# 11.x — Chapter 11 summary and quiz

ALEX[1] SEPTEMBER 28, 2024

Nice work. Function templates may seem pretty complex, but they are a very powerful way to make your code work with objects of different types. We'll see a lot more template stuff in future chapters, so hold on to your hat.

## Chapter Review

**Function overloading** allows us to create multiple functions with the same name, so long as each identically named function has different set of parameter types (or the functions can be otherwise differentiated). Such a function is called an **overloaded function** (or **overload** for short). Return types are not considered for differentiation.

When resolving overloaded functions, if an exact match isn't found, the compiler will favor overloaded functions that can be matched via numeric promotions over those that require numeric conversions. When a function call is made to function that has been overloaded, the compiler will try to match the function call to the appropriate overload based on the arguments used in the function call. This is called **overload resolution**.

An **ambiguous match** occurs when the compiler finds two or more functions that can match a function call to an overloaded function and can't determine which one is best.

A **default argument** is a default value provided for a function parameter. Parameters with default arguments must always be the rightmost parameters, and they are not used to differentiate functions when resolving overloaded functions.

**Function templates** allow us to create a function-like definition that serves as a pattern for creating related functions. In a function template, we use **type template parameters** as placeholders for any types we want to be specified later. The syntax that tells the compiler we're defining a template and declares the template types is called a **template parameter declaration**.

The process of creating functions (with specific types) from function templates (with template types) is called **function template instantiation** (or **instantiation**) for short. When this process happens due to a function call, it's called **implicit instantiation**. An instantiated function is called a **function instance** (or **instance** for short, or sometimes a **template function**).

**Template argument deduction** allows the compiler to deduce the actual type that should be used to instantiate a function from the arguments of the function call. Template argument deduction does not do type conversion.

Template types are sometimes called **generic types**, and programming using templates is sometimes called **generic programming**.

In C++20, when the auto keyword is used as a parameter type in a normal function, the compiler will automatically convert the function into a function template with each auto parameter becoming an

independent template type parameter. This method for creating a function template is called an **abbreviated function template**.

A **non-type template parameter** is a template parameter with a fixed type that serves as a placeholder for a constexpr value passed in as a template argument.

---

## Quiz time

### Question #1

1a) What is the output of this program and why?

```
#include <iostream>

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

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

int main()
{
    short s { 5 };
    print(s);

    return 0;
}
```

The output is int 5. Converting a short to an int is a numeric promotion, whereas converting a short to a double is a numeric conversion.

Show Solution (javascript:void(0))[2]

1b) Why won't the following compile?

```
#include <iostream>

void print()
{
    std::cout << "void\n";
}

void print(int x=0)
{
    std::cout << "int " << x << '\n';
}

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

int main()
{
    print(5.0f);
    print();

    return 0;
}
```

Because parameters with default arguments aren't counted for resolving overloaded functions, the compiler can't tell whether the call to print() should resolve to print() or print(int x=0).

Show Solution (javascript:void(0))[2]

1c) Why won't the following compile?

```
#include <iostream>

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

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

int main()
{
    print(5);

    return 0;
}
```

Converting an int to a long or a double is a numeric conversion

Show Solution (javascript:void(0))[2]

---

**Question #2**

---

**> Step #1**

Write a function template named `add()` that allows the users to add 2 values of the same type. The following program should run:

```
#include <iostream>

// write your add function template here

int main()
{
    std::cout << add(2, 3) << '\n';
    std::cout << add(1.2, 3.4) << '\n';

    return 0;
}
```

and produce the following output:

```
5
4.6
```

Show Solution (javascript:void(0))[2]

---

**> Step #2**

Write a function template named `mult()` that allows the user to multiply one value of any type (first parameter) and an integer (second parameter). The second parameter should not be a template type. The function should return the same type as the first parameter. The following program should run:

```cpp
#include <iostream>

// write your mult function template here

int main()
{
    std::cout << mult(2, 3) << '\n';
    std::cout << mult(1.2, 3) << '\n';

    return 0;
}
```

and produce the following output:

```
6
3.6
```

Show Solution (javascript:void(0))[2]

---

**> Step #3**

Write a function template named `sub()` that allows the user to subtract two values of different types. The following program should run:

```cpp
#include <iostream>

// write your sub function template here

int main()
{
    std::cout << sub(3, 2) << '\n';
    std::cout << sub(3.5, 2) << '\n';
    std::cout << sub(4, 1.5) << '\n';

    return 0;
}
```

and produce the following output:

```
1
1.5
2.5
```

Show Solution (javascript:void(0))[2]

---

**Question #3**

What is the output of this program and why?

```cpp
#include <iostream>

template <typename T>
int count(T) // This is the same as int count(T x), except we're not giving the parameter a name since we don't use the parameter
{
    static int c { 0 };
    return ++c;
}

int main()
{
    std::cout << count(1) << '\n';
    std::cout << count(1) << '\n';
    std::cout << count(2.3) << '\n';
    std::cout << count<double>(1) << '\n';

    return 0;
}
```

Show Solution (javascript:void(0))[2]

---

**Question #4**

What is the output of this program?

```cpp
#include <iostream>

int foo(int n)
{
    return n + 10;
}

template <typename T>
int foo(T n)
{
    return n;
}

int main()
{
    std::cout << foo(1) << '\n'; // #1

    short s { 2 };
    std::cout << foo(s) << '\n'; // #2

    std::cout << foo<int>(4) << '\n'; // #3

    std::cout << foo<int>(s) << '\n'; // #4

    std::cout << foo<>(6) << '\n'; // #5

    return 0;
}
```

Show Solution (javascript:void(0))[2]

**Next lesson**
F.1 Constexpr functions

3

**Back to table of contents**

4

**Previous lesson**
11.10 Using function templates in multiple files

5

6

B U URL INLINE CODE C++ CODE BLOCK HELP!

Leave a comment...

Name*

Email*

Notify me about replies:

POST COMMENT

Find a mistake? Leave a comment above!

Avatars from https://gravatar.com/[9] are connected to your provided email address.

42 COMMENTS

Newest

Copernicus
May 13, 2025 3:43 am PDT

Question #4
Got a couple wrong, because I thought the complier would always prefer ta non-template function when given the choice.
11
12 - thought complier would prefer non-template version
4
2
16 - thought complier would prefer non-template version

1 Reply

**Kacper**

Reply to Copernicus [10] June 25, 2025 2:37 pm PDT

So do I

0 Reply

**Copernicus**

May 13, 2025 3:36 am PDT

Question #3

1

2

1

2

0 Reply

**Copernicus**

May 13, 2025 3:32 am PDT

Question #2

Step 1:

```
#include <iostream>

template <typename T>
T add(T firstNumber, T secondNumber)
{
    return firstNumber + secondNumber;
}

int main()
{
    std::cout << add(2, 3) << '\n';
    std::cout << add(1.2, 3.4) << '\n';

    return 0;
}
```

Step 2:

```cpp
#include <iostream>

template <typename T>
T mult(T firstNumber, int secondNumber)
{
    return firstNumber * secondNumber;
}

int main()
{
    std::cout << mult(2, 3) << '\n';
    std::cout << mult(1.2, 3) << '\n';

    return 0;
}
```

Step 3:

```cpp
#include <iostream>

template <typename T, typename U>
auto sub(T firstNumber, U secondNumber)
{
    return firstNumber - secondNumber;
}

int main()
{
    std::cout << sub(3, 2) << '\n';
    std::cout << sub(3.5, 2) << '\n';
    std::cout << sub(4, 1.5) << '\n';

    return 0;
}
```

0 Reply

**Copernicus**

May 13, 2025 3:26 am PDT

Question #1
a: int 5, short s gets promoted to an int.
b: ambiguous match
c: ambiguous match, both are conversations

0 Reply

**DLTX**

February 8, 2025 8:52 am PST

Hey Alex, I just wanted to say that I've really been enjoying learning C++ from you! it's been a great learning experience. That said, I sometimes feel like I'm not practicing enough. Could you recommend some good places to find practice projects or exercises? I don't want to just read , I want to apply what I'm learning. Thanks :D

0 Reply

**Alex** Author

Reply to DLTX [11] February 11, 2025 5:21 pm PST

Just added: https://www.learncpp.com/cpp-tutorial/cpp-faq/#morepractice

3 Reply

**Joaquin**

January 27, 2025 6:47 pm PST

Hi Alex! I've been working through these tutorials since last August. I know C++ is a very broad language and it's not possible to cover everything, but have you gone over/are you planning to go over variadic templates? What is a good place to study them considering I already read chapter 11 and I'm currently going over chapter 27?

P.S.: Thank you so much for these tutorials! I was afraid of learning C++ and I stumbled upon this website while reading about learning C++ in reddit. It's a very complete and in-depth tutorial/guide and I'm sure everyone who uses these free learning resources is as thankful as me.

3 Reply

**Alex** Author

Reply to Joaquin [12] January 29, 2025 7:30 pm PST

At some point I will cover them. But I doubt soon. There are so many other topics of interest and areas of higher use that need improvement. I can't advise on where the best place to study them is, as I haven't done a lot of research yet.

0 Reply

**Dev Mangal**

Reply to Alex [13] April 8, 2025 2:13 pm PDT

i know that u said you cant tell us the best place to study them yet, but do you feel like after completing the series, other than practice we would need to cover other topics in c++ (other than dsa) to pursue a career in it? By career i mean like becoming a developer or going into competitive programming and all

0 Reply

**rasp**

January 18, 2025 1:59 am PST

```cpp
#include <iostream>

//#define CP2020

#ifndef CP2020

template <typename T>
T add(T x, T y){
    return x+y;
}

template <typename T, typename U>
T mult(T x, U y){
    return x*y;
}

template <typename T, typename U>
auto sub(T x, U y){
    return x-y;
}

#endif

#ifdef CP2020
auto add(auto x, auto y){
    return x+y;
}
auto mult(auto x, auto y){
    return x*y;
}
auto sub(auto x, auto y){
    return x-y;
}
#endif

int main()
{
    // std::cout << add(2, 3) << '\n';
    // std::cout << add(1.2, 3.4) << '\n';

    // std::cout << mult(2, 3) << '\n';
    // std::cout << mult(1.2, 3) << '\n';

    std::cout << sub(3, 2) << '\n';
    std::cout << sub(3.5, 2) << '\n';
    std::cout << sub(4, 1.5) << '\n';

    return 0;
}
```

0 Reply

walaza

January 9, 2025 8:26 pm PST

What exactly do the empty brackets `<>` mean in the context of templates? I've also noticed them when you type out the code that gets generated by the compiler when functions for a specific type are instantiated.

0 Reply

**PooperShooter**
December 28, 2024 4:18 pm PST

lightwork no reaction

1 Reply

**Jaximus Prime**
December 28, 2024 3:39 pm PST

For Step 3 for subtraction Would this code also worK

```
#include <iostream>

// write your sub function template here
template <typename T>
T sub(T x, T y) {

    return x - y;
}

int main()
{
    std::cout << sub(3, 2) << '\n';
    std::cout << sub<double>(3.5, 2) << '\n';
    std::cout << sub<double>(4, 1.5) << '\n';

    return 0;
}
```

0 Reply

**WilQ**
Reply to Jaximus Prime [14] December 31, 2024 6:36 pm PST

Hello. I am also learning, so don't quote me on this.
Anyways, your code does work, but it doesn't solve the question the author had asked. After you wrote your code, you were supposed to run it with the given function calls meant to be pasted in main():

```
int main() {
        std::cout << sub(3, 2) << '\n';
    std::cout << sub(3.5, 2) << '\n';
    std::cout << sub(4, 1.5) << '\n';
}
```

Furthermore, the question also asks you to create code that "allows the user to subtract two values of different types," essentially asking you to create two typenames in the template. Your code only allows for one (although two through type conversions) because there is only one `typename` in the template, thus allowing for only one type substitution.
Overall, your code does work, but you didn't solve it the way the author intended to.

*Last edited 5 months ago by WilQ*

1 Reply

# Links

1. https://www.learncpp.com/author/Alex/
2. javascript:void(0)
3. https://www.learncpp.com/cpp-tutorial/constexpr-functions/
4. https://www.learncpp.com/
5. https://www.learncpp.com/cpp-tutorial/using-function-templates-in-multiple-files/
6. https://www.learncpp.com/chapter-11-summary-and-quiz/
7. https://www.learncpp.com/cpp-tutorial/global-random-numbers-random-h/
8. https://www.learncpp.com/cpp-tutorial/returning-stdvector-and-an-introduction-to-move-semantics/
9. https://gravatar.com/
10. https://www.learncpp.com/cpp-tutorial/chapter-11-summary-and-quiz/#comment-609987
11. https://www.learncpp.com/cpp-tutorial/chapter-11-summary-and-quiz/#comment-607510
12. https://www.learncpp.com/cpp-tutorial/chapter-11-summary-and-quiz/#comment-607084
13. https://www.learncpp.com/cpp-tutorial/chapter-11-summary-and-quiz/#comment-607165
14. https://www.learncpp.com/cpp-tutorial/chapter-11-summary-and-quiz/#comment-605830
15. https://g.ezoic.net/privacy/learncpp.com