4.9. Шаблонные функции, синтаксис их объявления. Перегрузка и специализация шаблонных функций, разница между этими понятиями (на примере). Эвристические правила разрешения перегрузки: «частное лучше общего», «точное соответствие лучше приведения типа». Примеры разрешения перегрузки между шаблонными функциями.

1. **Синтаксис объявления шаблонных функций:**

```
template <typename T>
T max(const T& a, const T& b) {
    return a > b ? a : b;
}
```

2. **Перегрузка:**

```
template <typename T, typename U>
void f(T, U) { std::cout << 1; }

template <typename T>
void f(T, T) { std::cout << 2; }

void f(int, double) { std::cout << 3; }
```

   Рассмотрим несколько вызовов f и посмотрим что выведется

   - `f(0, 0)` - выведется 2, так как он «более частная» чем 1, то есть подходит в меньшем количестве случаев + в ней не требуется приведение типов как в 3
   - `f(0, 0.0)` - выведется 3, так как она более частная чем 1, а 2 просто не подходит, так как непонятно, чему равно T
   - `f(0.0, 0)` - выведется 1, так как в 3 нужно сделать 2 приведения типа, а это хуже чем точное совпадение.

3. **Специализация:** по стандарту в функциях запрещена частичная специализация, так как этот механизм уже реализован в виде перегрузки.

```
template <typename T, typename U>
void f(T, U) { std::cout << 1; }

template <typename T>
void f(T, T) { std::cout << 2; }

template <>
void f(int, int) { std::cout << 3; }
```

   Важно запомнить, что «хозяйкой» специализации является функция, объявленная над ней. Рассмотрим вызов `f(0, 0)`. Очевидно, что в перегрузке выберется вторая версия, а затем выберется специализация и в итоге выведется 3.

   Поменяем в коде функции 2 и 3 местами. Теперь хозяйкой специализации стала первая функция. В перегрузке все так же побеждает вторая, но специализации у нее больше нет, поэтому выведется 2.

   Итог: Специализация не участвует в перегрузке, рассматривается только если ее хозяйка победила

4.10. Специализация шаблонов. Синтаксис объявления специализации для функций и классов. Разница между перегрузкой и специализацией шаблонных функций на примере. Частичная и полная специализация шаблонов классов. Примеры: реализация hash для нестандартного типа, реализация is_same, реализация remove_reference.

1. **Специализации классов:**

```cpp
template <typename T>
struct vector {};

template <typename T> // частичная специализация
struct vector<T*> {}; // можно T&, const T, T[], etc.

template <> // полная явная специализация
struct vector<bool> {};
```

2. **Реализация hash:** (нигде не было лишь мои догадки. после консультации уточню)

```cpp
struct A { std::string field; }
template <typename T>
struct hash {};
template <>
struct hash<A> {
    size_t operator () (const A& a) {
        return std::hash<std::string>(a.field); // как-то хэшируем a
    }
};
```

3. **Реализация is_same:**

```cpp
template <typename U, typename V>
struct is_same {
    static const bool value = false;
};

template <typename U>
struct is_same<U, U> {
    static const bool value = true;
};
```

4. **Реализация remove_reference:**

```cpp
template <typename T>
struct remove_reference {
    using type = T;
};

template <typename T>
struct remove_reference<T&> {
    using type = T;
};

template <typename T>
struct remove_reference<T&&> {
    using type = T;
};
```

4.11. Простейшие compile-time вычисления с помощью шаблонной рекурсии. Вычисление чисел Фибоначчи в compile-time с помощью шаблонной рекурсии. Проверка на простоту числа N за O(N) в compile-time с помощью шаблонной рекурсии.

Вычисление N-ого числа Фибоначчи

```
template <size_t N>
struct Fibonacci {
  static const size_t value = Fibonacci<N-1>::value + Fibonacci<N-2>::value;
};

template <>                                  template <>
struct Fibonacci<1> {                        struct Fibonacci<0> {
    static const size_t value = 1;               static const size_t value = 0;
};                                           };
```

Проверка числа N на простоту за $O(N)$

```
template <size_t N, size_t D>
struct IsPrimeHelper { // проверка что N не делится на все числа ≤ D
    static const bool value = (N % D == 0 ? false
                                          : IsPrimeHelper<N, D-1>::value);
};

template <size_t N>
struct IsPrimeHelper<N, 1> { // база
    static const bool value = true;
};

template <size_t N>
struct IsPrime {
    static const bool value = IsPrimeHelper<N, N-1>::value;
};

template <>
struct IsPrime<1> { // отдельно случай для 1
    static const bool value = false;
};
```

Замечание: Максимальная глубина шаблонной рекурси по дефолту равна 1024. Ее можно увеличить с помощью флага `-ftemplate-depth=`новая глубина (но если сильно увеличить могут вылезти другие страшные ошибки)