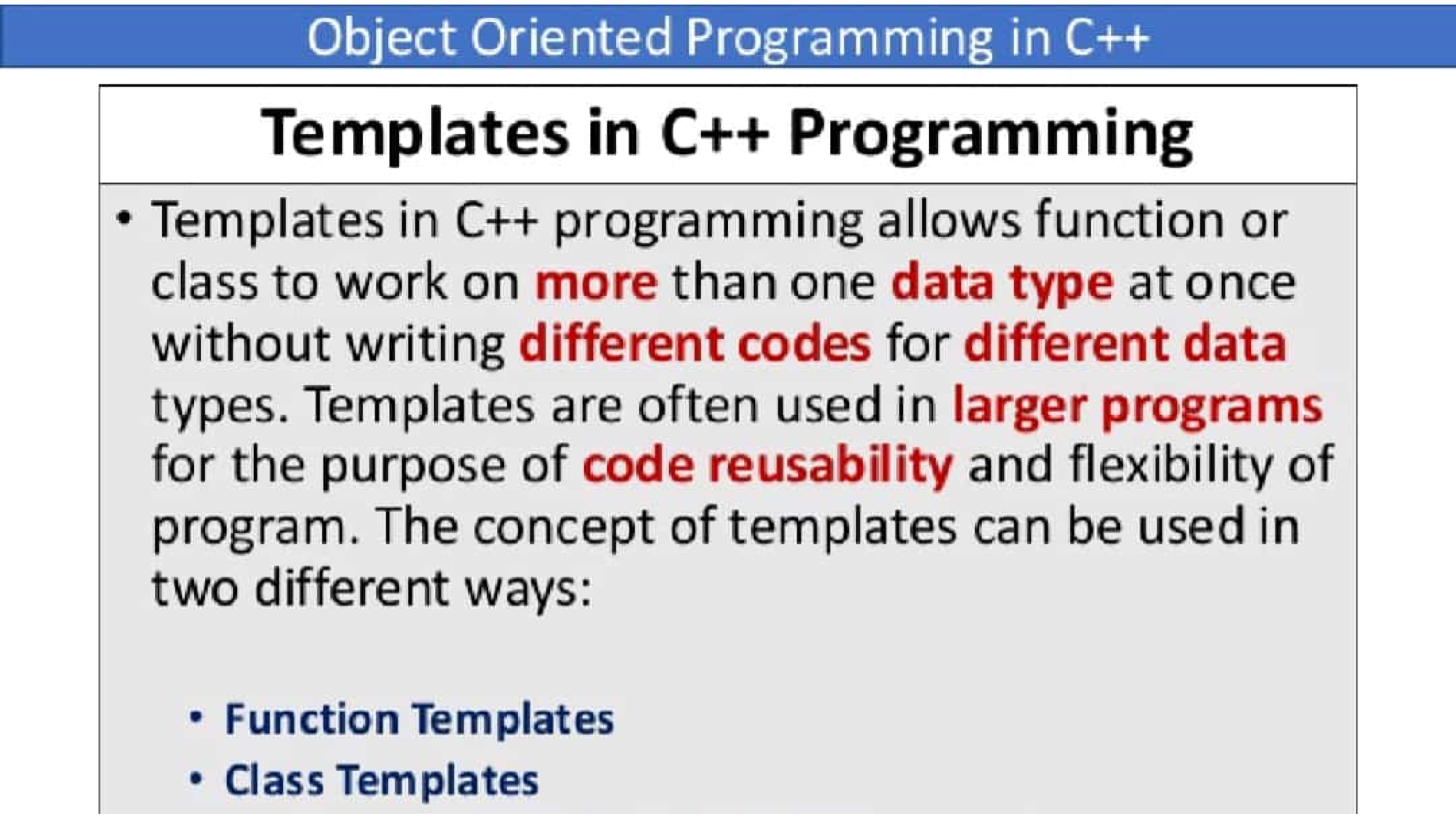
Templates in C++



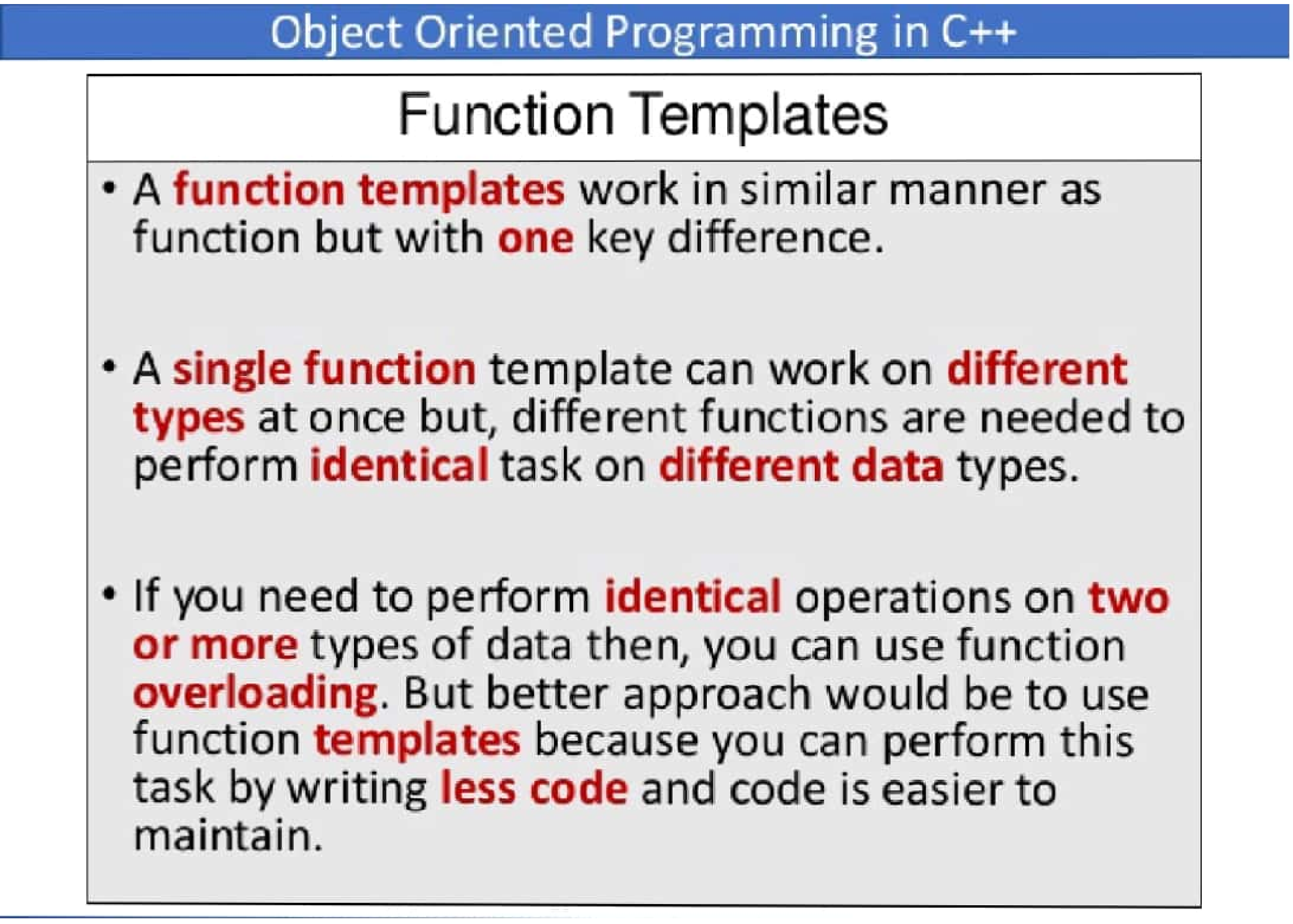
Templates in C++ Programming

- Templates in C++ programming allows function or class to work on **more** than one **data type** at once without writing **different codes** for **different data** types. Templates are often used in **larger programs** for the purpose of **code reusability** and flexibility of program. The concept of templates can be used in two different ways:

  - **Function Templates**
  - **Class Templates**

Templates in C++ Programming

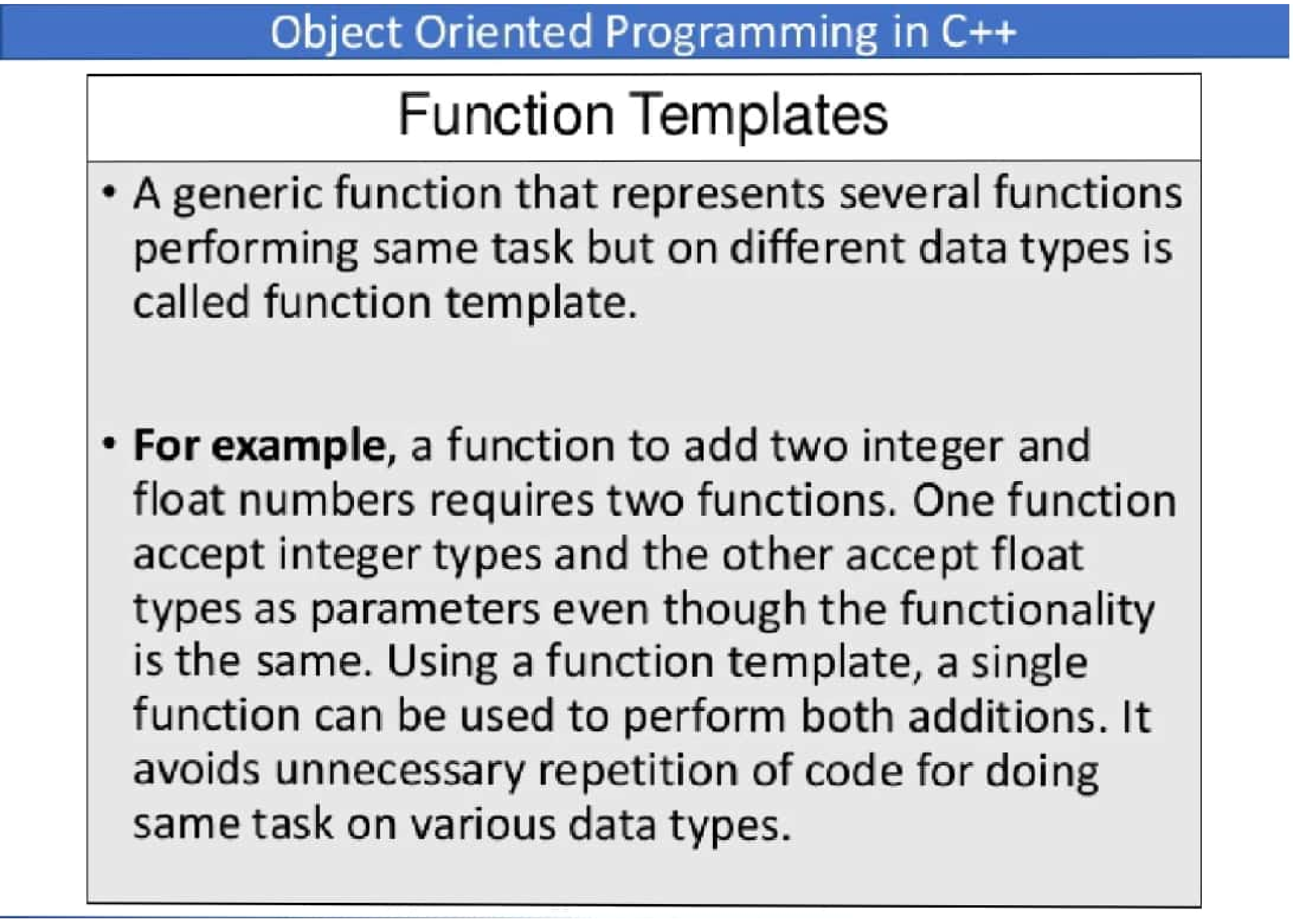
- Templates allow programmer to create a common class or function that can be used for a variety of data types. The parameters used during its definition is of generic type and can be replaced later by actual parameters. This is known as the concept of **generic programming**. The main advantage of using a template is the reuse of same algorithm for various data types, hence saving time from writing similar codes.
- **For example**, consider a situation where we have to sort a list of students according to their roll number and their percentage. Since, roll number is of integer type and percentage is of float type, we need to write separate sorting algorithm for this problem. But using template, we can define a generic data type for sorting which can be replaced later by integer and float data type.



Object Oriented Programming in C++

Function Templates

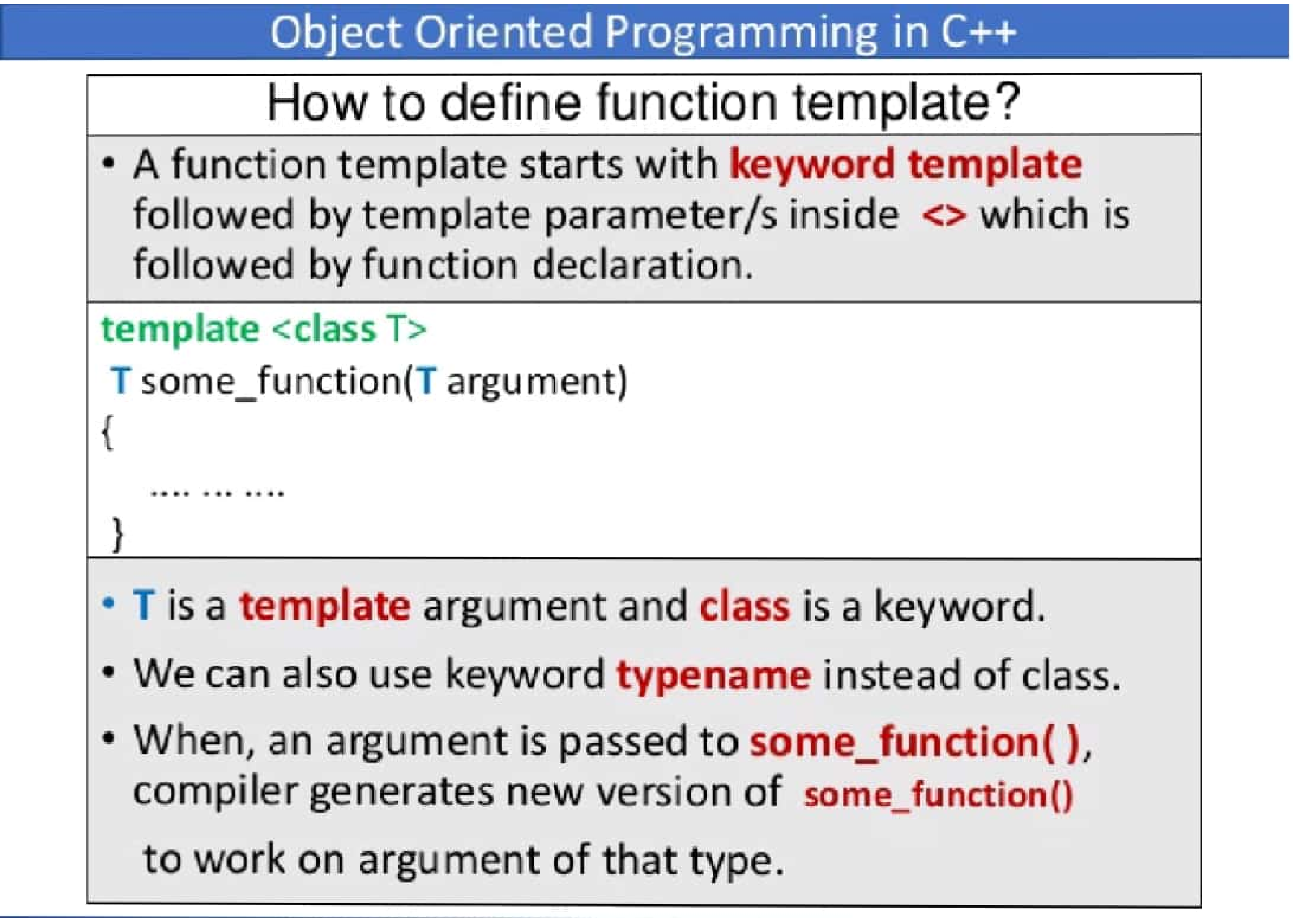
- A **function templates** work in similar manner as function but with **one** key difference.
- A **single function** template can work on **different types** at once but, different functions are needed to perform **identical** task on **different data** types.
- If you need to perform **identical** operations on **two or more** types of data then, you can use function **overloading**. But better approach would be to use function **templates** because you can perform this task by writing **less code** and code is easier to maintain.



Function Templates

- A generic function that represents several functions performing same task but on different data types is called function template.

- **For example**, a function to add two integer and float numbers requires two functions. One function accept integer types and the other accept float types as parameters even though the functionality is the same. Using a function template, a single function can be used to perform both additions. It avoids unnecessary repetition of code for doing same task on various data types.

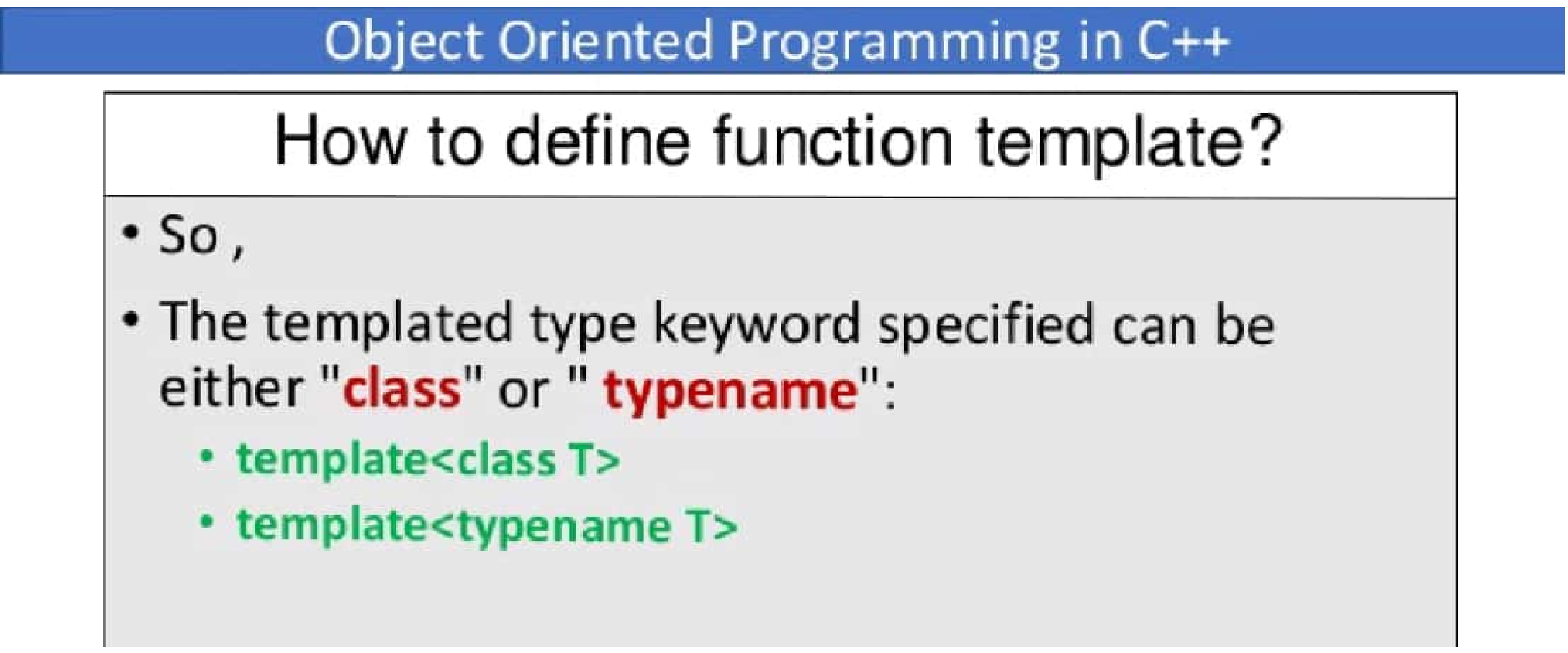


How to define function template?

- A function template starts with **keyword template** followed by template parameter/s inside **<>** which is followed by function declaration.

```
template <class T>
 T some_function(T argument)
{
    .... ... ....
 }
```

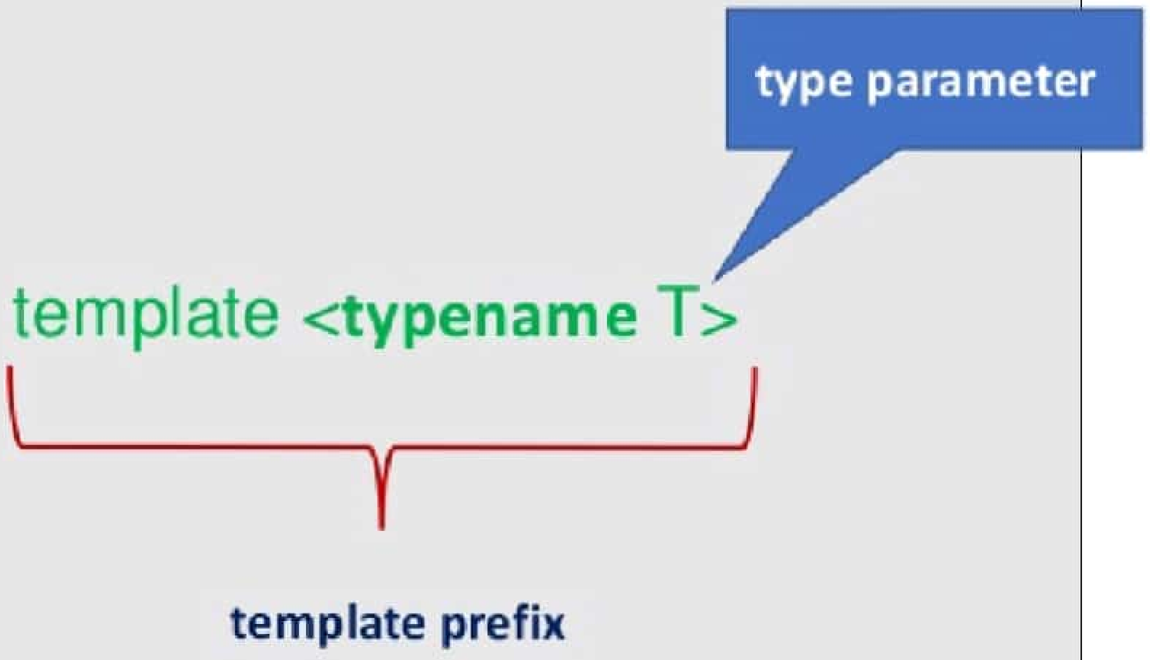
- **T** is a **template** argument and **class** is a keyword.
- We can also use keyword **typename** instead of class.
- When, an argument is passed to **some_function( )**, compiler generates new version of **some_function()** to work on argument of that type.



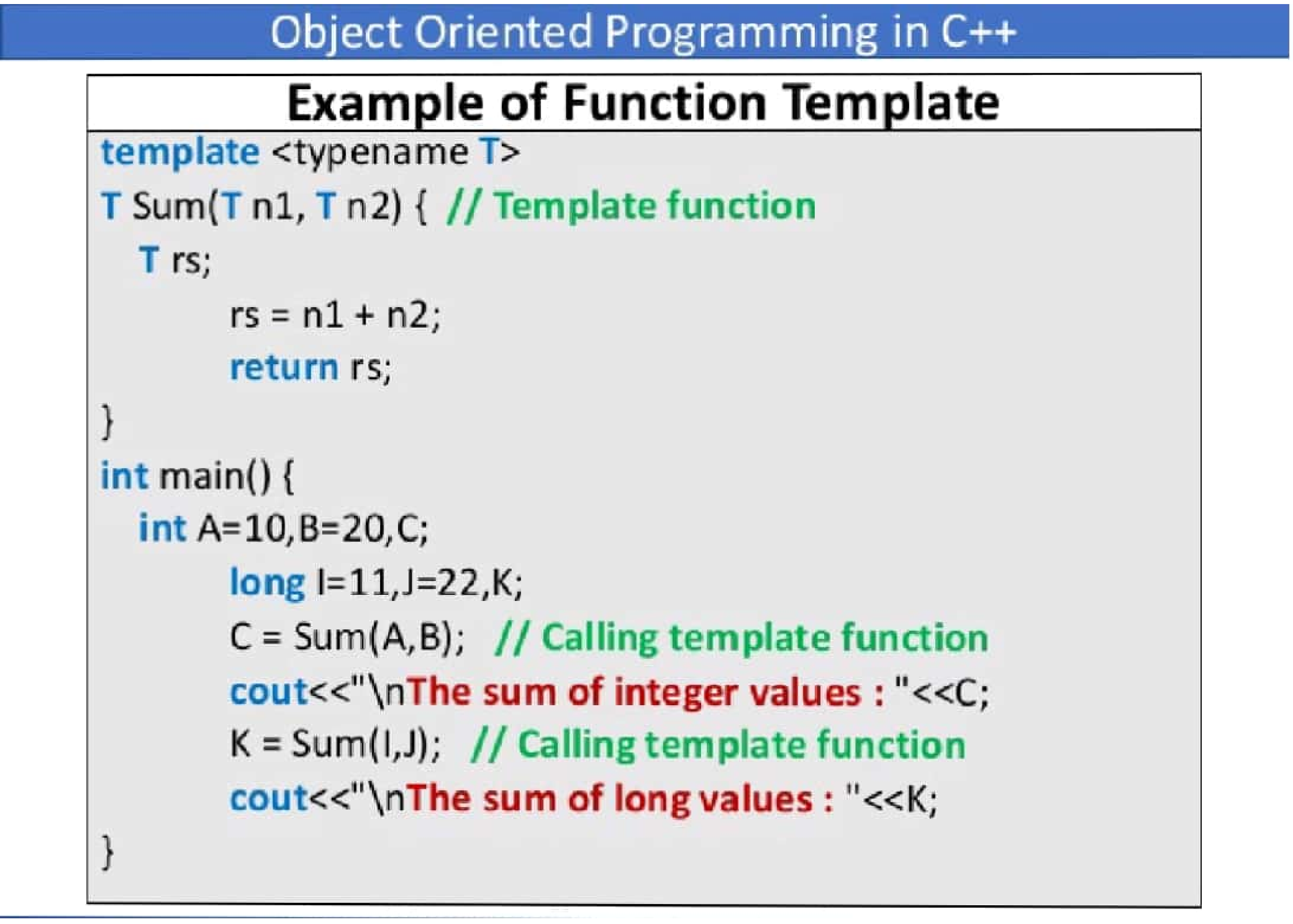
How to define function template?

- So ,
- The templated type keyword specified can be either "**class**" or " **typename**":
  - **template<class T>**
  - **template<typename T>**

Function Templates



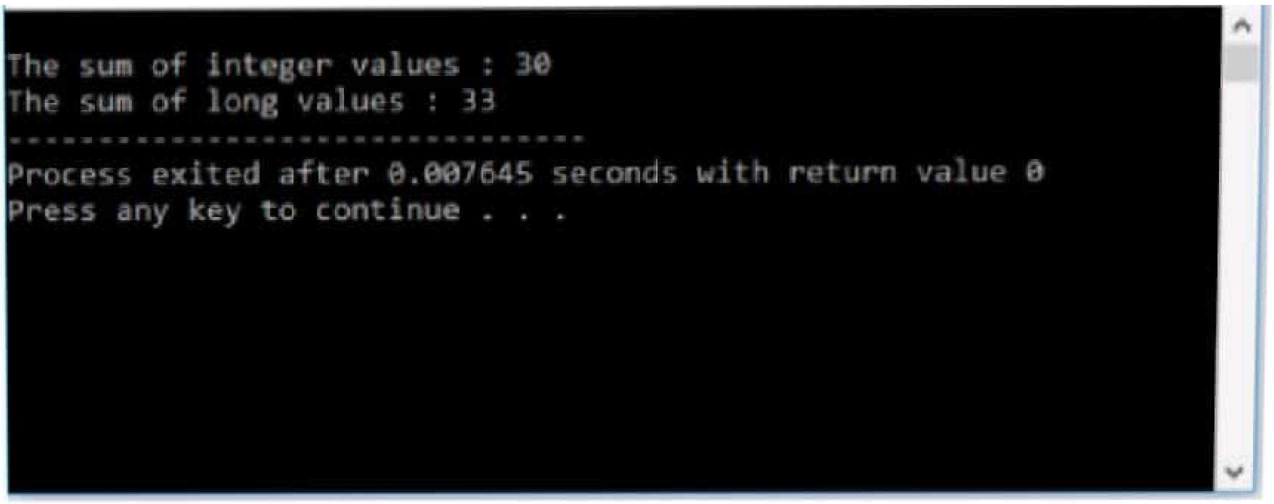
```
template <typename T>
```



Object Oriented Programming in C++

Example of Function Template

```
template <typename T>
T Sum(T n1, T n2) {  // Template function
   T rs;
         rs = n1 + n2;
         return rs;
}
int main() {
   int A=10,B=20,C;
         long I=11,J=22,K;
         C = Sum(A,B);   // Calling template function
         cout<<"\nThe sum of integer values : "<<C;
         K = Sum(I,J);   // Calling template function
         cout<<"\nThe sum of long values : "<<K;
}
```



```
The sum of integer values : 30
The sum of long values : 33
--------------------------------
Process exited after 0.007645 seconds with return value 0
Press any key to continue . . .
```

Function Templates

Example to show you function template use less code than function overloading

Function Overloading Example

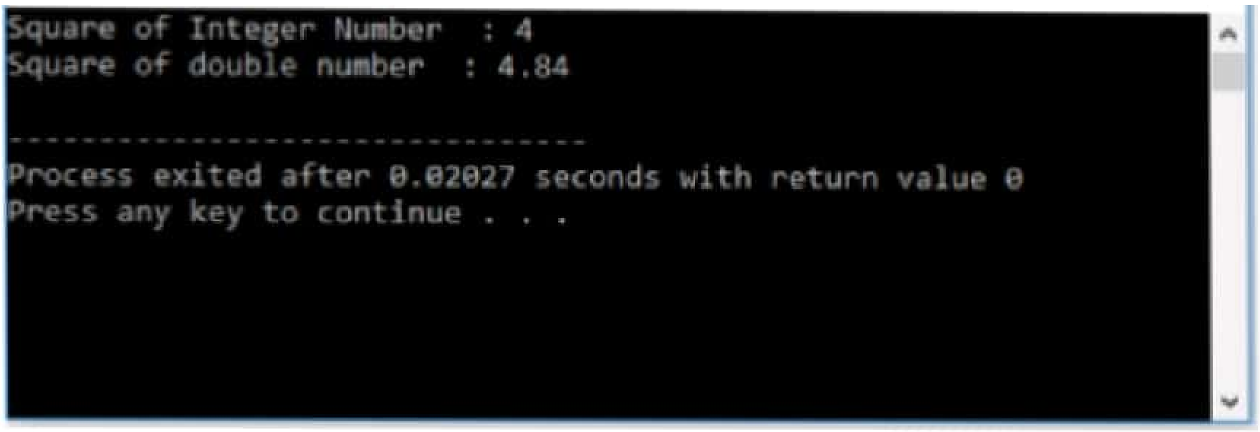
Overloaded functions specified for each data type

Function Overloading Example-1

```
#include <iostream>
using namespace std;
int square (int x)
{
  return x * x;
}
double square (double x)
{
  return x * x;
}
```

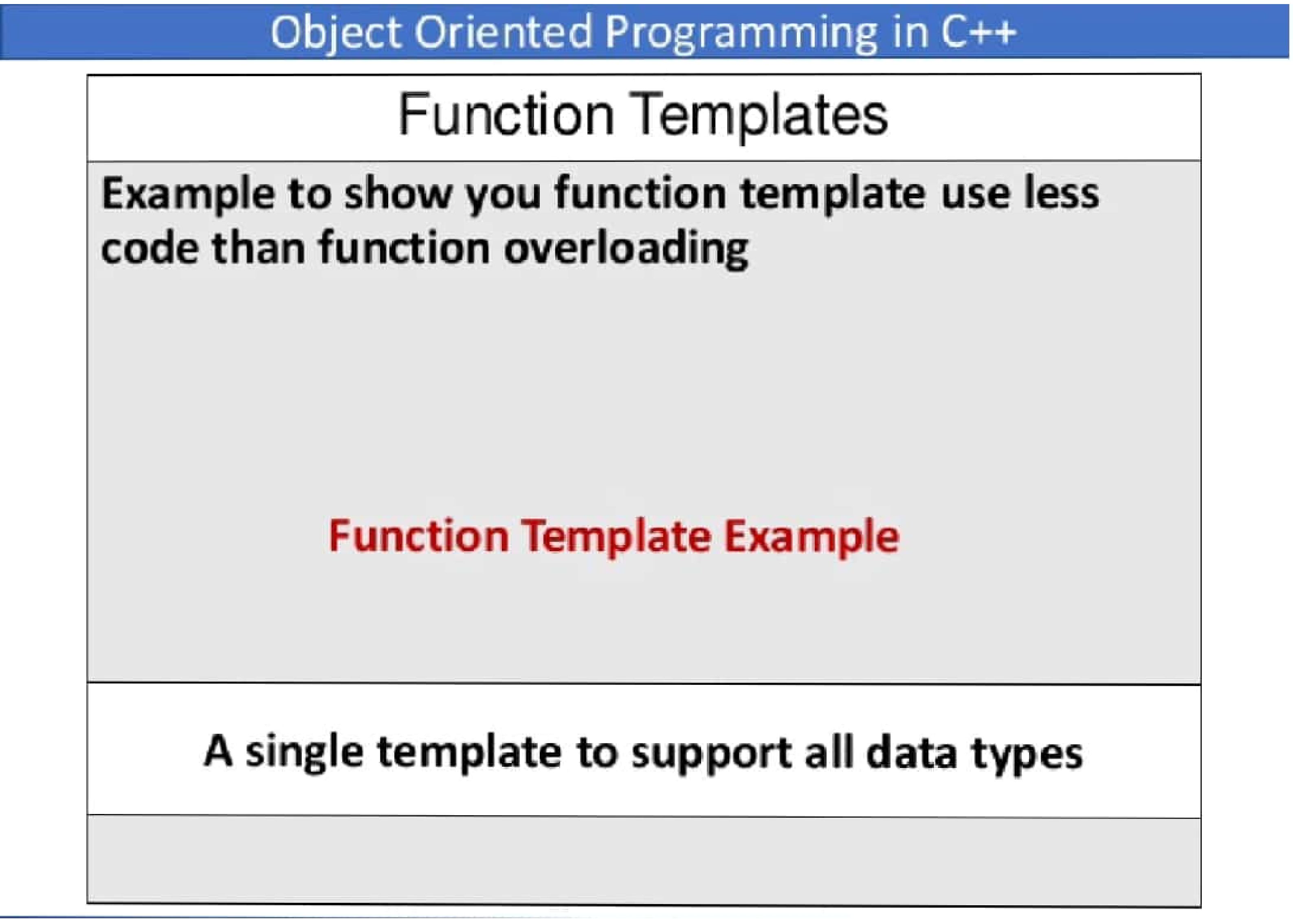
Function Overloading Example-2

```
int main()
{
  int   i, ii;
  double d, dd;
  i = 2;
  d = 2.2;
  ii = square(i);
  cout << "Square of Integer Number "<< " : " << ii << endl;
  dd = square(d);
  cout<< "Square of double number "<< " : " << dd << endl;
return 0;
}
```



```
Square of Integer Number  : 4
Square of double number  : 4.84

--------------------------------
Process exited after 0.02027 seconds with return value 0
Press any key to continue . . .
```



Object Oriented Programming in C++

Function Templates

Example to show you function template use less code than function overloading

Function Template Example

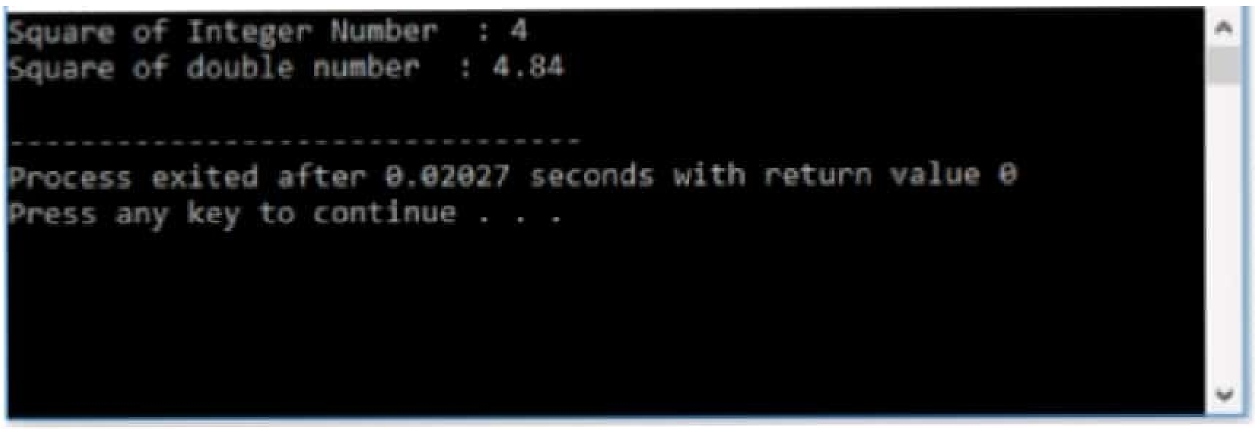
A single template to support all data types

Function Template Example-1

```
#include <iostream>
using namespace std;
template <typename T>
T square(T x)
{
  T result;
  result = x * x;
  return result;
}
```

Function Template Example-2

```
int main()
{
  int   i, ii;
  double d, dd;
  i = 2;
  d = 2.2;
  ii = square(i);
  cout << "Square of Integer Number "<< ": " << ii << endl;
  dd = square(d);
  cout<< "Square of double number "<< " : " << dd << endl;
return 0;
}
```



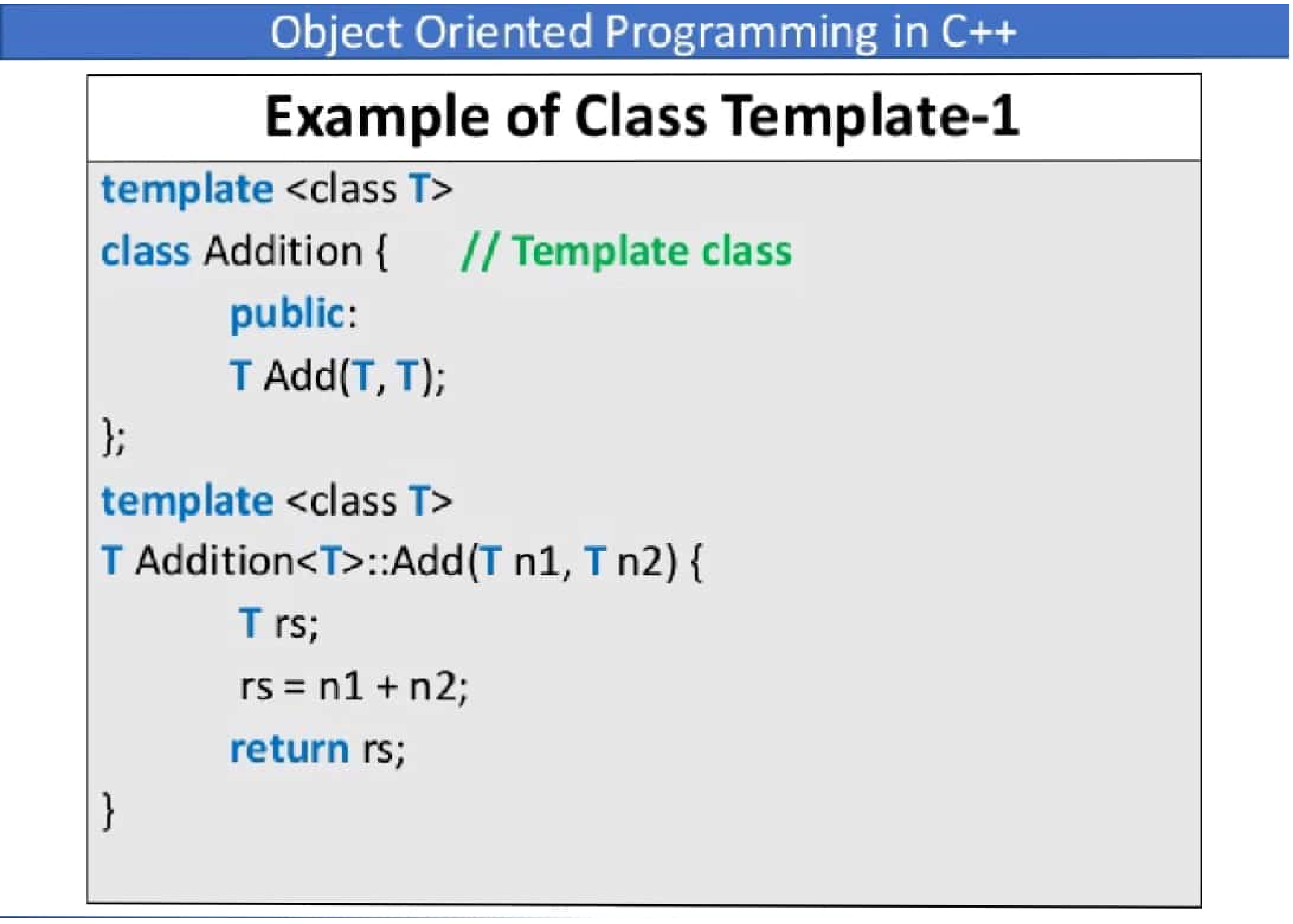
```
Square of Integer Number  : 4
Square of double number  : 4.84

--------------------------------
Process exited after 0.02027 seconds with return value 0
Press any key to continue . . .
```

Class Template

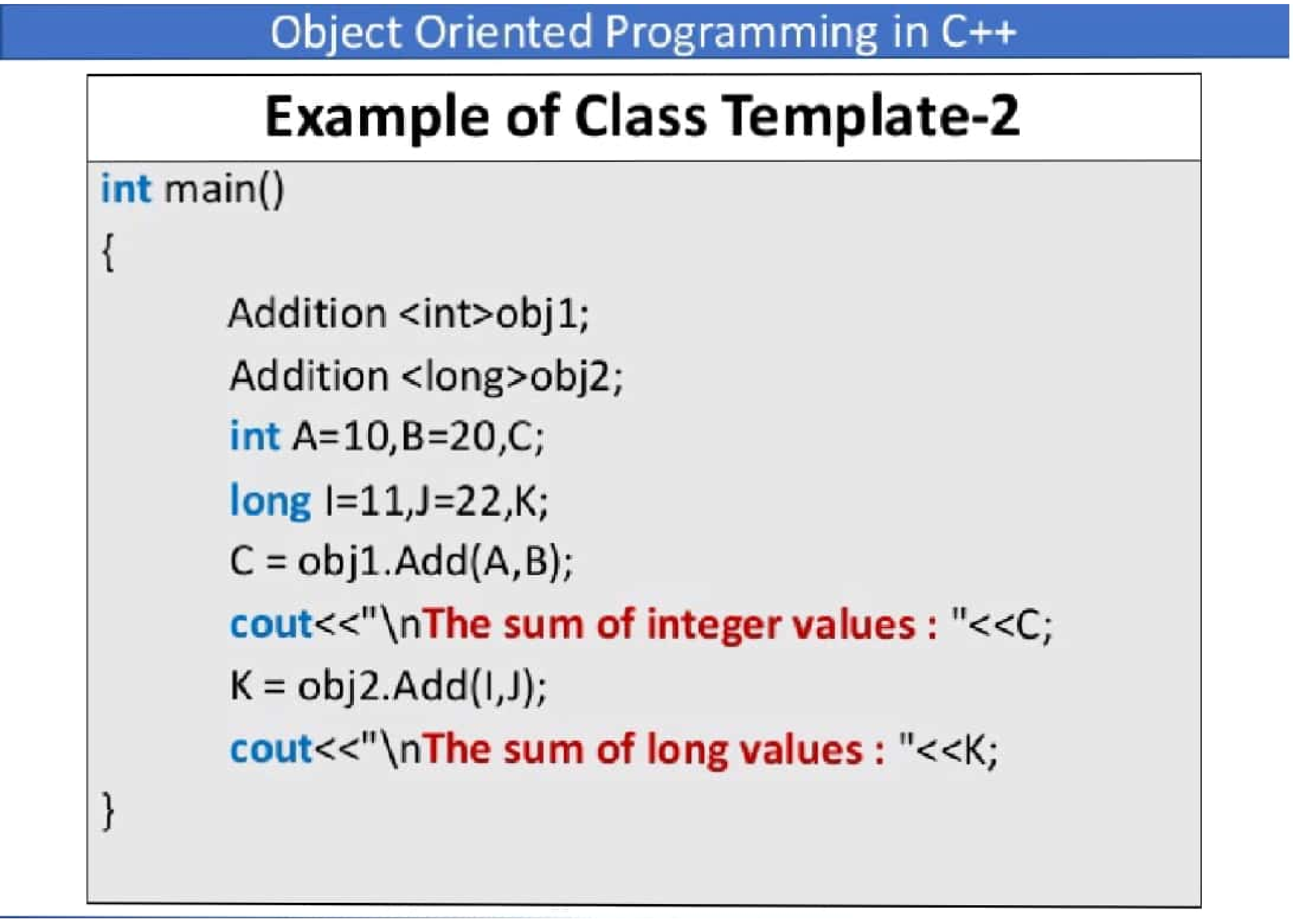
- Like function template, a class template is a common class that can represent various similar classes operating on data of different types. Once a class template is defined, we can create an object of that class using a specific basic or user-defined data types to replace the generic data types used during class definition.
- **Syntax of Class Template**

```
template <class T1, class T2, ...>
class classname
{
   attributes;
   methods;
};
```



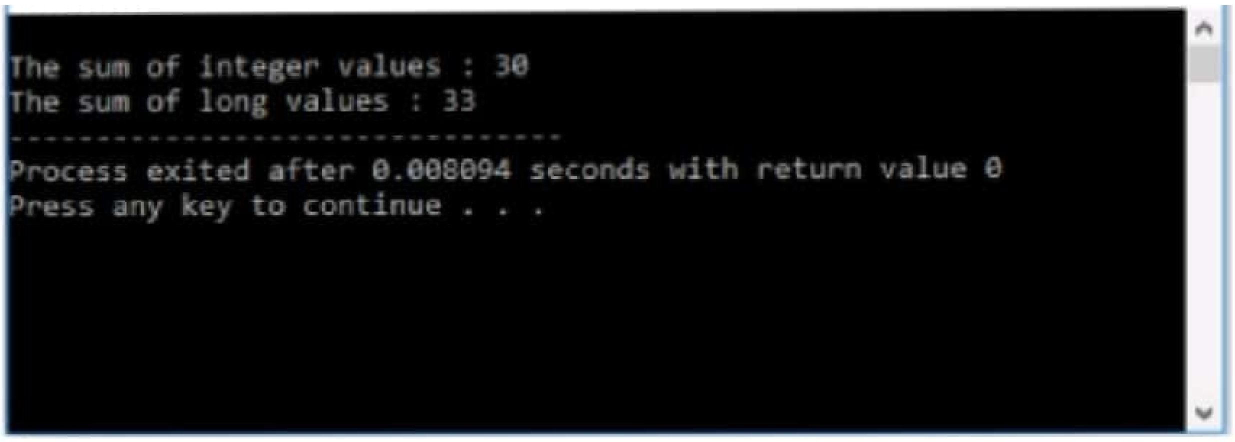
Example of Class Template-1

```
template <class T>
class Addition {      // Template class
        public:
        T Add(T, T);
};
template <class T>
T Addition<T>::Add(T n1, T n2) {
         T rs;
         rs = n1 + n2;
        return rs;
}
```

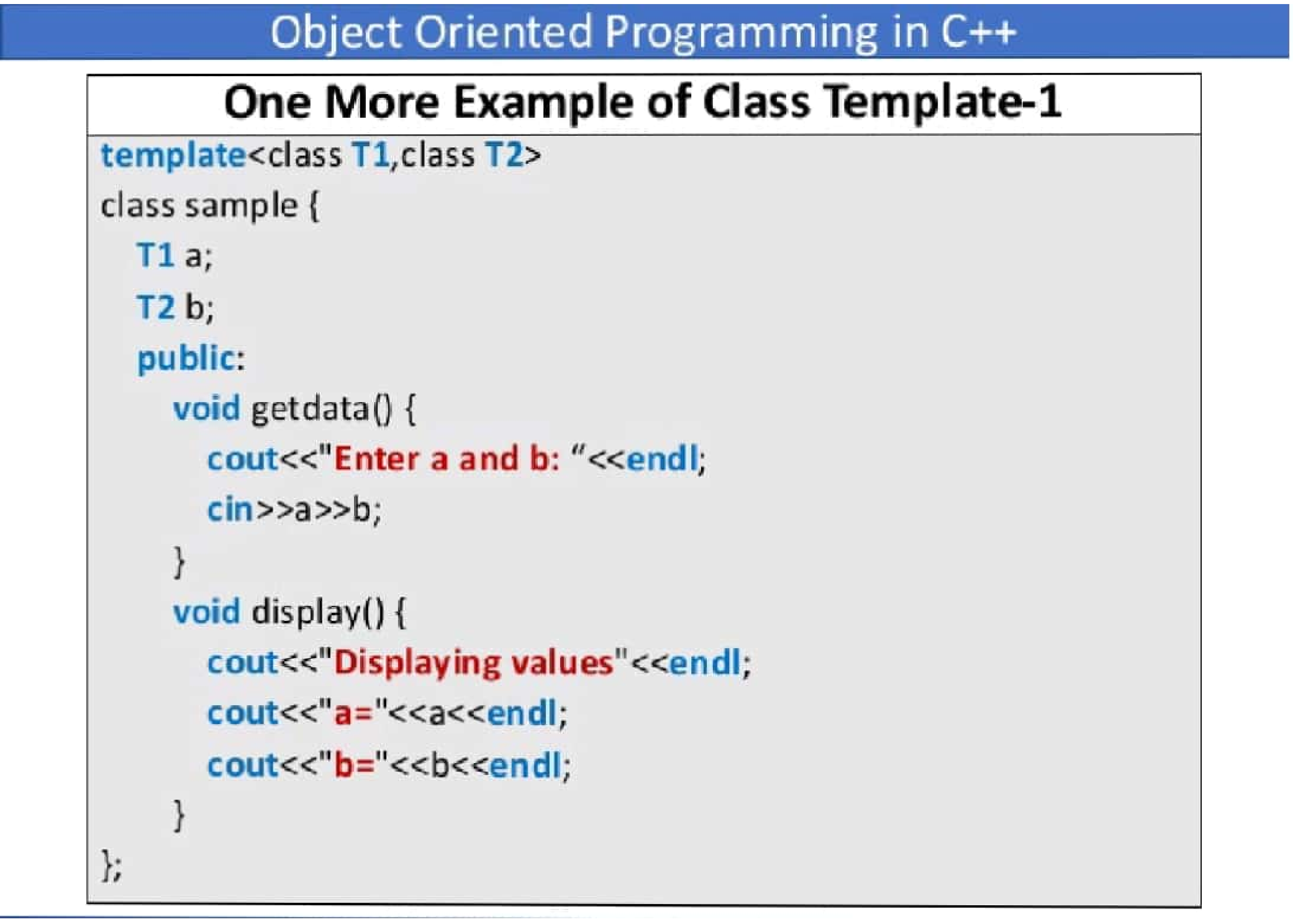


Example of Class Template-2

```
int main()
{
        Addition <int>obj1;
        Addition <long>obj2;
        int A=10,B=20,C;
        long I=11,J=22,K;
        C = obj1.Add(A,B);
        cout<<"\nThe sum of integer values : "<<C;
        K = obj2.Add(I,J);
        cout<<"\nThe sum of long values : "<<K;
}
```



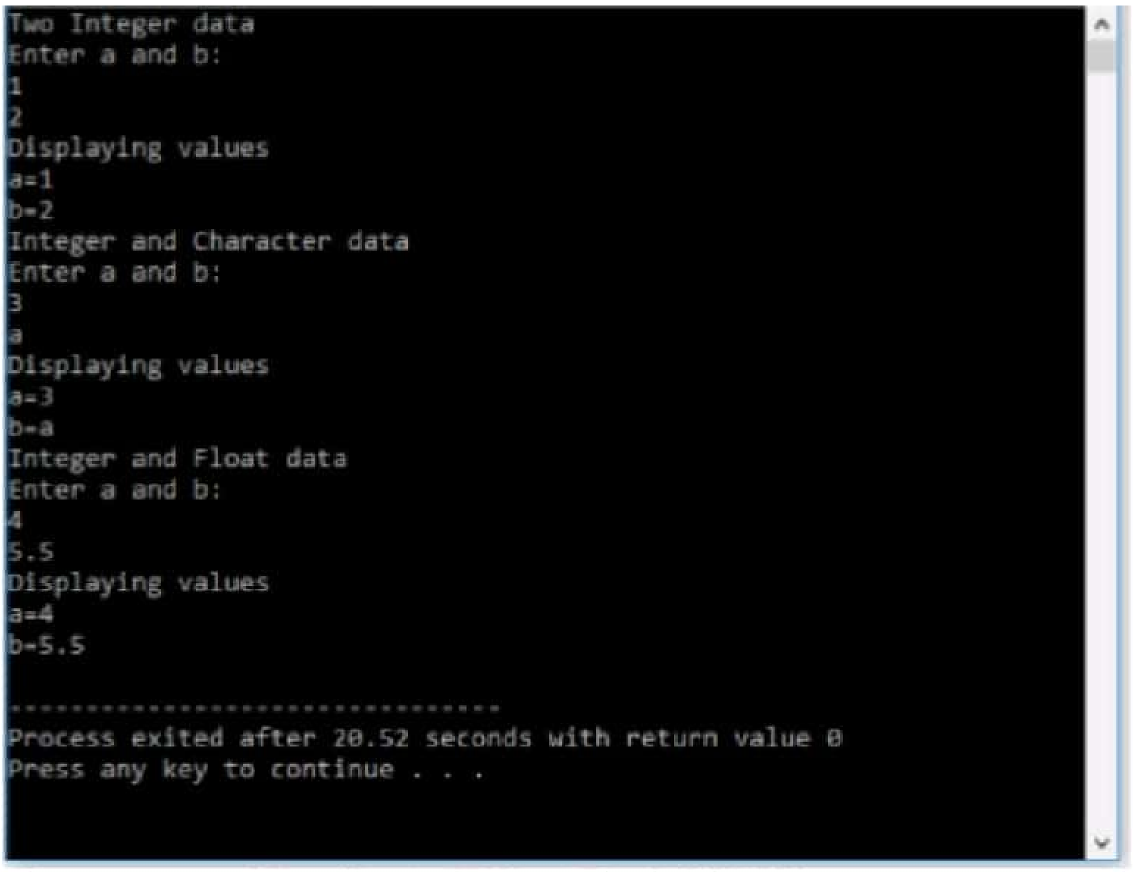
```
The sum of integer values : 30
The sum of long values : 33
--------------------------------
Process exited after 0.008094 seconds with return value 0
Press any key to continue . . .
```



Object Oriented Programming in C++

One More Example of Class Template-1

```
template<class T1,class T2>
class sample {
   T1 a;
   T2 b;
   public:
      void getdata() {
         cout<<"Enter a and b: “<<endl;
         cin>>a>>b;
      }
      void display() {
         cout<<"Displaying values"<<endl;
         cout<<"a="<<a<<endl;
         cout<<"b="<<b<<endl;
      }
};
```



```
Two Integer data
Enter a and b:
1
2
Displaying values
a=1
b=2
Integer and Character data
Enter a and b:
3
a
Displaying values
a=3
b=a
Integer and Float data
Enter a and b:
4
5.5
Displaying values
a=4
b=5.5

--------------------------------
Process exited after 20.52 seconds with return value 0
Press any key to continue . . .
```

Explanation of Previous Program

- In this program, a template class *sample* is created. It has two data a and b of generic types and two methods: *getdata()* to give input and *display()* to display data. Three object s1, s2 and s3 of this class is created. s1 operates on both integer data, s2 operates on one integer and another character data and s3 operates on one integer and another float data. Since, *sample* is a template class, it supports various data types.
- A class created form a class template is called a template class. The syntax foe defining an object of a template class is:

```
Classname <type> objectname(argument_list)
```