Decltype (revision 7): proposed wording

Programming Language C++

Document no: N2343=07-0203

Jaakko Järvi
Texas A&M University
College Station, TX
jarvi@cs.tamu.edu

Bjarne Stroustrup
AT&T Research
and Texas A&M University
bs@research.att.com

Gabriel Dos Reis
Texas A&M University
College Station, TX
gdr@cs.tamu.edu

2007-07-18

1 Introduction

We suggest extending C++ with the `decltype` operator for querying the type of an expression.

This document is a revision of the documents N2115=06-0185 [JSR06b], N1978=06-0048 [JSR06a], N1705=04-0145 [JSR04], 1607=04-0047 [JS04], N1527=03-0110 [JS03], and N1478=03-0061 [JSGS03], and builds also on [Str02]. We only include the proposed wording; for rationale and other discussion of the feature, see the earlier revisions.

2 Proposed wording

Section 2.11 Keywords [lex.key]

Add `decltype` to Table 3.

Chapter 5 Expressions [expr]

Paragraph 8 should read:

> ~~Clause 5 specifies for some operators that some of their operands are *unevaluated operands*~~ In some contexts, *unevaluated operands* appear (5.2.8, 5.3.3, 7.1.5.2). An unevaluated operand is not evaluated. [*Note:* In an unevaluated operand, a non-static class class member may be named (5.1) and naming of objects or functions does not, by itself, require that a definition be provided (3.2). — *end note*]

Section 7.1.5 Type specifiers [dcl.type]

Change paragraph 1 as indicated:

> As a general rule, at most one *type-specifier* is allowed in the complete *decl-specifier-seq* of a *declaration*. The only exceptions to this rule are the following:

— `const` can be combined with any other type specifier except itself. ~~`const` and `volatile` can be combined with any other type specifier. However, redundant cv-qualifiers are prohibited except when introduced through the use of typedefs (7.1.3), or template type arguments (14.3), in which case the redundant cv-qualifiers are ignored.~~

— `volatile` can be combined with any other type specifier except itself.

Section 7.1.5.1 The cv-qualifiers [dcl.type.cv]

Paragraph 1 should be:

There are two *cv-qualifiers*, `const` and `volatile`. If a *cv-qualifier* appears in a *decl-specifier-seq*, the *init-declarator-list* of the declaration shall not be empty. [*Note:* 3.9.3 describes how cv-qualifiers affect object and function types. — *end note*] Redundant cv-qualifications are ignored. [*Note:* For example, those could be introduced by using typedefs. — *end note*]

Section 7.1.5.2 Simple type specifiers [dcl.type.simple]

In paragraph 1, add the following to the list of simple type specifiers:

`decltype` (*expression*)

To Table 9, add the line:

`decltype` (*expression*)	the type as defined below

Add a new paragraph after paragraph 3:

The type denoted by `decltype(e)` is defined as follows:

1. If `e` is an *id-expression* or a class member access (5.2.5 [expr.ref]), `decltype(e)` is defined as the type of the entity named by `e`. If there is no such entity, or `e` names a set of overloaded functions, the program is ill-formed.
2. If `e` is a function call (5.2.2 [expr.call]) or an invocation of an overloaded operator (parentheses around `e` are ignored), `decltype(e)` is defined as the return type of that function.
3. Otherwise, where `T` is the type of `e`, if `e` is an lvalue, `decltype(e)` is defined as `T&`, otherwise `decltype(e)` is defined as `T`.

The operand of the `decltype` specifier is an unevaluated operand (clause 5 [expr]).

[*Example:*

```
const int&& foo();
int i;
struct A { double x; }
const A* a = new A();
decltype(foo()); // type is const int&&
decltype(i);     // type is int
decltype(a->x);  // type is double
decltype((a->x)); // type is const double&
```

— *end example*]

Section 14.6.2.1 [temp.dep.type] Dependent types

Add a case for `decltype` in paragraph 6:

> A type is dependent if it is:
>
> - denoted by `decltype(`*expression*`)`, where *expression* is type-dependent ([temp.dep.expr]).

Section 9.3.2 The this pointer ([class.this])

Paragraph 1 should start:

> In the body of a nonstatic (9.3) member function, the keyword `this` is ~~a non-lvalue~~ an rvalue expression ...

Editing note: this change is not intended to change semantics, and it is not strictly necessary for decltype.

References

[JS03] J. Järvi and B. Stroustrup. Mechanisms for querying types of expressions: Decltype and auto revisited. Technical Report N1527=03-0110, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming Language C++, September 2003. `http://anubis.dkuug.dk/jtc1/sc22/wg21/docs/papers/2003/n1527.pdf`.

[JS04] Jaakko. Järvi and Bjarne Stroustrup. Decltype and auto (revision 3). Technical Report N1607=04-0047, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming Language C++, March 2004.

[JSGS03] Jaakko Järvi, Bjarne Stroustrup, Douglas Gregor, and Jeremy Siek. Decltype and auto. C++ standards committee document N1478=03-0061, April 2003. `http://anubis.dkuug.dk/jtc1/sc22/wg21/docs/papers/2003/n1478.pdf`.

[JSR04] Jaakko. Järvi, Bjarne Stroustrup, and Gabriel Dos Reis. Decltype and auto (revision 4). Technical Report N1705=04-0145, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming Language C++, September 2004.

[JSR06a] Jaakko. Järvi, Bjarne Stroustrup, and Gabriel Dos Reis. Decltype (revision 5). Technical Report N1978=06-0048, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming Language C++, April 2006.

[JSR06b] Jaakko Järvi, Bjarne Stroustrup, and Gabriel Dos Reis. Decltype (revision 6): proposed wording. Technical Report N2115=06-0185, ISO/IEC JTC 1, Information technology, Subcommittee SC 22, Programming Language C++, November 2006.

[Str02] Bjarne Stroustrup. Draft proposal for "typeof". C++ reflector message c++std-ext-5364, October 2002.

3 Acknowledgments

We are grateful to Jeremy Siek, Douglas Gregor, Jeremiah Willcock, Gary Powell, Mat Marcus, Daveed Vandevoorde, David Abrahams, Andreas Hommel, Peter Dimov, and Paul Mensonides, Howard Hinnant, Jens Maurer, and Jason Merrill for their valuable input in preparing this proposal.