C/C++ Nodes

In this chapter:

Attributes

Nodes describing an "attribute" as it appeared in the source. The following attribute nodes are available:

Attribute

Node describing an "attribute" as it appeared in the source. For example:
`int x __attribute__ ((unused)); // Declaration with an Attribute`

Attribute Argument

Node describing the arguments of an attribute. For example:

```
void f() __attribute__ ((interrupt("IRQ"))); // Attribute Argument "interrupt"
```

Attribute Group

Node describing an attribute group. For example:

```
void fx1() { /* ... */; }
void fx2() __attribute__((weak, alias("fx1")));// Both attributes have the same Attribute Group
```

Constants

Constant value of any type. The following constant nodes are available:

Integer Constant

Constant integer expression. For example:

```
int i = 10; // Integer Constant, Value is 10
```

Real Constant

Constant expression that is a real number. For example:
`double d = 34.75; // Real Constant, Representation is "34.75"`

String Constant

Constant string expression. For example:

```
char *pc = "John Doe"; // String Constant, Value is "John Doe"
```

Template Parameter Constant

A constant template parameter in template class declaration. For example:

```
template <unsigned N> class Temp{}; // Template Parameter Constant: N
```

bool Constant

A constant value of true or false.

enum Constant

An enum identifier declared in body of enum declaration. For example:

```
enum E { A, B, C};    // enum Constant: A, B, C
```

nullptr constant

A pointer literal of type std::nullptr_t. Example:

```
int* pi = nullptr; // nullptr constant
```

```
bool b = (pi == nullptr); // nullptr constant
```

Declarations

Nodes representing various declarations in user code.

Some declaration nodes can be accessed from both their declaration and expressions that reference the declaration. In these cases, the `IsDecl` property returns a boolean, allowing users to distinguish between the actual declaration and references to that declaration.

The following declaration nodes are available:

- Explicit Template Instance
- Friend
- Functions
- Parameter
- Variables

Explicit Template Instance

Explicit template instances for types, functions and variables. For example:

```
template <typename T> void func(){}
template void func<int>(); // Explicit Template Instance
template <typename T> class C { T m1; };
template class C <int>; // Explicit Template Instance
```

Friend

Friend function declaration. For example:

```
void friendFunc();
class Foo {
public:
    friend void friendFunc();     // Friend
};
```

Functions

Function declarations. This includes Global and Member functions..

Global Function

Global function declaration. For example:

```
void globalFunc1();
   // Global Function, IsDecl is true
   // IsImplementation is false
   // HasVoid is true
void globalFunc2() { }
   // Global Function, IsDecl is true
   // IsImplementation is true
   // HasEllipsis is true
```

Member Function

Member function declaration. For example:

```
class Foo {
public:
        Foo () {};          // Member Function, IsDefaultConstructor is true
        void func();  // Member Function
};
```

Parameter

Parameters include **parameter** and **this**.

Parameter

Function parameter declaration. For example:

```
void func(int i); // i is parameter whose type is int
```

This

Parameter usage. For example:

```
class T{
T* func(T t){return this; } //this parameter
}
```

Variables

Variables include global, local, and member variables.

Global Variable

Global Variable. For example:

```
int global;    // Global Variable, IsDecl is true
global = 3;    // Global Variable, IsDecl is false
```

Local Variable

Local Variable. For example:

```
void func() {
   int local;   // Local Variable, IsDecl is true
   local = 3;   // Local Variable, IsDecl is false
}
```

Member Variable

Member Variable. For example:

```
class Foo {
public:
  Foo() : member(0) { }  // Member Variable, IsDecl is false
private:
  int member;            // Member Variable, IsDecl is true
};
```

Expressions

Includes all types of expression nodes:

- Assignment
- Bitwise
- Comparison
- Logical
- Miscellaneous
- Numerical

Assignment

Includes all expression nodes used for assignment.

++a

Preincrement expression. For example:

```
int i = 1;
++i;            // Preincrement ++a
```

--a

Pre-decrement expression. For example:

```
int i = 1;
--i;            // Predecrement --a
```

a%=b

Mod-equals operator expression. For example:

```
int i = 2;
int j = 3;
j %= i;        // Mod-equals a%=b
```

a&=b

Bitwise-and-equals operator expression. For example:

```
int i = 2;
int j = 3;
j &= i;        // Bitwise-and-equals a&=b
```

a*=b

Multiply-equals operator expression. For example:

```
int i = 2;
int j = 3;
j *= i;       // Multiply-equals a*=b
```

a++

Post increment expression. For example:

```
int i = 1;
i++;                    // Postincrement a++
```

a+=b

Plus-equals operator expression. For example:

```
int i = 1;
int j = 0;
j += i;                 // Plus-equals a+=b
```

a--

Post decrement expression. For example:

```
int i = 1;
i--;                   // Postdecrement a--
```

a-=b

Minus-equals operator expression. For example:

```
int i = 1;
int j = 0;
j -= i;                 // Minus-equals a-=b
```

a/=b

Divide-equals operator expression. For example:

```
int i = 4;
int j = 2;
j /= i;                 // Divide-equals a/=b
```

a<<=b

Left-shift-equals operator expression. For example:

```
int i = 2;
int j = 3;
j <<= i;               // Left-shift-equals a<<=b
```

a=b

Assignment operator expression. For example:

```
int i = 1;
int j = 0;
j = i;                  // Assignment a=b
```

a>>=b

Right-shift-equals operator expression. For example:

```
int i = 2;
int j = 3;
j >>= i;                // Right-shift-equals a>>=b
```

a^=b

Bitwise-xor-equals operator expression. For example:

```
int i = 2;
int j = 3;
j ^= i;                  // Bitwise-xor-equals a^=b
```

a|=b

Bitwise-or-equals operator expression. For example:

```
int i = 2;
int j = 3;
j |= i;                  // Bitwise-or-equals a|=b
```

Bitwise

Expressions involving non-assignment bitwise operators.

a&b

Bitwise 'and' expression. For example:

```
int z = 73, y = 0;
y = z & 0x0f;          // Bitwise 'and' a&b
```

a^b

Bitwise XOR expression. For example:

```
int z = 73, y = 0;
y = z | 0x0f;          // Bitwise XOR a^b
```

a|b

Bitwise 'or' expression. For example:

```
int x = 73, y = 0;
y = x | 4;              // Bitwise 'or' a|b
```

~a

Bitwise 'not' expression. For example:

```
int a = ~3;            // Bitwise 'not' ~a
```

Comparison

Expressions that compare values.

a!=b

Not-equal operator expression. For example:

```
bool func(int x) {
    if (x!=3) {        // a!=b
        return false;
    }
    return true
}
```

a<=b

'Less than or equals' expression. For example:

```
bool func(int x) {
    if (x <= 3) {       // a<=b
        return false;
    }
    return true
}
```

a<b

'Less than' expression. For example:

```
bool func(int x) {
    if (x<3) {            // a<b
        return false;
    }
    return true
}
```

a==b

Equality operator expression. For example:

```
bool func(int x) {
    if (x==3) {          // a==b
        return false;
    }
    return true
}
```

a>=b

'Greater than or equals' expression. For example:

```
bool func(int x) {
    if (x >= 3) {       // a>=b
        return false;
    }
    return true
}
```

a>b

'Greater than' expression. For example:

```
bool func(int x) {
    if (x > 3) {        // a>b
        return false;
    }
    return true
}
```

Logical

Comparison expressions involving logical operators.

!a

Logical 'not' expression. For example:

```
bool func(int x) {
    if (!x) {             // !a
        return false;
    }
    return true
}
```

a&&b

Logical 'and' expression. For example:

```
bool func(int x, int y) {
    if (x && y) {       // a&&b
        return false;
    }
    return true
}
```

a||b

Logical 'or' expression. For example:

```
bool func(int x, int y) {
    if (x || y) {             // a||b
        return false;
    }
    return true
}
```

Miscellaneous

Miscellaneous expressions that don't fit in other categories.

&a

Address of expression. For example:

```
int i = 0;
int *j = &i;               // &a
```

*a

Dereference expression. For example:

```
int *i = new int();
*i = 5;                    // *a
```

a->b

Dereference operator expression. For example:

```
class A {
public:
    int field;
};

main() {
    A *a = new A;
    a->field = 1; // Dereference a->b
}
```

alignof

Alignment of type (`alignof` in C++11 or `_Alignof` in C11). For example:

```
size_t a = alignof(int); // alignment of type int
```

Cast

Expression that casts from one type to another. For example:

```
void func() {
    char c = 'c';
    int i = (int)c;  // normal (C-style) cast
}
```

Generic Selection

A C11 generic selection expression. For example:

```
int is_int = _Generic(10, int: 1, default: 0); // Generic expression
```

GNU Statement Expression

A compound statement enclosed in parentheses that may appear as an expression in GNU C. For example:

```
int foo();
void bar(int p){
 p=({/* GNU Statement Expression start */
  int y = foo ();
  int z;
  if(y>0)z = y;
  else z = -y;
  z;
 });
}
```

Initializer List

Expression containing initializations, used in member functions. For example:

```
class Foo {
public:
   Foo() _x(0), _y(0) {}          // Initializer list
private:
   int _x;
   int _y;
};
```

Overloaded

An expression using an overloaded operator. For example:

```
#include <iostream.h>
void func()
{
   cout << "Hello, how old are you?"   // Overloaded <<
        << endl;                      // Overloaded <<
   int age;
   cin >> age;                        // Overloaded >>
}
```

Label

Label in goto statement. For example:

```
void func()
{
        int i;
        goto label1; //label with name 'label1'
        | := 0;
label1:
        |:=1;
}
```

a(b)

Function call expression. For example:

```
void foo(int i) {}
void bar(int x) {
    foo(x);      // Function call a(b).
                 // Left Hand Side is Function foo,
                 // Right Hand Side is Parameter x
}
```

a,b

Compound statement list. For example:

```
void func() {
    for (int i = 0; i >=0, i < 10; i++) {
       // a,b in for condition
    }
}
```

a->*b

Arrow-star expression used in pointers to member functions. For example:

```
class Foo {
public:
   void func();
};
typedef void(Foo::*mpf)();
void bar(Foo *f) {
  mpf ampf = &Foo::func;
  (f->*ampf)();               // a->*b
}
```

a.*b

Dot-star expression used in pointers to member functions. For example:

```
class Foo {
public:
   void func();
};
typedef void(Foo::*mpf)();
void bar(Foo f) {
  mpf ampf = &Foo::func;
  (f.*ampf)();                // a.*b
}
```

a.b

Dot operator expression. For example:

```
class Foo {
public:
  void func();
  int zzz;
};
int bar() {
  Foo f;
  f.func();                   // a.b
  return f.zzz;               // a.b
}
```

a::b

Obsolete. Formerly used to represent scope reference expressions. Use ScopePrefix instead.

a?b:c

Ternary operator expression. For example:

```
bool func(int x) {
  return x ? true : false;    // Ternary operator a?b:c
}
```

a[b]

Array reference. For example:

```
char array[] = "John Doe";
char c = array[0];            // array reference a[b]
```

asm

Inlined assembly expression. For example:

```
void func() {
   asm(mov eax, ebp);         // asm
}
```

delete

Delete operator expression. For example:

```
int *i = new int();
delete(i);                    // delete
```

ellipses (...)

Ellipses expression used for catch parameters. For example:

```
void func() {
  try {
  } catch(...) {}             // catch parameter is (...)
}
```

new

New operator expression. For example:

```
int *i = new int[20];   // new
```

sizeof

Sizeof operator expression. For example:

```
int *i = malloc(20*sizeof(int));  // sizeof
```

throw

Throw operator expression. For example:

```
void func() {
   throw;                         // throw
}
```

typeid

Typeid operator expression used for RTTI. For example:

```
class Shape {};
void f(Shape& r) {
   typeid(r);                     // typeid
```

noexcept

Noexcept operator. For example:

```
bool func(int x, int y) {
    return noexcept(x + y) || noexcept(func(x, y);
}
```

Vacuous destructor call

Explicit destructor call for simple types or classes that do not have a type. For simple types, this represents a pseudo-destructor call. For example:

```
struct S {} s;
s.S::~S();

typedef int INT;
int i;
i.INT::~INT();
```

Lambda

Lambda expression. For example:

```
#include <algorithm>
#include <cmath>
void mySort(float* x, unsigned N)
{
std::sort(x, x + N, [](float a, float b)
{ return a < b; }); // Lambda expression
}
```

Numerical

Numerical expressions that are not assignments.

+a

Positive signed expression. For example:

```
int func(int num) {
   return +num;               // +a
}
```

-a

Negative signed expression. For example:

```
int func(int num) {
   return -num;               // -a
}
```

a%b

Modulo expression. For example:

```
int j = 2;
int k = 5 % j;                // Mod a%b
```

a*b

Multiplication expression. For example:

```
int j = 1;
int k = 3 * j;                // Multiplication a*b
```

a+b

Addition expression. For example:

```
int j = 1;
int k = 3 + j;                // Multiplication a+b
```

a-b

Subtraction expression. For example:

```
int j = 1;
int k = 3 - j;           // Subtraction a-b
```

a/b

Division expression. For example:

```
int j = 2;
int k = 5 / j;           // Division a/b
```

a<<b

Left-shift expression. For example:

```
int i = 1, j = 1, k = 1;
k = i << j;              // Left-shift a<<b
```

a>>b

Right-shift expression. For example:

```
int i = 1, j = 1, k = 1;
k = i >> j;              // Right-shift a>>b
```

General

The following nodes do not easily fit into a particular category.

- Argument
- BaseClassDerivation
- BaseType
- File
- Lambda Capture
- Template Argument
- Translation Unit

Argument

Actual argument (actual parameter) of a function call. Returned by the "Arguments" property. For example:

```
void foo(int x);
void bar() {
    int param1;
    foo(param1);    // Argument: param1
}
```

BaseClassDerivation

Represents a possible inheritance path led from the parent class represented by BaseType to the child class. For example:

```
class C { };
class D : public C { };
class Final : public D, public C { };
```

For `Final` class, there are two BaseClassDerivation of class C: one direct and one through class D. There is also one BaseClassDerivation of class D.

BaseType

A Base Class of another class. For example:

```
class A : public B {} ; //public direct BaseClass B
```

File

File containing source code. Used for rules about relationships of nodes within a file.

Lambda Capture

Simple or init capture on a capture list. Includes also implicit captures provided through a capture-default. For example:

```
void foo() {
int w;
[&, a, b, z = 10]
{return w;}; // Four Lambda Captures: a, b, z, w
}
```

Template Argument

Represents actual argument used for instantiation of a template class or template function. For example:

```
template <typename T> class C {};
C<int> c;   // Template Argument: int
```

Translation Unit

Represents current translation unit.

Name Spaces

Nodes involving the use of namespaces.

- namespace
- using

namespace

Namespace declaration. For example:

```
namespace ns { // Name Space
 class Foo {}
};
```

using

Use of 'using' namespace keyword. For example:

```
#include <vector>
using namespace std; // using
```

- namespace
- using

Statements

General node for all types of statements.

Block

Block statement. Normally contains other statements. Begins and ends with curly braces. For example:

```
void func() {       // block
}
```

Case Lable

Represents a `case` or `default` label in a `switch` statement. For example:

```
void func(int i) {
    switch(i) {
    case 1:             // Case Label
        break;
    case 3:             // Case Label
    default:            // Case Label
        break;
    }
  }
```

Simple

A statement that does not fall into any other statement category. The simple statement's body usually contains an expression. For example:

```
void func(int & i) {
    i++;            // Simple statement
}
```

Empty

An empty `;` statement. For example:

```
void func() {
    if(true);             // Empty statement
}
```

break

Break statement. For example:

```
bool bar();
void func() {
   while(1) {
      if (bar()) {
          break;    // break
      }
   }
}
```

case

Case statement. For example:

```
void func(int i) {
    switch(i) {
    case 1: { break; }  // case
    case 2: { break; }  // case
    }
}
```

catch

Catch statement used for exception handling. For example:

```
void bar();
void func(int i) {
    try {
       bar();
    } catch (...) {     // catch
    }
}
```

continue

Continue statement used in loops. For example:

```
bool bar(int);
void func(int i) {
    while(i--) {
       if (bar(i)) {
          continue;     // continue
       }
    }
}
```

default

Default statement used inside a switch statement. For example:

```
void func(int i) {
    switch(i) {
    case 1: { break; }
    case 2: { break; }
    default: { break; }            // default
    }
}
```

do whjile

Do while loop statement. For example:

```
void func(int i) {
    do {                            // do while
        i--;
    } while (i);
}
```

for

For loop statement. For example:

```
void bar();
void func() {
   for (int i = 0; i < 10; i++) {  // for
      bar();
   }
}
```

for range

Range-based for loop statement. For example:

```
void bar();
void func() {
   for (auto iterator : range_expr) {  // for
      bar();
   }
}
```

goto

Goto statement. For example:

```
void func(bool done) {
   if (done) {
      goto end;             // goto
   }
   end:
}
```

if

If statement. For example:

```
void func(bool done) {
   if (done) {             // if
       return;
   }
}
```

return

Return statement. For example:

```
int func() {
   return 0;               // return
}
```

switch

Switch statement. For example:

```
void func(int i) {
    switch(i) {             // switch
    case 1: { break; }
    case 2: { break; }
    default: { break; }
    }
}
```

try

Try statement. For example:

```
void bar();
void func(int i) {
    try {                   // try
        bar();
    } catch (...) {
    }
}
```

while

While statement. For example:

```
void func(int i) {
   while (i--) {  // while
   }
}
```

Types

All types that can be used for variable declaration.

- Complex
    - Array
    - Builtin
    - Class
    - Decltype
    - Enum
    - Function
    - Reference
    - struct
    - Template Parameter
    - Typedef
    - Union
- Primitive
    - bool
    - char
    - double
    - float
    - int
    - long
    - long double
    - pointer
    - short
    - void
    - wchar_t
    - std::nullptr_t

Complex

Non-primitive types. This category contains the following types:

Array

Array type. For example:

```
char buf[1024];
      // buf is Variable of Type Array or Type char.
```

Builtin

Non-primitive type builtin for a given compiler.

Class

User-defined class type. For example:

```
class Foo { }; // class
```

Decltype

Type defined with the `decltype` specifier. For example:

```
int i;
decltype(i) k;            // k is Variable of Type Decltype to Type int.
```

Enum

User-defined enum type. For example:

```
enum E { }; //Enum type
```

Function

Function type. Used for function pointers. For example:

```
void foo(void (*ptr)(int)) {   // ptr is Pointer of Type Function
    ptr(3);
}
```

Reference

Reference type. Used for passing by reference. For example:

```
void func(int &i) {  // i is Parameter of Type Reference to Type int
   i = 3;
}
```

struct

Struct type. For example:

```
struct Foo { }; // struct
```

Template Parameter

A template parameter in template class declaration. For example:

```
template <typename T> class Temp{}; // Template Parameter: T
```

Typedef

Typedef type. For example:

```
typedef int INT;
INT x;
      // x is Variable of Type Typedef to Type int.
```

Union

Union type. For example":

```
union MyUnion {   // union
    int x;
    char *y;
};
```

Primitive

All primitive types. This category contains the following types:

bool

Primitive type bool. For example:

```
bool b; // b is Variable of Type bool
```

char

Primitive type char. For example:

```
char c; // c is Variable of Type char
```

double

Primitive type double. For example:

```
double d; // d is Variable of Type double
```

float

Primitive type float. For example:

```
float f; // f is Variable of Type float
```

int

Primitive type int. For example:

```
int i; // i is Variable of Type int
```

long

Primitive type long. For example:

```
long x; // x is Variable of Type long
```

long double

Primitive type long double. For example:

```
long double d; // d is Variable of Type long double
```

pointer

Pointer type. For example:

```
char * name;
 // name is Variable of Type pointer to type char.
```

short

Primitive type short. For example:

```
short x; // x is Variable of Type short
```

void

Primitive type void. Used for void pointers. For example:

```
void * p;
 // p is Variable of Type pointer to type void.
```

wchar_t

Primitive type wchar_t. For example:

```
wchar_t c; // c is Variable of Type wchar_t
```

std::nullptr_t

The type of pointer literal. Example:

```
decltype(nullptr) t; // variable of std::nullptr_t type
```