risk that it may be destroyed before use, or "long" if there is a risk of running out of memory because temporaries are not destroyed in time. Any resolution will probably break at least a few existing programs.

Alternatives

Many different resolutions may be considered for the destruction of temporaries:

- After first use
- At end of statement
- At the next branching point
- At end of block
- At end of function
- After last use (which effectively requires garbage collection)
- Leave undefined (the current situation).

Conditional expressions (i.e., `&&`, `||` and `?:`) which generate branching complicate the issue further. There are two possible resolutions:

- Use the same rule for all temporaries, which means that the implementation may have to generate additional code and variables to keep track of which temporaries were actually created.
- Add a rule which says that intermediate temporaries created in `&&`, `||` and `?:` are destroyed at the end of the conditional branch, i.e., earlier than other temporaries.

A few papers have been written on the issue. The most important paper is [Koenig, 1992], which has the basic discussion and many examples. Turner has argued in favour of very short-lived temporaries [Turner, 1991], and Pennello has argued against destruction at end-of-block [Pennello, 1992].

Boston straw vote

A straw vote was conduted at the Boston meeting in November 1992, in order to determine the committee's preferences. People could indicate which (possibly many) alternatives they were willing to accept, and which they could not accept. For unconditionally constructed temporaries the alternatives were

1. Destroy at end of statement (EOS)
2. Destroy at end of block (EOB)

For temporaries created in conditional expressions the alternatives were

a. Destroy at the end of the conditional branch (EOCB)
b. Destroy with other temporaries (which may require setting runtime flags)

The result of the straw vote was:

	Point of destruction	Willing to accept	Cannot accept
1a	EOS + EOCB	38	0
2a	EOB + EOCB	25	6
1b	EOS + EOS	34	1
2b	EOB + EOB	28	8
3	Unspecified	6	32

Leaving the lifetime unspecified is clearly not an acceptable situation. There is in my view a small but significant preference for destruction at end-of-statement over destruction at end-of-block.

The case for EOCB (which is not suggested in this paper) is not strong. EOCB is more difficult to describe than destroying all temporaries at EOS, although EOCB seems to be easier to implement.

Mail reflector discussion

During subsequent exchanges on the extensions mail reflector, it was shown that the phrase "end of statement" is not quite appropriate, although there was agreement on what it should mean. The appropriate phrase seems to be "end of full expression," which is defined in Standard C.

There was also some discussion if sequence points (also defined in Standard C) could be used for defining when temporaries are destroyed. Sequence points cannot be used unconditionally; for example, there is a sequence point after an argument list has been evaluated but before the function is called. The phrase "end of full expression" seems to handle all cases.

Examples

Most examples used in the discussion are presented in [Koenig, 1992], and will not be repeated here. However, a few "new" ones have shown up recently on the mail reflector.

In the examples that follow, I will use a string class which uses `operator +` for concatenation, and has a member function `cstr()` which returns a pointer to internal storage. The value returned by `cstr()` is guaranteed to be garbage when the corresponding string object has been destroyed. The names used in the examples are:

```
String s, t, u;
const char* p;
int n;
extern int g(const String &);
extern String f();
extern String operator + (const String &, const String &);
```

The first question is what temporaries we discuss in the case of `?:` expressions. This case presents no problems:

```
u = reverse ? t + s : s + t;
```

What must happen is that either `t+s` or `s+t` must stay around long enough to copy its value into the temporary that holds the result of `?:`. Then the latter temporary is yielded as the result. After that the temporaries are destroyed in reverse order. In other words:

```
if reverse goto X;
construct leftbranch = 1;
construct temp1 = s + t;
```

```
   construct temp0 = temp1;
   goto Y;
X: construct leftbranch = 0;
   construct temp2 = t + s;
   construct temp0 = temp2;
Y: u = temp0;
   destroy temp0;
   if leftbranch goto A;
   destroy temp2;
   goto B;
A: destroy temp1;
B:
```

Here, `temp0` represents the result of `?:` and should stay around until end of full expression as usual. It's `temp1` and `temp2` that may or may not need destruction at the end of the full expression, depending on the condition.

Now of course in this example, since `temp0`, `temp1`, and `temp2` are the same type, it is possible as an optimization to alias them:

```
   if reverse goto X;
   construct temp0 = s + t;
   goto Y;
X: construct temp0 = t + s;
Y: u = temp0;
   destroy temp0;
```

but that is perhaps beside the point.

The next two are more interesting. They will both work under the proposed rules, but only the second one will work under the EOCB rule. The first one will not work under EOCB, because `cstr()` returns a pointer to internal storage used by the intermediate temporary created for `t+s` or `s+t`; this storage has been destroyed when `strlen()` is called, together with the associated string object.

```
n = strlen(reverse ? (t + s).cstr() : (s + t).cstr());
n = strlen((reverse ? t + s : s + t).cstr());
```

A similar case which will work under the proposed rules but not EOCB is:

```
puts(s.length() + t.length() ? (char *) s+t : "none");
```

Reference parameters bound to temporaries exist for the duration of the function call, so the value returned by `f()` is available in `g()`:

```
g(f());
```

A general observation is that all temporaries created in evaluating the arguments of a function call exist for the duration of the called function; that may not be true under the EOCB rule.

Besides the issue of "initializer within the full expression" there is also the issue of "full expression within the initializer:"

```
const int& y = g(f());
```

Does the temporary created to initialize the argument of `g()` last as long as the reference `y`? The answer is no, because `y` is not bound directly to the value returned by `f()`.

If is also worth noting that left-hand and right-hand parts of a comma-expression are not full expressions:

```
e1, e2;
```

Temporaries created in `e1` are not destroyed until the end of the statement, which has been used to implement locking, for example. This behaviour is different from

```
e1; e2;
```

where the temporaries are destroyed at the end of each expression statement.

4. Acknowledgements

Most of this document is directly based on discussion on the extensions mail reflector. I would like to thank David Cok, Bill Gibbons, James Kanze, Andrew Koenig, Richard Minner, Tom Plum, Martin O'Riordan, Jerry Schwarz, John Skaller and Mark Terribile, for their contributions.

5. References

ANSI (1989): "Programming Language C, American National Standard X3.159-1989." Technical Report, American National Standards Institute.

ISO (1990): "Programming languages — C, International standard ISO/IEC 9899." Technical Report, International Standards Organization.

KOENIG, A. (1992): "Lifetime of temporaries." Technical Report, AT&T Bell Laboratories. Document numbers X3J16/92-0020 and WG21/N0098.

PENNELLO, T. (1992): "Some issues in temporaries destroyed at end of block." Technical Report, MetaWare, Inc. Document numbers x3J16/92-0126 and WG21/N0203.

TURNER, P. K. (1991): "Proposal for short-lived temporary objects." Technical Report, Language Processors, Inc. ANSI document number X3J16/91-0019.