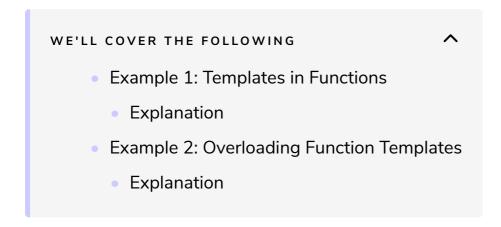
## - Examples

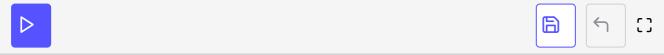
Let's take a look at the examples of function templates in this lesson.



## Example 1: Templates in Functions #

```
// templateFunctionsTemplates.cpp
#include <iostream>
#include <string>
#include <vector>
template <typename T>
void xchg(T& x, T& y){
 T t = x;
 x = y;
  y =t;
template <int N>
int nTimes(int n){
  return N * n;
int main(){
  std::cout << std::endl;</pre>
  bool t = true;
  bool f = false;
  std::cout << "(t, f): (" << t << ", " << f << ") "<< std::endl;
  xchg(t, f);
  std::cout << "(t, f): (" << t << ", " << f << ") "<< std::endl;
  std::cout << std::endl;</pre>
  int int2011 = 2011;
```

```
int int2014 = 2014;
std::cout << "(int2011, int2014): (" << int2011 << ", " << int2014 << ") "<< std::endl;
xchg(int2011, int2014);
std::cout << "(int2011, int2014): (" << int2011 << ", " << int2014 << ") "<< std::endl;
std::cout << std::endl;</pre>
std::string first{"first"};
std::string second{"second"};
std::cout << "(first, second): (" << first << ", " << second << ") "<< std::endl;</pre>
xchg(first, second);
std::cout << "(first, second): (" << first << ", " << second << ") "<< std::endl;</pre>
std::cout << std::endl;</pre>
std::vector<int> intVec1{1, 2, 3, 4, 5};
std::vector<int> intVec2{5, 4, 3, 2, 1};
std::cout << "vec1: ";
for (auto v: intVec1)std::cout << v << " ";</pre>
std::cout << "\nvec2: ";</pre>
for (auto v: intVec2)std::cout << v << " ";</pre>
std::cout << std::endl;</pre>
xchg(intVec1, intVec2);
std::cout << "vec1: ";</pre>
for (auto v: intVec1)std::cout << v << " ";</pre>
std::cout << "\nvec2: ";</pre>
for (auto v: intVec2)std::cout << v << " ";</pre>
std::cout << std::endl;</pre>
std::cout << "\n\n";</pre>
std::cout << "nTimes<5>(10): " << nTimes<5>(10) << std::endl;
std::cout << "nTimes<10>(5): " << nTimes<10>(5) << std::endl;</pre>
std::cout << std::endl;</pre>
```



## Explanation #

In the example above, we've declared two function templates: xchg and nTimes in lines 8 and 15. xchg swaps the values passed as arguments. The only non-type, we use is N in the function templates nTimes. nTimes returns the N times of the number passed n. We have initialized multiple instances to check for functions in lines 31 and 32, lines 39 and 40, and lines 46 and 47.

## Example 2: Overloading Function Templates #

```
#include <iostream>
void xchg(int& x, int& y){ // 1
 int t = x;
 x = y;
 y = t;
}
template <typename T>
                            // 2
void xchg(T& x, T& y){
 T t = x;
 x = y;
 y = t;
template <typename T>
void xchg(T& x, T& y, T& z){
 xchg(x, y);
 xchg(x, z);
int main(){
  std::cout << std::endl;</pre>
 int intA = 5;
  int intB = 10;
  int intC = 20;
  double doubleA = 5.5;
  double doubleB = 10.0;
  std::cout << "Before: " << intA << ", " << intB << std::endl;</pre>
  xchg(intA, intB); // 1
  std::cout << "After: " << intA << ", " << intB << std::endl;</pre>
  std::cout << std::endl;</pre>
  std::cout << "Before: " << doubleA << ", " << doubleB << std::endl;</pre>
  xchg(doubleA, doubleB);
                              // 2
  std::cout << "After: " << doubleA << ", " << doubleB << std::endl;</pre>
  std::cout << std::endl;</pre>
 xchg<>(intA, intB);  // explicit 2
xchg<int>(intA, intB);  // explicit 2: xchg<int>
                                    // ERROR explicit xchg<double>
  // xchg<double>(intA, intB);
  std::cout << "Before: " << intA << ", " << intB << ", " << intC << std::endl;</pre>
  xchg(intA, intB, intC);
  std::cout << "After: " << intA << ", " << intB << ", " << intC << std::endl;
  std::cout << std::endl;</pre>
```





In the above example, we used the concept of function overloading by calling xchg with different arguments passed to the function. We used the xchg function with different data types by passing two arguments and three arguments. In line 37, the non-template function is called, whereas, on all other calls to xchg(), the template function is used. The call xchg<double, double>(intA, intB) would be fine, when xchg would take its arguments by value.

In the next lesson, we'll solve an exercise related to function templates.