Proposal for C2x
WG14 N2510

Title:	Allowing unnamed parameters in a function definition
Author, affiliation:	Aaron Ballman, GrammaTech
Date:	2020-04-09
Proposal category:	New features
Target audience:	Developers working on combined C and C++ code bases

Abstract: There are times when the identifier for a function parameter is unnecessary (such as when the object is unused within the call), but C requires the identifier to be present nonetheless. This paper proposes lifting that restriction.

Prior art: C++

Allowing unnamed parameters in a function definition

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2510
Revises Document No: N2480
Date: 2020-04-09

Summary of Changes

N2510

- Amended the footnote to say “by name”
- Added the results of the straw poll taken at the Corona meeting

N2480

- Updated a few places where parameter identifiers are now optional

N2381

- Original proposal

Introduction and Rationale

In C17, it is permissible for function declarations to elide the identifier in a parameter declaration, but it is a constraint violation to elide an identifier for a function parameter in the definition of a function. Some C users may find the restriction surprising because of the inconsistency between function declarations and definitions. Further, this restriction is not in place for C++ and this can lead to surprising compiler diagnostics for users who are compiling header files in mixed language modes [0, 1].

There are several circumstances in which it is beneficial for program maintainability and expressing programmer intent to allow a programmer to elide these identifiers.

A common idiom when designing interfaces involving callbacks is to allow the programmer to pass in additional information to the API calling the callback which is passed to the callback itself. Consider the `SSL_CTX_set_client_hello_cb()` function from OpenSSL [2], which has an interface like:

```
typedef int (*SSL_client_hello_cb_fn)(SSL *s, int *al, void *arg);
void SSL_CTX_set_client_hello_cb(SSL_CTX *c, SSL_client_hello_cb_fn *f,
                                 void *arg);
```

The caller of `SSL_CTX_set_client_hello_cb()` can pass a `void *` argument to this function and the same value will be passed along to the eventual callback call through the function pointer. If the caller of `SSL_CTX_set_client_hello_cb()` does not need to pass extra information through `arg`, they may wish to explicitly demonstrate to the compiler that the argument is unused within the callback, despite the argument being required as part of the API contract. By eliding the identifier for the last parameter, the programmer can explicitly signal to the compiler that they do not intend to use the information passed as the argument.

Popular compiler implementations often have an option to warn on unused declarations, including function parameters that are not used [3] and allowing the user to elide the name of a parameter is a way for a programmer to explicitly specify their intent to the compiler and silence such diagnostics.

Proposal

This paper proposes allowing the identifier in a parameter declaration to be elided in both function declarations (as it is today in C17) and in function definitions (moving forward in C2x), as shown below.

C17	C2x

`int SSL_callback(SSL *s, int *al,`
`  void *);`
 	
`int SSL_callback(SSL *s, int *al,`
`  void *arg) {`
`  // ... code that uses s and al,`
`  // but not arg.`
`  (void)arg; // Silences diagnostics.`
`  return 0;`
`}`
	`int SSL_callback(SSL *s, int *al,`
	`  void *);`
	
	`int SSL_callback(SSL *s, int *al,`
	`  void *) {`
	`  // ... code that uses s and al.`
	`  return 0;`
	`}`

Interaction With K&R C Function Definitions

There is a question as to whether eliding parameter names in a K&R C function definition can cause conflicts due to typedefs. Specifically, whether it is ambiguous without lookahead to determine if a token is the type in parameter type list (sans name) or an identifier in an identifier list. Consider a code example like:

```
typedef int foo;
void func();
void func(foo) double foo; { }
```

If we allowed parameter identifiers to be elided in function definitions, does the definition of `func()` introduce a function with a parameter type list or an identifier list?

This is a non-issue because the previous code is non-conforming in C17 already. C17 6.7.6.3p11 states:

> If, in a parameter declaration, an identifier can be treated either as a typedef name or as a parameter name, it shall be taken as a typedef name.

Additionally, C17 6.9.1p6 states:

> If the declarator includes an identifier list, each declaration in the declaration list shall have at least one declarator, those declarators shall declare only identifiers from the identifier list, and every identifier in the identifier list shall be declared. An identifier declared as a typedef name shall not be redeclared as a parameter. …

Indeed, I cannot find a C compiler that accepts the above code in the presence of the `typedef` [4], but all of the compilers tested accepted the code in the absence of the `typedef` [5], which suggests the standard is reasonably clear on this point.

Based on the belief that there are not problematic interactions with K&R C function definitions, this paper does not propose any changes related to such definitions.

Straw Poll Results

At the Corona virtual meeting, we took a straw poll on including N2480 into C2x with a modification to add "by name" to the footnote. The results of that straw poll were: 14/2/1, which was consensus to adopt the proposal for C2x. The committee did not express interest in discussing the paper again at a future meeting, but due to the current lack of a project editor, asked for N2480 to be revised to the updated wording to ease integration into the working draft. N2510 is that revision.

Proposed Wording

The wording proposed is a diff from the committee draft of ISO/IEC 9899-2017. Green text is new text, while ~~red~~ text is deleted text.

Modify 6.9.1p5:

If the declarator includes a parameter type list, ~~the declaration of each parameter shall include an identifier, except for the special case of a parameter list consisting of a single parameter of type **void**, in which case there shall not be an identifier. N~~no declaration list shall follow. If such a declarator consists of a single parameter of type **void**, the parameter declarator shall not include an identifier.

Modify 6.9.1p7:

The declarator in a function definition specifies the name of the function being defined and the ~~identifiers and~~ types (and optionally the names) of all the parameters; the declarator also serves as a function prototype for later calls to the same function in the same translation unit. The type of each parameter is adjusted as described in 6.7.6.3; the resulting type shall be a complete object type.

Modify 6.9.1p9:

Each parameter has automatic storage duration; its identifier, if any[N)], is an lvalue.174) The layout of the storage for parameters is unspecified.

Add a new footnote N):

[N)] A parameter that has no declared name is inaccessible by name within the function body.

Acknowledgements

I would like to recognize the following people for their help in this work: Lars Gullik Bjønnes, Jens Gustedt, and Robert Seacord.

References

[0] parameter name omitted, C++ vs C. https://stackoverflow.com/questions/8776810/parameter-name-omitted-c-vs-c/8776886

[1] Why the unnamed parameter warning discrepency between C and C++?. https://stackoverflow.com/questions/12181852/why-the-unnamed-parameter-warning-discrepency-between-c-and-c

[2] SSL_CTX_set_client_hello_cb. https://www.openssl.org/docs/man1.1.1/man3/SSL_CTX_set_client_hello_cb.html

[3] https://godbolt.org/z/HaiZFq

[4] https://godbolt.org/z/sdD31P

[5] https://godbolt.org/z/vP4LAT