0.1 C++11

Small amount of intro text here. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse non nisi ex. Ut non sollicitudin est. Donec laoreet, risus pretium egestas condimentum, ex eros tempor diam, quis congue nisi metus in tellus. Proin tempor ac lectus nec elementum. Maecenas augue turpis, pellentesque sed eros sit amet, tincidunt pretium nulla. Nunc lacus ligula, ullamcorper a porta eget, fringilla sed orci. Maecenas eget ultricies risus. Donec varius vehicula diam.

0.1.1 `auto`

Small amount of intro text here. The `auto` keyword was repurposed[1] in C++11 to act as a *placeholder* type. When used instead of a type as part of a variable declaration, the compiler will use the same rules as template argument deduction to deduce the type of the variable.

Description

Description of the feature here. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse non nisi ex. Ut non sollicitudin est. Donec laoreet, risus pretium egestas condimentum, ex eros tempor diam, quis congue nisi metus in tellus. Proin tempor ac lectus nec elementum. Maecenas augue turpis, pellentesque sed eros sit amet, tincidunt pretium nulla. Nunc lacus ligula, ullamcorper a porta eget, fringilla sed orci. Maecenas eget ultricies risus. Donec varius vehicula diam.

Automatic type deduction rules Sub-section that describes a particular aspect of the feature in abstract terms. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse non nisi ex. Ut non sollicitudin est. Donec laoreet, risus pretium egestas condimentum, ex eros tempor diam, quis congue nisi metus in tellus.

EXAMPLE:

Proin tempor ac lectus nec elementum. Maecenas augue turpis, pellentesque sed eros sit amet, tincidunt pretium nulla. Nunc lacus ligula, ullamcorper a porta eget, fringilla sed orci. Maecenas eget ultricies risus. Donec varius vehicula diam.

Listing 1: example caption for recordcount

```
int recordCount;
while (cursor.next()) { ++recordCount; }
    //                   ^~~~~~~~~~~~~
    //                 Undefined behavior.
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse non nisi ex. Ut non sollicitudin est. Donec laoreet, risus pretium egestas condimentum, ex eros tempor diam, quis congue nisi metus in tellus. Proin tempor ac lectus nec elementum. Maecenas augue turpis, pellentesque sed eros sit amet, tincidunt pretium nulla. Nunc lacus ligula,

[1] Footnote. `inline` code in footnote. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

ullamcorper a porta eget, fringilla sed orci. Maecenas eget ultricies risus. Donec varius vehicula diam.

Listing 2: missing caption

```
auto recordCount; // Compile-time error.
while (cursor.next()) { ++recordCount; }
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse non nisi ex. Ut non sollicitudin est.

- Donec `laoreet`, risus pretium egestas condimentum, ex eros tempor diam, quis congue nisi metus in tellus.
- Proin tempor ac lectus nec elementum. Maecenas augue turpis.
- Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse non nisi ex. Ut non sollicitudin est.

Use Cases

Small amount of intro text can optionally be here. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse non nisi ex. Ut non sollicitudin est. Donec laoreet, risus pretium egestas condimentum, ex eros tempor diam.

Avoiding type repetition Sub-section that describes a particular aspect of the feature in concreteterms. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse non nisi ex. Ut non sollicitudin est. Donec laoreet, risus pretium egestas condimentum, ex eros tempor diam, quis congue nisi metus in tellus. Proin tempor ac lectus nec elementum[2].

Listing 3: missing caption

```
int x = 10;
auto y = x;
```

Maecenas augue turpis, pellentesque sed eros sit amet, tincidunt pretium nulla. Nunc lacus ligula, ullamcorper a porta eget, fringilla sed orci. Maecenas eget ultricies risus. Donec varius vehicula diam.[3]

Potential Pitfalls

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse non nisi ex. Ut non sollicitudin est. Donec laoreet, risus pretium egestas condimentum, ex eros tempor diam, quis congue nisi metus in tellus. Proin tempor ac lectus nec elementum. Maecenas augue

[2]The relative position of `decltype(range.sort())` in the signature of `sortRangeImpl` is not significant, as long as it is visible to the compiler during template substitution. This particular example (shown in the main text) makes use of a function parameter that is defaulted to `nullptr`. Alternatives involving a trailing return type or a default template argument are also viable:

[3]For more information on foobar, see **lakos20**, section 1.2.3, pp 208-234, especially Figure 1-35, p. 215.

turpis, pellentesque sed eros sit amet, tincidunt pretium nulla. Nunc lacus ligula, ullamcorper a porta eget, fringilla sed orci. Maecenas eget ultricies risus. Donec varius vehicula diam.[4]

In the example above, `localhost` will be initialized in the following manner[5]:

1. Constructor **(0)** will be invoked;
2. On line **(1)**, execution will be delegated to constructor **(2)**;
3. The body of constructor **(2)** will be executed;
4. The body of constructor **(1)** will be executed.

This feature, when used in conjunction with *explicit instantiation definitions*, can significantly improve compilation times for a set of translation units that often instantiate common templates:

Listing 4: code 1

```
void code()
{

}
```

Listing 5: code 2

```
void code()
{

}
```

Attempting to compile `main.cpp` on its own will produce a linker error along the lines of:

Listing 6: missing caption

```
undefined reference to `Vector2D<float>::normalize()'
```

The linker error is expected as the inclusion of `vector2d.h` suppresses implicit instantiation of `Vector2D<float>`. Note that `iVec` is not affected, as the `Vector2D<int>` instantiation does take place.

[4] **lakos20**, section 0.5, pp 34-42

```
template <typename Range>
auto sortRangeImpl(Range& range, int) -> decltype(range.sort(), void());
    // The comma operator is used to force the return type to `void`,
    // regardless of the return type of `range.sort()`.

template <typename Range, typename = decltype(std::declval<Range&>().sort()>
auto sortRangeImpl(Range& range, int);
    // `std::declval` is used to generate a reference to `Range` that can be
    // used in an unevaluated expression
```

[5] Footnote. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Block code in footnote:

```
abcdef
inline
virtual
```

Readability concerns Using `auto` can hide all information regarding a variable's type, increasing cognitive overhead for the readers. In conjunction with unclear variable naming, disproportionate usage of `auto` can make code unreadable. E.g.

Listing 7: missing caption

```
int main(int argc, char** argv)
{
    const auto args0 = parseArgs(argc, argv);
        // The behavior of `parseArgs` is unclear.

    const std::vector<std::string> args1 = parseArgs(argc, argv);
        // It is obvious what `parseArgs` does.
}
```

While it may be necessary to read `parseArgs`'s contract at least once to fully understand its behavior, an explicit type in the usage site helps readers understand its purpose CWG1655.

Listing 8: missing caption

```
testing column width for code
123456789A123456789B123456789C123456789D123456789E123456789F123456789G123456789H
this is 80 characters
```

C++11 introduces three new types of string literal, which provide strong guarantees on the encoding of character sequences:

Encoding	Example	Character Type
UTF-8	`u8"Hello"`	`char`
UTF-16	`"Hello"`	`char16_t`
UTF-32	`U"Hello"`	`char32_t`

Raw string literals enable developers to embed strings in a program's source code without requiring to escape special character sequences and preserving whitespace, with the goal of enhancing readability. The syntax of the feature is easily understood through an example showing a *regular expression* embedded in source code:

Listing 9: missing caption

```
//            delimiter and round parenthesis
//                     v~~~           v~~~
const char* regex = R"xxx([0-9]\(".*"\))xxx";
//                    ^     ^~~~~~~~~~~~
//                    |      string contents
//                    |
//                    | uppercase R
```

Lack of interface restrictions In generic code, even if concrete types are dependent on template arguments, `auto` is needlessly lax. It is always possible to identify a *concept*[6] which provides information regarding operations allowed on a type to the reader [see @AGE86, pp. 33-35], albeit specifying it in code is cumbersome.[7]

In some particular cases, concepts also carry important semantic meaning that could be lost by using `auto`. E.g.

Listing 10: missing caption

```
Packet* PacketCache::findFirstCorruptPacket() const
{
    auto it = std::begin(this->d_packet);

    static_assert(IsRandomAccessIterator<decltype(it)>::value,
                  "`it` must be a random access iterator.");

    return it == std::end(this->d_employees) ? nullptr
                                             : &*it;
}
```

List initialization The meaning of `auto` completely changes when using *list initialization*: `std::initializer_list` is always deduced.

Listing 11: missing caption

```
auto example0 = 0; // Copy initialiation, deduced as `int`.
auto example1(0);  // Direct initialiation, deduced as `int`.
auto example2{0};  // List initialiation, deduced as `std::initializer_list<int>`.
```

This surprising behavior contradicts the idea of "uniform initialization" and has been widely regarded as a mistake and rectified in C++14.

The `decltype` keyword allows inspecting the declared type of an entity or the type and value category of an expression. What `decltype` yields as the result depends on the provided argument:

- With an unparenthesized *id-expression*[8] or unparenthesized *class member access expression*[9], `decltype` yields the *"declared type"*[10] of the given expression.
- With any other expression of type `T`, `decltype` yields:
  - `T&&` if the value category of the expression is *xvalue*;
  - `T&` if the value category of the expression is *lvalue*;
  - `T` if the value category of the expression is *prvalue*.

Similarly to `sizeof`, the provided expression is not evaluated.

[6] Authors' Note: We will have some footnotes that are authors' notes.

[7] Unless explicitly specified, the *underlying type* of non-strongly typed enumerations is [*implementation-defined*].

[8] Footnote. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

[9] Footnote. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

[10] Footnote. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Glossary

word to be defined
Definition text follows. At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga.

word to be defined
Definition text follows. Itaque earum rerum hic tenetur a sapiente delectus, ut aut reiciendis voluptatibus maiores alias consequatur aut perferendis doloribus asperiores repellat.

word to be defined
Definition text follows.Nam libero tempore, cum soluta nobis est eligendi optio cumque nihil impedit quo minus id quod maxime placeat facere possimus, omnis voluptas assumenda est, omnis dolor repellendus.

word to be defined
Definition text follows. At vero eos et accusamus et iusto odio dignissimos ducimus qui blanditiis praesentium voluptatum deleniti atque corrupti quos dolores et quas molestias excepturi sint occaecati cupiditate non provident, similique sunt in culpa qui officia deserunt mollitia animi, id est laborum et dolorum fuga.

Index

Symbols
$, 31
%, 12
&, 31

A
a very long entry to test the column width, 31
anise, 31
apple
    braebrun, 31
    cameo, 31
    fuji, 31
    gala, 31
    granny smith, 31
    red delicious, 31
apricots, 31
avocado, 31

B
banana, 31
basil, 31
bibendum, 12
    dapibus, 12
blackberry, 31

C
cabbage, 31
celery, 31
chervil, 31
chives, 31
cilantro, 31
`codeword`, 12
corn, 31
cucumber, 31

D
dates, 31
dill, 31

E
eggplant, 31
endive, 31

F
fennel, 31
fig, 31
`function`, 31

G
garlic, 31
grapes, 31

H
horseradish, 31
how about another long entry, 31
huckleberry, 31

J
jicama, 31

K
kale, 31
kiwi, 31

L
leeks, 31
lemon, 31
lettuce
    boston bibb, 31
    iceberg, 31
    mesclun, 31
    red leaf, 31
lime, 31
lorem, *See* lobortis

M
majoram, 31
mango, 31
maybe another long entry for this test, 31
melon
    canary, 31
    cantaloupe, 31
    honeydew, 31
    watermelon, 31
mushrooms
    button, 31
    porcini, 31
    portabella, 31
    shitake, 31

N
nectarine, 31
nutmeg, 31

O
okra, 31

onion
red, 31
vidalia, 31
yellow, 31
orange, 31

P
papaya, 31
parsley, 31
peaches, 31
peppers
ancho, 31
bell, 31
habeñeros, 31
jalapeños, 31
pablaños, 31
perhaps yet another long entry for this test, 31
plantains, 31
plums, 31
potatoes
red-skinned, 31
russet, 31
yukon gold, 31
pumpkin, 31

Q
quince, 31

R
radicchio, 31
radish, 31
raspberry, 31
rosemary, 31
rutabaga, 31

S
shallots, 31
spinach, 31
squash, 31
still another long entry for column width testing, 31

T
thyme, 31
tomatillo, 31
tomatoes
cherry, 31
grape, 31
heirloom, 31
hybrid, 31
roma, 31
`typeof`, 31

U
ugly fruit, 31

V
verbena, 31
viverra, *See also* neque

X
xacuti masala, 31

Y
yams, 31
yet another very long entry for column width test, 31

Z
zucchini, 31