# Delayed Evaluation Parameters

Document number: N4360
Date: 2015-01-22
Project: Programming Language C++, Evolution Working Group
Reply-to: Douglas Boffey <douglas.boffey@gmail.com>

## Table of Contents



## Introduction

This proposal provides Delayed Evaluation Parameters (DEPs), which are parameters to a constructor, function or member function that are only evaluated on first use.

## Motivation and Scope

Most of the time, function parameters are evaluated prior to invocation of a function, but there are occasions where that is not what is needed.

Some of the reasons are:

a) evaluation of a parameter might invoke undefined behaviour in some circumstances. Take, for example, the idiom:

```
if (a && /* some expression involving *a */)
```

where a is a pointer to memory, or *nullptr*.

b) a parameter might be expensive to construct and then destruct. If it is only sometimes needed, it would make sense only to evaluate it in those circumstances.

c) code might need to be included in some builds, e.g. debug builds, that shouldn't be included in other builds, e.g. production builds. For example, code that checks preconditions, postconditions and invariants.

d) it may be that only details of a variable type are required. For example, a user-implementation of *sizeof* for variables would look like:

```
template <typename T>
std::size_t my_sizeof(inline T t) {
  return sizeof(T);
}
```

e) logging may be optionally enabled. Given, for example,

```
log(level) << expensive_expression
```

it is convenient not to evaluate *expensive_expression*.

Currently, c) and e) are usually achieved by using macro functions, expanding to something like:

```
if (/* enabled */) /* something potentially involving an expensive expression */
```

## Impact on the Standard

This code introduces no breaking behaviour.

## Design Decisions

There are a number of decisions:

a) Should the parameter be evaluated once only (cache-like), or on every occurrence (instance-like)?

This proposal is for a cache-like solution, because:

- efficiency
  - an instance-like solution might require multiple constructions or assignments
  - an instance-like solution might inhibit optimisations, e.g. moving code out of loop
- effect
  - there could be problems if the DEP was an rvalue
  - multiple evaluations might produce unexpected behaviour. Take, for instance,

    ```
    void fn(inline int a) {/* … */}
    // …
    int a{1};
    fn(++a);
    ```

    It could be difficult to predict what a would end up as.
- documentation: it would be easier to document the conditions under which the DEP is evaluated rather than the number of times it is evaluated.

b) When should the parameter be evaluated?

A parameter needs to be evaluated whenever the value is needed, a reference of it is taken, or a pointer is made to point to it. The functions '*sizeof*', '*typeid*', '*alignas*' and '*decltype*' would not cause the DEP to be evaluated, as only the type of the DEP is required. Any references or pointers to the parameter should be to the cached value, not the code generating the value.

c) What about a DEP used within a DEP of a subsidiary function?

The DEP should not automatically be evaluated, but if it has not been evaluated, the subsidiary function would be responsible for evaluating it.

If the subsidiary function evaluates the DEP, that evaluation is visible in the outer function.

For example:

```
int bar(inline int a) {
  return a;
}

void foo(inline int b) {
```

```
  // b has not been evaluated yet
  bar(b);
  // foo sees that b has been evaluated and has access to the
value
}
```

d) Should the DEP be a wrapper (like *std::function*) or a separate object (like a lambda)?

As it would not make sense to refer to the code evaluating the value, the code should be as if it were a wrapper around the parameter.

e) What nomenclature should be used?

Three options:

1. a symbol: this would be inconsistent with existing practice, where symbols are part of the type.
2. introduce a new keyword, e.g. lazy: any new keyword is a potential for breaking code, and there is no obvious way, consistent with the current syntax, of introducing a context-sensitive keyword.
3. use an existing keyword: the keyword, '*inline*' bests describes the effect of a DEP. It is proposed that it should be put at the beginning of a DEP declaration, to mirror inline variables.

f) A conforming implementation can execute a function taking a DEP with an already-evaluated value rather than a thunk, provided that the evaluation has no observable side effects (and without specifying how an implementation might do such a thing). For example, it would be permitted that an implementation generate multiple versions of a function taking one or more DEPs and choose between them.

## Technical Specifications

TBD

## Future Enhancements

Unless the parameter is evaluated in a try block, it would be subject to any exception restrictions of the callee, although the code is owned by the caller. It might be better to permit an exception specification with the DEP.

## Acknowledgements

My thanks go to sasho648, whose proposal inspired this proposal. Also to Nicola Gigante, David Krauss and Matthew Woehlke for their insightful comments.

## References

See https://groups.google.com/a/isocpp.org/forum/#!topic/std-proposals/ftrFqbyu7pQ for a discussion on this

Other, similar proposals can be found on https://groups.google.com/a/isocpp.org/d/topic/std-proposals/kwDmEOQL7oE/discussion and https://groups.google.com/a/isocpp.org/d/topic/std-proposals/DnxRfo_uQl8/discussion