- Examples

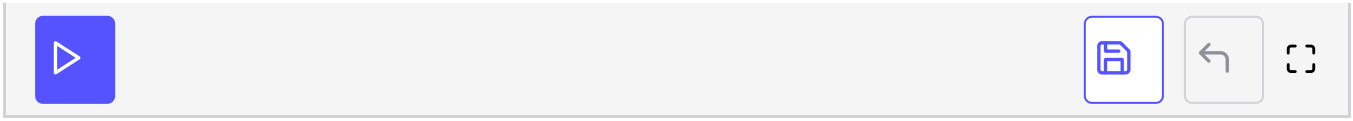
In this lesson, we will look at some examples for template specialization.

WE'LL COVER THE FOLLOWING

Example 1

```
// TemplateSpecialization.cpp

#include <iostream>

class Account{
public:
  explicit Account(double b): balance(b){}
  double getBalance() const {
    return balance;
  }
private:
  double balance;
};

template <typename T, int Line, int Column>
class Matrix{
  std::string getName() const { return "Primary Template"; }
};

template <typename T>
class Matrix<T,3,3>{
  std::string name{"Partial Specialization"};
};

template <>
class Matrix<int,3,3>{};
```

```
template<typename T>
bool isSmaller(T fir, T sec){
  return fir < sec;
}

template <>
bool isSmaller<Account>(Account fir, Account sec){
  return fir.getBalance() < sec.getBalance();
}

int main(){

  std::cout << std::boolalpha << std::endl;

  Matrix<double,3,4> primaryM;
  Matrix<double,3,3> partialM;
  Matrix<int,3,3> fullM;

  std::cout << "isSmaller(3,4): "  << isSmaller(3,4) << std::endl;
  std::cout << "isSmaller(Account(100.0),Account(200.0)): "<< isSmaller(Account(100.0),Accour

  std::cout << std::endl;

}
```



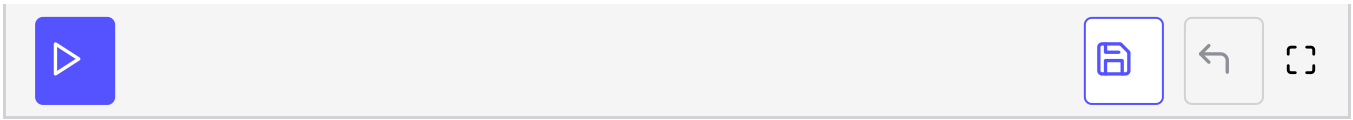
Explanation

In the above code, we are modifying the example that we have used in template arguments example.

- **Primary** template is called when we use values other than `Matrix<data_type, 3, 3>` (line 43).
- **Partial** specialization is called when we instantiate `Matrix<data_type, 3, 3>` where `data_type` is not `int` (line 44).
- **Full** specialization is called when we explicitly use `int` as a data type: `Matrix<int, 3, 3>` (line 45)
- **Full** specialization of the function template `isSmaller` is only applicable for `Account`s. This allows the functions to compare two `Account`s based on their balance (line 48).

Example 2

```
// templateSpecializationFull.cpp

#include <iostream>
#include <string>

template <typename T>
T min(T fir, T sec){
  return (fir < sec) ? fir : sec;
}

template <>
bool min<bool>(bool fir, bool sec){
  return fir & sec;
}

int main(){

  std::cout << std::boolalpha << std::endl;

  std::cout << "min(3.5, 4.5): "  << min(3.5, 4.5) << std::endl;
  std::cout << "min<double>(3.5, 4.5): "  << min<double>(3.5, 4.5) << std::endl;

  std::cout << "min(true, false): "  << min(true, false) << std::endl;
  std::cout << "min<bool>(true, false): "  << min<bool>(true, false) << std::endl;

  std::cout << std::endl;

}
```



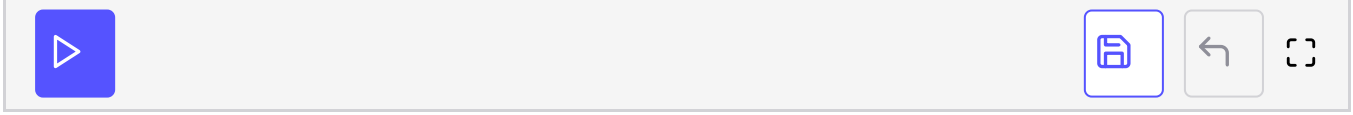
Explanation

- In the above example, we defined a full specialization for `bool`. The primary and the full specialization are implicitly invoked in the lines (20 and 23) and explicitly invoked in the lines (21 and 24).

Example 3

```
//TemplateSpecializationExternal.cpp

#include <iostream>


template <typename T=std::string, int Line=10, int Column=Line>
class Matrix{
public:
  int numberOfElements() const;
};

template <typename T, int Line, int Column>
int Matrix<T,Line,Column>::numberOfElements() const {
  return Line * Column;
}
```

```
template <typename T>
class Matrix<T,3,3>{
public:
  int numberOfElements() const;
};

template <typename T>
int Matrix<T,3,3>::numberOfElements() const {
  return 3*3;
}


template <>
class Matrix<int,3,3>{
public:
  int numberOfElements() const;
};

int Matrix<int,3,3>::numberOfElements() const {
  return 3*3;
}

int main(){

  std::cout << std::endl;

  Matrix<double,10,5> matBigDouble;
  std::cout << "matBigDouble.numberOfElements(): " << matBigDouble.numberOfElements() << std:

  // Matrix matString;    // ERROR
  Matrix<> matString;
  std::cout << "matString.numberOfElements(): " << matString.numberOfElements() << std::endl;

  Matrix<float> matFloat;
  std::cout << "matFloat.numberOfElements(): " << matFloat.numberOfElements() << std::endl;

  Matrix<bool,20> matBool;
  std::cout << "matBool.numberOfElements(): " << matBool.numberOfElements() << std::endl;

  Matrix <double,3,3> matSmallDouble;
  std::cout << "matSmallDouble.numberOfElements(): " << matSmallDouble.numberOfElements() <<

  Matrix <int,3,3> matInt;
  std::cout << "matInt.numberOfElements(): " << matInt.numberOfElements() << std::endl;

  std::cout << std::endl;

}
```



Explanation

- In the above example, we have set the value of `line` to 10 by default (line 6. We use the value of `line` as a default for `column`.

- The method `numberOfElements()` returns the product of both numbers as a result. If we call the `Matrix` with arguments, these passed arguments override the default.
- For `float` and `string`, it returns the 100 since no arguments are passed, and the default arguments are used (line 51).

Let's move on to **C**uriously **R**ecurring **T**emplate **P**atterns (CRTP) in C++ templates.