## 1. Шаблон функции для нахождения максимума

**Задача:** Напишите шаблон функции maxValue, который принимает два аргумента любого типа (с поддержкой оператора >) и возвращает наибольший из них.
**Дополнительно:** Проверьте работу функции для int, double и std::string.

```
#include <iostream>
#include <string>

template <typename T>
T maxValue(T a, T b) {
    return (a > b) ? a : b;
}

int main() {
    std::cout << maxValue(5, 10) << std::endl;          // 10
    std::cout << maxValue(3.14, 2.71) << std::endl;     // 3.14
    std::cout << maxValue("apple", "orange") << std::endl; // "orange" (лексикографическое сравнение)
    return 0;
}
```

---

## 2. Проверка типов в шаблоне

**Задача:** Напишите шаблон функции isEqual, который сравнивает два значения. Если типы разные, выведите ошибку на этапе компиляции (используйте static_assert).

```
#include <iostream>
#include <type_traits>

template <typename T, typename U>
bool isEqual(T a, U b) {
    static_assert(std::is_same<T, U>::value, "Error: Types must be the same!");
    return a == b;
}

int main() {
    std::cout << isEqual(5, 5) << std::endl;    // OK
    // std::cout << isEqual(5, 5.0) << std::endl; // Ошибка компиляции
    return 0;
}
```

---

## 3. Специализация шаблона для строк

**Задача:** Создайте шаблон функции printType, который выводит тип переданного значения. Затем сделайте явную специализацию для const char*, чтобы выводилось "C-style string".

```
#include <iostream>
#include <typeinfo>

template <typename T>
void printType(T value) {
    std::cout << "Type: " << typeid(T).name() << std::endl;
```

```
}

template <>
void printType<const char*>(const char* value) {
    std::cout << "Type: C-style string" << std::endl;
}

int main() {
    printType(42);              // Type: int (или i)
    printType(3.14);            // Type: double (или d)
    printType("Hello, world!"); // Type: C-style string
    return 0;
}
```

---

## 4. Шаблон функции с разными типами

**Задача:** Напишите шаблон функции multiply, который принимает два аргумента разных типов и возвращает их произведение. Проверьте с int и double.

```
#include <iostream>

template <typename T, typename U>
auto multiply(T a, U b) -> decltype(a * b) {
    return a * b;
}

int main() {
    std::cout << multiply(3, 4.5) << std::endl;   // 13.5 (double)
    std::cout << multiply(2.5, 4) << std::endl;   // 10.0 (double)
    return 0;
}
```

---

## 5. Явная конкретизация шаблона

**Задача:** Создайте шаблон функции square, который возвращает квадрат числа. Сделайте явную конкретизацию для int, чтобы она выводила дополнительное сообщение.

```
#include <iostream>

template <typename T>
T square(T x) {
    return x * x;
}

template <>
int square<int>(int x) {
    std::cout << "Squaring an int: ";
    return x * x;
}

int main() {
    std::cout << square(5.5) << std::endl;  // 30.25
    std::cout << square(5) << std::endl;    // "Squaring an int: 25"
    return 0;
```

```
}
```