In C++, how can I make a member function default parameter depend on this?

devblogs.microsoft.com/oldnewthing/20231206-00

December 6, 2023

Raymond Chen

In C++, you can define a default parameter, but among the constraints on the default parameter is that it may not be `this` or anything that depends on `this` like a member variable or call to another member function.

```
struct Sample
{
    int increment;
    void add(int v = increment); // not allowed
    void notify_all(Sample* source = this); // not allowed
};
```

Although it is not allowed, you can fake it:

```
struct Sample
{
    int increment;

    void add(int v);
    void add() { add(increment); }

    void notify_all(Sample* source);
    void notify_all() { notify_all(this); }
};

Sample s;

s.add(2); // adds 2
s.add(); // adds s.increment

s.notify_all(); // uses source = s
s.notify_all(other); // uses source = other
```

This is an overload, rather than a default parameter. Whereas default parameters allow you to call the same function in multiple ways, overloads are separate functions entirely. This means that creating a pointer to a member function is much more frustrating for overloads since you have to specify which overload you are trying to use.

```
void (S::*f)() = &S::add; // selects S::add()
void (S::*f)(int) = &S::add; // selects S::add(int)
auto f = &S::add; // error: ambiguous
auto f = static_cast<void(S::*)()>(&S::add); // selects S::add()
```

This complexity with producing pointers to overloaded member functions bit us earlier when we tried to use `winrt::capture`. We can work around this by providing non-overloaded alternatives for cases where you would prefer that each overload have a unique name.

```
struct Sample
{
    int increment;

    void add(int v);
    void add() { add(increment); }
    void add0() { add(); }
    void add1(int v) { add(v); }

    void notify_all(Sample* source);
    void notify_all() { notify_all(this); }
    void notify_all0() { notify_all(); }
    void notify_all1(Sample* source) { notify_all(source); }
};

s.add();  // this calls the 0-parameter version
s.add(2); // this calls the 1-parameter version

auto f = &S::add; // error: ambiguous
auto f = &S::add0; // This is the zero-parameter version
auto f = &S::add1; // This is the one-parameter version
```