- Examples

In this lesson, we will look at a few examples of class templates.

WE'LL COVER THE FOLLOWING ^ Example 1 Explanation Example 2 Explanation Example 3 Explanation

Example 1#

```
// templateClassTemplate.cpp
                                                                                         6
#include <iostream>
class Account{
public:
  explicit Account(double amount=0.0): balance(amount){}
  void deposit(double amount){
    balance+= amount;
  void withdraw(double amount){
    balance-= amount;
  double getBalance() const{
    return balance;
private:
  double balance;
};
template <typename T, int N>
class Array{
```

```
public:
  Array()= default;
  int getSize() const;
private:
 T elem[N];
};
template <typename T, int N>
int Array<T,N>::getSize() const {
  return N;
int main(){
  std::cout << std::endl;</pre>
  Array<double,10> doubleArray;
  std::cout << "doubleArray.getSize(): " << doubleArray.getSize() << std::endl;</pre>
  Array<Account,1000> accountArray;
  std::cout << "accountArray.getSize(): " << accountArray.getSize() << std::endl;</pre>
  std::cout << std::endl;</pre>
```







[]

Explanation

- We created two Array class objects, i.e., doubleArray and accountArray in lines 45 and 48.
- By calling the generic function getSize() in line 37, we can access the size of different objects.

Example 2

```
// templateClassTemplateMethods.cpp
#include <algorithm>
#include <iostream>
#include <vector>

template <typename T, int N>
class Array{

public:
    Array()= default;

template <typename T2>
    Array<T, N>& operator=(const Array<T2, N>& arr){
    elem clear():
```

```
elem.insert(elem.begin(), arr.elem.begin(), arr.elem.end());
         return *this;
 int getSize() const;
  std::vector<T> elem;
};
template <typename T, int N>
int Array<T, N>::getSize() const {
 return N;
int main(){
 Array<double, 10> doubleArray{};
 Array<int, 10> intArray{};
 doubleArray= intArray;
 Array<std::string, 10> strArray{};
 Array<int, 100> bigIntArray{};
 // doubleArray= strArray;
                                     // ERROR: cannot convert 'const std::basic_string<char</pre>
 // doubleArray= bigIntArray; // ERROR: no match for 'operator=' in 'doubleArray =
```

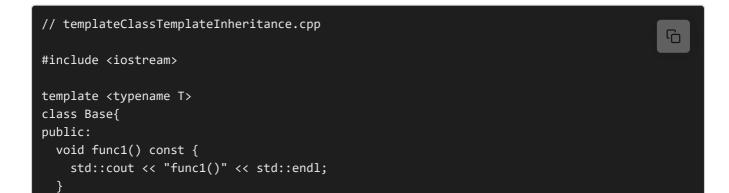




Explanation

- In the example above, we initialized two instances of Array class namely, doubleArray and intArray in lines 32 and 33.
- We use the generic = operator to copy the intArray elements to doubleArray in line 35. When you look carefully, you see that the generic is only applicable when both arrays have the same length.

Example 3



```
void func2() const {
    std::cout << "func2()" << std::endl;</pre>
  void func3() const {
    std::cout << "func3()" << std::endl;</pre>
};
template <typename T>
class Derived: public Base<T>{
public:
  using Base<T>::func2;
  void callAllBaseFunctions(){
    this->func1();
    func2();
    Base<T>::func3();
};
int main(){
  std::cout << std::endl;</pre>
  Derived<int> derived;
  derived.callAllBaseFunctions();
  std::cout << std::endl;</pre>
```







Explanation

We have implemented a Base and Derived classes. Derived is publicly inherited from Base and may use callAllBaseFunctions in line 24, the methods func1, func2, and func3 from the Base class.

- Make the name dependent: The call this->func1 in line 25 is dependent. The name lookup will consider in this case, all base classes.
- Introduce the name into the current scope: The expression using Base<T>:::func2 (line 22) introduces func2 into the current scope.
- Call the name fully qualified: Calling func3 fully qualified (line 27) will break a virtual dispatch and may cause new surprises.

We created a Derived class object named derived. By using this object, we can access the base class functions by calling method callAllBaseFunctions.

In the next lesson, we will discuss the friends of class templates.