nodiscard("should have a reason")

JeanHeyd Meneide <phdofthehouse@gmail.com>

September 23rd, 2019

Document: n2430
Audience: WG14
Proposal Category: New Features
Target Audience: General Developers, Compiler/Tooling Developers

Abstract:

Many functions return a value, however, not all function return values are of equal importance to the caller. The recent `[[nodiscard]]` attribute allows compilers to issue a diagnostics, but only hands the user a generic error message. This proposal enhances the `[[nodiscard]]` attribute in the same manner as the `[[deprecated]]` attribute, giving developers the same power to guide their users to better APIs with the aid of the compiler by providing a `string-literal` attribute argument clause.

Introduction

Document N2267 introduced a new attribute `[[nodiscard]]` in the C2x working paper. This has provided significant improvements in reminding programmers of the safety issues of discarding the return value of a function. The `[[nodiscard]]` attribute has helped prevent a serious class of software bugs, but sometimes it is hard to communicate exactly **why** a function is marked as `[[nodiscard]]` and perhaps what actions should be taken to rectify the issue.

This paper supplies an addendum to allow a person to add a string attribute token to let someone provide a small reasoning or reminder for why a function has been marked `[[nodiscard("potential memory leak")]]`.

Design Considerations

This paper is an enhancement of a preexisting feature to help programmers provide clarity with their code. Anything that makes the implementation warn or error should also provide some reasoning or perhaps point users to a knowledge base or similar to have any questions they have about the reason for the nodiscard attribute answered.

Consider the following code example, before and after the change:

```
#define FOO_BASE 0xBA51CF00

#define FOO_LINK_TYPE 1

struct foo { /* ... */ };
[[nodiscard]] int foo_get_value(struct foo*);
```

Status Quo:

```
[[nodiscard]]
foo* foo_create(int, struct foo*);
[[nodiscard]]
int foo_compare(struct foo*, struct foo*);

// Always > 0
const int kHandles = ...;

int main (int, char*[]) {

  foo* foo_handles[kHandles + 1] = { };
  foo_handles[0] = create(BASE_FOO, NULL);
  for (int i = 1; i < kHandles; ++i) {
    foo_handles[i] = create(FOO_LINK_TYPE, foo_handles[0])
  }

  /* sometime later */

  for (int i = 0; i < kHandles,
    foo_compare(foo_handles[0], foo_handles[i]), foo_get_value(foo_handles[i]) > 0;
    // ^ warning: function return value marked nodiscard was discarded
    ++i) {
      /* process... */
  }

  return 0;
}
```

⚠️ - warning, but it is a generic warning; what exactly went wrong here?

With Proposal:

```
[[nodiscard("memory leaked")]]
struct foo* foo_create(int, struct foo*);
[[nodiscard("value of foo comparison unused")]]
int foo_compare(struct foo*, struct foo*);

// Always > 0
const int kHandles = ...;

int main (int, char*[]) {

  struct foo* foo_handles[kHandles + 1] = { };
  foo_handles[0] = create(BASE_FOO, NULL);
  for (int i = 1; i < kHandles; ++i) {
    foo_handles[i] = create(FOO_LINK_TYPE, foo_handles[0])
  }

  /* sometime later */

  for (int i = 0; i < kHandles,
    foo_compare(foo_handles[0], foo_handles[i]), foo_get_value(foo_handles[i]) > 0;
    // ^ warning: function return marked nodiscard was discarded - value of foo comparison
    //     unused
    ++i) {
      /* process... */
  }
```

```
    return 0;
}
```

✔️ - warning much more clearly makes it obvious that a comma was used with the return value of `foo_compare`, and not `&&`.

The design is very simple and follows the lead of the deprecated attribute. We propose allowing a string literal to be passed as an attribute argument clause, allowing for `[[nodiscard("use the returned token with lib_foobar")]]`. The key here is that there are some nodiscard attributes that have different kinds of "severity" versus others.

Adding a reason to nodiscard allows implementers of the standard library, library developers, and application writers to benefit from a more clear and concise error beyond `error:<line>: value marked [[nodiscard]] was discarded`. This makes it easier for developers to understand the intent for return values for the used libraries (and understand from which individual expression errors originate in complex expressions).

Implementation Experience

This is in the official C++ Standard, and has been [merged into Clang already](). A patch is out for [GCC](). It would be good to maintain parity with C++ to allow headers that work in both languages to continue to use the same syntax, since this is going to be an increasingly useful existing practice.

Proposed Wording

This proposed wording is currently relative to Working Paper N2385. The intent of this wording is to allow for the `[[nodiscard]]` attribute to be able to take a string literal.

Changes

Rewrite §6.7.11.2 "The nodiscard attribute"'s **Constraint** subsection as follows:

> The nodiscard attribute shall be applied to the identifier in a function declarator or to the definition of a structure, union, or enumeration type. It shall appear at most once in each attribute list. If an attribute argument clause is present, it shall have the form:
>
> (*string-literal*)

Add additional clauses in the **Semantics** subsection as follows:

> A name or entity declared without the nodiscard attribute can later be redeclared with the attribute and vice-versa. Redeclarations using different forms of the attribute (with or without the attribute-argument-clause or with different attribute-argument-clauses) are allowed.
>
> [Note: Thus, an entity initially declared without the attribute can be marked as nodiscard by a subsequent redeclaration. However, after an entity is marked as nodiscard, later redeclarations do not remove the nodiscard from the entity. — end note]

Add a third example after the first two in the **Recommended Practice** subsection as follows:

```
[[nodiscard("must check armed state")]]
bool arm_detonator(int);

void call(void) {
  arm_detonator(3);
  detonate();
}
```

A diagnostic for the call to `arm_detonator` using the *string-literal* in the *attribute-argument-clause* is encouraged.