



HAL
open science

Add annotations for unreachable control flow (slides)

Jens Gustedt

► **To cite this version:**

| Jens Gustedt. Add annotations for unreachable control flow (slides). 2021. hal-03328557

HAL Id: hal-03328557

<https://inria.hal.science/hal-03328557>

Preprint submitted on 30 Aug 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Add annotations for unreachable control flow

ISO/IEC JTC 1/SC 22/WG14 **N2757**

Jens Gustedt

INRIA – Camus

ICube – ICPS
Université de Strasbourg

inria
informatiques mathématiques



<https://modernc.gforge.inria.fr/>



Table of Contents

1 Motivation

2 Existing practice

3 Variations

4 Questions to WG14

How to distinguish coding errors from UB?

A program with potential UB

```
#include <stdio.h>
#include <assert.h>

int main(int argc, char* argv[static argc+1]) {
    return printf("we see %s\n", argv[2]);
}
```

- This program is undefined when called with less than two arguments.
- It is usually optimized to a tail call to `printf`.
- Some diagnostics may be produced for the access to `argv[2]`.



How to distinguish coding errors from UB?

Declaring an intent (current C)

```
#include <stdio.h>
#include <assert.h>

int main(int argc, char* argv[static argc+1]) {
    if (argc <= 2) abort();
    else return printf("we see %s\n", argv[2]);
    return puts("this should never be reached");
}
```

- This program is defined under all circumstances.
- The comparison cannot not be removed during optimization.
- The two programs are not equivalent.
- In general, there are no diagnostics printed.

How to distinguish coding errors from UB?

Declaring an intent (our proposal)

```
#include <stdio.h>
#include <assert.h>

int main(int argc, char* argv[static argc+1]) {
    if (argc <= 2) unreachable("two_arguments_needed");
    else return printf("we_see_%s\n", argv[2]);
    return puts("this_should_never_be_reached");
}
```

- This program conveys the intent that two arguments are expected.
- The comparison can be removed during optimization.
- The call to `puts` can be removed during optimization.
- The program is equivalent to the first.

Table of Contents

- 1 Motivation
- 2 Existing practice
- 3 Variations
- 4 Questions to WG14

Existing practice

C and C++

```
gcc          __builtin_unreachable()  
clang       __builtin_unreachable()  
icc         __builtin_unreachable()  
Visual C++  assume(false)
```

Our proposal is based on P0627R3 for C++.

Other programming languages

- Rust has

```
unsafe std::hint::unreachable_unchecked!();
```


Table of Contents

1 Motivation

2 Existing practice

3 Variations

4 Questions to WG14

Add a string to convey a reason

Simple wrapper implementations for existing features

```
#define unreachable(...) __unreachable(__VA_ARGS__ "")
```

- For compilers that implement `__builtin_unreachable()`

```
#define __unreachable(REASON) \
((void)REASON, __builtin_unreachable())
```

- For compilers that implement `assume(false)`

```
#define __unreachable(REASON) \
((void)REASON, assume(false))
```

By that, the argument to **unreachable** has to be either empty, or a white-space separated sequence of string literals.

Add an expression to name the case: **attest**

Simple wrapper implementations for existing features

- For compilers that implement `__builtin_unreachable()`

```
#define attest(...) \
  ((__VA_ARGS__) ? (void)0 : __builtin_unreachable())
```

- For compilers that implement `assume(false)`

```
#define attest(...)  assume(!!( __VA_ARGS__ ))
```



Table of Contents

- 1 Motivation
- 2 Existing practice
- 3 Variations
- 4 **Questions to WG14**

Do we want such a feature?

Question 1

Does WG14 want a feature similar to **unreachable** with no arguments along the lines of N2757, either provided as

- proper syntax,
- an attribute,
- a function interface or
- a macro interface?

How do we want such a feature?

Question 2

Does WG14 want such a feature to be provided as a functon interface?

Question 3

Does WG14 want to add such a functon interface to `<assert.h>`?

How do we want that feature to be named?

Question 4

Does WG14 want to use the name **unreachable** for the feature?

Do we want to add reason?

Question 5

Does WG14 want to add the possibility to add a string that provides a reason to the feature?

Question 6

Shall such a string be required to be a string literal?

Question 7

Shall such a string be required to be compatible to `char const*` ?

