

## Ответы на контрольные вопросы:

### 1 вопрос: О каких принципах следует помнить при разработке функций?

→ Если какой-то блок действий в программе выполняется некоторое число раз, большее единицы, и его логика не является специфичной и особенной только для данной программы, то имеет смысл загнать этот блок в функцию, предварительно описав её, а после в каждой части программы, где это будет необходимо, просто её вызывать.

→ Второй посыл, вытекающий из первого: чтобы логика функции на всякий случай не оказалась специфичной, желательно подбирать "говорящие" названия как для самой функции, так и для переменных, которые в ней используются. Опять же, это сделает нашу функцию более универсальной.

→ Писать желательно маленькие функции, если функция слишком большая - можно разбить её на несколько небольших (при этом можно не учитывать проверку -пред и пост- условий и ошибок).

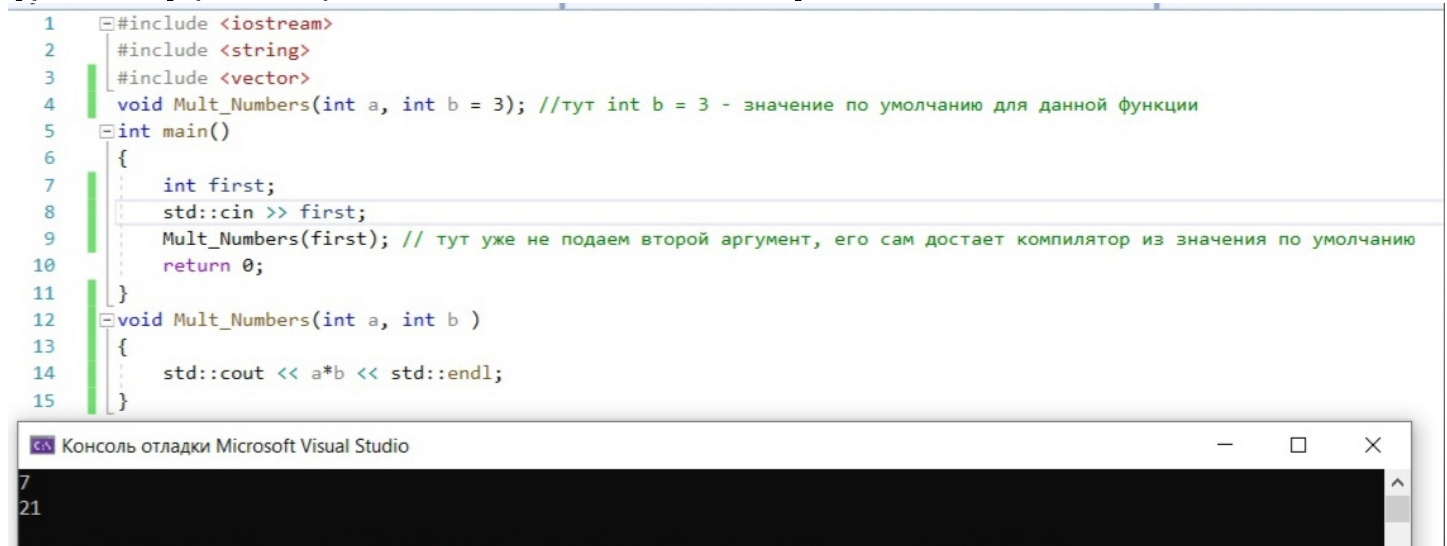
### 2 вопрос: В чём заключается концепция встраивания вызовов функций?

Основная идея: ускорение программы ценой занимаемого места.

Если функция встроенная, то всякий раз, когда мы будем её вызывать, компилятор будет заменять вызов функции фактическим кодом из функции. То есть мы не будем прыгать в отдельную ячейку памяти, где у нас лежит эта функция, а как бы просто подставляем тело функции на место вызова. Естественно, если тело функции достаточно велико, а функция используется достаточно часто, то мы много потеряем в памяти.

### 3 вопрос: Какие аргументы функции могут иметь значения по умолчанию?

Аргумент по умолчанию – это такой аргумент функции, который можно не указывать при вызове функции. Аргумент по умолчанию добавляется компилятором автоматически.



```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  void Mult_Numbers(int a, int b = 3); //тут int b = 3 - значение по умолчанию для данной функции
5  int main()
6  {
7      int first;
8      std::cin >> first;
9      Mult_Numbers(first); // тут уже не подаем второй аргумент, его сам достает компилятор из значения по умолчанию
10     return 0;
11 }
12 void Mult_Numbers(int a, int b )
13 {
14     std::cout << a*b << std::endl;
15 }
```

Если же в программе необходимо использовать другое значение вместо значения аргумента по умолчанию, то тогда при вызове функции мы пишем нужное значение (в примере выше было бы MultNumbers(first, second) )

Аргументы по умолчанию объявляются после переданных (обычных) аргументов.

Проблема может возникнуть только с `char* = nullptr`;

### 4 вопрос: На основании чего разрешается выбор перегруженной функции?

(At first) Параметры функции - это имя функции, число аргументов и типы аргументов. Поэтому если у функций одинаковое имя, но разные параметры, то умный компилятор считает их разными.

*Перегрузка функций* - это создание функций с одинаковыми именами, но разными полными именами (параметрами).

(!) Тип возвращаемого значения не входит в полное имя функции.

Когда можно использовать перегрузку?

→ Тип аргументов. Функцию в том числе определяет и тип её аргументов. Однако если в самой программе заранее не известно, с каким типом данных мы будем работать, но для любого из предполагаемых нами сам алгоритм в теле функции будет выполняться, то имеет смысл создать несколько перегруженных функций, которые будут отличаться лишь типом аргументов (ведь по сути обе они выполняют одно и то же). Например, если нам нужно посчитать сумму элементов массива, а их тип заранее не известен, то можно сделать несколько однотипных по своей работе функций с одинаковым именем.

→ Количество аргументов. Рассмотрим на примере вычисления площади прямоугольника: оригинальная функция работает в сантиметрах (неправильная дробь). Однако пользователь может ввести значения в метрах и сантиметрах (так называемая правильная дробь). Тогда на вход мы получим четыре аргумента, вместо двух. Чтобы не возиться с проверками или просьбами пользователю самостоятельно осуществить перевод (в некоторых случаях это будет не так легко), можно сделать две функции с одним именем, которые будут отличаться количеством аргументов и наличием шкалы перевода в теле одной из них.