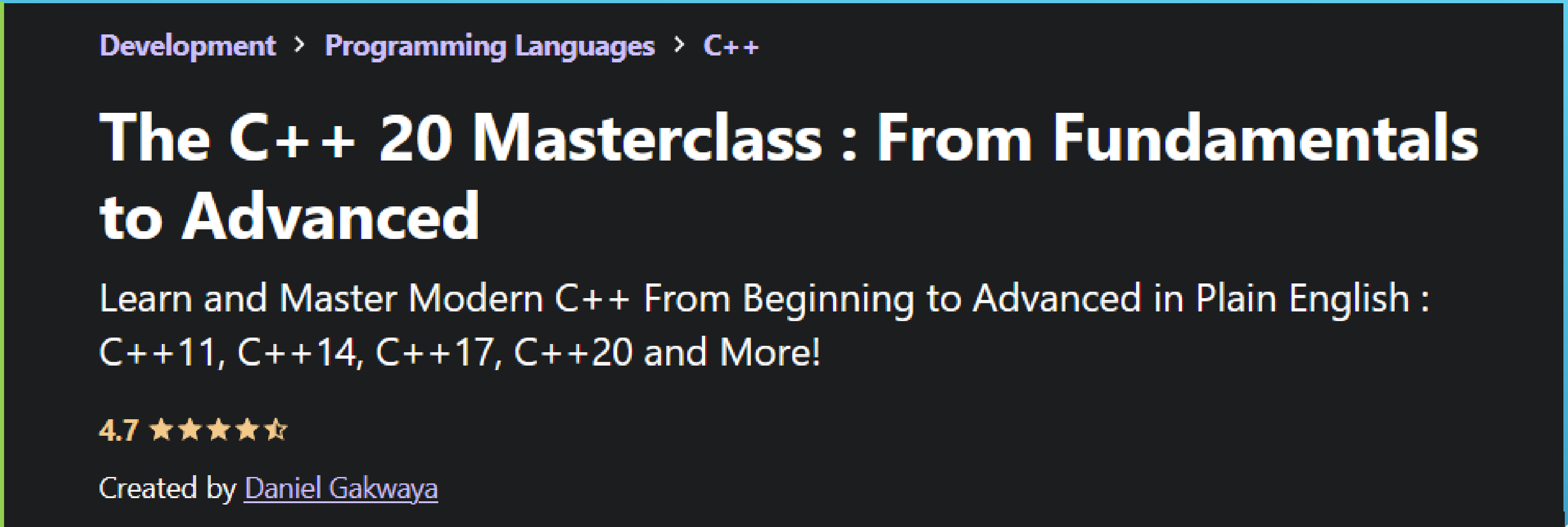
Slides



Development › Programming Languages › C++

The C++ 20 Masterclass : From Fundamentals to Advanced

Learn and Master Modern C++ From Beginning to Advanced in Plain English : C++11, C++14, C++17, C++20 and More!

4.7 ★★★★☆

Created by Daniel Gakwaya

Section : Function Overloading

Slide intentionally left empty

Function Overloading : Introduction

```cpp
int max(int a, int b);
double max( double a , double b);
std::string max( const std::string& a, const std::string& b);
```

```
max(a,b);
```

Slide intentionally left empty

Overloading with different parameters

Parameter differences

- Order
- Number
- Types

max() overloads

```
int max(int a, int b){
    return (a>b)? a : b;
}

/*
//Can't overload on the return type. Compiler error
double  max(int a, int b){
    return (a>b)? a : b;
}
*/

double max(double a, double b){
    return (a>b)? a : b;
}
std::string_view  max(std::string_view a, std::string_view b){
     return (a>b)? a : b;
}
```

Calling the overloads

```
int int_value1{41};
int int_value2{29};

double double_value1{47.2};
double double_value2{55.01};

std::string_view first{"Hello"};
std::string_view second{"World"};


std::cout << "max (" << int_value1 << "," << int_value2 << ") : "
            << max(int_value1,int_value2) << std::endl;

std::cout << "max (" << 5 << "," << 7 << ") : "
            << max(5,7) << std::endl;

std::cout << "max (" << double_value1 << "," << double_value2 << ") : "
            << max(double_value1,double_value2) << std::endl;

std::cout << "max (" << first << "," << second << ") : "
            << max(first,second) << std::endl;

std::cout << "max (dog,cat) : " <<  max("dog","cat") << std::endl;
```

Slide intentionally left empty

Overloading with pointer parameters

Pointers to different types are different types

```cpp
double max(double * numbers, size_t count){
    double maximum{0};

    for(size_t i{0}; i < count ;++i){
        if(numbers[i]> maximum)
            maximum = numbers[i];
    }
    return maximum;

}

int max(int * numbers, size_t count){
    int maximum{0};

    for(size_t i{0}; i < count ;++i){
        if(numbers[i]> maximum)
            maximum = numbers[i];
    }
    return maximum;

}
```

Equivalent declarations

```
int max(int * numbers, size_t count);

int max(int numbers[], size_t count);

int max(int numbers[10],size_t count);
```

Slide intentionally left empty

Overloading with reference parameters

Ambiguous functions

```
//Ambiguous calls
void say_my_name(const std::string& name){
    std::cout << "Your name is (ref) : " << name << std::endl;
}

void say_my_name( std::string name){
    std::cout << "Your name is (non ref) : " << name << std::endl;
}
```

Both functions all valid. The compiler doesn’t know which one to choose

```
// Ambiguous calls
std::string name{"Daniel"};

say_my_name(name); // Compiler error : Ambiguous call.

say_my_name("Daniel");// Compiler error : The compiler knows how to take
                        // a temporary const char* string and turn that into
                        // a reference. More details on this later in the course.
                        // So both functions are still valid for this call, thus, AMBIGUOUS
```

Implicit conversions with references : WATCH OUT!

```cpp
//Implicit conversions with references
double max(double a, double b){
    std::cout<< "double max called" << std::endl;
    return (a>b)?a:b;
}

int& max(int& a, int& b){
    std::cout << "int max called" << std::endl;
    return (a>b)?a:b;
}
```

Implicit conversions with references : WATCH OUT!

```
char a{45};
char b{62};

int maximum = max(a,b); // double version called
std::cout << "max : " << maximum << std::endl;
```

Potential solution

```
char a{45};
char b{62};

int int_a {static_cast<int>(a)};
int int_b {static_cast<int>(b)};

maximum = max(int_a,int_b);
std::cout << "max : " << maximum << std::endl;
```

Potential solution

```
const int& max(const int& a,const int& b){
    std::cout << "int max called" << std::endl;
    return (a>b)?a:b;
}
```

Slide intentionally left empty

Overloading with const parameters by value

Equivalent functions : REDEFINITION!

```cpp
int max(int a, int b){
    return (a > b)? a : b;
}

int max(const int a, const int b){
    return (a > b)? a : b;
}

int main(int argc, char **argv)
{
    std::cout << "Hello World in C++20!" << std::endl;
    return 0;
}
```

const only in definition

```cpp
//int min(const int a,const int b);
int min(int a, int b);

int main(int argc, char **argv)
{
    std::cout << "Hello World in C++20!" << std::endl;
    return 0;
}

int min(const int a,const int b){
       ++a; // Compiler error
        return (a < b)? a : b;
}
```

Slide intentionally left empty

Overloading with const pointer and pointer to const parameters

Valid unique overloads. No REDEFINITION

```
int max(int*a , int* b){
    std::cout << "max with int* called" << std::endl;
    return (*a > *b)? *a : *b;
}

int max(const int* a, const int* b){
    std::cout << "max with cont int* called" << std::endl;
    return (*a > *b)? *a : *b;
}
```

Valid unique overloads. No REDEFINITION

```
int a{10};
int b{12};
const int c{30};
const int d{15};

int maximum = max(&a,&b);
std::cout << "max : " << maximum << std::endl;

maximum = max(&c,&d);
std::cout << "max : " << maximum << std::endl;
```

Equivalent in the eyes of the compiler. REDEFINITION!

```
int min(const int* a, const int* b){
    return (*a < *b)? *a : *b;
}

int min(const int* const a, const int* const b){
    return (*a < *b)? *a : *b;
}
```

```
int a{10};
int b{12};

const int* p_a {&a};
const int* p_b {&b};

int minimum = min(p_a,p_b);
std::cout << "min : " << minimum << std::endl;
```

Showing addresses where pointers live

```
int min(const int* const a, const int* const b){
    std::cout << "&a : " << &a << std::endl;
    std::cout << "&b : " << &b << std::endl;
    return (*a < *b)? *a : *b;
}

int main(int argc, char **argv)
{
    int a{10};
    int b{12};
    const int* p_a {&a};
    const int* p_b {&b};

    std::cout << "p_a : " << p_a << std::endl;
    std::cout << "p_b : " << p_b << std::endl;

    int minimum = min(p_a,p_b);
    std::cout << "min : " << minimum << std::endl
                                                 ;

    return 0;
}
```

Slide intentionally left empty

Overloading with const references

Valid unique overloads. No REDEFINITION

```
//These functions retun a copy, not a refernce in any way
int max(int& a, int&b){
    std::cout << "max with int& called" << std::endl;

    //Can change a and b through the reference
    a = 200;

    return (a > b)? a : b;
}

int max(const int& a, const int& b){
    std::cout << "max with const int& called" << std::endl;

    //Can NOT change a and b through the reference
    //a = 200; // Will give a compiler error.
    return (a > b)? a : b;
}
```

Valid unique overloads. No REDEFINITION

```
int a{45};
int b{85};

std::cout << std::endl;
std::cout << "first call : " << std::endl;

int max1 = max(a,b);
std::cout << "max1 : " << max1 << std::endl;

const int& ref_a = a;
const int& ref_b = b;

std::cout << std::endl;
std::cout << "first call : " << std::endl;

int max2 = max(ref_a,ref_b);
std::cout << "max2 : " << max2 << std::endl;
```

Slide intentionally left empty

Overloads with default parameters

```
void print_age(int age = 33);

void print_age(long int age = 44);

int main(int argc, char **argv)
{
    print_age();

    return 0;
}


void print_age(int age ){
    std::cout << "Your age is( int version)  : " << age << std::endl;
}

void print_age(long int age){
    std::cout << "Your age is (long int version) : " << age << std::endl;
}
```

Slide intentionally left empty

Function Overloading : Summary

```cpp
int max(int a, int b);
double max( double a , double b);
std::string max( const std::string& a, const std::string& b);
```

```
max(a,b);
```

Overloads with different parameters

```
int max(int a, int b){
    return (a>b)? a : b;
}

/*
//Can't overload on the return type. Compiler error
double  max(int a, int b){
    return (a>b)? a : b;
}
*/

double max(double a, double b){
    return (a>b)? a : b;
}
std::string_view  max(std::string_view a, std::string_view b){
     return (a>b)? a : b;
}
```

Overloads with pointer parameters

```
double max(double * numbers, size_t count){
    double maximum{0};

    for(size_t i{0}; i < count ;++i){
        if(numbers[i]> maximum)
            maximum = numbers[i];
    }
    return maximum;

}

int max(int * numbers, size_t count){
    int maximum{0};

    for(size_t i{0}; i < count ;++i){
        if(numbers[i]> maximum)
            maximum = numbers[i];
    }
    return maximum;

}
```

Overloads with reference parameters

```
//Ambiguous calls
void say_my_name(const std::string& name){
    std::cout << "Your name is (ref) : " << name << std::endl;
}

void say_my_name( std::string name){
    std::cout << "Your name is (non ref) : " << name << std::endl;
}
```

Overload with reference parameters

```cpp
//Implicit conversions with references
double max(double a, double b){
    std::cout<< "double max called" << std::endl;
    return (a>b)?a:b;
}

int& max(int& a, int& b){
    std::cout << "int max called" << std::endl;
    return (a>b)?a:b;
}
```

Overloads with const parameters by value

```
int max(int a, int b){
    return (a > b)? a : b;
}

int max(const int a, const int b){
    return (a > b)? a : b;
}

int main(int argc, char **argv)
{
    std::cout << "Hello World in C++20!" << std::endl;
    return 0;
}
```

Overloading with pointer to const

```
int max(int*a , int* b){
    std::cout << "max with int* called" << std::endl;
    return (*a > *b)? *a : *b;
}

int max(const int* a, const int* b){
    std::cout << "max with cont int* called" << std::endl;
    return (*a > *b)? *a : *b;
}
```

Overloading with const pointer

```
int min(const int* a, const int* b){
    return (*a < *b)? *a : *b;
}

int min(const int* const a, const int* const b){
    return (*a < *b)? *a : *b;
}
```

Overloading with const references

```
//These functions retun a copy, not a refernce in any way
int max(int& a, int&b){
    std::cout << "max with int& called" << std::endl;

    //Can change a and b through the reference
    a = 200;

    return (a > b)? a : b;
}

int max(const int& a, const int& b){
    std::cout << "max with const int& called" << std::endl;

    //Can NOT change a and b through the reference
    //a = 200; // Will give a compiler error.
    return (a > b)? a : b;
}
```

Function overloads with default arguments

```
void print_age(int age = 33);

void print_age(long int age = 44);

int main(int argc, char **argv)
{
    print_age();

    return 0;
}


void print_age(int age ){
    std::cout << "Your age is( int version)  : " << age << std::endl;
}

void print_age(long int age){
    std::cout << "Your age is (long int version) : " << age << std::endl;
}
```

Slide intentionally left empty