8.12 — Default arguments

ALEX JULY 4, 2021

A default argument is a default value provided for a function parameter. For example:

```
void print(int x, int y=10) // 10 is the default
argument
{
    std::cout << "x: " << x << '\n';
    std::cout << "y: " << y << '\n';
}
```

When making a function call, the caller can optionally provide an argument for any function parameter that has a default argument. If the caller provides an argument, the value of the argument in the function call is used. If the caller does not provide an argument, the value of the default argument is used.

Consider the following program:

```
#include <iostream>

void print(int x, int y=4) // 4 is the default
argument
{
    std::cout << "x: " << x << '\n';
    std::cout << "y: " << y << '\n';
}

int main()
{
    print(1, 2); // y will use user-supplied argument
2
    print(3); // y will use default argument 4
```

This program produces the following output:

```
x: 1
y: 2
x: 3
y: 4
```

In the first function call, the caller supplied explicit arguments for both parameters, so those argument values are used. In the second function call, the caller omitted the second argument, so the default value of `4` was used.

When to use default arguments

Default arguments are an excellent option when a function needs a value that has a reasonable default value, but for which you want to let the caller override if they wish.

For example, here are a couple of function prototypes for which default arguments might be commonly used:

```
int rollDie(int sides=6);
void openLogFile(std::string
filename="default.log");
```

> **Author's note**
>
> **Because the user can choose whether to supply a specific argument value or use the default value, a parameter with a default value provided is sometimes called an optional parameter. However, the term *optional parameter* is also used to refer to several other types of parameters (including parameters passed by address, and parameters using std::optional), so we recommend avoiding this term.**

Multiple default arguments

A function can have multiple parameters with default arguments:

```
#include <iostream>

void print(int x=10, int y=20, int z=30)
{
    std::cout << "Values: " << x << " " << y << " " << z <<
'\n';
}

int main()
{
    print(1, 2, 3); // all explicit arguments
    print(1, 2); // rightmost argument defaulted
    print(1); // two rightmost arguments defaulted
    print(); // all arguments defaulted

    return 0;
}

```

The following output is produced:

```
Values: 1 2 3
Values: 1 2 30
Values: 1 20 30
Values: 10 20 30
```

C++ does not (as of C++20) support a function call syntax such as `print(,,3)` (as a way to provide an explicit value for `z` while using the default arguments for `x` and `y`. This has two major consequences:

1. **Default arguments can only be supplied for the rightmost parameters. The following is not allowed:**

```
void print(int x=10, int y); // not
allowed
```

Rule

Default arguments can only be provided for the rightmost parameters.

2. **If more than one default argument exists, the leftmost default argument should be the one most likely to be explicitly set by the user.**

Default arguments can not be redeclared

Once declared, a default argument can not be redeclared (in the same file). That means for a function with a forward declaration and a function definition, the default argument can be declared in either the forward declaration or the function definition, but not both.

```
#include <iostream>

void print(int x, int y=4); // forward declaration

void print(int x, int y=4) // error: redefinition of default argument
{
    std::cout << "x: " << x << '\n';
    std::cout << "y: " << y << '\n';
}
```

Best practice is to declare the default argument in the forward declaration and not in the function definition, as the forward declaration is more likely to be seen by other files (particularly if it's in a header file).

in foo.h:

```
#ifndef FOO_H
#define FOO_H
void print(int x, int y=4);
#endif
```

in main.cpp:

```
#include "foo.h"
#include <iostream>

void print(int x, int y)
{
    std::cout << "x: " << x << '\n';
    std::cout << "y: " << y << '\n';
}

int main()
{
    print(5);

    return 0;
}
```

Note that in the above example, we're able to use the default argument for function `print()` because `main.cpp` #includes `foo.h`, which has the forward declaration that defines the default argument.

> **Best practice**
>
> **If the function has a forward declaration (especially one in a header file), put the default argument there. Otherwise, put the default argument in the function definition.**

Default arguments and function overloading

Functions with default arguments may be overloaded. For example, the following is allowed:

```
void print(std::string string)
{
}

void print(char ch=' ')
{
}

int main()
{
    print("Hello, world"); // resolves to print(std::string)
    print('a'); // resolves to print(char)
    print(); // resolves to print(char)

    return 0;
}
```

The function call to `print()` acts as if the user had explicitly called `print(' ')`, which resolves to `print(char)`.

Now consider this case:

```
void print(int x);
void print(int x, int y = 10);
void print(int x, double y = 20.5);
```

Parameters with default values will differentiate a function overload (meaning the above will compile).
However, such functions can lead to potentially ambiguous function calls. For example:

```
print(1, 2); // will resolve to printValues(int, int)
print(1, 2.5); // will resolve to printValues(int,
double)
print(1); // ambiguous function call
```

In the last case, the compiler is unable to tell whether `print(1)` should resolve to `print(int)` or one of the two function calls where the second parameter has a default value. The result is an ambiguous function call.

Summary

Default arguments provide a useful mechanism to specify values for parameters that the user may or may not want to override. They are frequently used in C++, and you'll see them a lot in future lessons.

Next lesson
8.13 Function templates

Back to table of contents

Previous lesson
8.11 Function overload resolution and ambiguous matches

Leave a comment... Put C++ code between triple-backticks (markdown style):```Your C++ coc

Name*

Email*

Avatars from https://gravatar.com/ are connected to your provided email address.

Notify me about replies: POST COMMENT

DP N N FOUT

Newest