# Alias attribute for function declarations

### Examples

char\* find\_key(const char \*s) [[alias(find\_key, s)]];

void seek\_blank(const char \*\*s) [[alias(\*s after, \*s before)]];

int sanitize(char \*s, char \*end) [[alias(end, s)]];

char\* func(Struct Str \*a, char \* p[]) [[alias(func, a->s),
alias(p[1] after, a->s]];

### Wording

#### Syntax

*alias\_argument\_clause:*

**(***aliased\_ident*, *aliased\_ident***)**

*aliased\_ident*

*unary-expression when*opt

*when*

**before**

**after**

#### Constrains

1 The unary expresion shall have no side effects, shall have pointer type and shall denote a pointer based on one of the function parameters or the return value or another identifier with a declaration visible at the point the function is declared, or based in any subobject of these or the pointed-to memory by these, recursively.

2 If a unary expression denotes a pointer based on the return value the *when* specification shall not be present. If the lifetime of the object to which the identifier refers ends when the function returns the *when* specification, if present, shall not be **after**.

#### Semantics

3 The word **before** or **after** specifies when the value of the pointer which the identifier denotes is to be considered: upon entry to the function or immediately after the function returns. If it is not present it defaults to **before**, except for the return value, for which it is allays **after**. The return value is identified by the name of the function.

4 If the “when” specifications, explicit or implicit, are both **before,** the alias attribute indicates that the two identifiers denoted by the *aliased\_ident* expressions may alias each other within the function body; i.e., may point to the same location in memory during their executions of one call to the function, or that any pointers based on them in the function body may.

5 If one of the “when” specifications is **after** and the other is **beofre**, the alias attribute indicates that the value of the former when the function returns may be based on the value of the latter passed to the function, as a result of the execution of the function body.

6 If the “when” specifications are both **after**, the alias attribute indicates that their values when the function returns may be based in the same pointer as a result of the execution of the body of the function.

### When two pointers alias

A definition according to which to pointers alias if they point wihin the same storage instance is too restrictive, in that it results in pointers aliasing according to the definition but which never alias during program execution. For example:

unsigned char large\_buffer[10000];

unsigned char \*a= &large\_buffer[0], \*b= &large\_buffer[1000];

If the memory accessed trhough a or any pointer based on it is allways in the range a[0] - a[999], and that accessed trhough bin the range b[0] - b[499], say, the pointers will in fact never alias.

The definition for when two pointers alias, whether it be the one proposed here or a different one, should be placed somewhere else in the text of the standard, not inserted in the semantics section of the **alias** attribute.

### When one pointer is based on another

Consider the following example:

char \*g;

char\* func(char \*a, char \*b, char \*c) [[alias(a, b), alias(func, c)]] {

 void \*d = c;

 a[2]=0;

 \*d=0;

 return g;

}

Here the caller might pass for c a pointer based on a, and as a result a[2] and \*d may access the same memory location. This is not considered as the pointer d being based on a. Also, the alias(func, c) should be omited, even if g might be based on c. In the latter case, in place of alias(func, c) there should be

alias(c,g), alias(func,g)

### before and after

The aliasings *before* are of interest to the function: the compiler may use that information when translating the function for the purposes of optimization static analysis, etc. The aliasings *after* are of interest to the caller: the programmer knows two pointers may alias after the function call or one pointer is based on another pointer passed to the function. The compier may use that information for the code which follow the function.