Для закрепления материала по правилам выведения типов auto, шаблонов и decltype ответьте на следующие вопросы.

Вопрос 1. Какой тип будет у аргументов param в каждом из следующих вариантов вызова шаблонных функций?

Будьте внимательны – некоторые вызовы не скомпилируются. Каждый вызов пронумерован для удобства обсуждения с коллегами.

```
template <typename T>
void func1(T param) {
      // some very interesting code
}

template <typename T>
void func2(T& param) {
    // some very interesting code
}

template <typename T>
void func3(T* param) {
    // some very interesting code
}

template <typename T>
void func4(T&& param) {
    // some very interesting code
}

int main(int argc, char * argv[]) {

      int i = 10;
      const int ci = 15;
      const int &cir = ci;
      int * pi = &i;
      const int * cpi = &i;

      func1(i); // 1.1. param type ?
      func1(ci); // 1.2. param type ?
      func1(cir); // 1.3. param type ?
      func1(42); // 1.4. param type ?
      func1(pi); // 1.5. param type ?
      func1(cpi); // 1.6. param type ?
      func1({1}); // 1.7. param type ?
      func1({1, 2, 3}); // 1.8. param type ?

      func2(i); // 2.1. param type ?
      func2(ci); // 2.2. param type ?
      func2(cir); // 2.3. param type ?
      func2(42); // 2.4. param type ?
      func2(pi); // 2.5. param type ?
      func2(cpi); // 2.6. param type ?

      func3(i); // 3.1. param type ?
      func3(ci); // 3.2. param type ?
      func3(cir); // 3.3. param type ?
      func3(42); // 3.4. param type ?
      func3(pi); // 3.5. param type ?
      func3(cpi); // 3.6. param type ?
```

```
      func4(i); // 4.1. param type ?
      func4(ci); // 4.2. param type ?
      func4(cir); // 4.3. param type ?
      func4(42); // 4.4. param type ?
      func4(pi); // 4.5. param type ?
      func4(cpi); // 4.6. param type ?

    return 0;
}
```

Вопрос 2. Какой тип выведет auto в каждом из приведенных ниже случаев?

```
#include <utility>

int func_value() {
    return 42;
}

int& func_ref() {
    static int value = 42;
    return value;
}

const int& func_cref() {
    static const int value = 42;
    return value;
}

int* func_ptr() {
    static int value = 42;
    return &value;
}

const int * func_cptr() {
    static int value = 42;
    return &value;
}

const int * const func_ccptr() {
    static const int value = 42;
    return &value;
}

int&& func_rref() {
    int value = 42;
    return std::move(value);
}

int main(int argc, char * argv[]) {

    {
        auto v1 = func_value();     // 1.1 - v1 type ?
        auto& v2 = func_value();    // 1.2 - v2 type ?
        const auto& v3 = func_value();  // 1.3 - v3 type ?
        auto&& v4 = func_value();   // 1.4 - v4 type ?
    }
    {
        auto v1 = func_ref();     // 2.1 - v1 type ?
        auto& v2 = func_ref();    // 2.2 - v2 type ?
```

```
        const auto& v3 = func_ref();  // 2.3 - v3 type ?
        auto&& v4 = func_ref();   // 2.4 - v4 type ?
    }
    {
        auto v1 = func_cref();     // 3.1 - v1 type ?
        auto& v2 = func_cref();    // 3.2 - v2 type ?
        const auto& v3 = func_cref();  // 3.3 - v3 type ?
        auto&& v4 = func_cref();   // 3.4 - v4 type ?
    }
    {
        auto v1 = func_ptr();     // 4.1 - v1 type ?
        auto& v2 = func_ptr();    // 4.2 - v2 type ?
        const auto& v3 = func_ptr();  // 4.3 - v3 type ?
        auto&& v4 = func_ptr();   // 4.4 - v4 type ?
    }
    {
        auto v1 = func_cptr();     // 5.1 - v1 type ?
        auto& v2 = func_cptr();    // 5.2 - v2 type ?
        const auto& v3 = func_cptr();  // 5.3 - v3 type ?
        auto&& v4 = func_cptr();   // 5.4 - v4 type ?
    }
    {
        auto v1 = func_ccptr();     // 6.1 - v1 type ?
        auto& v2 = func_ccptr();    // 6.2 - v2 type ?
        const auto& v3 = func_ccptr();  // 6.3 - v3 type ?
        auto&& v4 = func_ccptr();   // 6.4 - v4 type ?
    }
    {
        auto v1 = func_rref();     // 7.1 - v1 type ?
        auto& v2 = func_rref();    // 7.2 - v2 type ?
        const auto& v3 = func_rref();  // 7.3 - v3 type ?
        auto&& v4 = func_rref();   // 7.4 - v4 type ?
    }
    {
        auto v1(10);    // 8.1 - v1 type ?
        auto v2 = 10;   // 8.2 - v2 type ?
        auto v3{10};    // 8.3 - v3 type ?
        auto v4 = {10};     // 8.4 - v4 type ?
        auto v5 = {1, 2, 3};    // 8.5 - v5 type ?
        auto v6 = {1, 2, 3.0};  // 8.6 - v6 type ?
    }
    return 0;
}
```