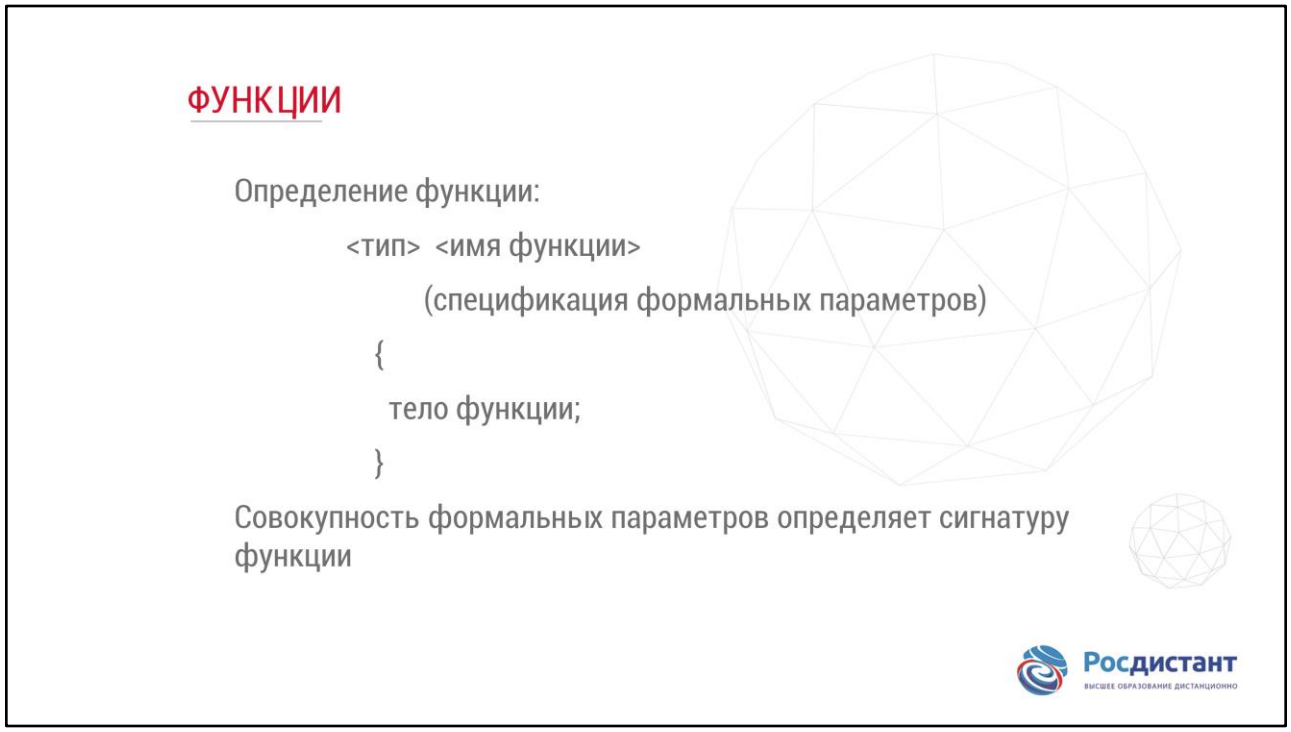

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО
ОСНОВЫ
ПРОГРАММИРОВАНИЯ
6 ЧАСТЬ




**Слайд 137**

**Тема 6. Функции**

«С увеличением объема программы становится невозможным удерживать в памяти все детали. Естественным способом борьбы со сложностью любой задачи является ее разбиение на части. В C++ задача может быть разделена на более простые и обозримые с помощью функций, после чего программу можно рассматривать в более укрупненном виде — на уровне взаимодействия функций.»

Функция – это основное понятие. Ранее говорилось о том, что каждая программа должна содержать единственную функцию с именем **main,** которая называется главная функция. Именно с этой функции начинаются вычисления в программе.

Помимо функции с именем **main**, в программу может входить произвольное количество не главных функций. Каждая функция должна быть определена или хотя бы описана до первого обращения к ней.

«Очень часто в программировании необходимо выполнять одни и те же действия. Например, мы хотим выводить пользователю сообщения об ошибке в разных местах программы, если он ввел неверное значение.

Функции — один из самых важных компонентов языка **C++.**

Любая функция имеет тип, также, как и любая переменная.

Функция может возвращать значение, тип которого в большинстве случаев аналогично типу самой функции.

При объявлении функции, после ее типа должно находиться имя функции и две круглые скобки - открывающая и закрывающая, внутри которых могут находиться один или несколько аргументов функции, которых также может не быть вообще.

После списка аргументов функции ставится открывающая фигурная скобка, после которой находится само тело функции.

В конце тела функции обязательно ставится закрывающая фигурная скобка. Функции очень сильно облегчают работу программисту и намного повышают читаемость и понятность кода, в том числе и для самого разработчика.»

## ФУНКЦИИ

```
#include <iostream>
using namespace std;
void fun_name()
{ cout << "Тело функции" << endl;
}
int main()
{ fun_name(); // Вызов функции
return 0;
}
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 138**

В определении функции указываются последовательность действий, выполняемых при ее вызове, называется телом функции. Структура не главных функций повторяет структуру главной функции.

Тело функции – это блок или составной оператор, то есть последовательность описаний и операторов, заключаются в фигурные скобки.

В определении функции также указываются имя функции, тип функции и совокупность формальных параметров. Необходимо учесть, что тип функции определяет тип возвращаемого результата.

Функция возвращает только один результат, если функция не возвращает никакого результата, то тип функции указывается с помощью служебного слова – **void**;

Имя функции - уникальный идентификатор, с помощью которого происходит вызов функции, обращение к функции.

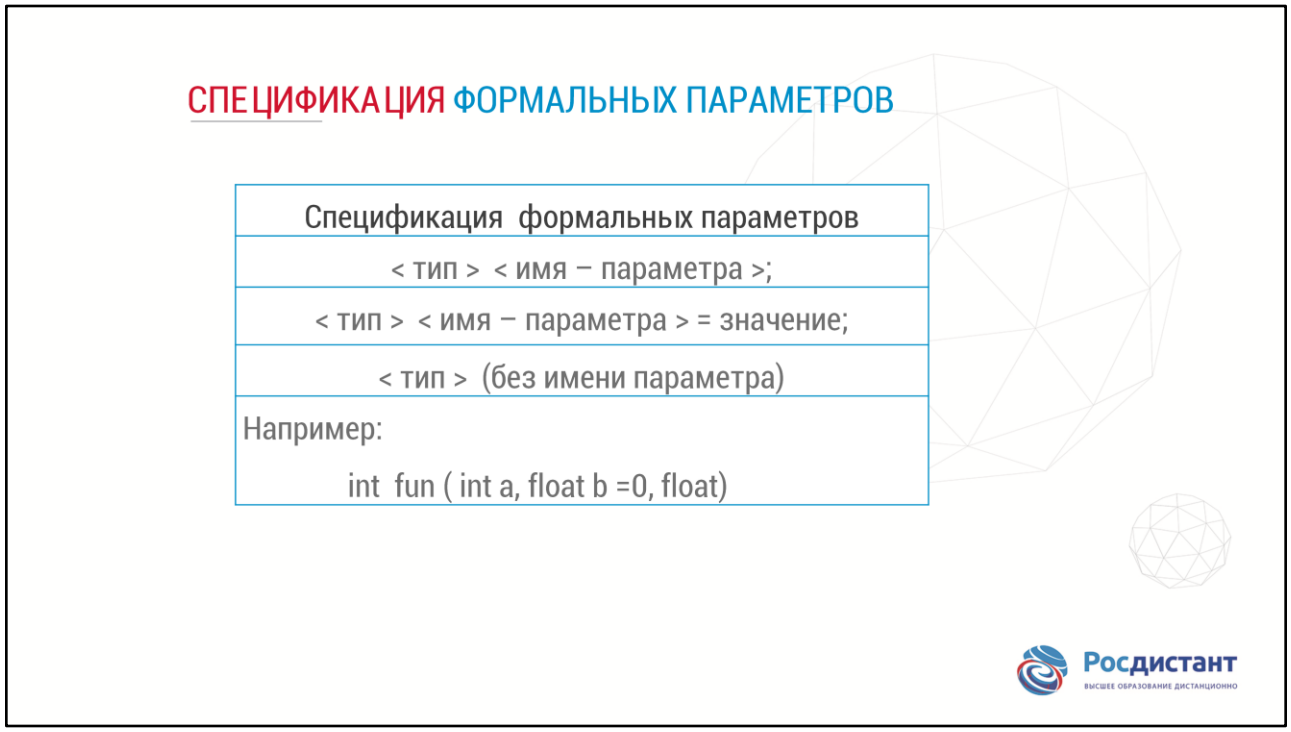
«Функция - это самостоятельная единица программы, которая спроектирована для реализации конкретной подзадачи. Функция является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно, в библиотеке.

Каждая функция выполняет в программе определенные действия.

Сигнатура функции определяет правила использования функции. Обычно сигнатура представляет собой описание функции, включающее имя функции, перечень формальных параметров с их типами и тип возвращаемого значения.

Семантика функции определяет способ реализации функции. Обычно представляет собой тело функции. После того, как произошел вызов функции, начинает работать данная функция. Если функцию нигде не вызвать, то этот код будет проигнорирован программой.

Переменные и константы, объявленные в разных функциях независимы друг от друга, они даже могут иметь одинаковые имена.»




**Слайд 139**

Остановимся подробно на списке формальных параметров.

В списке формальных параметров обязательно задается их тип, то есть формальные параметры - специфицируются.

На слайде приведены различные варианты спецификации формальных параметров. В первом варианте - указывается тип и имя формального параметра, во втором варианте - указывается тип, имя формального и определено значение формального параметра, и третий вариант - указан только тип формального параметра. Зарезервировано место в списке параметров, определена сигнатура функции. В дальнейшем можно внести изменения в функцию, ввести новую переменную, не изменяя вызывающей функции.

Совокупность формальных параметров определяет сигнатуру функции. Сигнатура функции зависит от количества формальных параметров, их типов и порядка следования.

Формальные параметры в теле функции не описываются, они используются при описании тела функции.

Список параметров может совсем отсутствовать. Но при этом, для соблюдения синтаксиса пустые круглые скобки рядом с именем функции - обязательны.

«Разбиение программ на функции дает следующие преимущества:

- Функцию можно вызвать из различных мест программы, что позволяет избежать повторения программного кода.
- Одну и ту же функцию можно использовать в разных программах.
- Функции повышают уровень модульности программы и облегчают ее проектирование.
- Использование функций облегчает чтение и понимание программы и ускоряет поиск и исправление ошибок.

С точки зрения вызывающей программы функцию можно представить как некий «черный ящик», у которого есть несколько входов и один выход. С точки зрения вызывающей программы неважно, каким образом производится обработка информации внутри функции. Для корректного использования функции достаточно знать лишь ее сигнатуру.»

## ВЫЗОВ ФУНКЦИИ

| Обращение к функции | |
|---|---|
| <имя функции> (список фактических параметров); | вызов функции |
| <тип> <имя–функции>(спецификация явных параметров , ...); | заголовок функции с переменным количеством формальных параметров |

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 140**

Вызов функций осуществляется с указанием имени функции и списка фактических параметров. На слайде приведена структура обращения к функции.

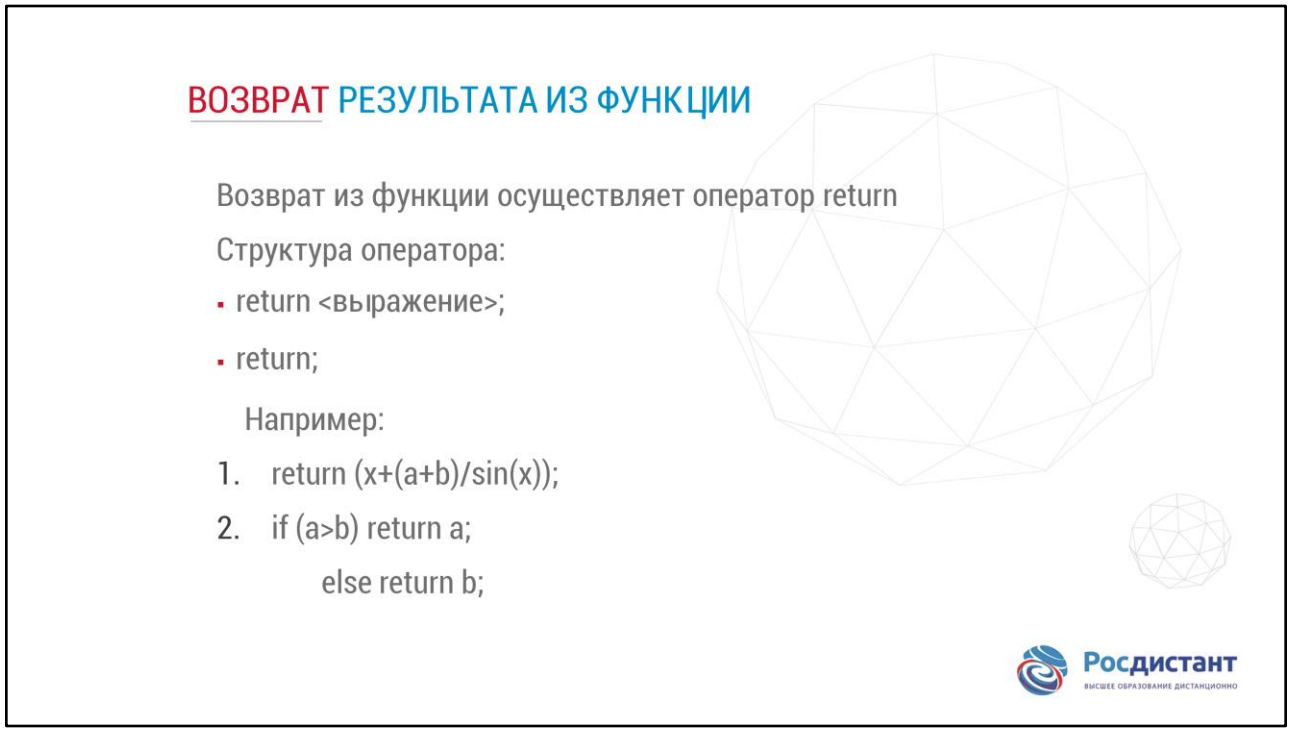
При обращении к функции, формальные параметры заменяются фактическими. Фактическими параметрами могут быть – переменные или константы. Формальные и фактические параметры должны совпадать по типу и порядку следования.

На языке **C++** допускается переменное количество формальных параметров.

В списке формальных параметров сначала указываются имена и типы обязательных параметров, затем после запятой ставится многоточие, три точки по требованию синтаксиса. Компилятору передается сообщение, что дальнейших анализ количества и типов параметров не нужен. Переход от одного фактического параметра к другому осуществляется с помощью указателей.

Фактический аргумент - это величина, которая присваивается формальному аргументу при вызове функции. Таким образом, формальный аргумент - это переменная в вызываемой функции, а фактический аргумент — это конкретное значение, присвоенное этой переменной вызывающей функцией. Фактический аргумент может быть константой, переменной или выражением. Если фактический аргумент представлен в виде выражения, то его значение сначала

вычисляется, а затем передается в вызываемую функцию. Если в функцию требуется передать несколько значений, то они записываются через запятую. При этом формальные параметры заменяются значениями фактических параметров в порядке их следования в сигнатуре функции.




**Слайд 141**

«Особенностью языка C++ является невозможность для функции определять внутри своего тела другую функцию. То есть не допускаются вложенные определения. Но вполне допускается вызов функции из другой функции.

Фактический аргумент - это величина, которая присваивается формальному аргументу при вызове функции. Формальный аргумент - переменная в вызываемой функции, а фактический аргумент - это конкретное значение, присвоенное этой переменной вызывающей функцией. Фактический аргумент может быть константой, переменной или выражением. Если фактический аргумент представлен в виде выражения, то его значение сначала вычисляется, а затем передается в вызываемую функцию. Если в функцию требуется передать несколько значений, то они записываются через запятую. При этом формальные параметры заменяются значениями фактических параметров в порядке их следования в сигнатуре функции. Как говорилось выше, функция возвращает один результат в вызывающую функцию.

По окончании выполнения вызываемой функции осуществляется возврат значения в точку ее вызова. Это значение присваивается переменной, тип которой должен соответствовать типу возвращаемого значения функции. Функция может передать в вызывающую программу только одно значение.»

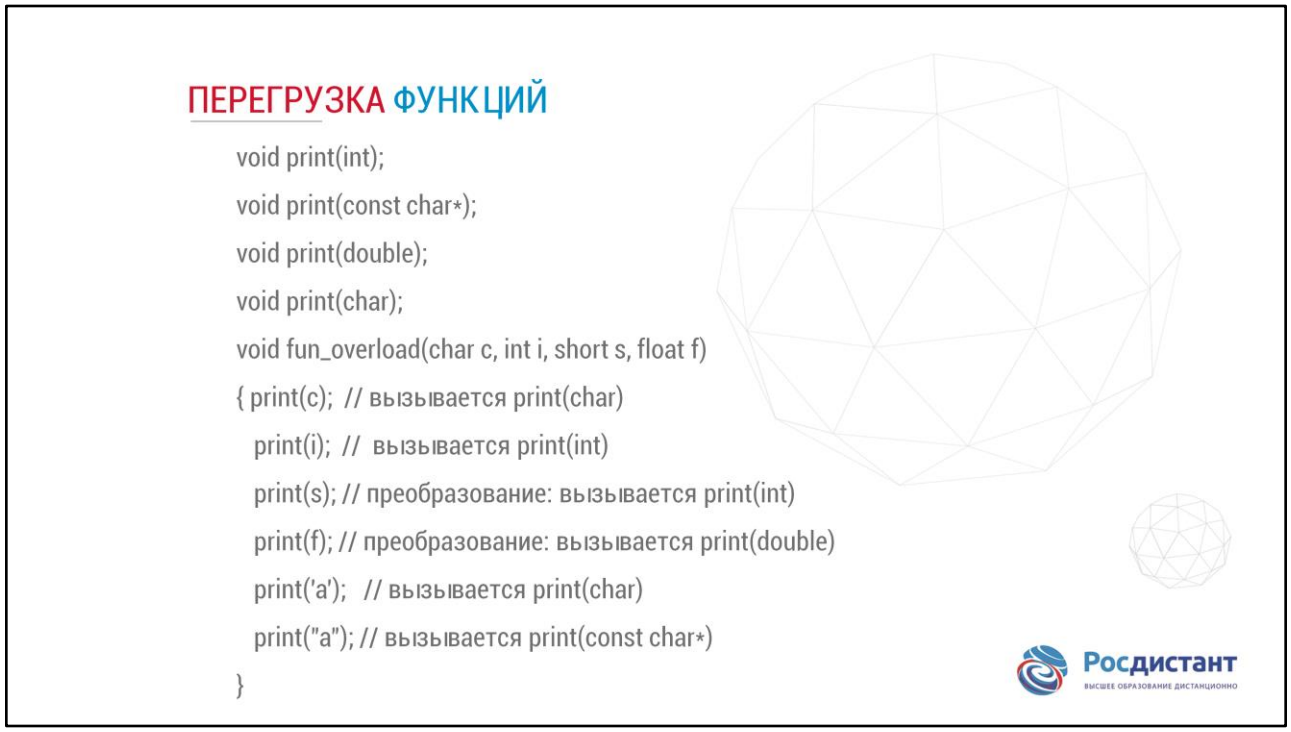
Важным оператором тела функции является оператор **return**.

Указанное выражение, в частном случае, может быть - имя переменной, которой в теле функции присвоено определенное значение.

Если функция не возвращает никакого результата, то выражение может отсутствовать.

Оператор **return** определяет точку выхода из функции, возврат в вызывающую функцию. В теле функции допускается наличие нескольких операторов **return**.

Этот оператор может присутствовать и в главной функции, тогда возвращаемое им значение анализируется операционной системой.




**Слайд 142**

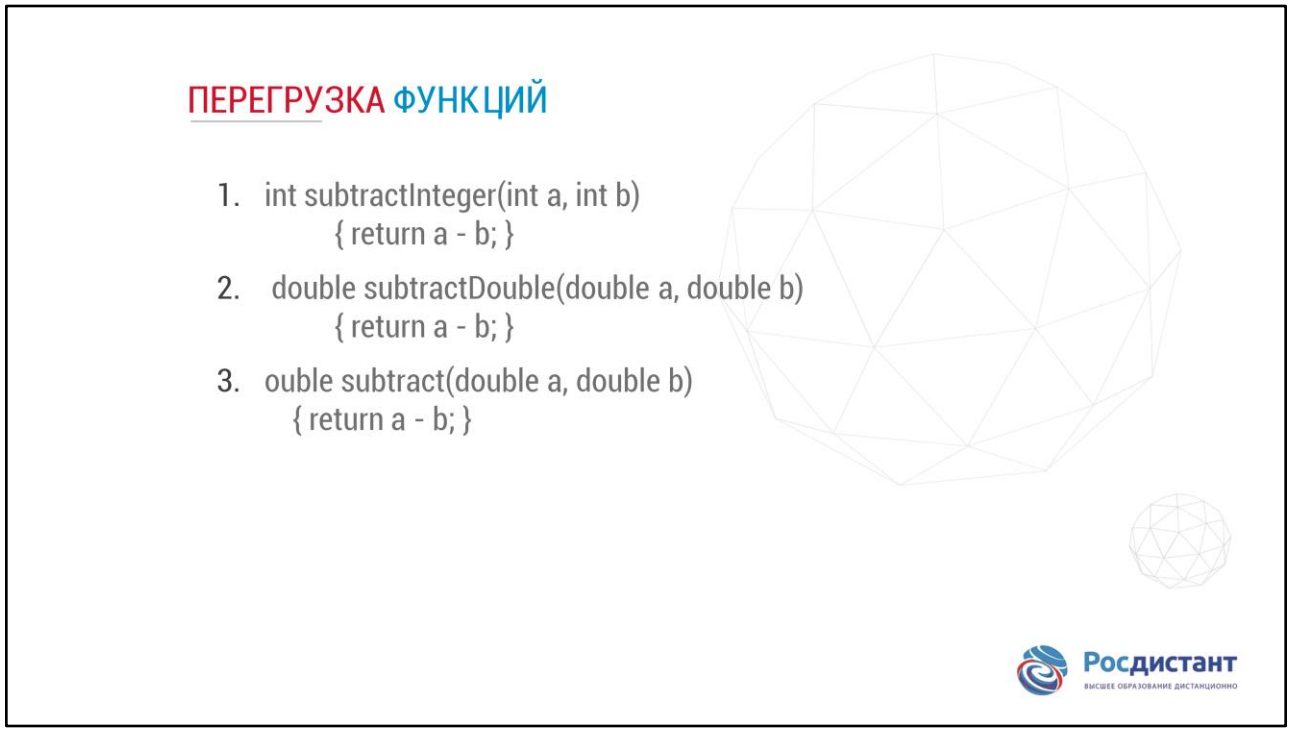
«Часто бывает удобно, чтобы функции, реализующие один и тот же алгоритм для различных типов данных, имели одно и то же имя. Использование нескольких функций с одним и тем же именем, но с различными типами параметров, называется перегрузкой функций. Обычно имеет смысл давать разным функциям разные имена. Если же несколько функций выполняет одно и то же действие над объектами разных типов, то удобнее дать одинаковые имена всем этим функциям. Перегрузкой имени называется его использование для обозначения операций над разными типами. Собственно, уже для основных операций **C++** применяется перегрузка. Действительно: для операций сложения есть только одно имя знак **+ ,** но оно используется для сложения и целых чисел, и чисел с плавающей точкой, и указателей. Такой подход легко можно распространить на операции, определенные пользователем, то есть на функции.

Для транслятора в таких перегруженных функциях общее только одно - имя. Очевидно, по смыслу такие функции сходны, но язык не способствует и не препятствует выделению перегруженных функций. Таким образом, определение перегруженных функций служит, прежде всего, для удобства записи.

При вызове функции с именем **print** транслятор должен разобраться, какую именно функцию **print** следует вызывать. Для этого сравниваются типы фактических параметров, указанные в вызове, с типами формальных параметров всех описаний функций с именем **print**. В результате вызывается та

функция, у которой формальные параметры наилучшим образом сопоставились с параметрами вызова, или выдается ошибка если такой функции не нашлось.

Если найдены два сопоставления по самому приоритетному правилу, то вызов считается неоднозначным, а значит ошибочным. Эти правила сопоставления параметров работают с учетом правил преобразований числовых типов для языка **C++ .**»




**Слайд 143**

«Итак, перегрузка функций - это возможность определять несколько функций с одним и тем же именем, но с разными параметрами.

На слайде приведены три примера. В первом примере выполняется операция вычитания с целыми числами. Однако, если нужно использовать числа типа с плавающей запятой, эта функция совсем не подходит, так как любые параметры типа **double** будут конвертироваться в тип **int**. В результате - будет теряться дробная часть значений.

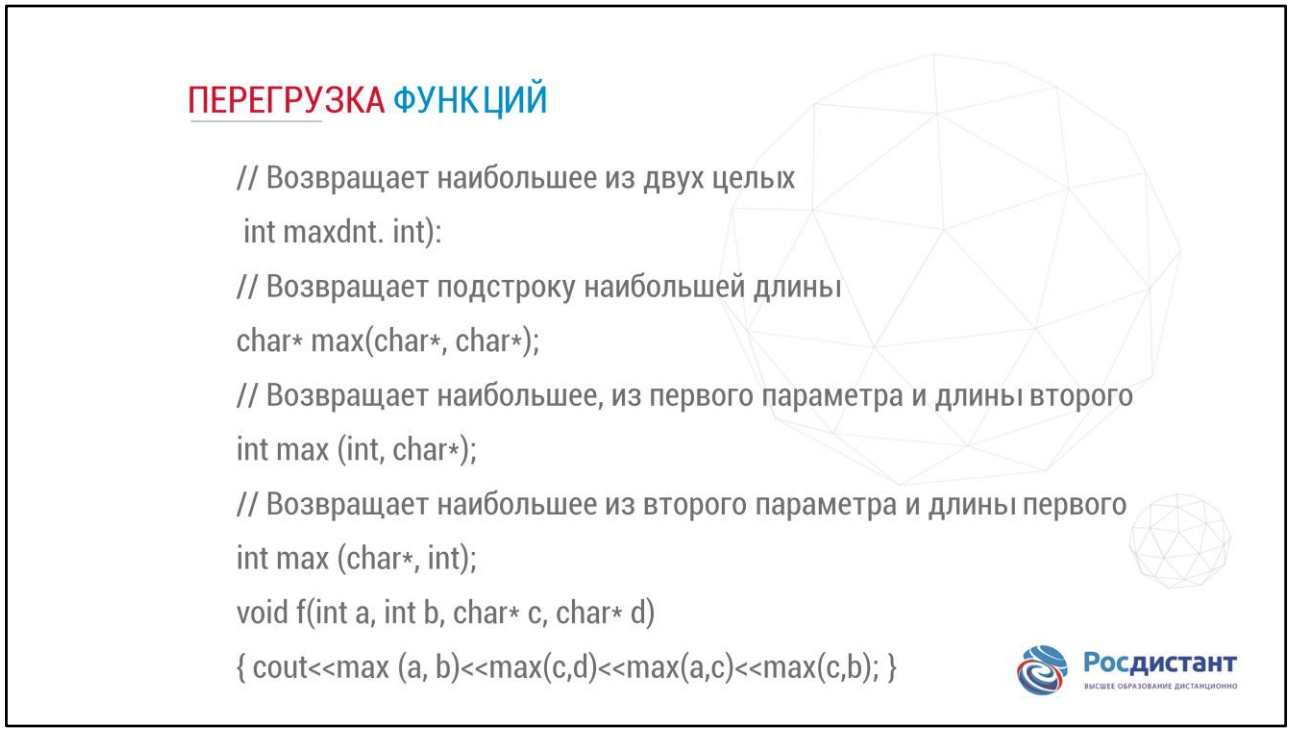
Одним из способов решения этой проблемы является определение второй функций, пример второй.

Но есть и лучшее решение - перегрузка функции. Мы можем просто объявить еще одну функцию **subtract ,** которая принимает параметры типа **double**, третий пример.

Может показаться, что произойдет конфликт имен, но это не так. Компилятор может определить сам, какую версию функции **subtract,** следует вызывать на основе аргументов, используемых в вызове функции. Если параметрами будут переменные тип **int**, то компилятор понимает, что мы хотим вызвать **subtract** с целыми параметрами. Если же мы предоставим два значения типа с плавающей запятой, то компилятор поймет, что мы хотим вызвать **subtract** с вещественными параметрами.

Правила обращения применяются в следующем порядке по убыванию их приоритета:

- Точное сопоставление: сопоставление произошло без всяких преобразований типа; или только с неизбежными преобразованиями.
- Сопоставление с использованием стандартных целочисленных преобразований, то есть символьный **тип char** в целый, тип **short** в целый, а также преобразований вещественный тип **float** в тип **double.**
- Сопоставление с использованием стандартных преобразований, например, целых в вещественные, беззнаковых в целые.»




**Слайд 144**

Рассмотрим еще один пример использования перегрузки функций.

«Компилятор определяет, какую именно функцию требуется вызвать, по типу фактических параметров. Этот процесс называется разрешением перегрузки. Тип возвращаемого функцией значения в разрешении не участвует. Механизм разрешения основан на достаточно сложном наборе правил, смысл которых сводится к тому, чтобы использовать функцию с наиболее подходящими аргументами и выдать сообщение, если такой не найдется. Допустим, имеется четыре варианта функции, определяющей наибольшее значение, как показано на слайде.»

При вызове функции **max** компилятор выбирает соответствующий типу фактических параметров вариант функции. В приведенном примере будут последовательно вызваны все четыре варианта функции. Если точного соответствия не найдено, выполняются продвижения порядковых типов в соответствии с общими правилами.

Например, логические и символьные в целые, вещественный тип **float** в вещественный тип **double,** целые в вещественный тип **double** или указателей в **void.** Следующим шагом является выполнение преобразований типа, заданных, а также поиск соответствий за счет переменного числа аргументов функций. Если соответствие на одном и том же этапе может быть получено более чем

одним способом, вызов считается неоднозначным и выдается сообщение об ошибке.

Неоднозначность может появиться при: преобразовании типа; использовании параметров-ссылок; использовании аргументов по умолчанию.

Перегруженные функции должны находиться в одной области видимости, иначе произойдет сокрытие аналогично одинаковым именам переменных во вложенных блоках.

## ФУНКЦИЯ - УКАЗАТЕЛЬ

```
#include<iostream>
#include<stdio.h >
#include<conio.h >
using namespace std;
char  *min_str(char s1[ ], char s2[ ])
{ int  i =0;
  while(s1[i]!='\0' &&  s2[i]!='\0') i++;
    if (s1[i]= ='\0') return (s1);
              else  return (s2); }
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 145**

Далее рассмотрим использование функций на конкретных примерах.

«Различают системные функции, входящие в состав систем программирования, и собственные функции. Системные функции хранятся в стандартных библиотеках, и пользователю не нужно вдаваться в подробности их реализации. Достаточно знать лишь их сигнатуру. Примером системных функций, используемых ранее, являются функции **cout** и **cin.**
Собственные функции - это функции, написанные пользователем для решения конкретной подзадачи.

Возможны только две операции с функциями: вызов и взятие адреса. Указатель, полученный с помощью последней операции, можно впоследствии использовать для вызова функции.

Отметим, что формальные параметры в указателях на функцию описываются так же, как и в обычных функциях. При присваивании указателю на функцию требуется точное соответствие типа функции и типа присваиваемого значения.»

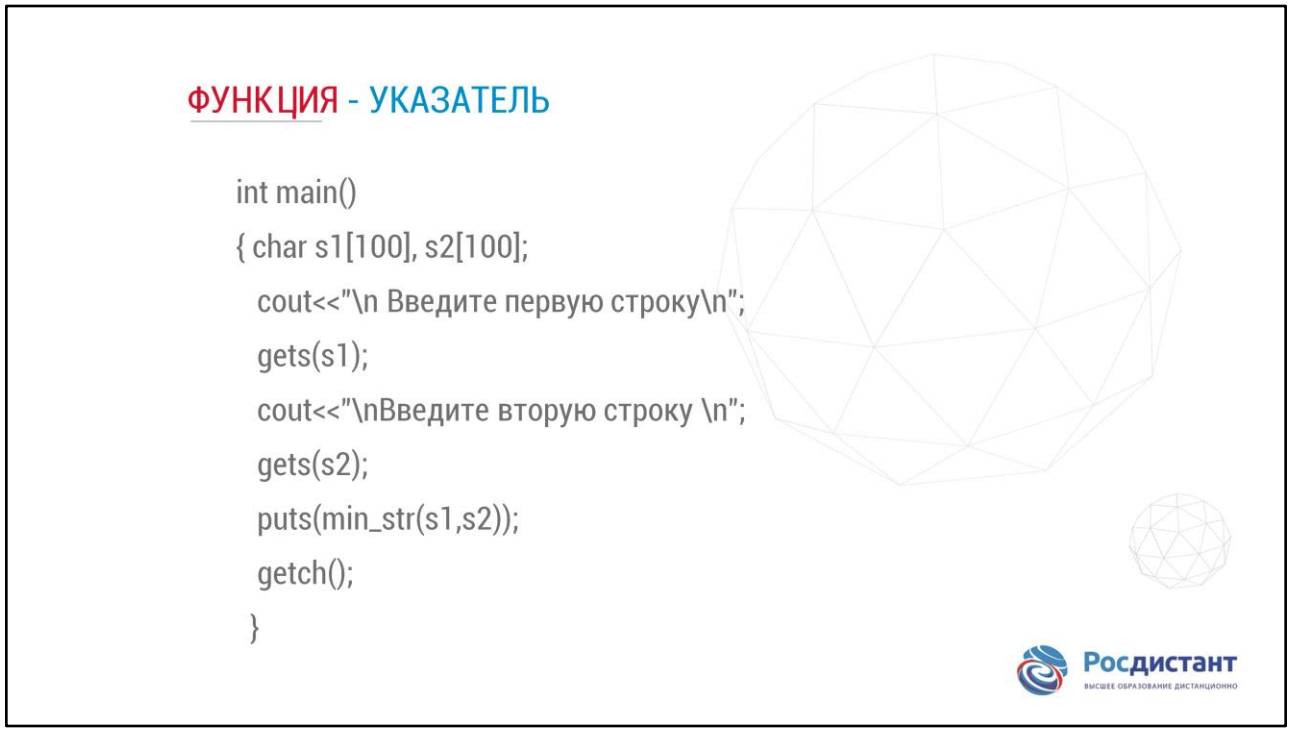
Итак, функция может возвращать результат в виде указателя. В этом случае и тип функции определяется как указатель.

На слайде приведен пример функции, которая возвращает результат в виде указателя. Поэтому, при определении функции, рядом с именем функции ставится символ звездочка.

Функция из двух строковых переменных вернет указатель на более короткую строку. Формальными параметрами являются имена массивов символов.

Так как информация, на которую ссылается указатель, имеет символьный тип, то и тип функции определен как символьный, то есть **char**.

В списке формальных параметров указаны имена двух массивов и пустые квадратные скобки. Размеры массивов можно не указывать. Сравнение продолжается до тех пор, пока в одной из строк не встретится признак конца строки, символ \0 **.** Проверяется условие, и возвращается указатель на более короткую строку.




**Слайд 146**

На слайде представлена главная функция, вызывающая функцию с именем **min_str,** рассмотренную на предыдущем слайде.

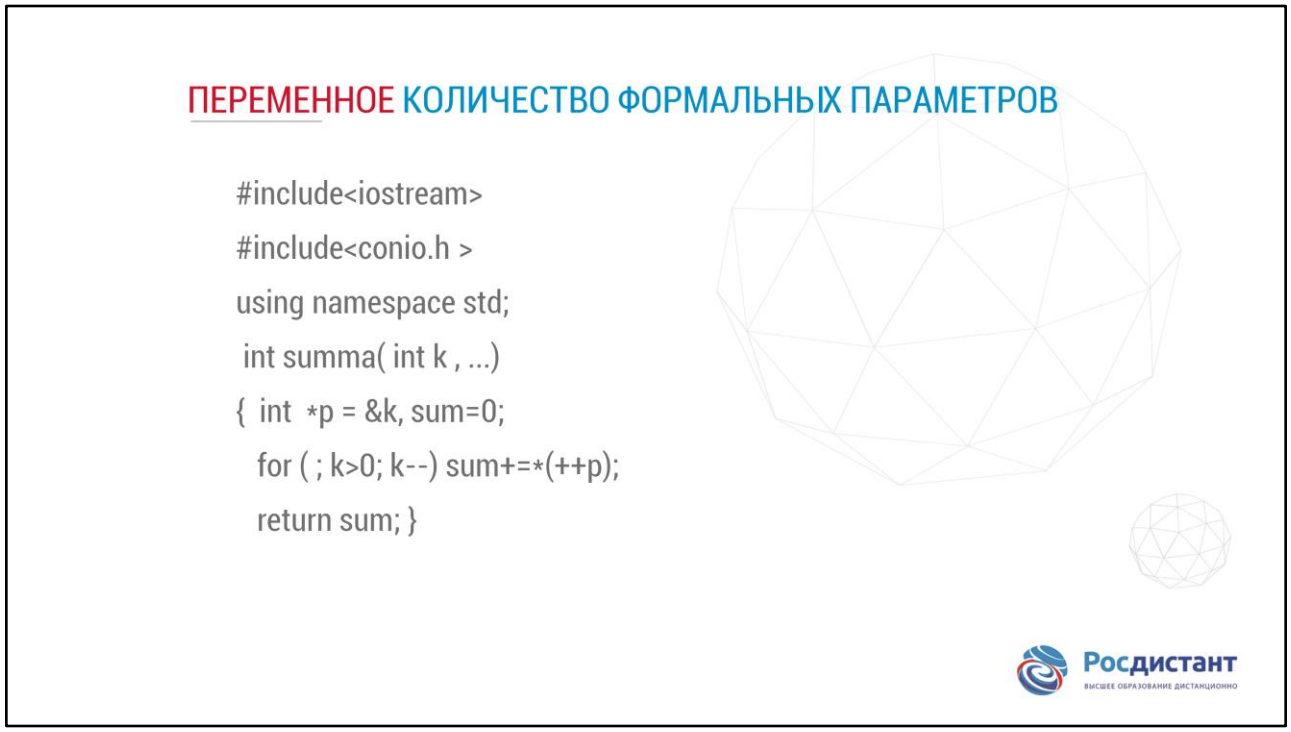
Вывод результата осуществляется с помощью функция **puts ,** поэтому в программе указан заголовочный файл **stdio.**

Каждую строку можно было бы вводить как элементы одномерного массива. Но при работе с цепочкой символов, удобнее рассматривать ее как строку. С клавиатуры поочередно вводятся две строки. При вводе цепочки символов с помощью функции **gets,** в конце этой строки автоматически сформируется признак конца строки. Именно по этому признаку, символу **\0** и выбирается, какая строка короче.

Если функция результата не возвращает, то обращение к ней может выглядеть как отдельный оператор. Если функция возвращает конкретный результат, то обращаться к ней необходимо в структуре какого-либо оператора, или в структуре функции, как в нашем примере. Функция **min_str** является фактическим параметром функции вывода строки **puts.** В данном случае функция **min_str** возвращает результат в виде указателя на ту строку, длина которой короче и содержимое строки выводится на печать.

«Существуют функции, в описании которых невозможно указать число и типы всех допустимых параметров. Тогда список формальных параметров

завершается многоточием, что означает - возможно, еще несколько параметров. Такие функции пользуются для распознавания своих фактических параметров недоступной транслятору информацией.»




**Слайд 147**

Во многих языках программирования формальные и фактические параметры у функций должны совпадать по типу, порядку следования и количеству. На языке **C++** формальные и фактические параметры обязательно должны совпадать по типу и порядку следования, но допустимо переменное количество формальных параметров.

Для того чтобы разобраться с этой особенностью построения функций на языке **C++** рассмотрим следующий пример. На слайде – пример программы с использованием функции с переменным количеством формальных параметров. В случае функции **summa** первый параметр является целочисленным. Очевидно, что раз параметр не описан, транслятор не имеет сведений для контроля и стандартных преобразований типа этого параметра. При описании функции сначала указан явный параметр **k,** целого типа, а затем – многоточие. Это означает, что при вызове этой функции можно указывать различное количество фактических параметров.

Данная функция подсчитывает сумму чисел. Количество чисел может быть различным.

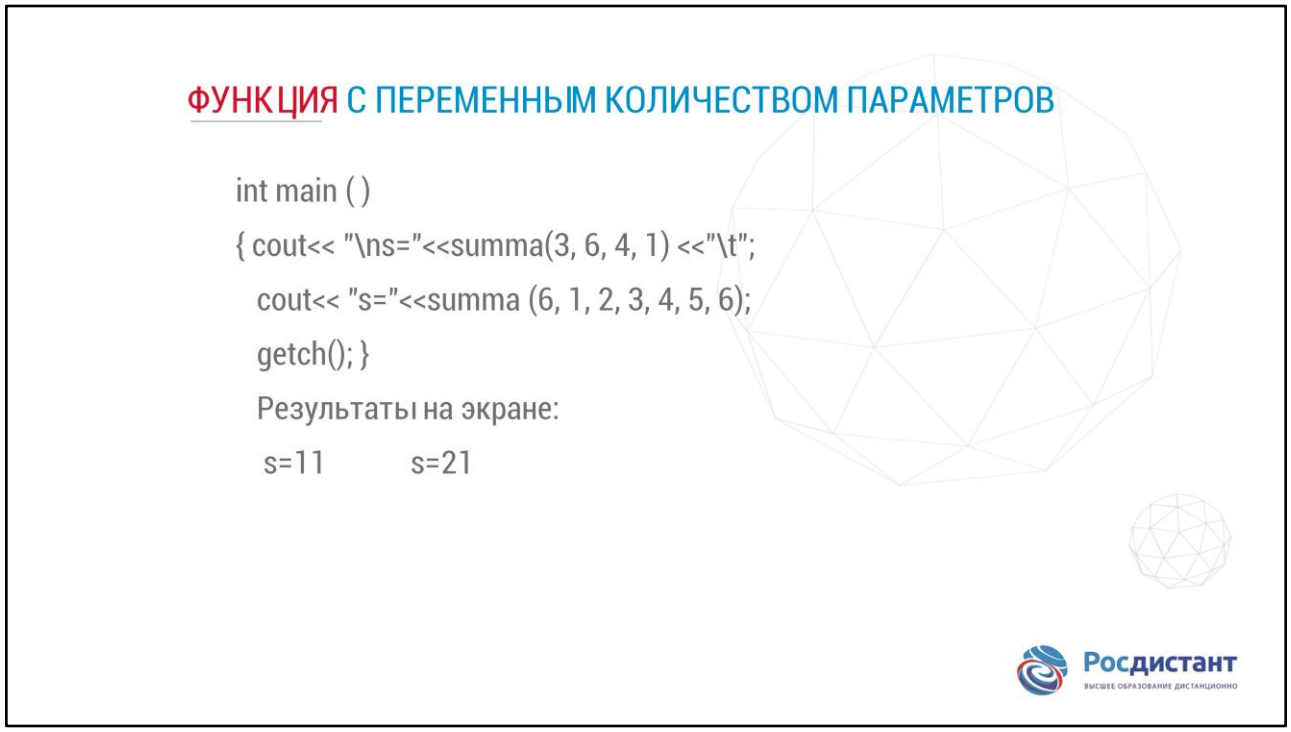
Образуется сигнатура функции с именем сумма, с переменной **k** – во главе. Переменная **k** – и определяет количество чисел.

Используя операцию &, операцию определения адреса, можно легко

определить, что элементы массива расположены в памяти последовательно. То есть, элементы нулевой, первый, второй и так далее размещены рядом, друг за другом. Мы уже знаем, что элементы массива расположены в памяти последовательно. Добавление единицы к указателю возвращает адрес памяти следующего объекта этого же типа данных.

Введена переменная указатель **p** - целого типа, благодаря которой и осуществляется переход от одного числа к другому. Первоначально этой переменной присваивается адрес переменной **k.** Увеличение значения указателя на единицу и соответствует перемещению на четыре байта, количество байт, отводимых под целые числа, то есть переходу к очередному числу. Данная функция возвращает сумму чисел.

«Чтобы обойти контроль типов параметров, лучше использовать перегрузку функций или стандартные значения параметров, чем параметры, типы которых не были описаны. Переменное количество формальных параметров становится необходимым только тогда, когда могут меняться не только типы, но и число параметров.»




**Слайд 148**

Хотя передача массива в функцию на первый взгляд выглядит так же, как передача обычной переменной, но **C++** обрабатывает массивы несколько иначе.

Когда обычная переменная передается по значению, то **C++** копирует значение аргумента в параметр функции. Поскольку параметр является копией, то изменение значения параметра не изменяет значение исходного аргумента.

Однако, поскольку копирование больших массивов — дело трудоёмкое, то **C++** не копирует массив при его передаче в функцию. Вместо этого передается фактический массив. И здесь мы получаем побочный эффект, позволяющий функциям напрямую изменять значения элементов массива!

На слайде представлена главная функция, в которой дважды выполняется обращение к одной и той же функции вычисления суммы целых чисел. При первом обращении указано четыре фактических параметра, первый параметр – это количество, а следующие три фактических параметра – это те числа, сумму которых надо подсчитать.

«Тип возвращаемого значения и типы параметров совместно определяют тип функции. Для вызова функции в простейшем случае нужно указать ее имя, за которым в круглых скобках через запятую перечисляются имена передаваемых аргументов. Вызов функции может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, который формирует функция.

Если тип возвращаемого функцией значения не **void**, она может входить в состав выражений или, в частном случае, располагаться в правой части оператора присваивания.»

При втором вызове функции указано семь фактических параметров.

Первый параметр - количество, а затем перечислены сами числа.

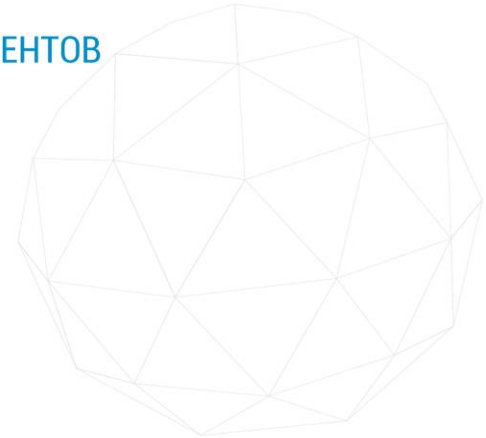
Вызов функции осуществляется в структуре функции вывода на печать. После вывода суммы чисел первой последовательности указан управляющий символ табуляции, после чего выводится сумма второй последовательности чисел. Оба результата выводятся на одной строке экрана.

Ограничением этой функции является тот факт, что она применима только для подсчета суммы целых чисел. Так как переход от одного числа к другому осуществляется с помощью указателя, то и все фактические параметры должны иметь целый тип.

Результаты вычислений продемонстрированы на слайде.

## ФУНКЦИЯ ПОИСКА СУММЫ ЭЛЕМЕНТОВ

```
#include<iostream>
#include<conio.h>
using namespace std;
int  sum_array(int n, int a[ ])
 { for (int i=0, summa=0; i <n; i++)
        if (a[ i ]>0) summa+=a[i];
   return summa; }
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 149**

«Если тип возврата функции не **void**, то она должна возвращать значение указанного типа, использовав оператор **return**. Единственно, исключением является функция **main**, которая возвращает 0, если не предоставлено другое значение.

Когда процессор встречает в функции оператор **return**, он немедленно выполняет возврат значения обратно. Любой код, который находится за оператором **return** в функции - игнорируется.

Функция может возвращать только одно значение через оператор **return** обратно. Это может быть либо число, либо значение переменной, либо выражение, у которого есть результат, либо определенное значение из набора возможных значений.»

На слайде представлена функция нахождения суммы элементов одномерного массива. Функция возвращает в главную функцию значение суммы элементов массива с помощью оператора **return**.

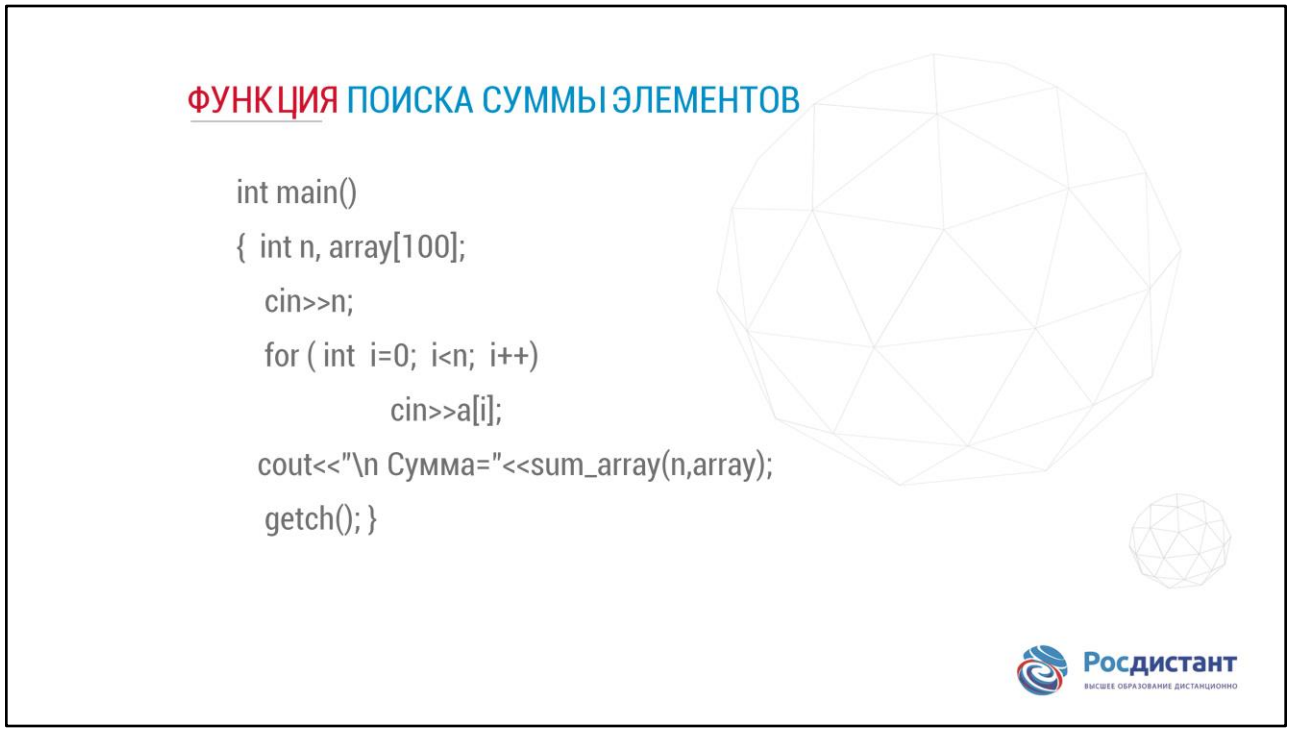
Формальными параметрами являются размер массива и сам массив. Обратите внимание на пустые квадратные скобки при описании второго формального параметра - массива. Это вполне допустимо, фактически в функцию передается указатель на массив, адрес начала массива, адрес нулевого элемента.

Переменная **i** – является индексом массива, она необходима для перебора

элементов массива. В переменной **summa** подсчитывается сумма элементов массива.

Обе переменные используются только в теле функции, являются локальными, описаны в теле функции и время их существования - от начала и до конца работы функции. Вне этой функции эти переменные не доступны. Время их существования – от начала и до конца работы функции, в которой описаны.

Обратите внимание, значению переменной **summa** присвоено значение ноль в структуре оператора **for,** во время инициализации. Как известно, инициализация выполняется только один раз в оператора цикла с параметром. В теме базовые алгоритмические структуры, мы подробно рассматривали оператор цикла с параметром.




**Слайд 150**

На слайде представлен код главной функции.

Описан массив целых чисел. Размер массива и элементы массива вводятся с клавиатуры.

Вызов функции подсчета суммы элементов массива указан в структуре функции вывода на экран **cout**. Возврат из функции осуществляется в ту же точку, из которой было обращение к функции и выводится значение суммы.

Обратите внимание на список фактических параметров. Указывается размер массива и имя массива при вызове функции.

«При использовании в качестве параметра массива в функцию передается указатель на его первый элемент, иными словами, массив всегда передается по адресу. При этом информация о количестве элементов массива теряется, и следует передавать его размерность через отдельный параметр.»

В функции использовался формальный параметр **a** – как имя массива, а в главной функции – имя массива обозначено как **array.** Это вполне допустимо!

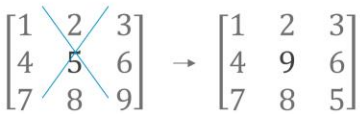
Еще раз напомним, что формальные и фактические параметры должны совпадать по типу и порядку следования.

При вызове функции, формальные параметры заменяются на фактические параметры, и именно с ними выполняются вычисления. Как известно,

формальные параметры в теле функции не описываются, их тип определяется в списке формальных параметров при определении функции.

## ОБРАБОТКА ДВУМЕРНЫХ МАССИВОВ

Найти максимальный элемент квадратной матицы и поменять его местами с элементом, стоящим на пересечении диагоналей



Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 151**

В теме указатели и адреса объектов, говорилось о том, что допустимы указатели на функции. Рассмотрим пример обработки двумерного массива с использованием функции, которая возвращает результат в виде указателя. На слайде приведена формулировка задачи. Приведен пример исходной и преобразованной матрицы, с конкретными числами.

При решении этой задачи необходимо учесть, так как матрица квадратная, количество строк и столбцов равны и является нечетным числом, иначе на пересечении диагоналей нет элемента. В программе обязательно надо выполнить проверку на нечетность размере массива, и, если условие нечетности не выполняется, выдать соответствующее сообщение. Вспомним свойство результативности алгоритма, которое гласит, что алгоритм должен приводить или к результату, или к сообщению о причинах его отсутствия!

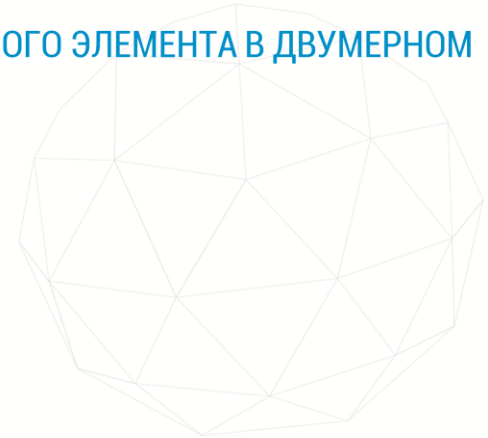
В данной задаче результатом работы функции является номер строки и номер столбца, на пересечении которых находится максимальный элемент массива. Зная индексы максимального элемента, можно будет обменять его с элементом на пересечении главной и побочной диагоналей, индексы которого легко подсчитываются, через размер массива. Известно, что на языке **C++,** функция возвращает только один результат. Поэтому, результат возвращает в виде указателя на максимальный элемент массива.

Опишем массив как глобальную переменную, вне всяких функций. Тогда массив будет доступен во всех функциях, включая главную, и отпадает необходимость передавать массив в функции через формальные параметры.

Глобальные переменные существуют от начала и до конца работы программы. Не стоит злоупотреблять использованием глобальных переменных.

## ФУНКЦИЯ ПОИСКА МАКСИМАЛЬНОГО ЭЛЕМЕНТА В ДВУМЕРНОМ МАССИВЕ

```
… int a[5][5];
int  *max_array(int n)
{  int max=a[0][0], imax=0, jmax=0;
   for( int  i=0;  i<n; i++)
    for ( int  j=0;  j<n; j++)
        if (a[i][j]>max) { max=a[i][j];
                          imax=i; jmax =j;}
    return &a[imax] [jmax]; }
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 152

«Все величины, описанные внутри функции, а также ее параметры, являются локальными. Областью их действия является функция. При вызове функции, как и при входе в любой блок, в стеке выделяется память под локальные автоматические переменные. Кроме того, в стеке сохраняется содержимое регистров процессора на момент, предшествующий вызову функции, и адрес возврата из функции для того, чтобы при выходе из нее можно было продолжить выполнение вызывающей функции. При выходе из функции соответствующий участок стека освобождается, поэтому значения локальных переменных между вызовами одной и той же функции не сохраняются.

Первая строка является описанием функции. Она задает функцию с параметром типа - целое, возвращающую значение типа - указатель. Описание функции необходимо для ее вызова. При вызове функции тип каждого фактического параметра сверяется с типом, указанным в описании функции, точно так же, как если бы инициализировалась переменная описанного типа. Это гарантирует надлежащую проверку и преобразования типов.»

На слайде представлена функция поиска максимального элемента в двумерном массиве целых чисел.

Звездочка перед именем при описании функции max_array  и означает, что функция возвращает в виде результата - указатель. Так как массив целых чисел,

то и возвращаемый результат, то есть тип функции, - целого типа. В списке формальных параметров указан размер массива.

Обратите внимание на локальные переменные – это значение максимального элемента массива max  и его индексы imax  и jmax . Они определены в теле функции, так как в главной функции они не используются.

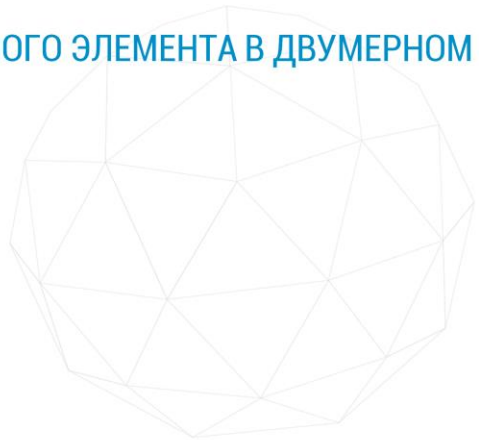
Необходимо определить номер строки и номер столбца, на пересечении которых находится максимальный элемент массива. Первоначально значению максимального присваивается значение элемента массива, расположенного на пересечении нулевой строки и нулевого столбца.

Поиск максимального элемента осуществляется во вложенных циклах.

Определив индексы максимального элемента, с помощью унарной операции определения адреса & , в главную функцию возвращается указатель на максимальный элемент массива, фактически адрес максимального элемента.

## ФУНКЦИЯ ПОИСКА МАКСИМАЛЬНОГО ЭЛЕМЕНТА В ДВУМЕРНОМ МАССИВЕ

```
int main()
{ int n;  cin>>n;
  if (n%2 !=0)
   {for ( int i=0; i<n; i++)
       for ( int j=0; j<n; j++)
            cin>>a[i][j];
    int  *p=max_array(n);
    swap( *p, a[n/2][n/2] );
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 153

На слайде представлено продолжение программы.

После ввода размера массива n  необходимо проверить, является ли оно нечетным числом. Проверка осуществляется в структуре логического оператора if . Вся последовательность операторов, в случае, когда указанное условие истинно, то есть введено нечетное число, описана в фигурных скобках.

Если введено четное число, то есть указанное условие не выполняется, на экране появится сообщение об ошибке ввода. Это будет реализовано в следующем фрагменте программы, на следующем слайде.

Переменные i  и j , используемые как индексы двумерного массива, описаны непосредственно в структуре оператора цикла. Это локальные переменные, доступные только в структуре этого оператора.

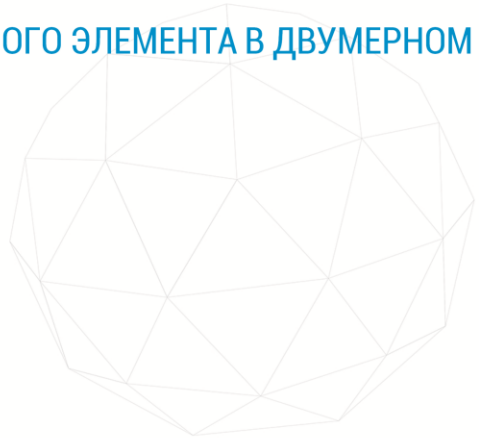
В программе определена переменная p  - как указатель, которой присваивается результат, возвращенный из функции max_array .

Обращение к функции осуществляется в операторе присваивания. Напомним, возврат из функции осуществляется в ту же точку, из которой происходил вызов функции. С помощью функции swap  меняются местами максимальный элемент с элементом на пересечении главной и побочной диагоналей. Обмен осуществляется через указатель на максимальный элемент.

Когда перед указателем p  ставится знак звездочка, осуществляется обращение к информации по заданному адресу, фактически к максимальному элементу.

## ФУНКЦИЯ ПОИСКА МАКСИМАЛЬНОГО ЭЛЕМЕНТА В ДВУМЕРНОМ МАССИВЕ

```
for ( int i=0; i<n; i++)
   { for (int j=0; j<n; j++)
            cout<<"\t"<<a[i][j];
      cout<<"\n"; }
}
else  cout<<"\n Ошибка ввода! \n";
... }
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 154

На слайде представлен фрагмент вывода преобразованного массива в виде матрицы.

Вывод осуществляется во вложенных циклах. Внутренний цикл позволяет вывести на экран элементы одной отдельно взятой строки. Внешний цикл позволяет перейти от строки к строке.

Обратите внимание, переменные, которые служат индексами двумерного массива, описаны с структуре операторов цикла.

Выделена часть оператора проверки условия, структура else .

Дополнительные фигурные скобки определяют последовательность действий, которые будут пройдены при выполнении условия задачи, то есть введено нечетное число. В противном случае, то есть введено четное число, на экране появится сообщение о некорректном вводе.

Итак, сделаем выводы: если необходимо вернуть из функции несколько результатов, можно воспользоваться указателями или глобальными переменными.

Например, в нашей задаче индексы максимального элемента массива можно было описать вне всяких функций, тогда они были бы доступны во всех функциях, включая главную. Отпадает необходимость в возврате значений из

функции. Но учтите, не стоит злоупотреблять глобальными переменными, значения которых сохраняются от начала и до конца работы программы.

## ПОИСК В МАССИВЕ

| Введенные обозначения | Пояснения |
|---|---|
| m | количество элементов в массиве |
| k | искомое значение |
| p | указатель на начало массива |
| a[][] | двумерный массив |
| b[] | одномерный массив |

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 155

Рассмотрим задачу: составить программу поиска некоторого заданного числа k в массиве целых чисел. Определить местоположение найденного числа в массиве. Использовать одну функцию как для поиска в одномерном массиве, так и для поиска в двумерном массиве. На слайде приведены введенные обозначения.

Составим универсальную функцию, к которой будем обращаться как для поиска определенного числа в одномерном массиве, так и в двумерном массиве. Единственным ограничением в данной задаче является тип элементов массива. Массив должен содержать целые числа. Это связано с тем, что переход от одного элемента массива к другому, как в одномерном, так и в двумерном, будет осуществляться через указатель. Указатель также должен иметь целый тип.

Увеличение значения указателя на единицу, то есть адреса на единицу, фактически означает переход от одного элемента массива к другому элементу. Вспомним, что элементы двумерного массива хранятся в памяти по строкам и с помощью указателя можно перебрать все элементы и двумерного массива.

Введены дополнительные переменные m , k и p . В одномерном массиве местоположение элемента в массиве определяется одним индексом. В двумерном массиве каждый элемент определяется двумя индексами. Функция

может вернуть только один результат. Так как эта функция будет использоваться и для поиска в одномерном массиве, и для поиска в двумерном массиве, результат возвращается в виде указателя на найденный элемент в массиве. При обращении к этой функции для поиска числа в двумерном массиве, необходимо будет указать количество элементов, то есть произведение количества строк на количество столбцов.

Через этот указатель будем определять местоположение найденного элемента как в одномерном массиве, так и в двумерном массиве.

## ФУНКЦИЯ ПОИСКА В МАССИВЕ

```
#include<iostream>
#include<conio.h
 using namespace std;
 int  *poisk( int  *a, int  m, int  k)
 {  for (int i=0; i<m;  i++)
     { if ( *a= =k) return (a);
       a++; }
   return (NULL); }
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 156

На слайде представлена функция поиска числа в массиве, которая возвращает результат в виде указателя на найденный в массиве элемент.

Первая часть определения функции задает ее имя, тип возвращаемого значения, а также типы и имена формальных параметров. Значение возвращается из функции с помощью оператора return .

Соответственно перед именем функции ставится символ звездочка. Тип функции целый, определяется типом элементов массива, типом возвращаемого результата.

В списке формальных параметров:

- указатель на начало массива, то есть имя массива;
- количество элементов в массиве, одномерном, или двумерном;
- искомое число k .

Имя массива является указателем на начало массива. Увеличение значения переменной a  на единицу, то есть увеличения адреса на единицу, при каждом прохождении через цикл соответствует переходу от одного элемента к другому.

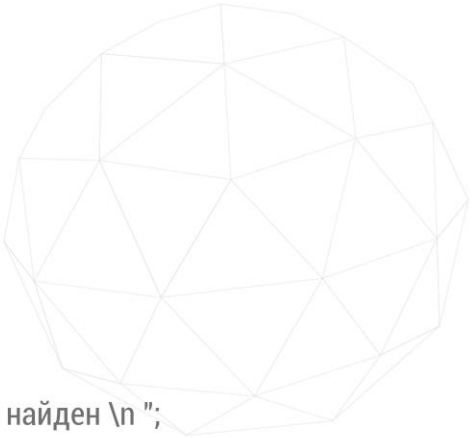
Напомним, что элементы двумерного массива расположены в памяти по строкам. Перебрав элементы одной строки, указатель переходит к перебору элементов следующей строки.

Через этот указатель определяется местоположение числа в массиве, как одномерном, так и в двумерном. Именно через указатель будем определять местоположение элемента.

Если число в массиве на найдено, функция вернет с помощью оператора return константу NULL .

## ПОИСК В ДВУМЕРНОМ МАССИВЕ

```
int main()
{  int *p, k, nS, nSt;
 int  a[2][3]={ {0, 1, 2},
                {3, 4, 5}};
 cin>>k;  p=poisk( *a, 6, k);
 if (p= =NULL) cout <<"\n Элемент не найден \n ";
  else { nS =(p - *a)/3; nSt =p - *a - nS*3;
         cout<<"\n a["<<nS<<"]["
        <<nSt<<"] ="<<*p; }
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 157**

Рассмотрим фрагмент использования описанной выше функции для поиска искомого числа в двумерном массиве.

Значения элементов массива, как одномерного, так и двумерного определены в программе. Значение искомого числа вводится с клавиатуры.

При первом обращении к функции **poisk ,** фактическими параметрами являются:

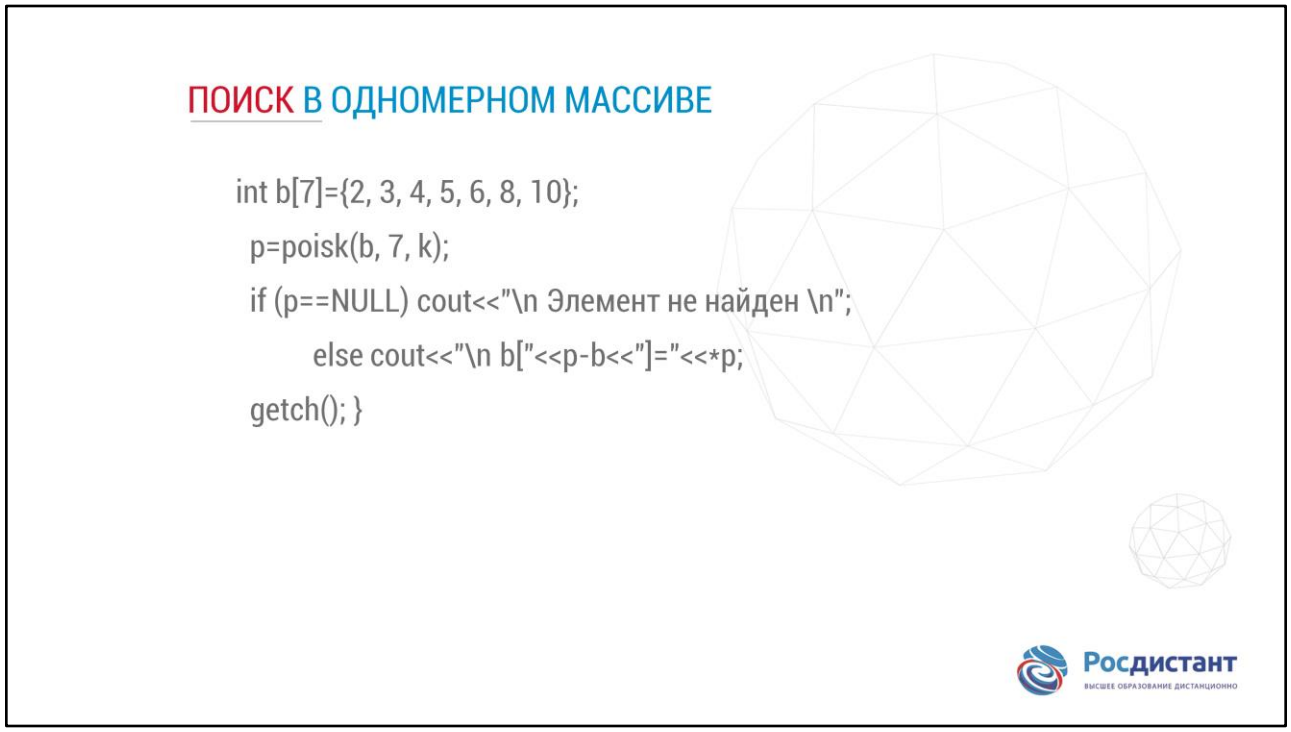
- имя двумерного массива,
- константа, определяющая количество элементов двумерного массива
- искомое число**.**

Введены две переменные, **S** - номер строки и **St** номер столбца. С помощью этих переменных по указанным формулам, через указатель, полученный как результат работы функции, определяется номер строки и номер столбца на пересечении которых находится найденный элемент. Двумерный массив инициализирован во время описания конкретными значениями. Обратите внимание, в формулах определения номера строки и номера столбца присутствует число три – это количество столбцов двумерного массива. Если размерность массива другая, в формулу необходимо внести изменения.

Обращение к функции **poisk** происходит в теле функции вывода **cout .** В функции вывода предусмотрен формат вывода найденного элемента с указанием номера

строки и номера столбца, на пересечении которых расположен найденный элемент. На экран выведется элемент с указанием его положения в двумерном массиве.

Если элемент не найден в двумерном массиве, выдается соответствующее сообщение. Для этого используется константа **NULL.**




**Слайд 158**

Следующий фрагмент демонстрирует обращения к той же функции **poisk** для нахождения искомого числа в одномерном массиве. Одномерный массив инициализирован конкретными значениями при объявлении.

Фактическими параметрами при обращении к функции являются имя одномерного массива, количество элементов в одномерном массиве и искомое число.

Функция вернет указатель на найденный элемент в одномерном массиве. Если число будет найдено в массиве, то на экран выведется элемент одномерного массива с указанием индекса, значение которого подсчитывается через указатель. Индекс элемента определяется как разность между указателем, то есть адресом найденного элемента и адресом начала массива. Индекс определяет положение элемента в общем наборе, массиве.

Если число в массиве не найдено, то есть функция возвращает константу **NULL** , появится соответствующее сообщение.

Если значения массивов не определять в явном виде в программе, то необходимо ввести дополнительные переменные как количество элементов в одномерном и двумерном массивах соответственно. И тогда можно использовать ту же функцию поиска.

На следующем слайде рассмотрим работу этой функции поиска на конкретных

примерах.

Разделение программы на функции позволяет избежать избыточности кода, поскольку функцию записывают один раз, а вызывать ее на выполнение можно многократно из разных точек программы. Процесс отладки программы, содержащей функции, можно лучше структурировать. Часто используемые функции можно помещать в библиотеки. Таким образом создаются более простые в отладке и сопровождении программы.

## РЕЗУЛЬТАТ РАБОТЫ ФУНКЦИИ

| | |
|---|---|
| 1. При k=2<br>Результат на экране:<br>a[0] [2] =2<br>b[0]=2 | Исходные массивы:<br>a[2][3]={ {0, 1, 2}, {3, 4, 5} };<br>b[7]={2, 3, 4, 5, 6, 8, 10}; |
| 2. При k=10<br>Результат на экране:<br>Элемент не найден<br>b[6]=10 | |

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 159

На слайде приведены два варианта результатов, при вводе различных значений переменной k.

Переменная, значение которой ищется в массивах, вводится с клавиатуры. Для двумерного массива – это общее количество элементов массива.

В правой колонке – исходные массивы. В правой колонке – значения элементов одномерного и двумерного массивов.

В первом случае, при значении искомой переменной равной двум, число будет найдено и в двумерном и в одномерном массиве. Положение найденного элемента в массивах определено с помощью индексов, указанных в квадратных скобках.

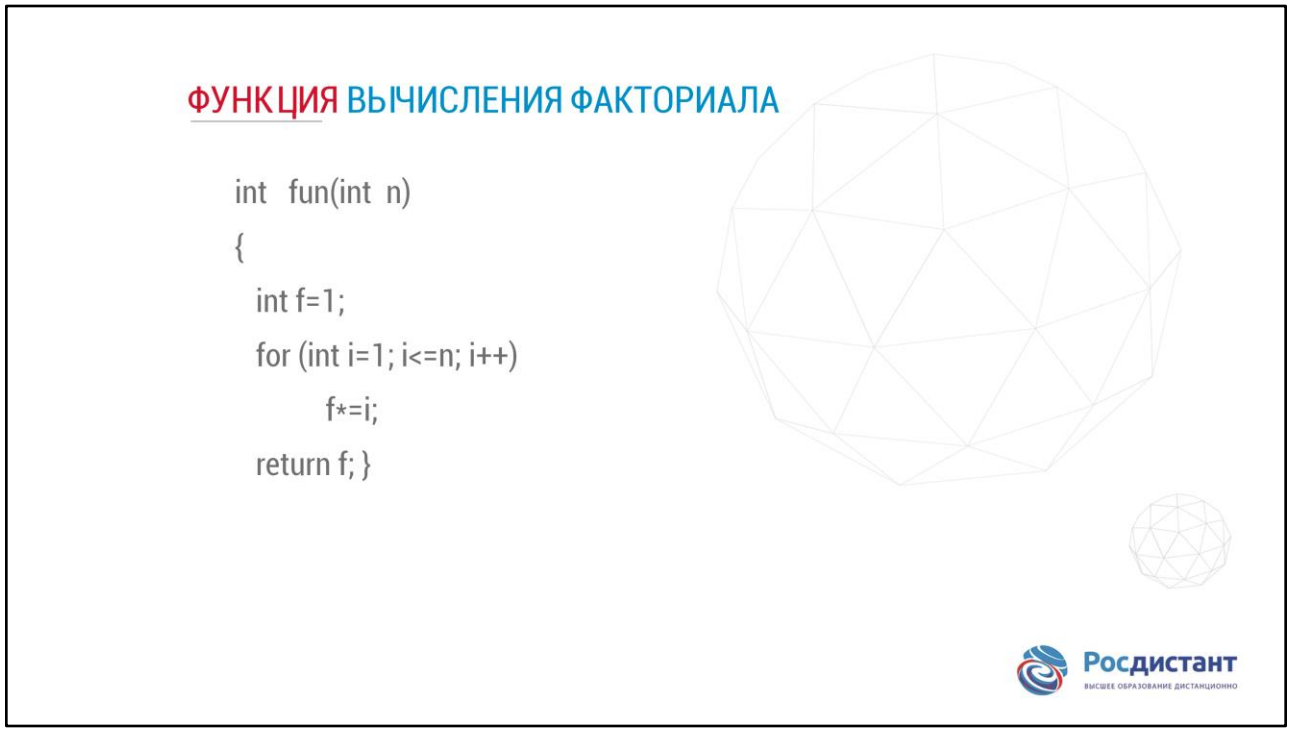
Во втором варианте, при значении искомой переменной равной десяти, число в одномерном массиве найдено, а в двумерном массиве число не найдено, о чем выдается сообщение.

«Так же часто бывает полезен массив указателей на функции. Например, можно реализовать систему меню для редактора с вводом, управляемым мышью, используя массив указателей на функции, реализующие команды.

Но как быть, когда нам потребуется вызвать функцию, которая должна получить несколько значений в результате вычислений. Здесь для этого проще всего

использовать так называемые функции типа void . Такие функции не содержат оператора return  и поэтому как бы ничего не передают наружу. Однако в C++ существует понятие глобальной переменной, то есть такой переменной, которая доступна в любой функции данной программы. Если, обратится к функции типа void  просто по имени, без использования присваивания, и внутри нее пересчитать глобальные переменные, то после этого в основной программе значения глобальных переменных изменятся.»

Напомним, что глобальные переменные описываются до открытия главной функции main .




**Слайд 160**

«Рекурсивная функция - это функция, которая вызывает саму себя. Это в случае прямой рекурсии. Существует и косвенная рекурсия - когда две или более функций вызывают друг друга. Когда функция вызывает себя, в стеке создаётся копия значений её параметров, после чего управление передаётся первому исполняемому оператору функции. При повторном вызове процесс повторяется.

Рекурсивные функции являются альтернативой циклам. Рекурсивные функции в основном используются для упрощения проектирования алгоритмов. В программировании рекурсия тесно связана с функциями, точнее именно благодаря функциям в программировании существует такое понятие как рекурсия или рекурсивная функция. Простыми словами, рекурсия – определение части функции через саму себя, то есть это функция, которая вызывает саму себя. Типичными рекурсивными задачами являются задачи: нахождения **n!** , числа Фибоначчи. Такие задачи мы уже решали, но с использованием циклов, то есть итеративно.»

На слайде - представлена функция вычисления факториала, произведения натуральных чисел от одного до **n .**

Для вычисления произведения используется цикл с параметром. Однако есть очень красивый подход замены циклов рекурсией.

Рекурсия – фундаментальное понятие в математике и компьютерных науках. В

программировании рекурсивной называется функция, которая обращается сама к себе

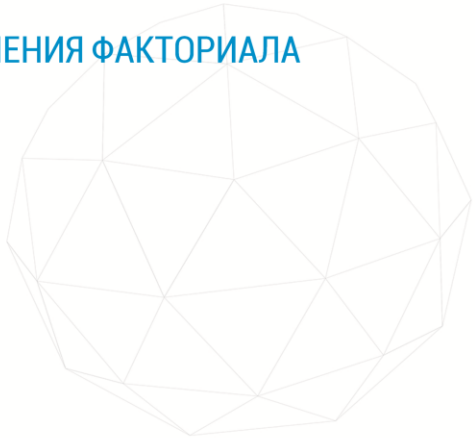
Рекурсивная функция не может вызывать себя до бесконечности, следовательно, важная особенность рекурсивной функции – наличие условия завершения, позволяющее функции прекратить вызывать себя

При выполнении рекурсивной функции, сначала происходит рекурсивное погружение, а затем возврат вычисленных результатов.

Рекурсивную функцию всегда можно преобразовать в не рекурсивную, итеративную, использующую циклы. И эта функция выполняла бы те же вычисления. И наоборот, используя рекурсию, любое вычисление, предполагающее использование циклов, можно реализовать, не прибегая к циклам.

## РЕКУРСИВНАЯ ФУНКЦИЯ ВЫЧИСЛЕНИЯ ФАКТОРИАЛА

```
...
int  fact ( int  n )
{  int  f;
   if  (n= =0)  f=1;
        else    f= n* fact ( n-1 );
   return f;
}
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 161**

«Вообще говоря, всё то, что решается итеративно можно решить рекурсивно, то есть с использованием рекурсивной функции. Всё решение сводится к решению основного или, как ещё его называют, базового случая. Существует такое понятие как шаг рекурсии или рекурсивный вызов. В случае, когда рекурсивная функция вызывается для решения сложной задачи выполняется некоторое количество рекурсивных вызовов или шагов, с целью сведения задачи к более простой.

Одной из сложных к пониманию тем в большом числе случаев оказывается тема рекурсии. Даже шутки есть: чтобы понять рекурсию, надо сначала понять рекурсию.

В программировании рекурсией называют зацикливание функций путём вызовов функций вызовами функций.

Различают прямую рекурсию и косвенную:

Прямая - это прямой вызов из некоторой функции этой же самой функции.

Косвенная - это вызов из первой функции второй функции, когда вторая функция в своём вычислении использует первую функцию.»

На слайде представлена рекурсивная функция вычисления факториала с именем **fact** . В теле функции введена переменная **f** для накапливания

произведения чисел, которой первоначально необходимо присвоить значение единица. Переменная является локальной, время ее существования от начала и до конца работы этой функции.

Функция многократно обращается сама к себе, каждый раз уменьшая значение формального параметра **n** на единицу. На слайде – это выделенный и подчеркнутый фрагмент.

Погружение в функцию продолжается до тех, пока значение параметра **n** не станет равна нулю. После этого происходит многократный выход из рекурсивной функции, и при каждом выходе – подсчитывается произведение.

Функция возвращает вычисленное произведение в вызывающую функцию с помощью оператора **return**.

## РЕКУРСИВНАЯ ФУНКЦИЯ ВЫЧИСЛЕНИЯ ФАКТОРИАЛА

| Погружение | Возврат |
|---|---|
| fact (5) -> 5* fact (4) | 1 |
| 4* fact (3) | 1*2 |
| 3* fact (2) | 2*3 |
| 2* fact (1) | 4*6 |
| 1 | 5*24 ->120 |

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 162**

«Рекурсивные вызовы функций работают точно так же, как и обычные вызовы функций. Однако, программа, приведенная выше, иллюстрирует наиболее важное отличие простых функций от рекурсивных: вы должны указать условие завершения рекурсии, в противном случае — функция будет выполняться бесконечно, фактически до тех пор, пока не закончится память в стеке вызовов.

Условие завершения рекурсии - это условие, которое, при его выполнении, остановит вызов рекурсивной функции самой себя.

Вызов функции влечет за собой некоторые дополнительные накладные расходы, связанные с передачей управления и аргументов в функцию, а также возвратом вычисленного значения. Поэтому итерационная процедура вычисления факториала будет несколько более быстрым решением. Чаще всего итерационные решения работают быстрее рекурсивных.
Любые рекурсивные процедуры и функции, содержащие всего один рекурсивный вызов самих себя, легко заменяются итерационными циклами.

Еще одним недостатком рекурсии является то, что ей может не хватать для работы стека. При каждом рекурсивном вызове в стеке сохраняется адрес возврата и передаваемые аргументы. Если рекурсивных вызовов слишком много, отведенный объем стека может быть превышен.»

Для того чтобы представить изменения параметров при каждом обращении к

рекурсивной функции и с какими значения происходит выход, рассмотрим конкретный пример вычисления факториала числа пять.

На слайде поэтапно рассмотрены значения, с которыми происходит погружение в рекурсивную функцию. Это продемонстрировано в первой колонке.

Функция вычисления факториала имеет один формальный параметр. При каждом повторном обращении к функции, значение фактического параметра уменьшается на единицу. Во второй колонке показан возврат значений рекурсивной функции на каждом шаге итерации. Возвращаемые результаты функции - фактически и есть произведение чисел.

## РЕКУРСИВНАЯ ФУНКЦИЯ

```
include<iostream>
#include<conio.h>
using namespace std;
int  f=0;
int  fun_rec ( float n )
 { if ( n == 1 ) return 1;
     else  if (n > 1 && n < 2)  return 0;
           else  { f++;
                 return fun_rec ( n/2 ); } }
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 163**

«Автор функции решает, что означает её возвращаемое значение. Некоторые функции используют возвращаемые значения в качестве кодов состояния для указания результата выполнения функции, успешно ли выполнение или нет. Другие функции возвращают определенное значение из набора возможных значений. Кроме того, существуют функции, которые вообще ничего не возвращают.»

Рассмотрим задачу: определить, является ли заданное целое число **n** точной степенью двойки. На слайде представлена рекурсивная функция, реализующая задачу. Формальным параметром функции является исходное число.

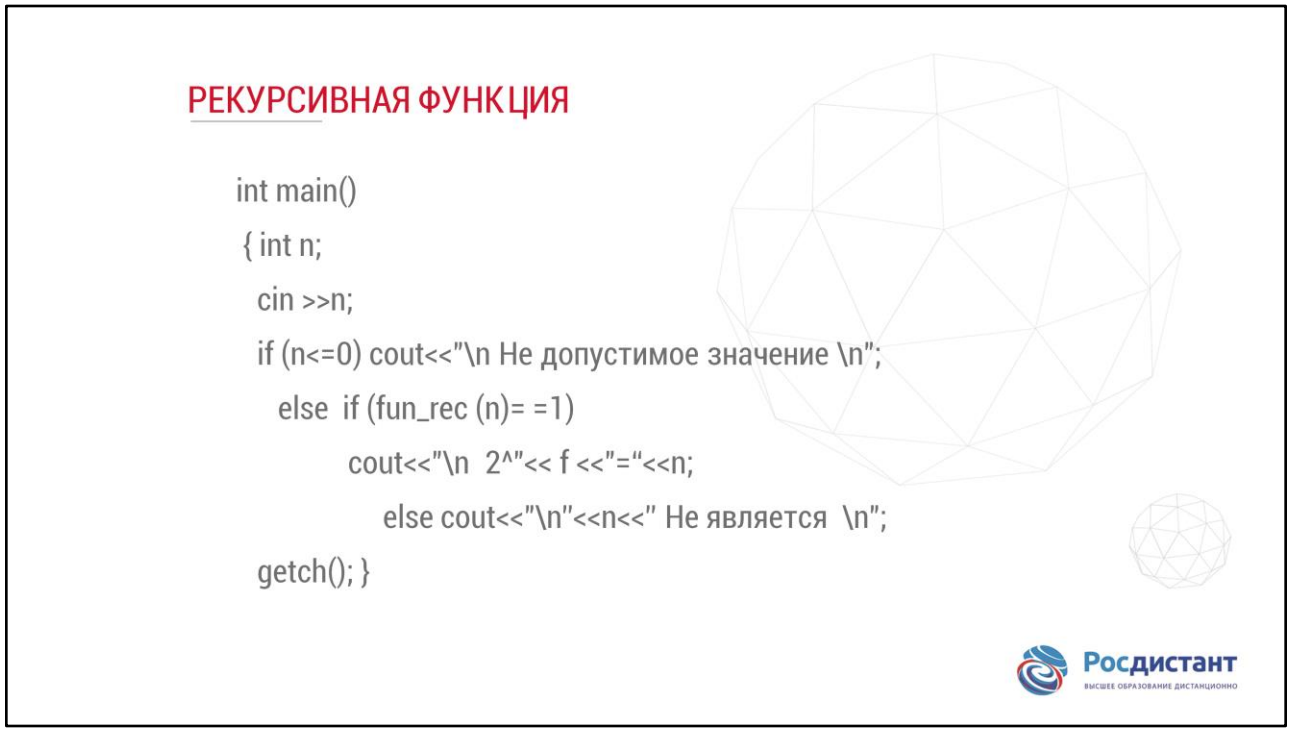
Функция многократно обращается сама к себе, подавая в виде формального параметра число, деленное на два. Этот фрагмент выделен на слайде.

Как известно, в рекурсивных функциях обязательно должно присутствовать условие прекращения рекурсии. В данной задаче процесс так называемого погружения продолжается до тех пор, пока в результате деления числа на два – результат не станет равен единице, значит число является точной степенью двойки. Вторым условием прекращения рекурсии является условие, когда в результате деления числа на два результат - дробное число.

Функция возвращает одно из двух значений: единица – если число является точной степенью двойки, и ноль – в противном случае.

При реализации этого алгоритма, если число является точной степенью двойки, можно определить эту степень. В функции присутствует еще одна переменная **f** , которая подсчитывает количество погружений, фактически определяет степень двойки.

Так как функция может вернуть только один результат, и это признак, является ли число точной степенью двойки, то переменная, определяющая степень двойки описана как глобальная переменная. Соответственно, значение переменной **f** можно будет использовать в главной функции.




**Слайд 164**

В главной функции анализируется значение введенного числа, и если оно – отрицательное число, в результате выполнения программы на печать выдается сообщение, что введенное число не допустимо и вычисления прекращаются.

Если функция не возвращает никакого результата, то обращение к ней можно описать в программе в отдельном выражении, как в отдельным оператором. Данная функция возвращает результат, поэтому обращение к рекурсивной функции происходит в структуре оператора проверки условия **if .**

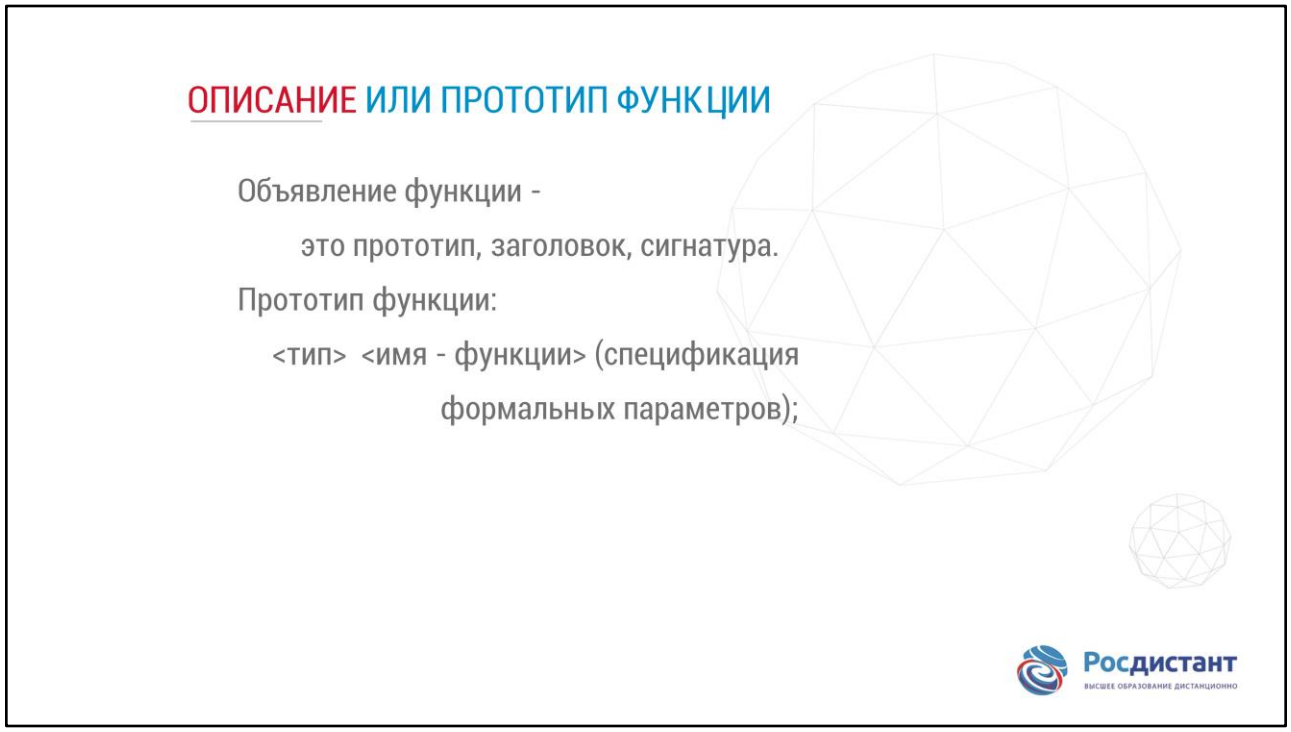
Если введенное число является точной степенью двойки, функция возвращает значение единица, в противном случае – функция возвращает значение ноль.

Если рекурсивная функция возвращает в виде результата единицу, на экране выдается какая это степень двойки.

Для определения степени двойки используется глобальная переменная **f ,** значение которой подсчитывалось в рекурсивной функции. При каждом обращении к рекурсивной функции значение переменной увеличивается на единицу. Переменная **f** описана и определена вне всяких функций.

Если рекурсивная функция вернет значение равное нулю, появится соответствующее сообщение, что введенное число не является точной степенью двойки.

«Глобальные переменные видны во всех функциях, где не описаны локальные переменные с теми же именами, поэтому использовать их для передачи данных между функциями очень легко. Тем не менее это не рекомендуется, поскольку затрудняет отладку программы и препятствует помещению функций в библиотеки общего пользования. Нужно стремиться к тому, чтобы функции были максимально независимы, а их интерфейс полностью определялся прототипом функции.»




**Слайд 165**

Функция должна быть определена до первого обращения к ней, или хотя бы описана.

На слайде показано описание функции, то есть, если функция описана ниже вызывающей, необходимо указать прототип функции. Указывается тип функции, имя функции и список формальных параметров. Обратите внимание, описание функции завершается точкой запятой после круглой скобки. Тело функции отсутствует.

«Поскольку прототип функции является стейтментом, то он также заканчивается точкой с запятой. Определение необходимо для корректной работы компановщика, линкера. Если вы используете идентификатор без его определения, то линкер выдаст вам ошибку.

В языке С++ есть правило одного определения, которое состоит из трех частей:

- внутри файла функция, объект, тип могут иметь только одно определение;
- внутри программы объект или обычная функция могут иметь только одно определение;
- внутри программы типы, шаблоны функций и встроенные функции могут иметь несколько определений, если они идентичны.

Нарушение первой части правила приведет к ошибке компиляции. Нарушение

второй или третьей части правила приведет к ошибке линкинга, компановки.

Объявление – это стейтмент, который сообщает компилятору о существовании идентификатора и о его типе. Объявление - это всё, что необходимо для корректной работы компилятора, но недостаточно для корректной работы линкера. Определение обеспечит корректную работу как компилятора, так и линкера.»

Определение функции, тело функции, может быть указано после вызывающей функции.

Допустима вложенность функций, когда одна функция, не главная, вызывает другую. При построении таких функций придерживаются правил, изложенных выше.

Функция может быть определена и в одном файле с вызывающей функцией, а может и в разных файлах.

Подключение функции к файлу происходит с помощью команды

препроцессора, как и для библиотечных, стандартных функций.

## ОПИСАНИЕ ИЛИ ПРОТОТИП ФУНКЦИИ

Задача: транспонировать квадратную матрицу целых чисел. Использовать прототипы функций для ввода, вывода и транспонирования матрицы

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 166**

Одной из наиболее распространенных проблем, с которой сталкиваются - где, когда и как эффективно использовать функции.

На слайде представлена формулировка задачи, при решении которой будут использованы прототипы функций. Если функция определена ниже вызывающей, необходимо в теле вызывающей функции использовать прототипы функций. На примере следующей задачи подробно рассмотрим, как в программе используются прототипы функций.

Предполагается построение двух функций:

- первая функция реализует вывод двумерного массива в виде матрицы. К этой функции в программе будем обращаться два раза: для вывода на экран исходного массива и для вывода преобразованного массива.
- вторая функция сортирует элементы каждой строки в порядке возрастания.

«В определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать. На имена параметров ограничений по соответствию не накладывается, поскольку функцию можно вызывать с различными аргументами. В прототипах имена компилятором игнорируются. Они служат только для улучшения читаемости программы.»

Формальными параметрами первой функции являются размер массива и имя

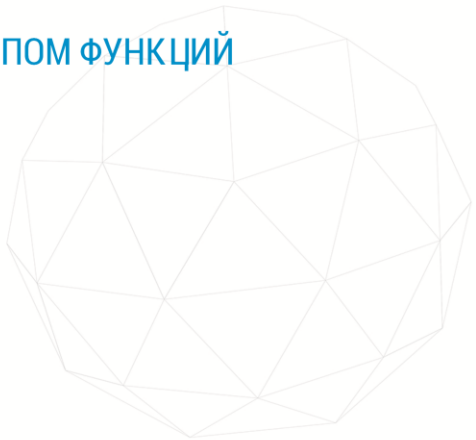
массива.

Формальным параметром второй функции является номер строки, элементы которой необходимо отсортировать.

Обе функции будут определены ниже главной функции, поэтому в теле главной функции необходимо указать прототипы перечисленных выше функций.

## ПРИМЕР ПРОГРАММЫ С ПРОТОТИПОМ ФУНКЦИЙ

```
... int main()
{  void input_mas(int , int a[][5]);
   void print_mas(int , int a[][5]);
   void tran_mas(int , int a[][5]);
   int n, a[5][5];   cin>>n;
   input_mas(n,a);
   cout<<"\n Исходная матрица\n";
   print_mas(n,a);
   tran_mas(n,a);
   cout<<"\n Транспонированная матрица\n";
   print_mas(n,a); getch(); }
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 167**

«Объявление функции предполагает определение таких понятий как прототип, заголовок, сигнатура. При определении функции задается ее имя, тип возвращаемого значения и список передаваемых параметров. Определение функции содержит, кроме объявления, тело функции, представляющее собой последовательность операторов и описаний в фигурных скобках.»

Функции, используемые в программе, описаны ниже обращения к этим функциям. Поэтому в главной функции указаны прототипы трех функций:

- **input_mas** – предназначена для ввода элементов матрицы;
- **print_mas** – используется для вывода на печать массива в виде матрицы;
- **tran_mas** – выполняет транспонирование исходной матрицы.

На слайде представлен код функции **main .** Обратите внимание, описание прототипа завершается точкой с запятой.

В списке формальных параметров прототипов функций размер массива обозначен только типом. При описании прототипа каждой функции указано имя функции и список формальных параметров. Это было необходимо для создания сигнатуры функции. Тип функций, то есть тип возвращаемого результата, указан **void ,** поэтому обращение к этим функциям представлено как отдельный оператор в программе.

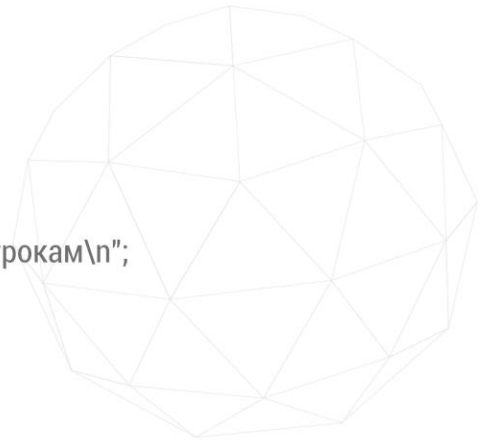
Опишем последовательность действий, выполняемых программой:

- обращение к функции **input_mas** для ввода исходной матрицы;
- функция **print_mas** выведет на печать исходную матрицу;
- функция **tran_mas** транспонирует исходную матрицу;
- повторное обращение к функции **print_mas** выведет на печать транспонированную матрицу.

Функции, к которым обращаются из главной функции, будут описаны ниже главной. Они представлены на следующих слайдах.

## ФУНКЦИЯ ВВОДА МАТРИЦЫ

```
void input_mas(int n, int a[][5])
 {
  cout<<"\nВведите элементы по строкам\n";
   for(int i=0; i<n; i++)
     for(int j=0; j<n; j++)
        cin>>a[i][j];
   return;
 }
```





Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 168**

«Естественным способом борьбы со сложностью любой задачи является ее разбиение на части. В **C++** задача может быть разделена на более простые и обозримые с помощью функций, после чего программу можно рассматривать в более укрупненном виде — на уровне взаимодействия функций. Это важно, поскольку человек способен помнить ограниченное количество фактов. »

На данном слайде описана функция ввода элементов массива по строкам. Ввод осуществляется во вложенных циклах.

«При вызове функции выделяется память под её формальные параметры, и каждому формальному параметру присваивается значение соответствующего фактического параметра. Семантика передачи параметров идентична семантике инициализации. В частности, проверяется соответствие типов формальных и фактических параметров и при необходимости выполняются либо стандартные, либо определённые пользователем преобразования типов. Существуют специальные правила для передачи массивов в качестве параметров.

Тело функции указано в фигурных скобках. При обращении к этой функции из главной, передается размер массива и указатель на первый элемент матрицы, элемент нулевой строки и нулевого столбца

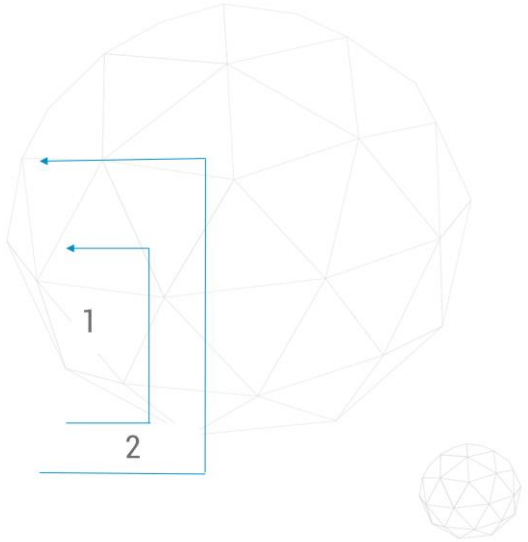
Как известно имя массива - это указатель на начало области, выделенной под хранение элементов массива.

Вспомните, при описании прототипа функции, переменная, определяющая размер массива, не была указана. Определен только тип формального параметра. При обращении к этой функции из главной указывается переменная, определенная как размер матрицы. Так как матрица квадратная, количество строк равно количеству столбцов.

Ввод элементов осуществляется по строкам. Это сообщение предусмотрено в теле функции.

## ФУНКЦИЯ ВЫВОДА МАТРИЦЫ

```
void print_mas(int n, int a[][5])
{
  for(int i=0; i<n; i++)
   { for(int j=0; j<n; j++)
   cout<<a[i][j]<<"\t";
       cout<<"\n";}
 return;
 }
```




Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 169**

«Все величины, описанные внутри функции, а также ее параметры, являются локальными. Областью их действия является функция. При вызове функции, как и при входе в любой блок, в стеке выделяется память под локальные автоматические переменные. Кроме того, в стеке сохраняется содержимое регистров процессора на момент, предшествующий вызову функции, и адрес возврата из функции для того, чтобы при выходе из нее можно было продолжить выполнение вызывающей функции. При выходе из функции соответствующий участок стека освобождается, поэтому значения локальных переменных между вызовами одной и той же функции не сохраняются.

Напомним, в определении, в объявлении и при вызове одной и той же функции типы и порядок следования параметров должны совпадать. На имена параметров ограничений по соответствию не накладывается, поскольку функцию можно вызывать с различными аргументами, а в прототипах имена компилятором игнорируются.»

На слайде продемонстрирована функция вывода на экран двумерного массива в виде матрицы. К этой функции мы обращаемся дважды: для вывода на печать исходной матрицы и транспонированной матрицы.

Вывод массива в виде матрицы осуществлен во вложенных циклах.

Структура вложенных циклов схематично показана на слайде.

Внутренний цикл выводит на печать элементы одной отдельно взятой строки. На слайде обозначен цифрой один.

Внешний цикл осуществляет переход от строки к строке матрицы, на слайде обозначен цифрой два. В конце каждой строки предусмотрен перевод курсора на начало следующей строки с помощью управляющего символа **\n**.

При определении функции указаны формальные параметры – размер массива и имя массива.

Обратите внимание, функция не возвращает результата, поэтому тип функции при описании указан с помощью служебного слова **void.**

## ФУНКЦИЯ ТРАНСПОНИРОВАНИЯ МАТРИЦЫ

```
void tran_mas(int n, int a[][5])
{
 for(int i=0; i<n; i++)
     for(int j=i; j<n; j++)
         swap(a[i][j],a[j][i]);
 return;
}
```



Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

**Слайд 170**

На слайде представлена функция транспонирования квадратной матрицы. Ранее мы рассматривали алгоритм транспонирования матрицы. Элементы верхнего треугольника меняются местами с элементами нижнего треугольника, ограниченных главной диагональю.

Обратите внимание на закон изменения параметра внутреннего цикла **j.** Именно это выражение позволяет транспонировать матрицу. Этот фрагмент подчеркнут на слайде.

Рассмотренная задача продемонстрировала возможность описывать функции ниже вызывающей, при этом нельзя забывать о прототипе функции.

Использование функций упрощает понимание кода. Фактически перечислена последовательность действий.

«Использование функций является первым шагом к повышению степени абстракции программы и ведет к упрощению ее структуры. Разделение программы на функции позволяет также избежать избыточности кода, поскольку функцию записывают один раз, а вызывать ее на выполнение можно многократно из разных точек программы. Процесс отладки программы, содержащей функции, можно лучше структурировать. Часто используемые функции можно помещать в библиотеки. Таким образом создаются более простые в отладке и сопровождении программы.

Следующим шагом в повышении уровня абстракции программы является группировка функций и связанных с ними данных в отдельные файлы, компилируемые раздельно. Получившиеся в результате компиляции объектные модули объединяются в исполняемую программу с помощью компоновщика. Разбиение на модули уменьшает время перекомпиляции и облегчает процесс отладки, позволяя отлаживать программу по частям. Это уменьшает общий объем информации, которую необходимо одновременно помнить при отладке. Разделение программы на максимально обособленные части является сложной задачей, которая должна решаться на этапе проектирования программы.»

На этом мы завершаем изучение темы - Функции.

---

Введение в языки программирования C и C++ | Уроки C++ - Ravesli / https://ravesli.com/urok-2-vvedenie-v-yazyki-programmirovaniya-c-i-s/

Т, А. Павловская С/С++ Программирование на языке высокого уровня/http://cph.phys.spbu.ru/documents/First/books/7.pdf

Введение в программирование | Уроки C++ - Ravesli/https://ravesli.com/urok-1-vvedenie-v-programmirovanie/

Технология программирования - Информатика, автоматизированные информационные технологии и системы/ https://studref.com/441961/informatika/tehnologiya_programmirovaniya

Бьерн Страуструп. Язык программирования С++ 11/ https://vk.com/doc-145125017_450056359

Язык СИ++ Учебное собие / http://5fan.ru/wievjob.php?id=4301