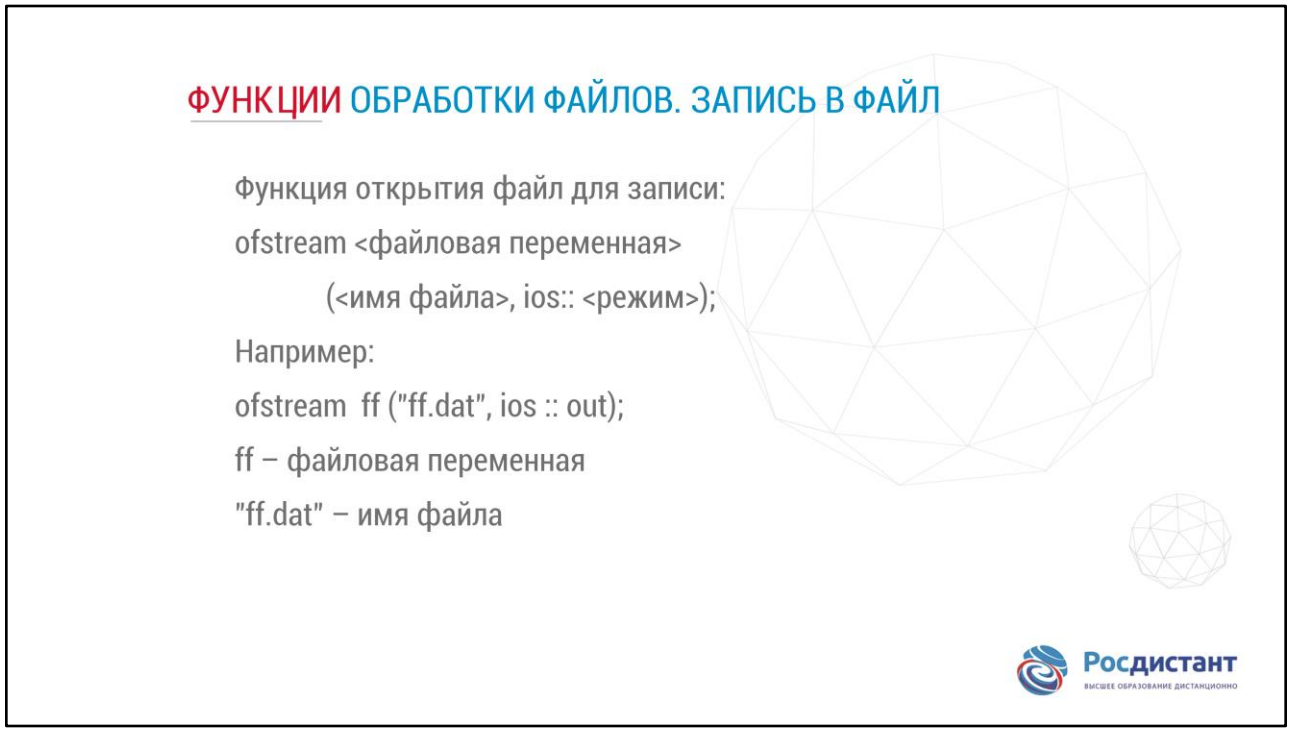

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО
ОСНОВЫ
ПРОГРАММИРОВАНИЯ
9 ЧАСТЬ



Слайд 229

Тема 9. Файл данных

«Большинство компьютерных программ работают с файлами, и поэтому возникает необходимость создавать, удалять, записывать, читать, открывать файлы. Хранение данных через переменные или массивы является временным, до конца работы программы. Для долговременного хранения предназначены файлы, размещенные на внешних носителях.

Файл – именованный набор байтов, который может быть сохранен на некотором накопителе. Под файлом понимается некоторая последовательность байтов, которая имеет своё, уникальное имя, В одной директории не могут находиться файлы с одинаковыми именами. Под именем файла понимается не только его название, но и расширение.»

Логически файл можно представить как цепочку байт, имеющих начало и конец. Работа с файлом предполагает подключение заголовочного файла **fstream** из стандартной библиотеки**.**

Для записи в файл, его необходимо открыть. Функция **ofstream** открывает файл для записи.

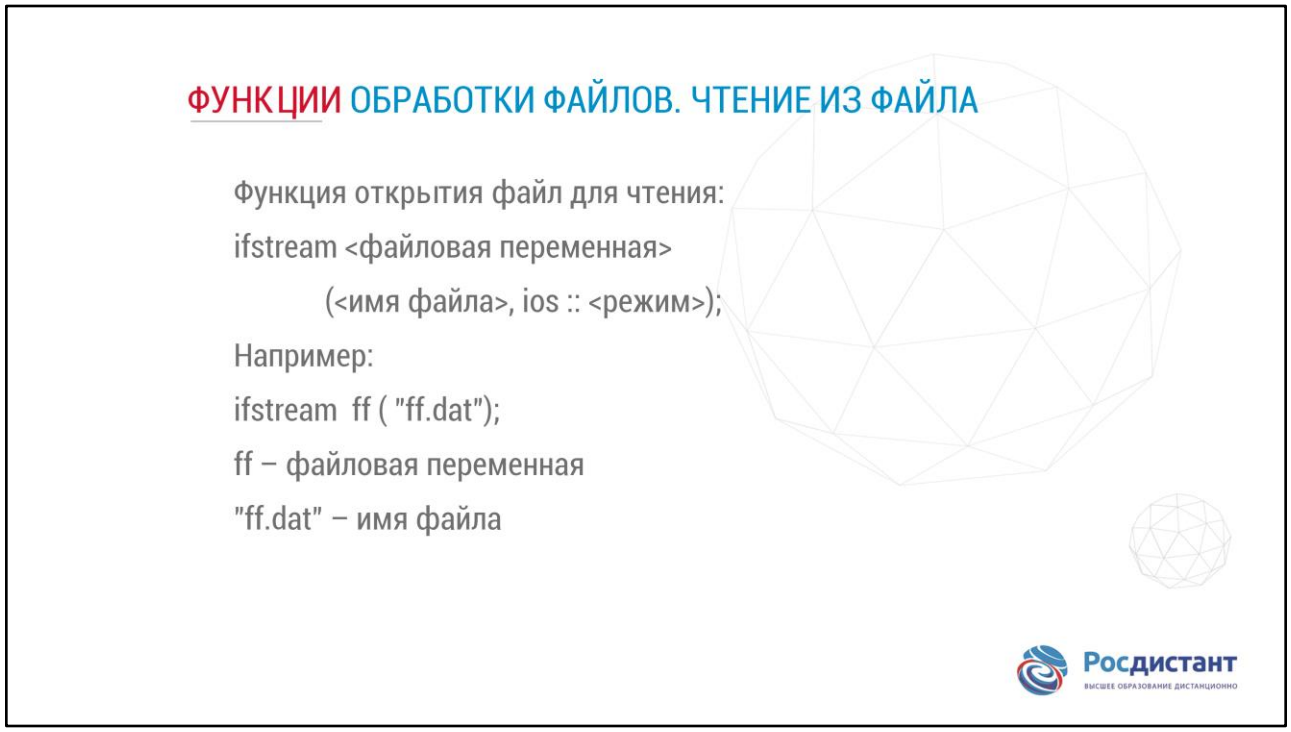
При обращении к файлу используют файловую переменную. Файловая переменная – это уникальный идентификатор, строится по все правилам построения идентификаторов на языке **C++** .

Указывается также имя файла и режим доступа к файлу. Синтаксис языка требует заключить имя файла в двойные кавычки. Если файл не находится в одной папке с программным кодом, то прописывают путь доступа к файлу.

Запись в файл определяется положением внутреннего указателя. Положение указателя меняется либо автоматически, либо за счет явного управления его положением. Положение внутреннего указателя определяется режимом открытия файла. При каждом обращении к файлу, то есть записи в файл, внутренний указатель автоматически перемещается в конец очередной порции информации, записанной в фай.

Файловая переменная, связанная с именем файла, фактически идентифицирует файл. Далее в программе имя файла не используется. Обращение к файлу осуществляется через файловую переменную. При повторном обращении к файлу, например, чтению из файла, в пределах одной программы нельзя использовать одну и ту же файловую переменную.

На слайде приведен пример, демонстрирующий открытие файла для записи.



Слайд 230

Работа с файлами предполагает операции:

- Создание файла;
- Запись в файл;
- Чтение из файла;
- Добавление информации, или как говорят дозапись, в уже существующий файл.

Рассмотрим процедуру чтения из файла. Прежде всего необходимо открыть файл для чтения. На слайде представлены процедуры открытия файла для чтения. С помощью функции **ifstream** файловую переменную связывают с именем файла.

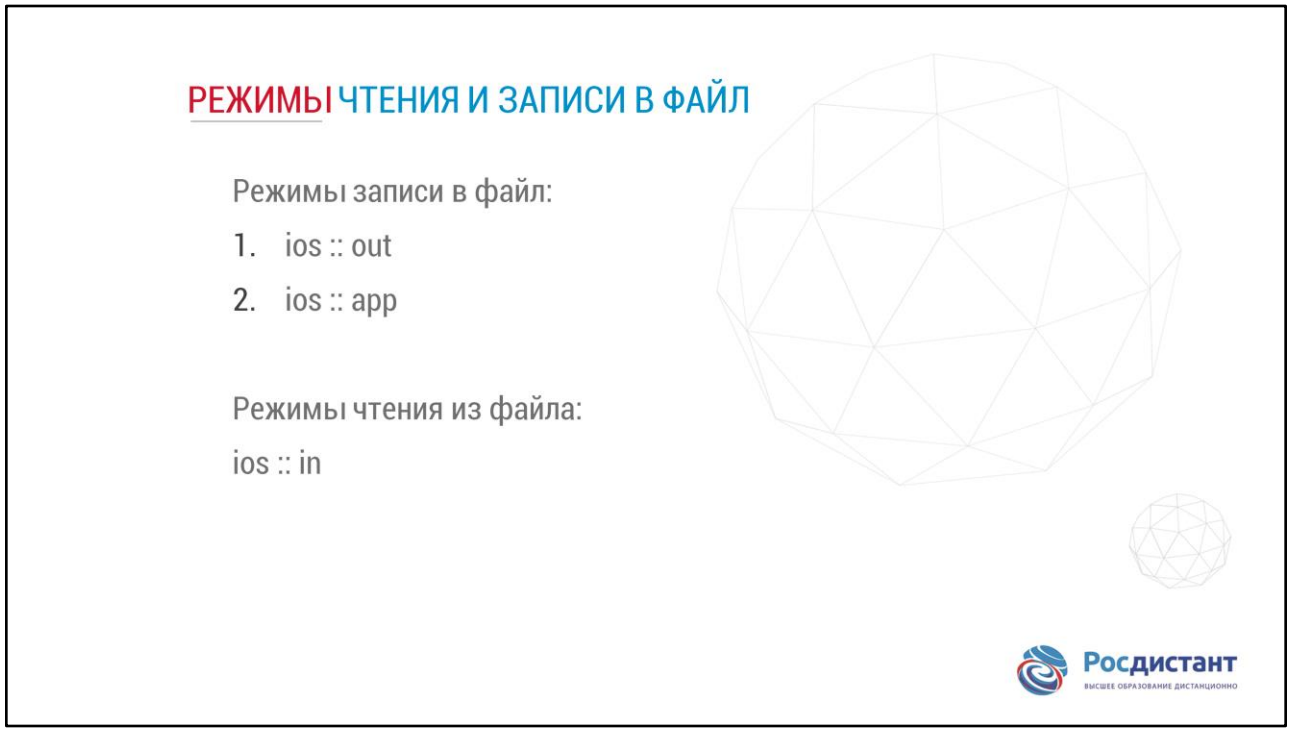
Рассмотрим функцию, которая открывает файл для чтения. Указывается файловая переменная, которая в дальнейшем идентифицирует файл в функциях обработки.

Как уже говорилось ранее, в файле есть внутренний указатель. Чтение из файла определяется положением внутреннего указателя, который может меняться либо автоматически, либо за счет явного управления его положением. При каждом чтении из файла, внутренний указатель перемещается на начало следующей компоненты файла.

Компонента файла – это та порция информации, которая была записана в файл при его создании. В файле можно хранить любую информацию. Файлы позволяют хранить информацию разного типа, объединяя ее в структуры, то есть создавать файлы структур. В следующих разделах мы подробно рассмотрим файлы структур.

Положение внутреннего указателя в файле зависит от режимов открытия файла. Если режим не указан, как в примере на слайде, то при открытии файла для чтения, внутренний указатель автоматически устанавливается в начало файла.

Использование функции открытия файла для чтения также предполагает подключение заголовочного файла **fstream .**



Слайд 231

При работе с файлами, может возникнуть ситуация, когда в пределах одной программы необходимо и записывать информацию в файл и читать информацию из файла. Подробно остановимся на режимах открытия файла для записи и чтения в пределах одной программы.

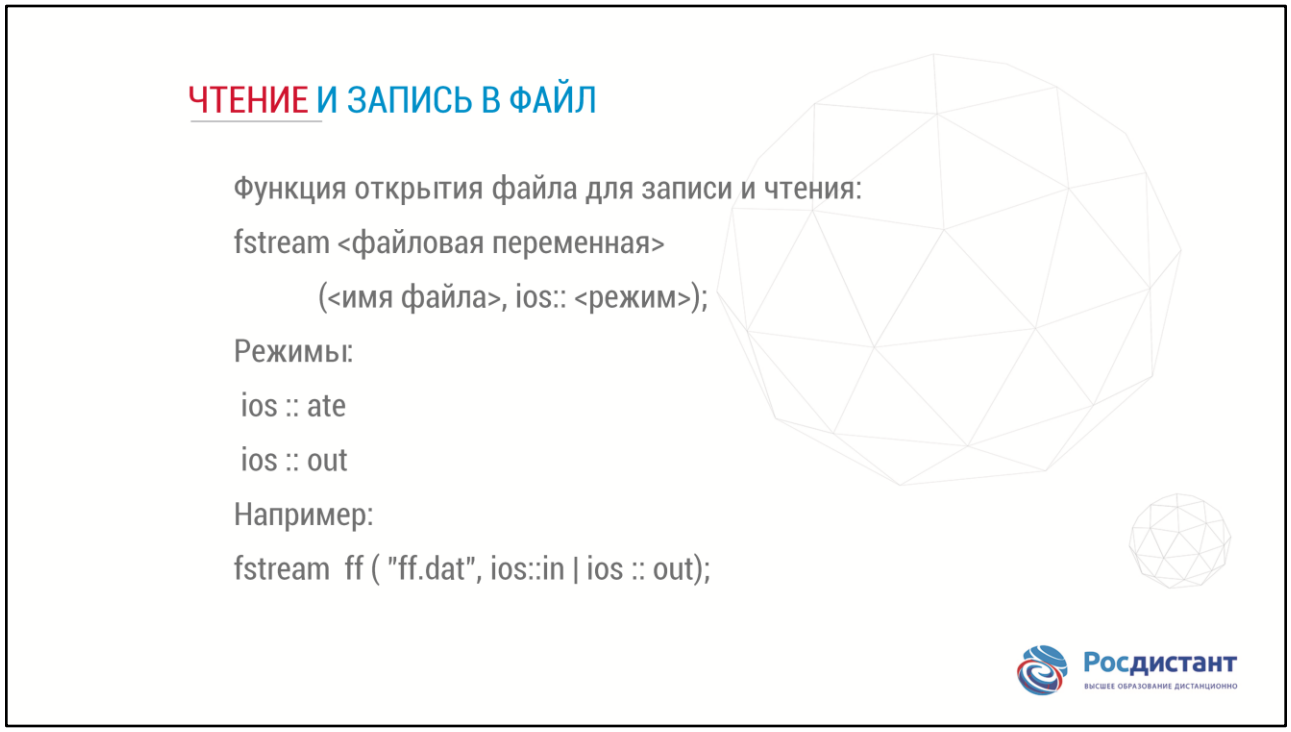
Рассмотрим два режима записи в файл.

Первый режим **ate** используется для создания нового файла, указатель устанавливается в начало файла. Если указать этот режим для уже существующего файла, можно потерять информацию, хранящуюся в файле.

Указатель установится в начало файла и при записи, старая информация, как говорится, затирается.

Второй режим **out** используется для добавления информации, дозаписи, в уже существующий файл, внутренний указатель устанавливается в конец файла.

Режим чтения из файла **in** устанавливается внутренний указатель в начало файла. Этот режим можно опустить, не указывать, он устанавливается автоматически.



Слайд 232

«Когда вы создаёте файловую переменную, связываемую с файлом по имени или полному пути файла, вы задаёте этой переменной тип, который может быть или **ifstream ,** или **ofstream ,** или **fstream .** Вот этот тип называют файловым потоком. В каком режиме окажется файл, если режим не указан явно, зависит от выбранного типа. Если тип чтение файла, то по умолчанию файл будет открываться для чтения, если тип запись в файл, то по умолчанию файл будет открываться для записи в файл. Есть ещё режим, который получается при комбинации двух режимов, режим чтения и записи в файл, в таком режиме файл по умолчанию будет открыт и для чтения, и для записи сразу.»

Для работы с файлами, в программе прежде всего надо открыть файл. На слайде продемонстрирована функция **fstream ,** позволяющая открыть файл для записи и чтения.

В функции открытия файла можно указать режимы работы.

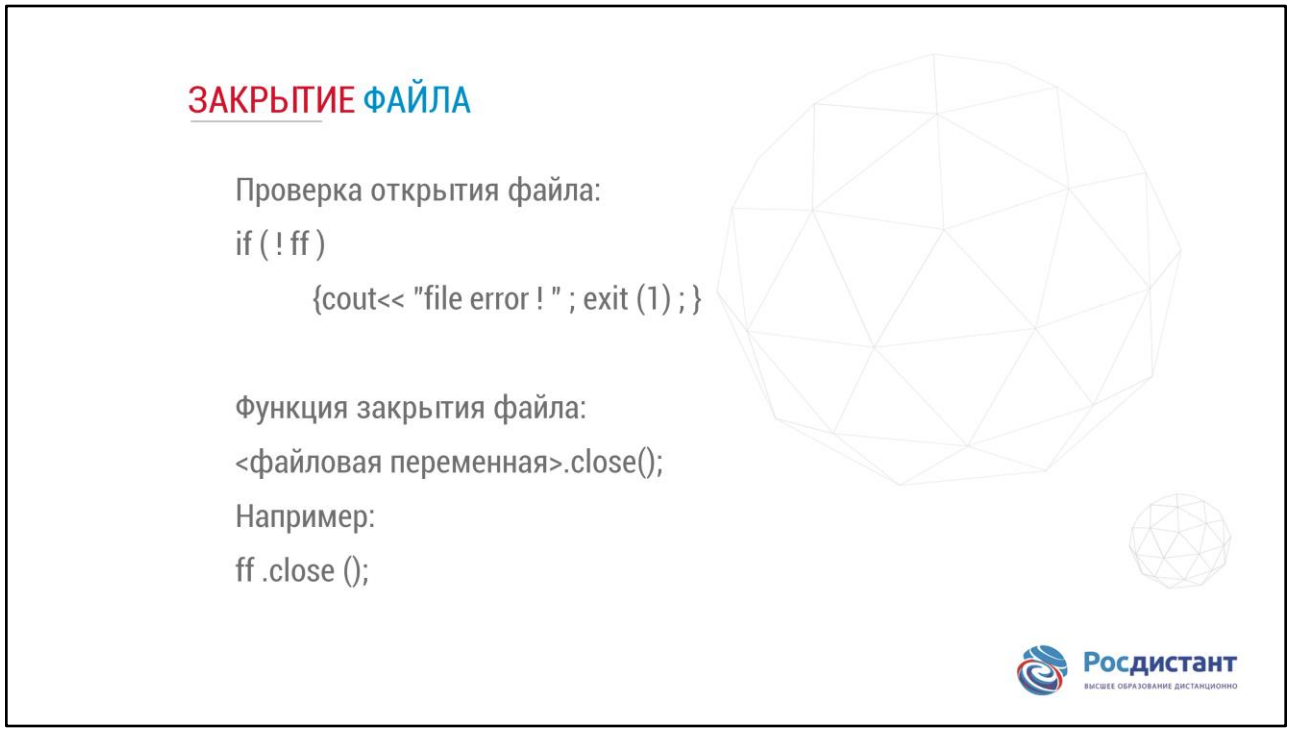
Первый режим **ate** - запись в файл, указатель автоматически устанавливается в конец файла.

Второй режим **out** - указатель устанавливается в начало файла.

Использование функции открытия файла для чтения и записи также предполагает подключение заголовочного файла **fstream .**

На слайде приведен пример открытия файла для записи и чтения. Несколько режимов можно объединять между собой. Объединение режимов достигается путём использования операции дизъюнкция, логическое ИЛИ. Знак этой операции ставится между режимами.

Где **ff** – файловая переменная, через которую осуществляется доступ к файлу, то есть запись в файл или чтение из файла. Файловая переменная связана с именем конкретного файла. При указании имени файла, можно также указать путь доступа к файлу, если файл не находится в текущей папке.



Слайд 233

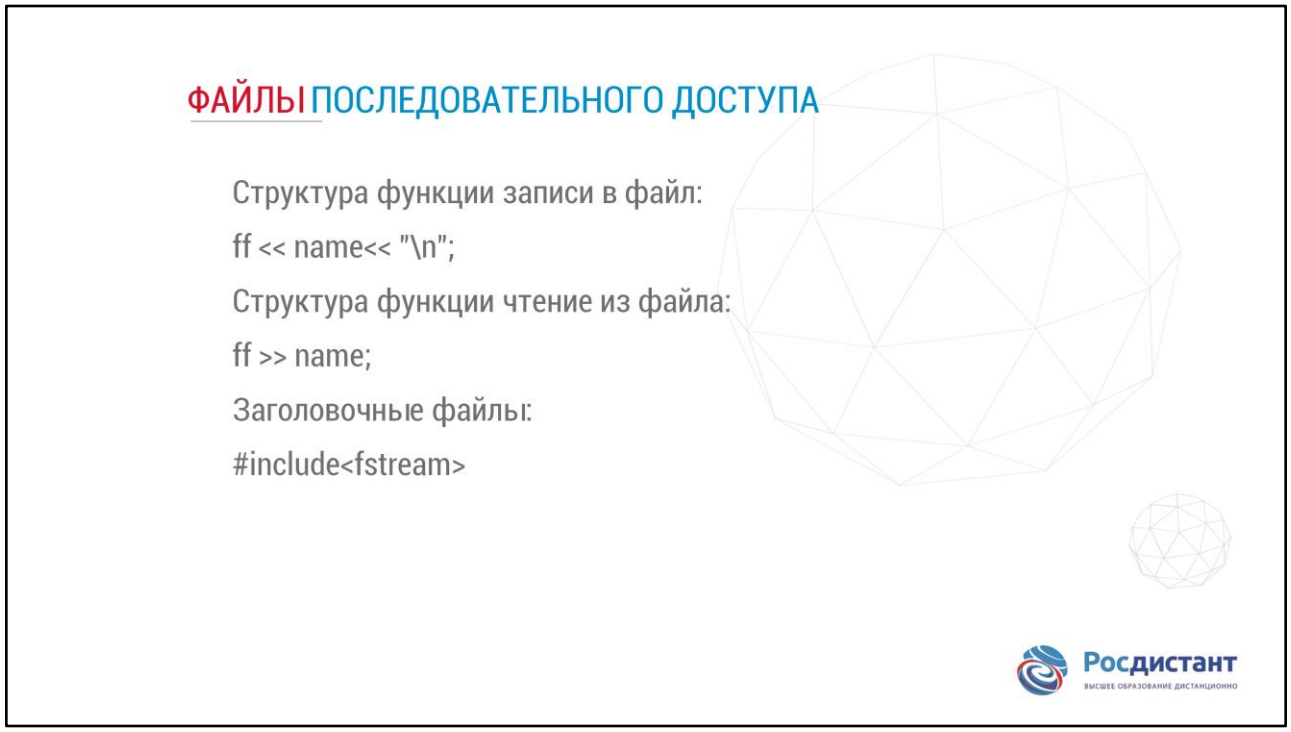
Работа с файлами предполагает проверку на существование файла как при записи в файл, так и при чтении из файла. На слайде приведен пример, демонстрирующий проверку на существование файла, где в проверке указывается файловая переменная, которая была связана с именем переменной при открытии. Используя операции открытия файла желательно осуществлять проверку на существование файла.

В логическом операторе **if** анализируется файловая переменная **ff .** Логическое выражение может принимать одно из двух значений: **true** или **false .**

Если файл существует, то есть открытие файла прошло удачно, значение указанного условия будет истинным. Если файл не существует, или не найден по указанному пути доступа к файлу, указанное логическое выражение выработает значение ложь. На экран выведется сообщение об ошибке, и функция **exit** прекратит выполнение программы.

Используя файлы данных, во избежание потери информации, перед завершением работы программы, желательно закрыть файл.

На слайде представлена функция **close ,** обеспечивающая закрытие файла. При обращении к этой функции указывается файловая переменная, связанная с именем файла при открытии. Необходимо указывать пустые круглые скобки рядом с именем функции, этого требует систаксис языка **C++**.



Слайд 234

По способу доступа к информации файлы делятся на файлы последовательного доступа и файлы произвольного доступа.

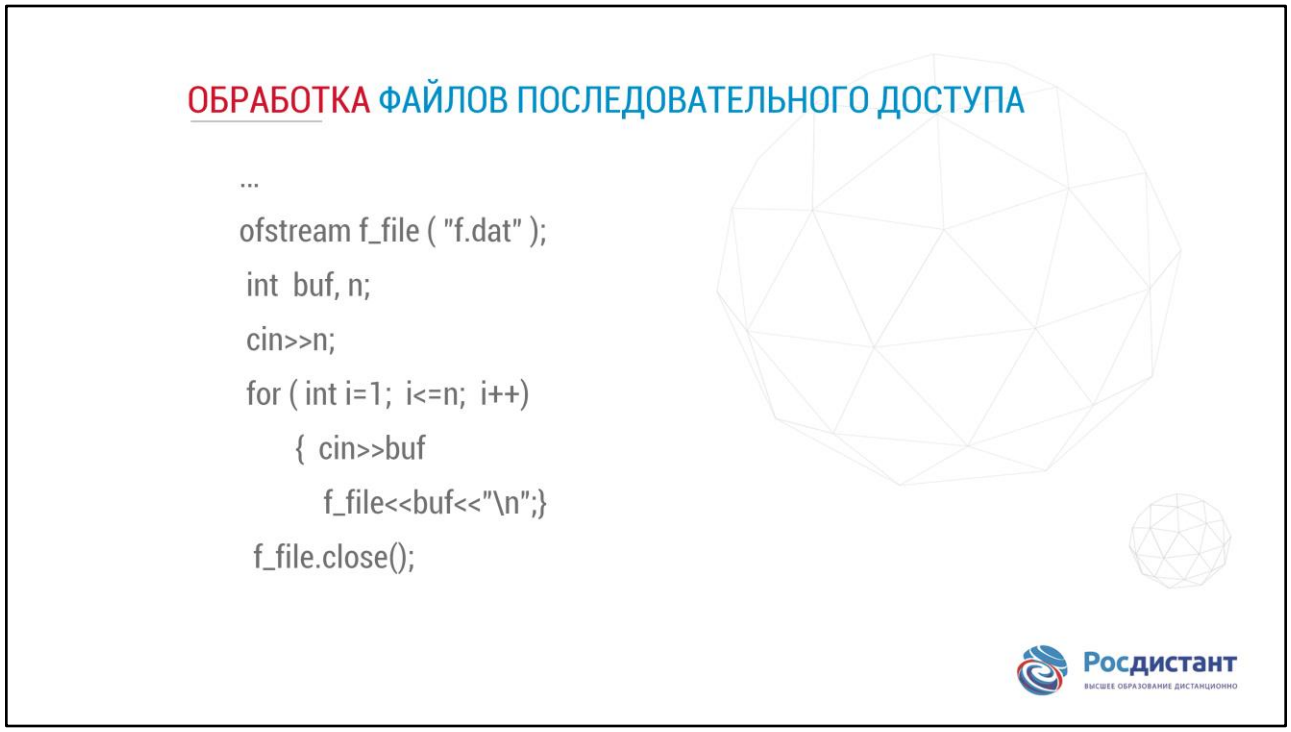
Файлы последовательного доступа – это файлы, хранящие информацию в неструктурированном виде. Поиск в таких файлах осуществляется последовательным считыванием информации из файла. Так же и обращение к определённому участку файла каждый раз требует выполнить действие - чтения с начала файла. Файл – это цепочка логически связанных байт. Как в свое время информация была записана в файл, также она и будет считываться из файла.

На слайде представлены процедуры записи и чтения одой компоненты в файлах последовательного доступа.

И запись, и чтение осуществляется через некоторую переменную **name .** Тип этой переменной, то есть размер компонент файла, определяет количество байт, которые записываются в файл или считываются из файла.

При записи информации в файл, после записи каждой компоненты, вносится константа **\n** . Это обеспечивает порцию считывания из файла. При каждом обращении к файлу считывается одна компонента файла.

Обратите внимание, запись в файл и чтение из файла аналогично организации потоков ввода-вывода информации. Работу этих функций обеспечивает стандартная библиотека **fstream .**



Слайд 235

На слайде представлен фрагмент создания файла последовательного доступа, содержащего целые числа.

Открыт файл для записи, указана файловая переменная **f_file** и имя файла. Имя файла, путь доступа к файлу, указывается в двойных кавычках. Далее в программе обращение к файлу будет происходить только через файловую переменную.

Для записи в файл выделена переменная **buf** - целого типа. Тип этой переменной определяет величину каждой компоненты файла, то есть порции записи в файл в байтах.

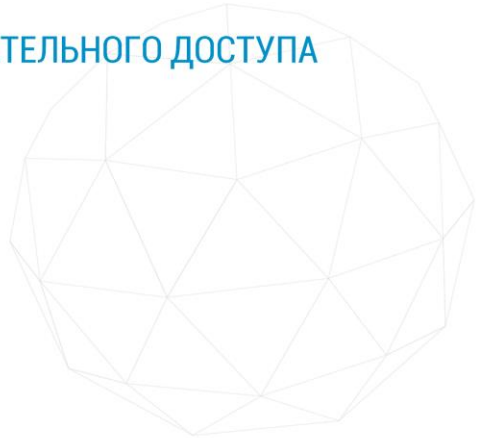
Переменная **n** определяет количество чисел, которые будут введены с клавиатуры и через буфер записываются в файл. Ввод осуществляется в цикле. Каждое введенное число, на каждом шаге итерации цикла, записывается в файл.

При завершении ввода чисел и записи в файл, внутренний указатель находится в конце файла. Исходный файл необходимо закрыть. При повторном открытии файла для чтения, указатель автоматически перейдет на начало файла.

Сформировать из созданного файла целых чисел два файла, один с четными числами исходного файла, другой файл - с нечетными числами. На следующем слайде приведен фрагмент программы решения этой задачи.

ОБРАБОТКА ФАЙЛОВ ПОСЛЕДОВАТЕЛЬНОГО ДОСТУПА

```
ifstream  ff_file ( "f.dat");
ofstream  fc_file("fc.dat");
ofstream  fn_file("fn.dat");
while ( ff_file>>x )
{ if  (buf %2= =0)
            fc_file<<x<<"\n" ;
     else  fn_file<<x<<"\n" ; }
ff_file.close();  fc_file.close();  fn_file.close();
```



Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 236

В представленном на слайде фрагменте открыты три файла:

- **ff_file** – для чтения из файла исходных чисел;
- **fc_file** – для создания файла четных чисел;
- **fn_file** – для создания файла нечетных чисел.

Обратите внимание, для открытия исходного файла используется новая файловая переменная. На предыдущем слайде, при открытии файла для записи, использовалась файловая переменная **f_file** .

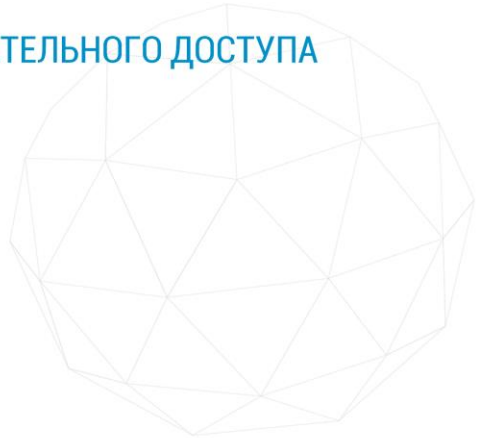
Чтение из исходного файла реализована в структуре оператора цикла с предусловием, Условие в операторе **while ,** условием продолжения цикла, и есть процедура чтения из файла. Это условие трактуется, как - пока читается из файла.

Каждое число, прочитанное из исходного файла, анализируется на четность и записывается либо в файл четных чисел, либо в файл нечетных чисел.

Все три файла необходимо закрыть. При закрытии каждого файла указывается своя файловая переменная.

ОБРАБОТКА ФАЙЛОВ ПОСЛЕДОВАТЕЛЬНОГО ДОСТУПА

```
ifstream ffc_file("fc.dat");
ifstream ffn_file("fn.dat");
cout<<"\nЧетные числа \n" ;
while (ffc_file>>buf)
      cout<< "\n "<<buf;
cout<<"\nНечетные числа\n" ;
while (ffn_file>>buf)
{ cout<< "\n "<<buf; }
 ffc_file.close();  ffn_file.close();
```



Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 237

Не забывайте о заголовочном файле, который необходимо подключить для реализации процедур обработки файлов.

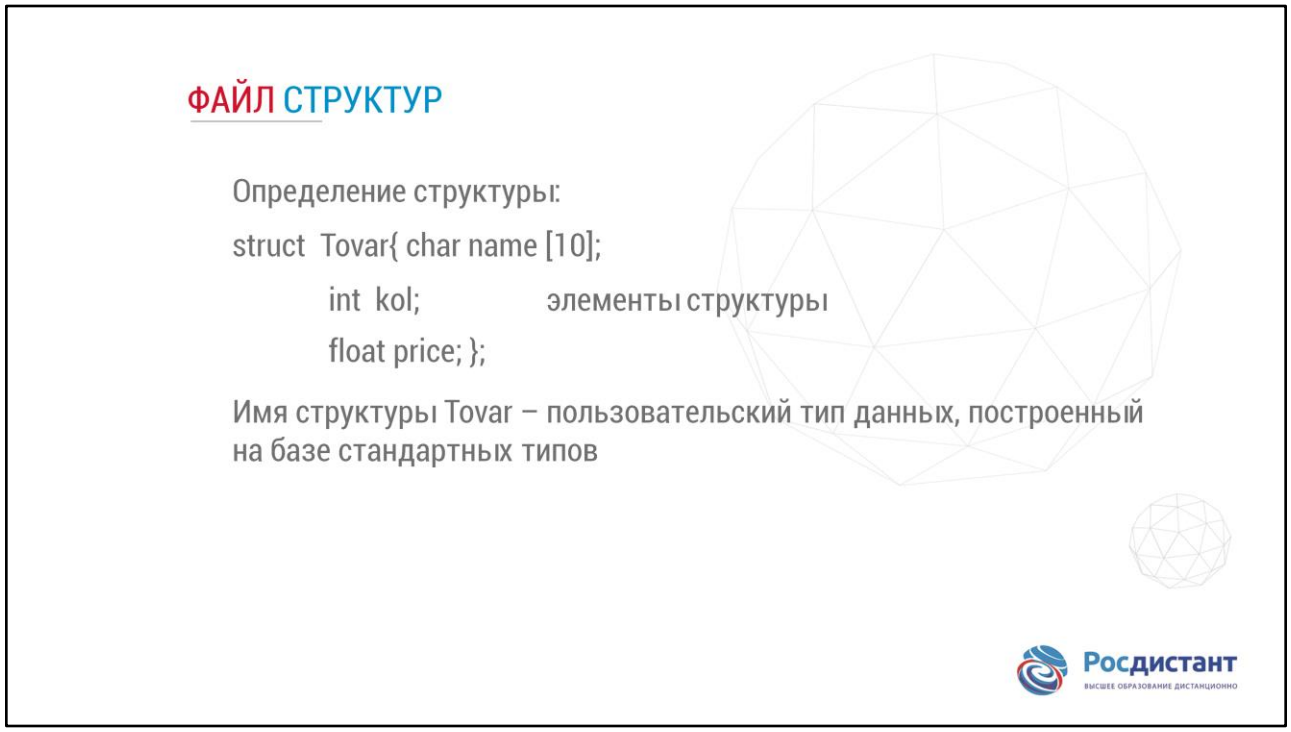
Заранее не известно, сколько чисел попало в файл четных чисел, и сколько в файл нечетных чисел. Поэтому для чтения информации из файлов используется оператор цикла с предусловием. В файлах при создании автоматически формируется признак конца файла. Цикл продолжается до тех пор, пока читается из файла, пока не встретился признак конца файла.

В цикле с предусловием, читается очередная компонента файла и выводится на экран. После каждой процедуры чтения из файла, внутренний указатель переносится в начало следующей компоненты файла.

Процедура последовательно выполняется сначала для файла четных чисел, а затем для файла нечетных чисел. Выводу на печать содержимого созданных файлов, предшествует сообщение на экран о содержимом файла.

Каждая компонента файла считывается в переменную с именем **buf** и выводится на печать.

Функция чтения из файла вставлена как условие продолжения цикла в операторе **while .**



Слайд 238

Далее рассмотрим работу с файлами, каждая компонента которых является структурой, содержащей информацию разного типа. Файлы структур могут быть как последовательного, так и произвольного доступа. Рассмотрим создание файлов структур последовательного доступа. На слайде приведен пример описания структуры, то есть описание каждой компоненты файла. Размер каждой компоненты зависит от элементов описанной структуры.

Рассмотрим задачу: создать файл последовательного доступа, содержащий информацию о товарах некоторого склада.

Информация о каждом товаре содержит:

- наименование товара;
- количество товаров каждого наименования;
- цена единицы товара.

Требуется распечатать информацию из созданного файла и подсчитать общую стоимость товаров на складе.

Определен пользовательский тип данных с именем **Tovar .**

Для записи и чтения из файла, необходимо ввести переменную-буфер, размер которой, то есть количество байт, определено описанной структурой.

СОЗДАНИЕ ФАЙЛА СТРУКТУР

```
#include<fstream>
#include<iostream>
#include<conio.h>
# define FILE "f.dat"
using namespace std;
int main()
{ struct  Tovar { char name[10];
                  int  kol;
                  float price; };
Ttovar buf;
```


Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 239

На слайде представлена программа создания файла последовательного доступа, каждая компонента которого имеет описанную структуру. Рассмотрим первый фрагмент программа решения этой задачи.

Подключены заголовочные файлы, обеспечивающие работу функций обработки файлов.

В данной программе используется препроцессорная команда **define ,** которая задаёт символическую константу. Символическая константа связывается с именем файла. При необходимости можно указать путь доступа к файлу. Имя файла задается в двойных кавычках. Далее в программе эта символическая константа используется вместо имени файла.

Описана структура, содержащая:

символьный массив под хранение наименования товара;

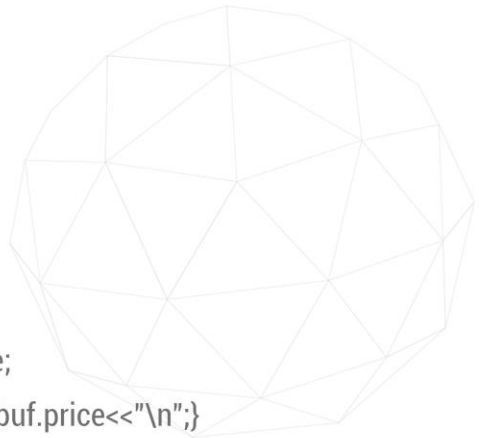
цена товара, переменная целого типа;

количество товаров, переменная вещественного типа.

Создан пользовательский тип – **Tovar .**

СОЗДАНИЕ ФАЙЛА СТРУКТУР

```
float sum=0; int n;
ofstream f (FILE);    // для записи
cin>>n;
for ( int i=1; i<=n; i++)
{ cin>>buf.name>>buf.kol>>buf>>price;
 f<<buf.name<<"\n"<<buf.kol<<"\n"<<buf.price<<"\n";}
 f.close();
```



Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 240

Для обращения к элементам структуры, введена переменная **buf .**

С помощью функции **ofstream** открывается файл последовательного доступа для записи.

С клавиатуры вводится количество наименований товаров. В цикле вводится информация о каждом наименовании товаров. Обращение к отдельным элементам структуры осуществляется с помощью переменной буфер.

Каждая единица информации записывается в файл, и завершается, или как говорят, подпирается символом **\n .**

Это позволит в дальнейшем, при чтении из файла, получить информацию согласно описанной выше структуре: наименование, цену, количество. При чтении из файла необходимо соблюдать этот же порядок. Как говорилось выше, файл - это последовательность байт. Если при чтении из файла нарушить последовательность, информация считается из файла, но будет не читабельна.

Обратите внимание, как происходит обращение к отдельным элементам структуры – через имя переменной, имеющей тип – структура, и имя конкретного поля. Между ними ставится точка.

По окончании процедуры записи в файл, внутренний указатель находится в конце файла.

Закрываем файл, чтобы открыть его для чтения.

СОЗДАНИЕ ФАЙЛА СТРУКТУР

```
ifstream ff (FILE);  // для чтения
 for ( int  i=1;  i<=n;  i++)
 { ff>>buf.name>>buf.kol>>buf.price;
    sum+=buf.kol * buf.price;}
 cout<<"\n  sum= "<<sum;
 ff.close();
 getch();}
```


Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 241

Следующий фрагмент программы демонстрирует чтение из файла и подсчет общей стоимости товаров на складе. Для подсчета общей стоимости товаров определена переменная **s .** С помощью функции **ifstream о**ткрывается файл для чтения. Используется новая файловая переменная **ff ,** для обращения к файлу. Вместо имени файла используется символьная константа.

При открытии файла для чтения внутренний указатель устанавливается автоматически в начало файла.

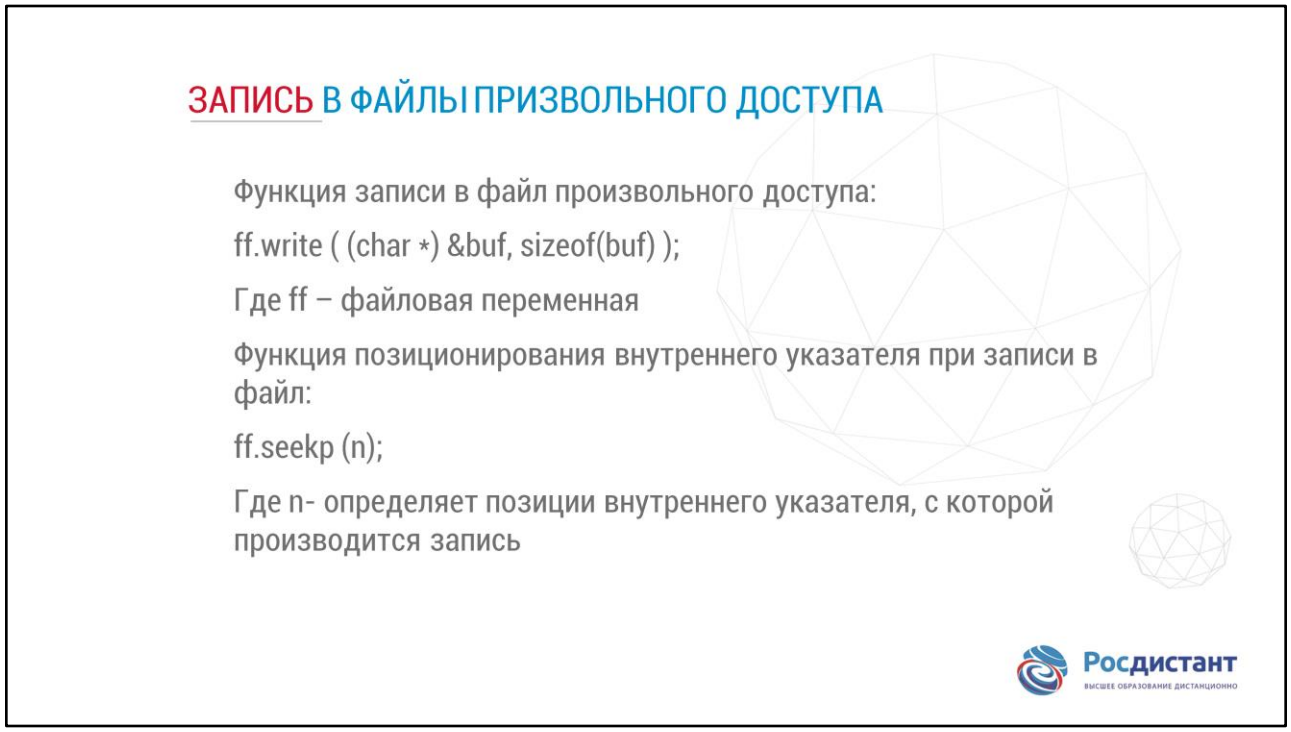
Чтение информации из файла выполняется в цикле, количество циклов определено введенным значением переменной **n .** При каждом обращению к файлу считывается единица информации из файла. В файлах последовательного доступа нельзя считать сразу всю структуру, всю информацию об одном наименовании товара. Надо помнить, как в свое время был создан файл, то есть записывалась информацию в файл, в такой же последовательности надо считывать информацию из файла. И если не соблюдать перечисленных выше правил, информация будет считана, но не адекватно внесенной в файл информации.

Для подсчета общей стоимости товаров на складе, определена вещественная переменная, которой было присвоено значение ноль в начале программы.

В конце программы необходимо закрыть файл с помощью функции **close .**

Посчитанная общая стоимость товаров на складе выводится на печать.

Если не указан путь доступа к файлу при его создании, файл будет создан в текущей папке, где и код программы.



Слайд 242

Перейдем к рассмотрению файлов произвольного доступа. Файлы последовательного доступа выигрывают у файлов с произвольным доступом по компактности, но проигрывают по скорости доступа к информации.

Еще раз напомним, что файлы последовательного доступа – это последовательность не структурированных байт. Файлы с последовательным доступом читаются от начала к концу, поэтому невозможно одновременно и считывать из них данные, и записывать таковые. Чтобы изменить одну запись файла последовательного доступа, его нужно весь записать заново.

Если требуется частый доступ к данным, хранящимся в некотором файле, следует использовать файлы с произвольным доступом.

Файлы с произвольным доступом - файлы, хранящие информацию в структурированном виде. Файлы произвольного доступа также как и файлы последовательного доступа, должны быть открыты для или записи, или для чтения, или для записи и чтения одновременно.

Запись в файл произвольного доступа реализует функция - **write** , а позиционирование - функция **seekp ,** где слово **put** означает - положить.

Работа этих функций означает - установить внутренний указатель на **n** байт от начала файла и записать значение переменной **buf** с размером **sizeof** байт, начиная с этой позиции вперед.

Где, **ff** – файловая переменная, которая связана с именем конкретного файла.

ФАЙЛЫ ПРОИЗВОЛЬНОГО ДОСТУПА

Функции управления внутренним указателем:

Функция	Назначение
seekg	устанавливает внутренний указатель в файле при чтении
seekp	устанавливает внутренний указатель в файле при записи
tellg	определяет позицию внутреннего указателя при чтении
tellp	определяет позицию внутреннего указателя при записи

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 243

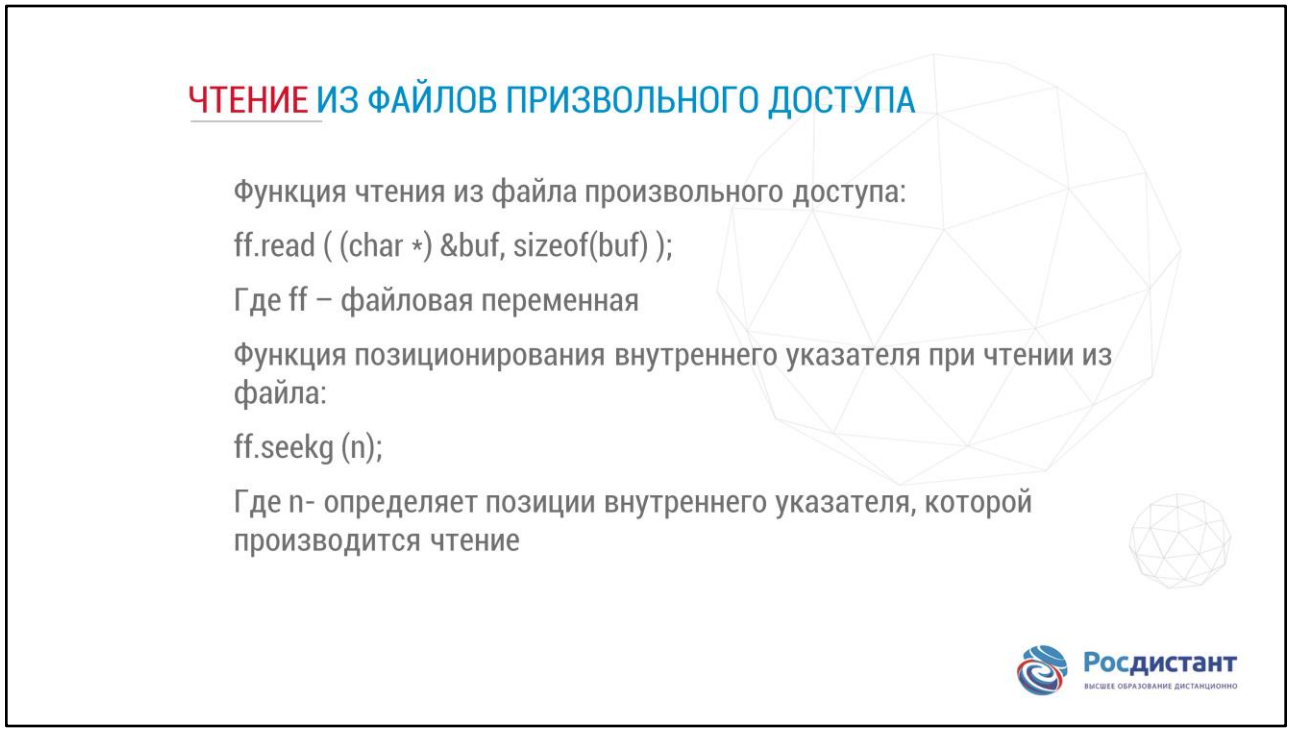
Выше рассматривались задачи обработки файлов последовательного доступа, содержащие информацию разного типа, объединенную в структуру.

Каждый элемент структуры последовательно записывался и последовательно считывался из файла.

Еще одним преимуществом файлов произвольного доступа, помимо позиционирования внутреннего указателя, является возможность записи и считывания из файла всей структуры одним обращением к файлу.

При обработке файлов, надо помнить, что как информация в свое время была записана в файл, так же и надо ее считывать, такими же порциями.

Если появляется необходимость вносить изменения в файл, то это сложно осуществить в файлах последовательного доступа. Управлять внутренним указателем в файлах произвольного доступа можно с помощью функций, представленных на слайде.



Слайд 244

Чтение из файлов произвольного доступа реализует функция - **read** , а позиционирование функция – **seekg ,** где слово **get** – означает взять.

Работа этих функций означает - установить внутренний указатель на **n** байт от начала файла и присвоить переменной **buf** , то есть считать, значение, записанное от позиции **n** на **sizeof** байт вперед, где, **ff** – файловая переменная.

Поиск в файлах произвольного доступа осуществляется в области адресов и завершается обращением непосредственно к искомому участку. Дисковое пространство, занимаемое таким файлом, поделено на одинаковые участки, записи, имеющие одинаковую структуру полей.

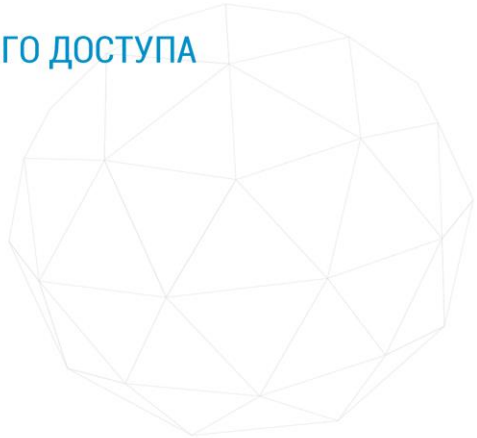
Напомним, операция **sizeof –** определяет объем памяти в байтах.

После каждой процедуры записи или чтения, в файлах произвольного доступа, внутренний указатель перемещается на начало следующей компоненты файла. Если необходимо читать компоненты файла подряд, то функцию позиционирования можно опустить.

Преимущество файлов произвольного типа заключается в том, что если создать файл структур, то можно и записать в файл и считать из файла всю структуру целиком.

ОБРАБОТКА ФАЙЛА ПРИЗВОЛЬНОГО ДОСТУПА

```
#include<fstream>
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{ // открыть файл для записи
  ofstream f_file("f.dat");
  int buf, n;
```



Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

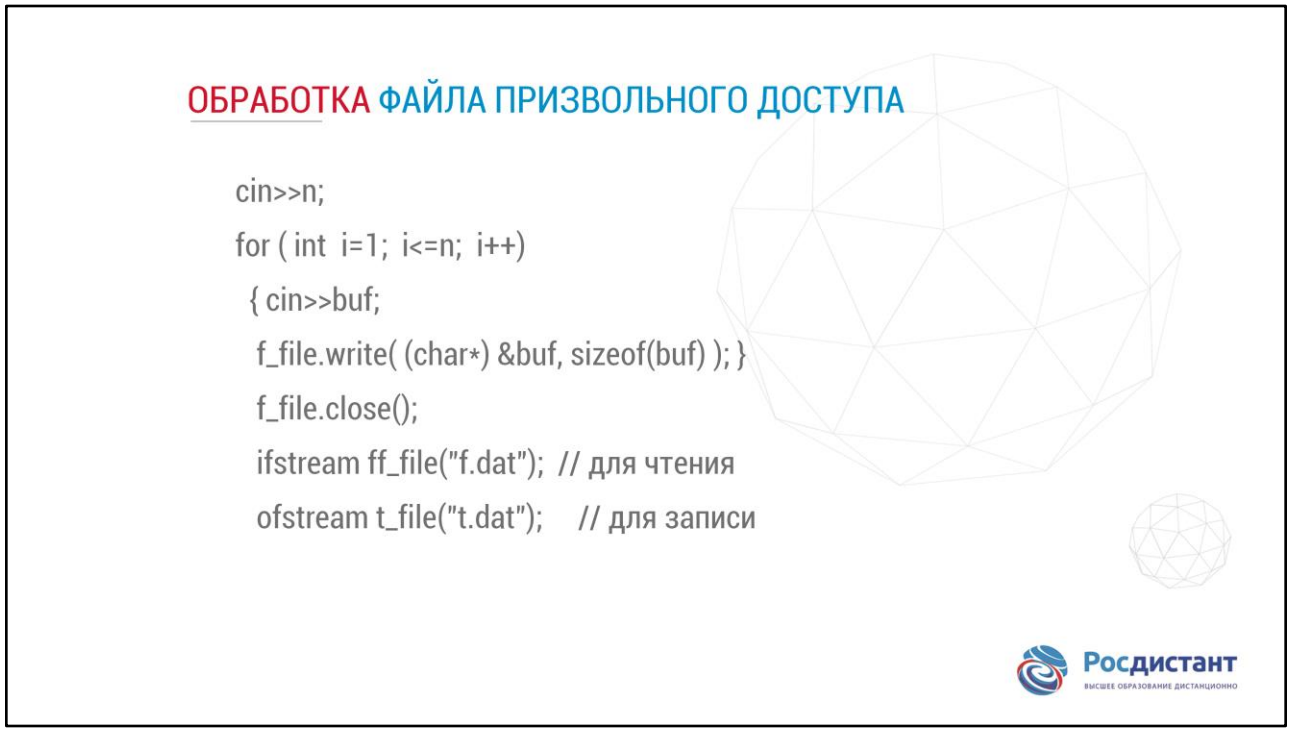
Слайд 245

Рассмотрим задачу: создать файл целых чисел. Из исходного файла сформировать новый, содержащий числа исходного файла, стоящие на четных местах.

Из условия задачи видно, что необходимо позиционирование внутреннего указателя для перебора чисел на четных местах. Для решения этой задачи создается файл произвольного доступа. Как известно, в файлах произвольного доступа есть возможность управлять внутренним указателем в файле. Именно с положения внутреннего указателя происходит запись в файл и чтение из файла.

Подключены заголовочные файлы **fstream** для применения функций обработки файлов и **iostream** для организации ввода-вывода информации.

С помощью файловой переменной **f_file** открыт доступ к файлу для записи. Введена переменная буфер, тип которой, согласно условию задачи, целый, то есть имеет тип компонент файла. Переменная **n** – это количество чисел, которые необходимо записать в файл.



Слайд 246

На слайде продемонстрирован фрагмент программы, создающий исходный файл целых чисел.

С клавиатуры вводится количество чисел **n .** В цикле вводятся целые числа с клавиатуры, и каждое введенное число записывается в файл. Нет необходимости в позиционировании внутреннего указателя в процессе создания нового файла. После записи очередной компоненты в файл произвольного доступа, внутренний указатель автоматически устанавливается в начало следующей компоненты.

Размер каждой компоненты файла определяется типом переменной - буфера.

Исходный файл необходимо закрыть.

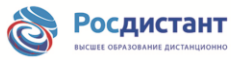
Далее открываются два файла:

- исходный файл для чтения;
- новый файл для записи, содержащий числа исходного файла, стоящие на четных местах. Имя файла указано в двойных кавычках.

Обратите внимание, при повторном открытии исходного файла используют новую файловую переменную **ff** .

ОБРАБОТКА ФАЙЛА ПРИЗВОЛЬНОГО ДОСТУПА

```
for ( int i=2; i<=n; i+=2)
 { ff_file.seekg( sizeof(buf)*(i-1) );   // позиционирование указателя
   ff_file.read ( (char*) &buf, sizeof(buf) );
   t_file.write ( (char*) &buf, sizeof(buf) ); }
ff_file.close();  t_file.close();
ifstream tt_file("t.dat");
cout<<"\n Созданный файл \n";
while ( tt_file.read( (char*) &buf, sizeof(buf) ) )
            cout<<"\n"<< buf;
tt_file.close(); getch(); }
```



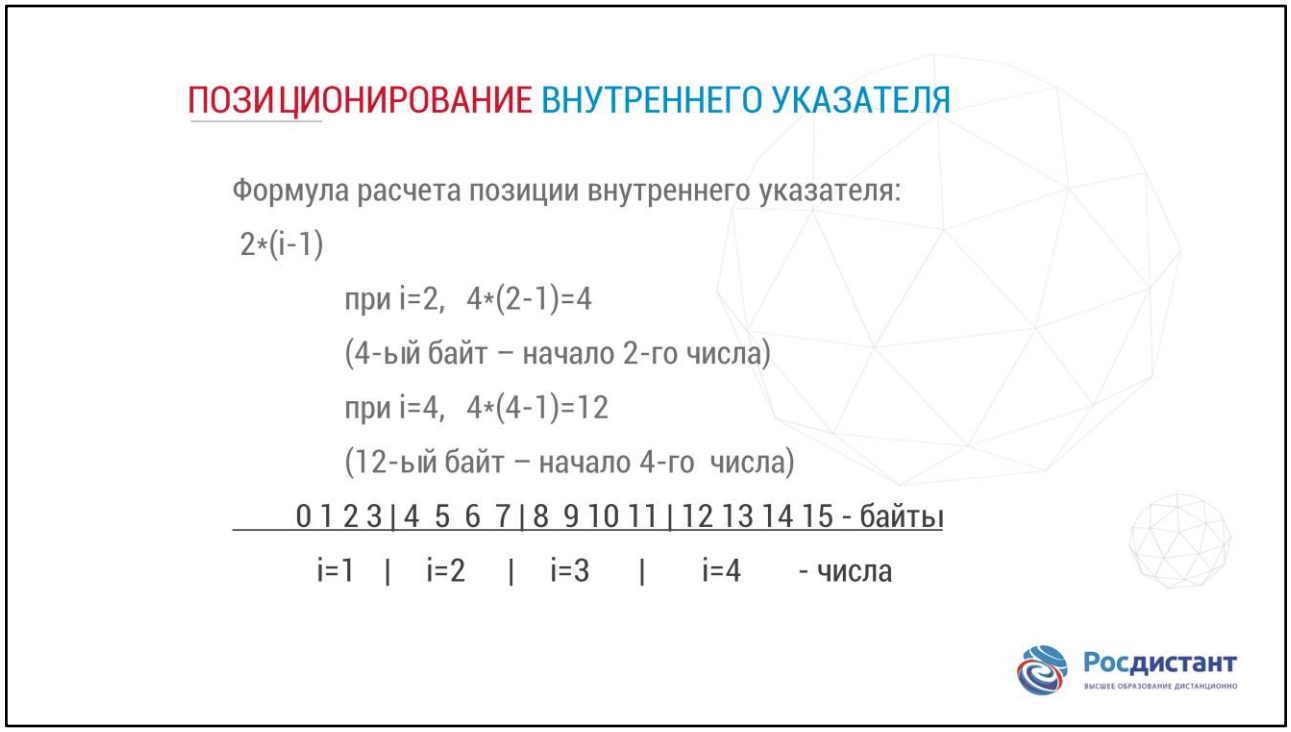
Слайд 247

На слайде следующий фрагмент программы демонстрирует создание нового файла.

В цикле считывается информация из исходного файла с помощью функции **read** и записывается в новый файл. В законе изменения параметра цикла, шаг изменения равен двум, что позволяет, благодаря указанной формуле, позиционировать внутренний указатель в исходном файле. Позиционирование осуществляет функция **seekp .**

Далее, необходимо **з**акрыть оба файла. Созданный файл открывается для чтения с указанием новой файловой переменной. В цикле с предусловием информация читается из файла и выводится на экран. Обратите внимание, что функция чтения из файла **read** одновременно является условием продолжения цикла, то есть пока читается из файла.

Обращение к файлам, то есть процедуры чтение из файла и процедура записи в файл, осуществляется с помощью переменной **buf** .



Слайд 248

На слайде подробно рассмотрен расчет позиционирования внутреннего указателя, так как из исходного файла необходимо считывать только числа, стоящи на четных местах. В различных системах количество байт под тот или иной тип данных может отличаться. На слайде приведен расчет для целочисленных переменных, занимающих четыре байта памяти. С помощью функции **sizeof** можно определить количество байт, отводимых под переменные разного типа.

Расчет зависит от размера компонент файла, в данной задаче это четыре байта, размер переменных целого типа, тип переменной, через которую происходит обращение к файлу.

По указанной формуле, рассчитывается байт, на который устанавливается внутренний указатель исходного файла и производится чтение очередной компоненты.

Расчет позиционирования внутреннего указателя всегда связан с размером компонент файла, то есть связано с количеством байт, которые необходимо считать из файла.

Слайд 249

Рассмотрим задачу: создать файл, содержащий информацию о товарах некоторого склада. Каждая компонента файла - структура, содержащая информацию о товарах некоторого склада: наименование товаров, количество каждого наименования товаров и цена единицы товара.

На слайде приведена структура, определяющая размер компонент файла, определяющая количество байт.

Переменная **buf** имеет пользовательский тип – структура. Размер этой переменной определено элементами структуры.

Обратите внимание, переменная **buf** описана непосредственно после определения структуры. В принципе, имя этой структуры при описании структуры можно опустить. Именно через переменную **buf** будет происходить обмен информацией с файлом.

ПРИМЕР – «ФАЙЛЫ ПРОИЗВОЛЬНОГО ДОСТУПА»

```
int  n;
cin>>n;
ofstream f_file("f.dat");
cout<<"\n  Введите информацию о товарах \n";
for ( int  i=1;  i<=n;  i++ )
  { cin>> buf.name>>buf.kol>>buf.price;
    f_file.write( ( char *) &buf, sizeof (buf) ); }
 f_file.close();
```


Слайд 250

С клавиатуры вводится количество наименований товаров, значение переменной **n .**

Открывается файл для записи. Ввод информации осуществляется в цикле с параметром.

На каждой итерации цикла с клавиатуры вводится информация об одном наименовании товара, отдельно наименование товара, цена и количество.

Функция **write** позволяет одним обращением всю структуру записать в файл. Размер компонент файла вычисляется автоматически операцией **sizeof .**

В данном случае нет необходимости в позиционирование внутреннего указателя файла.

По окончании процедуры записи в файл внутренний указатель находится в конце файла. Файл был открыт для записи.

ПРИМЕР – «ФАЙЛЫ ПРОИЗВОЛЬНОГО ДОСТУПА»

```
ifstream  ff_file ("f.dat");
cout<<"\nТовары склада \n";
while ( ff_file.read ( ( char *) &buf, sizeof (buf) ) )
{  cout<<"\n "<<buf.name<<"\t"
        <<buf.kol<<"\t"<<buf.price; }
ff_file.close();
getch(); }
```


Слайд 251

Для того чтобы просмотреть содержимое созданного файла, открываем файл для чтения.

Чтение из файла осуществляется в цикле с предусловием.

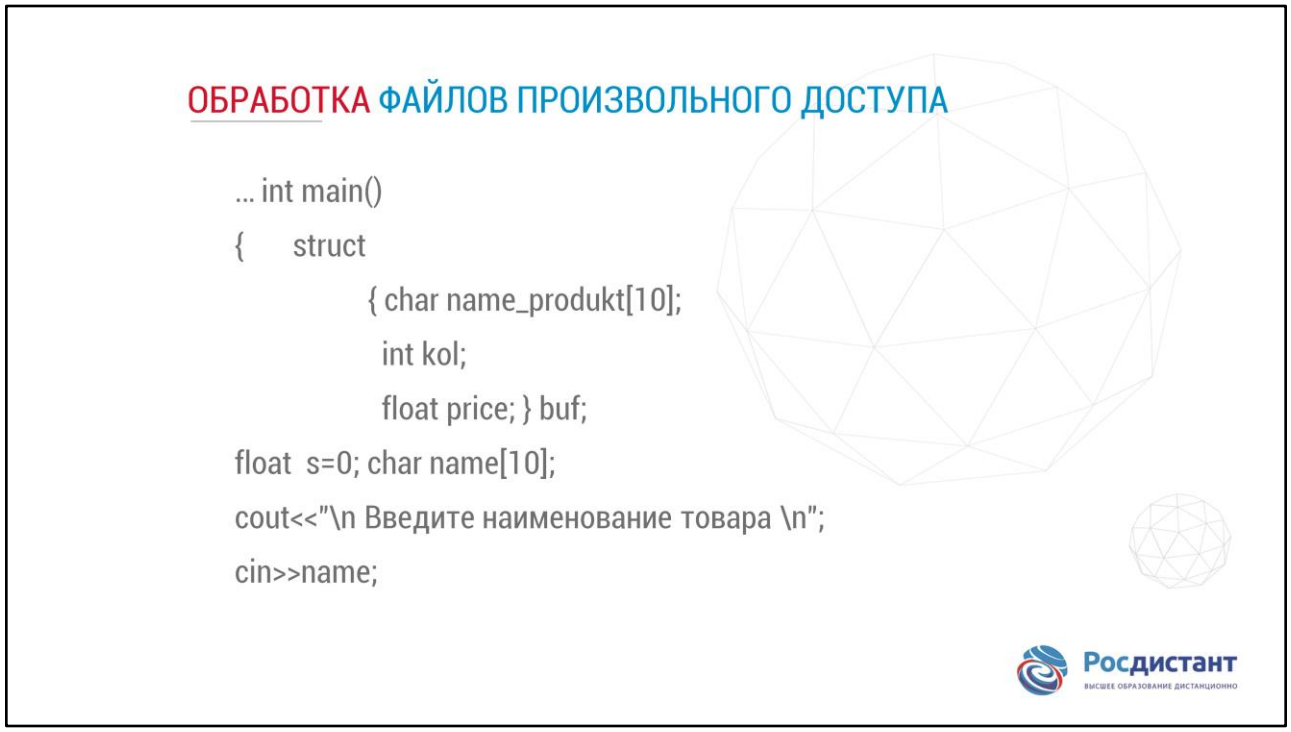
Обратите внимание, в качестве условия выполнения цикла в операторе **while** используется функция чтения из файла.

Каждое обращение к файлу соответствует чтению одной компоненты файла, равной размеру описанной выше структуру

Каждый элемент структуры через переменную **buf** выводится на экран.

Перед концом выполнения программы необходимо закрыть файл во избежание потери или порчи информации.

Файл, созданный этой программой, мы будем использовать в следующей программе.



Слайд 252

В данной задаче будем использовать базу данных, то есть файл, созданный в предыдущей программе.

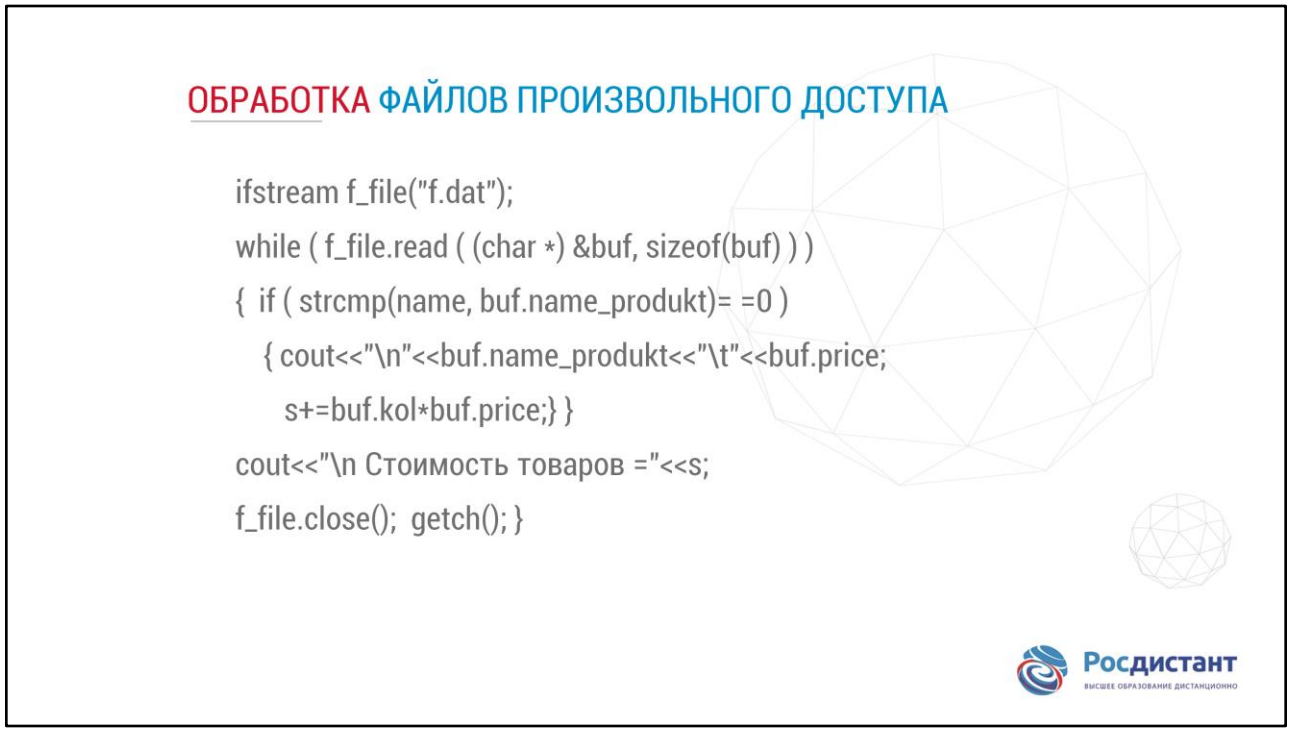
Необходимо подсчитать общую стоимость товаров определенного наименования в базе данных, в файле.

При обращении к файлу, надо точно знать, как он был в свое время создан, то есть структуру каждой компоненты файла.

Описана структура, которая должны совпадать со структурой, описанной в предыдущей задаче.

Введена переменная **s** для подсчета стоимости определенного товара в файле, значение которой надо подготовить – обнулить.

Для поиска по наименованию определенного товара в файле введена переменная **name ,** одномерный массив символов. Значение этой переменной вводится с клавиатуры.



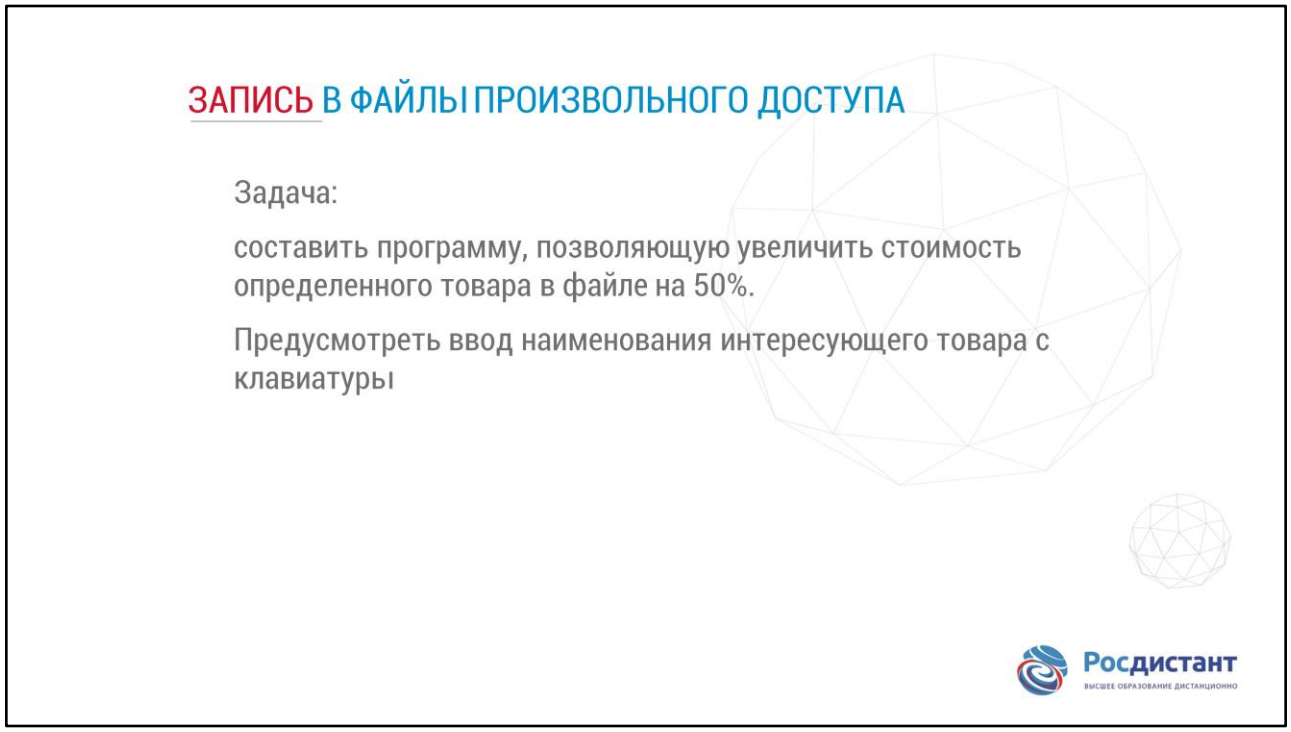
Слайд 253

Отрывается доступ к файлу, созданному в предыдущей программе. Можно указать путь доступа к файлу, если файл не в текущей папке.

Чтение информации из файла выполняется в цикле с предусловием. Чтение из файла, функция **write** , используется как условие продолжения цикла, то есть пока читается из файла.

Каждое обращение к файлу соответствует считыванию одной компоненты, размер которой, то есть количество считанных из файла байт, соответствует размеру описанной в начале программы структуре. Наименование структуры, имена переменных, которые являются элементами структуры могут не совпадать с описанием структуры при создании файла. Но, типы элементов структуры и последовательность элементов необходимо сохранить.

Считывается информация из файла в переменную-буфер buf и сравнивается с искомым в файле товаром. Функция strcmp сравнивает побайтно две переменные, искомое наименование товара и наименование товара из файла. Если переменные совпали, функция вырабатывает значение ноль. Информация выводится на печать и одновременно подсчитывается общая стоимость товаров на складе данного наименования. Данную функцию сравнения символьных переменных мы подробно ранее рассматривали в нашем курсе. В конце вычислений необходимо закрыть файл.



Слайд 254

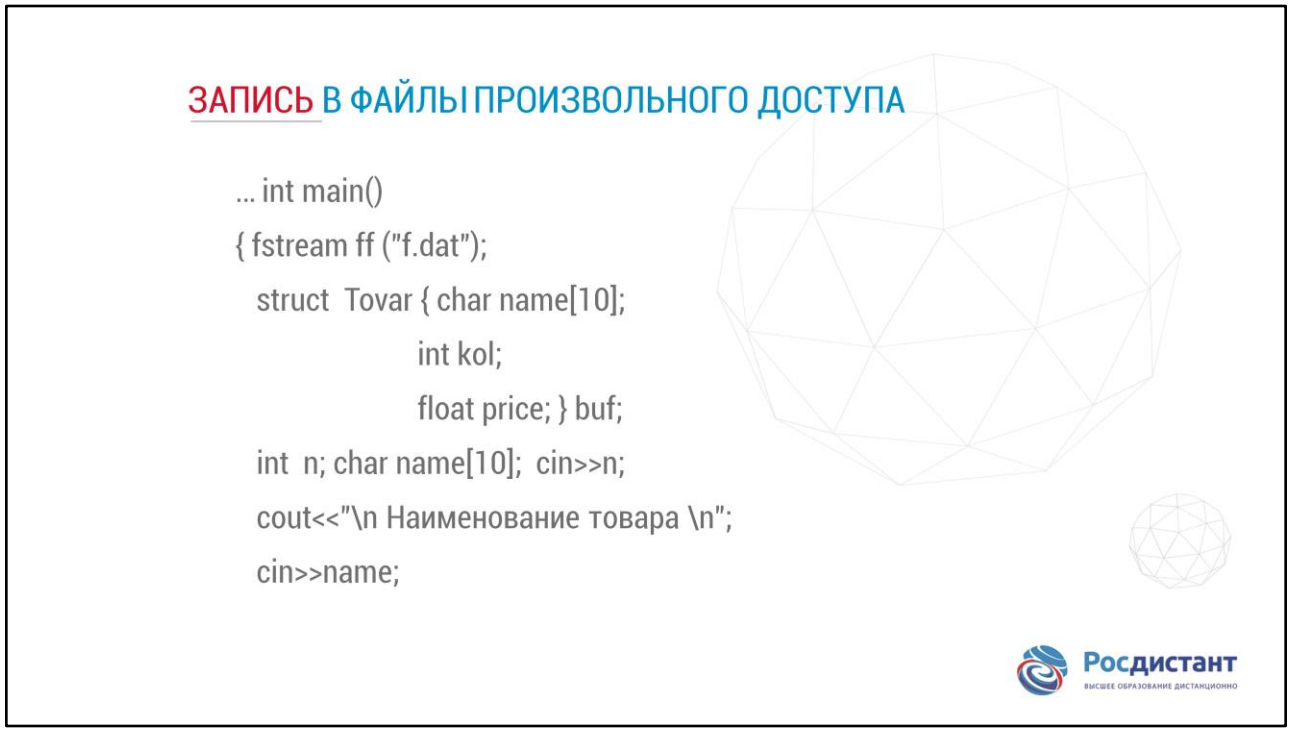
В задаче на слайде подразумевается внести изменения в уже существующий файл.

Наименование товара, стоимость которого надо изменить, вводится с клавиатуры.

Для решения этой задачи, необходимо точно знать, как файл был в свое время создан, то есть знать структуру файла, последовательностью расположения элементов структуры. Размер каждой компоненты определяется типом элементов структуры.

Задача предполагает чтение очередной компоненты файла, и, если это искомый товар, изменить стоимость и измененную информацию записать в файл на старое место. Это возможно, если применить функции управления внутренним указателем в файлах произвольного доступа, которые были рассмотрены выше.

Файл произвольного доступа позволяет позиционировать внутренний указатель, что и позволит изменять информацию, не создавая новый файл.



Слайд 255

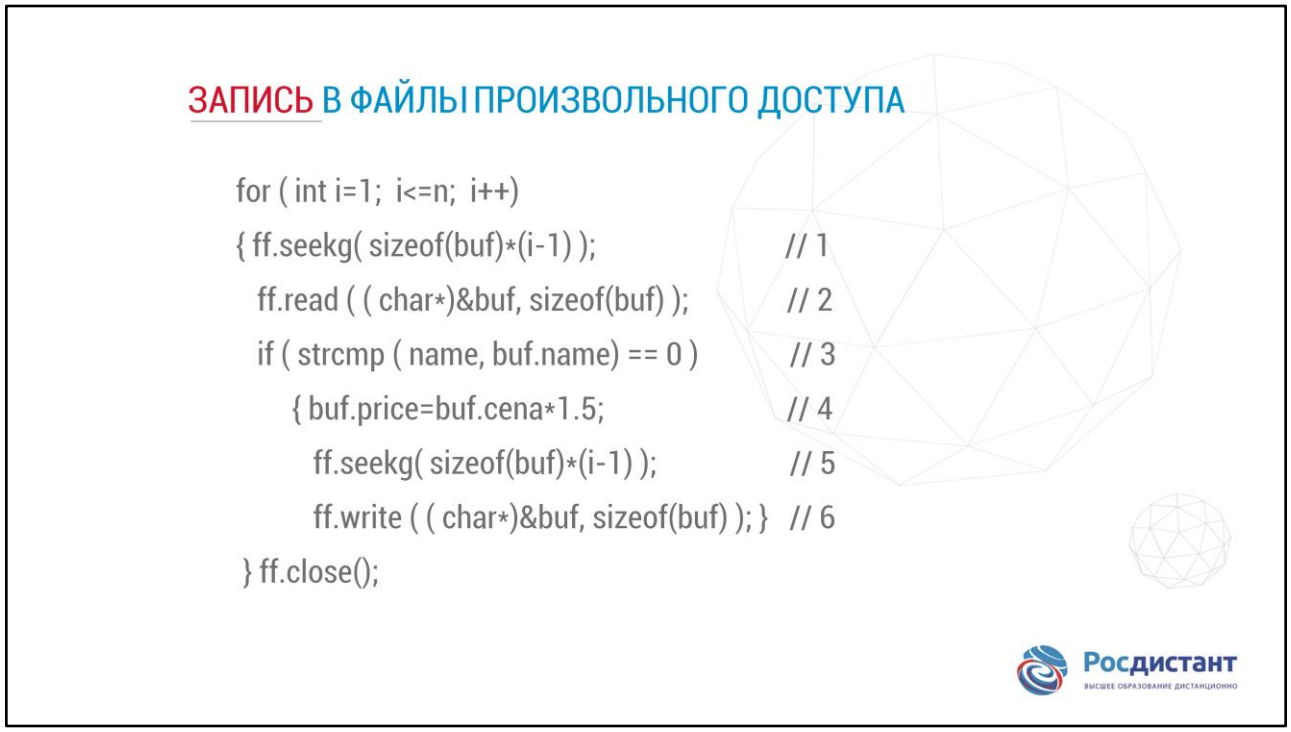
На слайде представлен фрагмент программы.

Обратите внимание, с помощью функции **fstream** открыт доступ к файлу для записи и чтения. Работа этой функции рассматривалась выше. Одновременно и читается из файла и записывается в файл, используя одну и ту же файловую переменную.

Описан пользовательский тип **Tovar** , который определяет структуру и размер компонент файла.

Переменная **buf** – будет использована при обращении к файлу, записи или чтении. Переменная **n** – общее количество различных наименований товаров в файле.

С клавиатуры вводится наименование товара, стоимость которого надо изменить.



Слайд 256

На слайде представлен следующий фрагмент программы. Комментарии помогут подробно рассмотреть этот фрагмент.

Использован цикл с параметром, где параметр цикла используется при позиционировании внутреннего указателя в файле.

Рассмотрим подробно:

- внутренний указатель устанавливается на начало компоненты файла с номером i ;
- считывается очередная компонента файла;
- сравнивается наименование товара в очередной компоненте файла с наименованием искомого товара;
- если условие выполнилось, изменяется стоимость данного товара;
- указатель возвращается назад, то есть на начало этой же компоненты;
- измененная информация записывается на старое место.

Обратите внимание, что изменяется только часть информации компоненты файла, а именно цена. Наименование товара и количество остаются неизменными. Записать в файл необходимо всю структуру.

ЧТЕНИЕ ИЗ ФАЙЛА ПРОИЗВОЛЬНОГО ДОСТУПА

```
ifstream tt ("f.dat");
cout<<"\n Измененный файл \n";
while( tt.read( ( char*)&buf, sizeof(buf) ) )
 { cout<<"\n"<< buf.name<<"\t"<<buf.price; }
tt.close();
getch();
}
```


Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

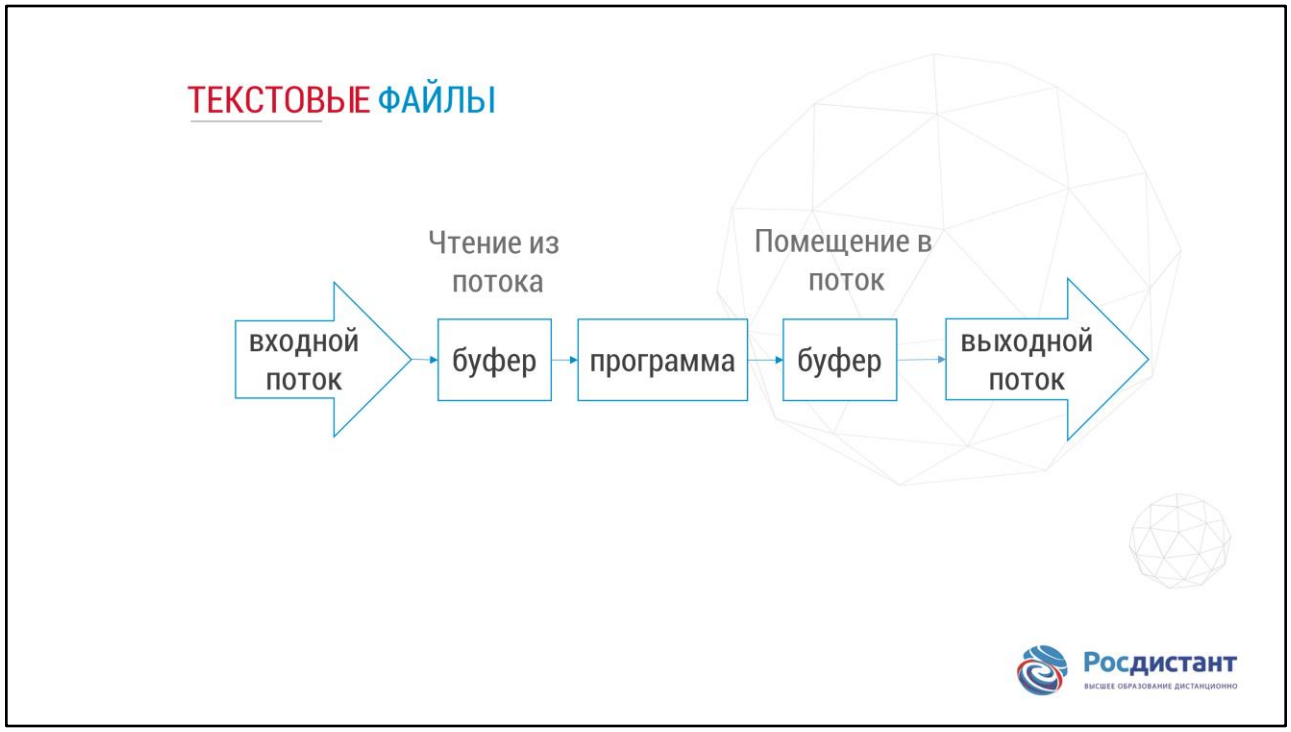
Слайд 257

Откроем файл только для чтения. Для открытия файла введена новая файловая переменная **tt .**

В цикле с предусловием, читаем информацию из файла и выводим на экран. Обратите внимание на условие продолжения цикла. Этим условием является функция чтения из файла. Условие трактуется так – пока читается из файла, то есть пока не будут считаны все компоненты файла. Естественно, в файле присутствует признак конца файла.

Размер каждой считанной компоненты определяется операцией **sizeof .** Считывается вся компонента из файла и каждый элемент структуры выводится на печать через переменную **buf .** Указывается имя структуры и через точку указывается имя конкретного элемента структуры. а именно наименование товара и цена.

Во избежание потери информации, перед выходом из программы, необходимо закрыть файл.



Слайд 258

«Файл – это именованная область внешней памяти, в которой хранится логически завершенный объем данных. Текстовый файл – это файл, в котором каждый символ из используемого набора символов хранится в виде одного байта, кода, соответствующего символу. Текстовые файлы разбиваются на несколько строк с помощью специального символа - конец строки.

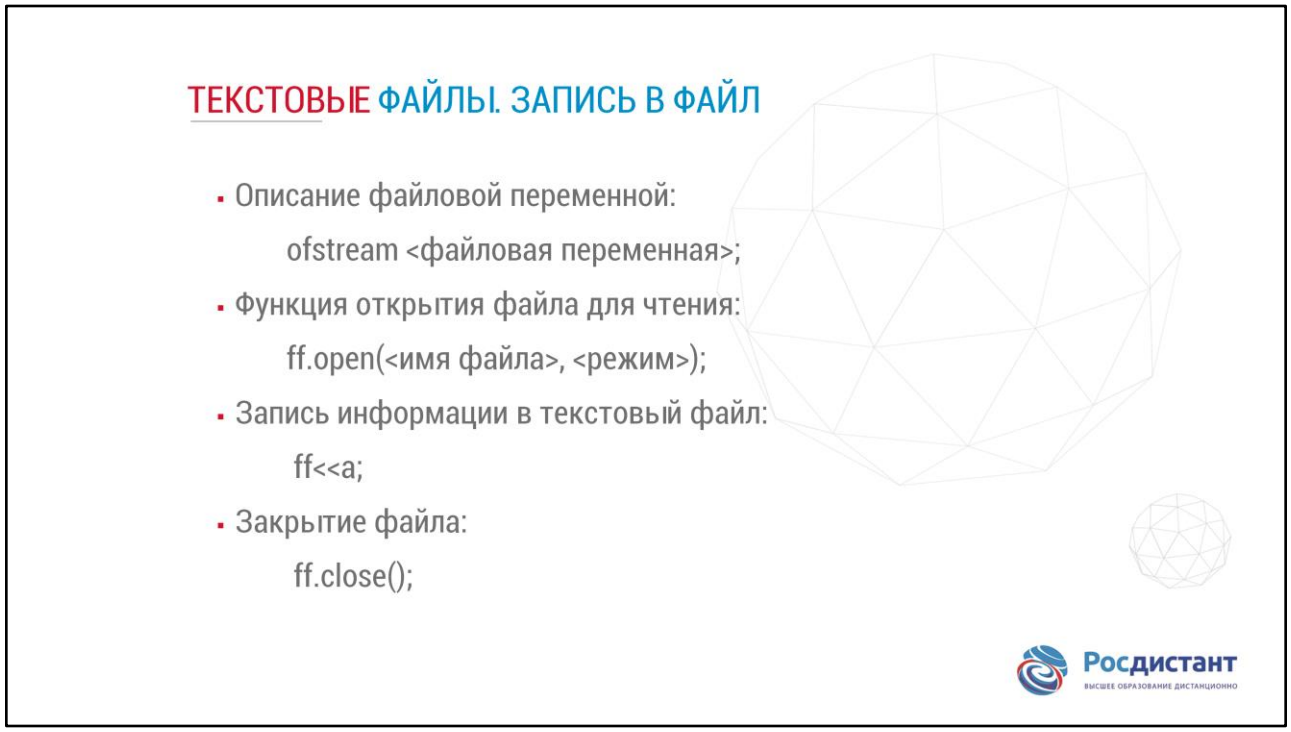
Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

Функции библиотеки ввода-вывода языка С++ , поддерживающие обмен данными с файлами на уровне потока, позволяют обрабатывать данные различных размеров и форматов, обеспечивая при этом буферизованный ввод и вывод. Таким образом, поток представляет собой этот файл вместе с предоставленными средствами буферизации. Чтение данных из потока называется извлечением, вывод в поток – помещением, или включением. Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен, оперативная память, файл на диске, клавиатура или принтер. Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти – буфер. Буфер накапливает байты, и фактическая передача данных выполняется после заполнения буфера. При вводе это дает возможность исправить ошибки, если данные из буфера еще не отправлены в программу.

Когда программа начинает выполняться, автоматически открываются следующие потоки:

- стандартный поток ввода;
- стандартный поток вывода;
- стандартный поток вывода сообщений об ошибках.

По умолчанию стандартному потоку ввода ставится в соответствие клавиатура, а потокам вывода соответствует экран монитора.»



Слайд 259

На слайде приведены основные функции при работе с текстовыми файлами в режиме записи в файл.

«Существуют два основных типа файлов: текстовые и двоичные. В предыдущих темах мы уже познакомились с двоичными файлами.

К текстовым файла прибегают в тех случаях, когда возникает необходимость обрабатывать большие объемы данных, не вводя их с клавиатуры.

Текстовыми файлы состоят из любых символов. Они организуются по строкам, каждая из которых заканчивается символом конца строки.

В конце текстового файла присутствует признак конца файла.

С помощью любого текстового редактора можно просмотреть содержимое текстового файла. При записи информации в текстовый файл, все данные преобразуются к символьному типу и хранятся в символьном виде.

В программах на языке С++ при работе с текстовыми файлами необходимо подключать библиотеки **iostream** и **fstream** .

Для того чтобы записывать данные в текстовый файл, необходимо:

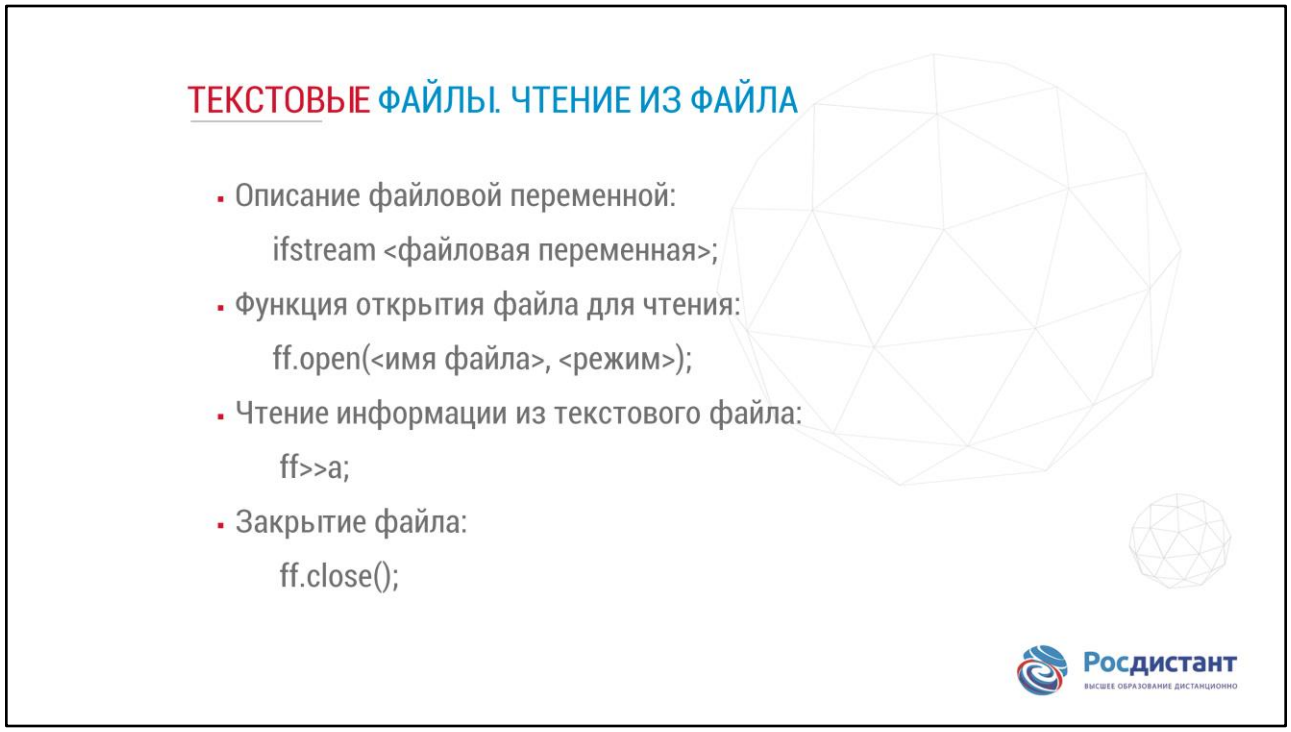
- описать файловую переменную типа **ofstream ;**
- открыть файл с помощью функции **open ;**

- вывести информацию в файл;
- обязательно закрыть файл.

В функции записи в текстовый файл указывается имя файла и режим открытия файла для записи.

После открытия файла в режиме записи, в него можно писать точно так же, как и при выводе на экран, только вместо стандартного устройства вывода **cout** необходимо указать имя открытого файла. Имя файла определяется через файловую переменную.»

Открытые на диске файлы после окончания работы с ними рекомендуется закрыть. После окончания работы файлы закрываются функцией **close** во избежание потери информации в файле**.**



Слайд 260

На слайде приведены основные функции при работе с текстовыми файлами в режиме чтения из файла.

«Для того, чтобы считать данные из текстового файла, необходимо:

- описать файловую переменную типа **ifstream ;**
- открыть файл с помощью функции **open ;**
- считать информацию из файла, при считывании каждой порции данных необходимо проверять, достигнут ли конец файла;
- закрыть файл.

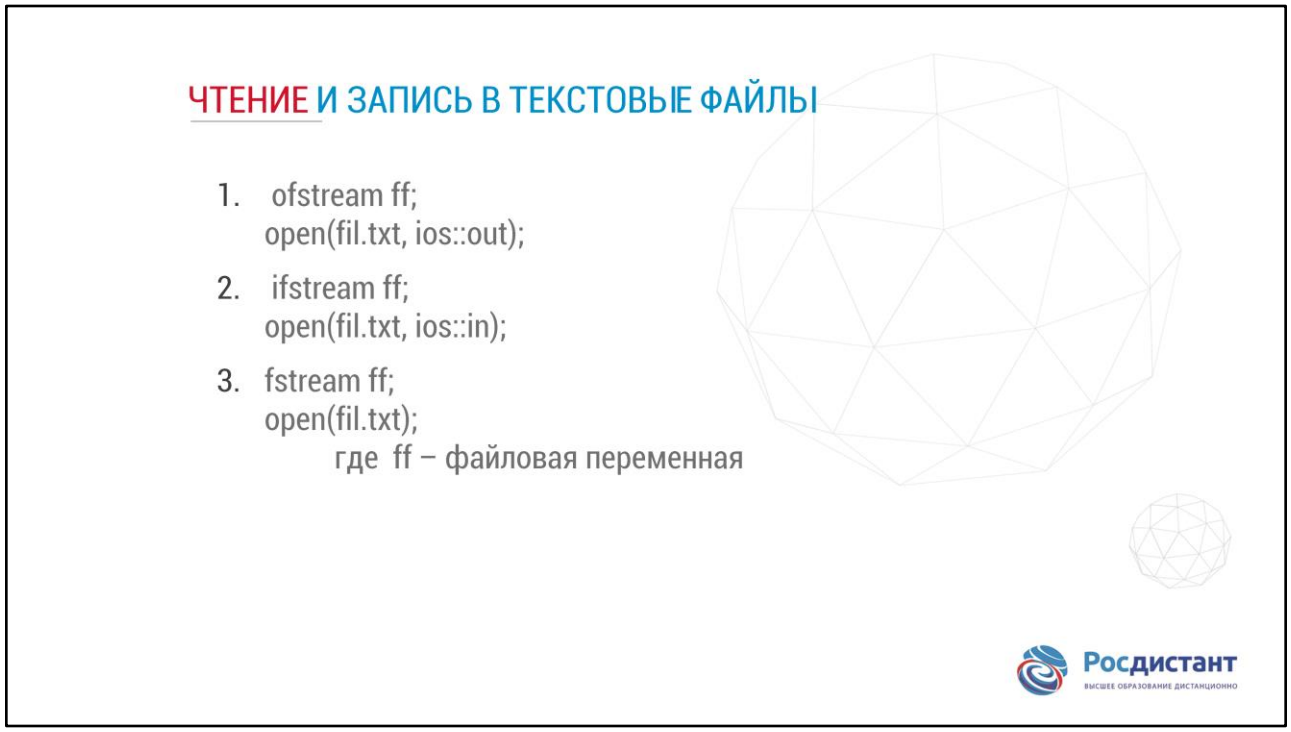
После открытия файла в режиме чтения из него можно считывать информацию точно так же, как и с клавиатуры. Вместо функции **cin** нужно указать имя потока, определенного через файловую переменную, из которого будет происходить чтение данных.»

На слайде приведены основные функции описания файловой переменной и открытие файла для чтения, а также представлен пример чтения информации из текстового файла.

Например, чтение из потока **ff** в переменную **a** представлена на слайде. Параметр, указывающий режим открытия файла может отсутствовать, в этом случае файл открывается в режиме по умолчанию для данного потока.

Указывая имя файла, можно указать путь доступа к файлу. В программах на языке **C++** при работе с текстовыми файлами в потоковом режиме необходимо подключать библиотеки **iostream** и **ofstream** .

После удачного открытия файла, в любом режиме, в файловой переменной **ff** будет храниться значение **true** - истина**,** в противном случае **false** - ложь**.** Это позволит проверить корректность операции открытия файла.



Слайд 261

На слайде приведены три примера:

1. открытие файла для записи;
2. открытие файла для чтения;
3. открытие файла для записи и чтения.

Если в пределах одной программы возникает необходимость и записывать информацию в файл и считывать информацию из файла, используют третий подход.

При открытии файла впервые, то есть при создании нового файла, в файле автоматически создается признак конца файла. При дозаписи в уже существующий файл, признак конца файла автоматически отодвигается.

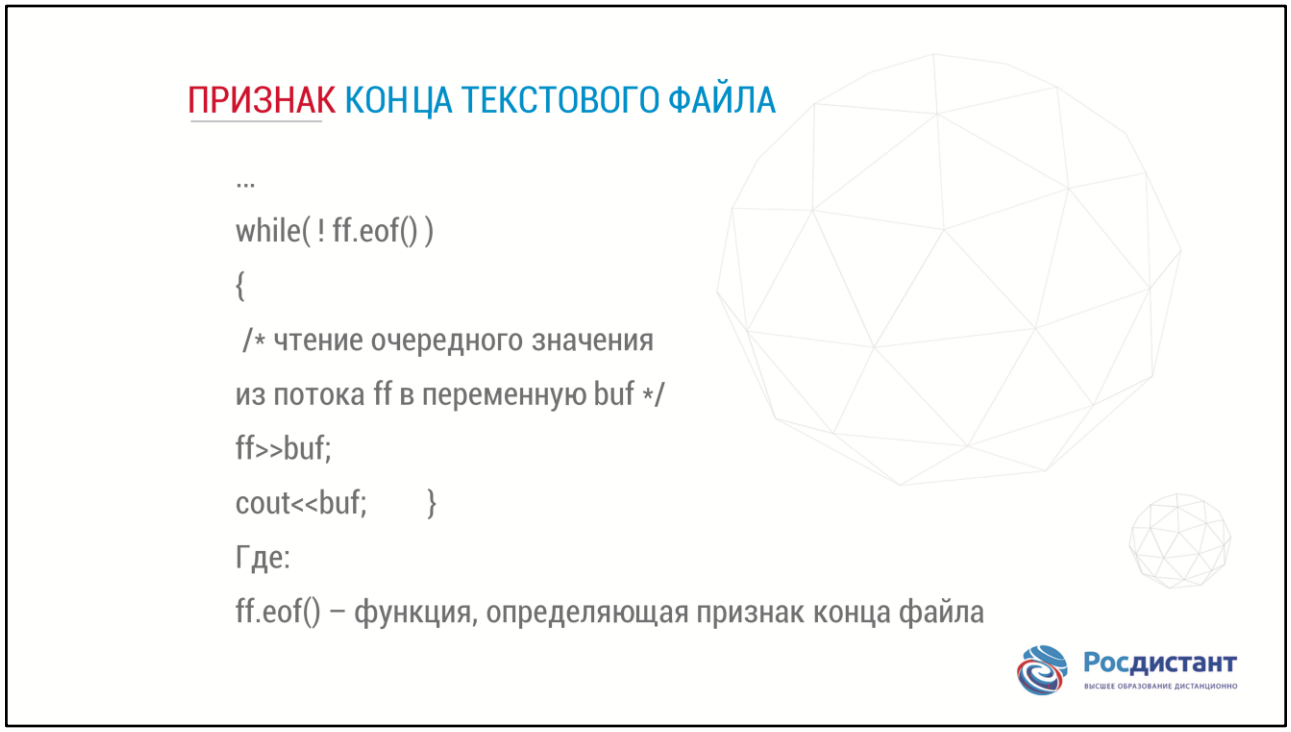
«После удачного открытия файл, в любом режиме, в файловой переменной **ff** будет храниться единицу, то есть **(true) ,** в противном - случае ноль, то есть **(false) .** Это позволит проверять корректность операции открытия файла.

Чтение из файла выполняется корректно до достижения конца файла. Поток как тип данных – это тип, который описывает объекты, реализующие процесс ввода и вывода данных в файлы.

Режимы открытия файлов – это наборы допустимых действий с открываемыми в

программе файлами.

После открытия файла в режиме записи, будет создан пустой файл, в который можно будет записывать информацию. Если необходимо открыть существующий файл, при этом сохранить его содержимое, то следует использовать определенный режим. »



Слайд 262

«Два числа в текстовом редакторе считаются разделенными, если между ними есть хотя бы один из символов: пробел, табуляция, символ конца строки. Хорошо, когда заранее известно, сколько и какие значения хранятся в текстовом файле.

Однако часто известен лишь тип значений, хранящихся в файле, при этом их количество может быть различным. Для решения данной проблемы необходимо считывать значения из файла поочередно, а перед каждым считыванием проверять, достигнут ли конец файла.

«Для определения конца файла используют функцию **eof** .

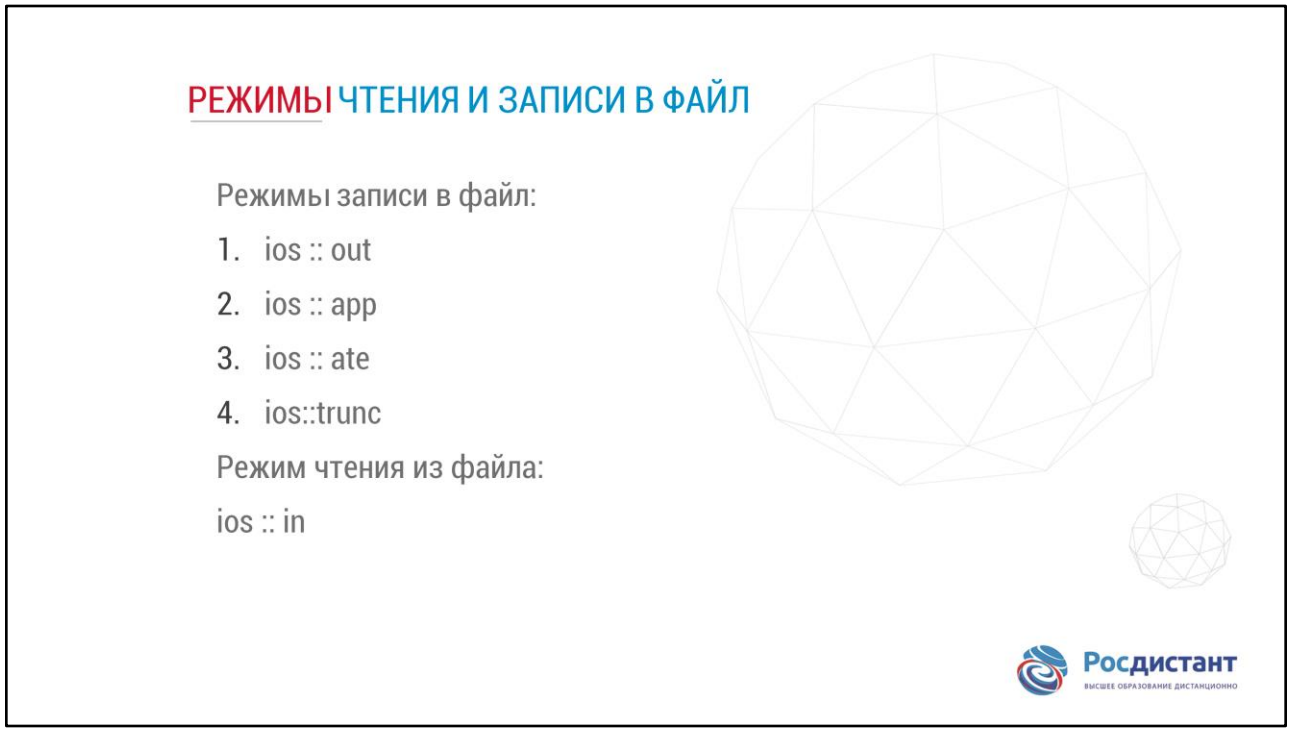
На слайде приведен фрагмент чтения информации из текстового файла в структуре оператора **while .** Цикл, то есть чтение из файла продолжается до тех пор, пока не достигнут конец файла.

Файловая переменная **ff –** может принимать одно из двух значений: **true** или **false** , в зависимости от того достигнут ли конец файла.

«Итак, подведем итоги:

1. На языке **C++** определен специальный тип данных – поток.
2. Каждый из потоков: **fstream** , **ifstream , ofstream** – служит для работы с файлами в определенном режиме.

3. Прототипы функций по работе с файлами в потоковом режиме находятся в стандартных библиотеках.
4. Перед началом работы с файлом его необходимо открыть, указав режим открытия.
5. Любой открытый в программе файл необходимо закрыть после использования.»



Слайд 263

Рассмотрим режимы записи и чтения из текстовых файлов.

Рассмотрим режимы записи:

- первый режим - открыть файл в режиме записи данных, при этом информация в существующем файле уничтожается. Этот режим устанавливается по умолчанию для потоков **ofstream ;**
- второй режим - открыть файл в режиме записи данных в конец файла;
- третий режим – передвинуть внутренний указатель в конец уже открытого файла;
- четвертый режим - очистить файл, действия, аналогичные первому режиму;

Режим чтения из файла:

открыть файл в режиме чтения данных, режим является режимом по умолчанию для потоков **ifstream .**

Режимы, которые устанавливаются по умолчанию, можно не указывать.

ОБРАБОТКА ТЕКСТОВЫХ ФАЙЛОВ

```
#include<iostream>
#include<fstream>
#include<conuo.h>
using name std;
int main()
{ int i, n;
  double buf;
  ofstream f;
  ff.open("file.txt", ios::out);
```


Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Слайд 264

Рассмотрим пример обработки текстовых файлов.

«В программах зачастую необходимо обрабатывать данные больших объемов, причем сами данные и результаты обработки требуется сохранять относительно долгое время. Проблему выделения дополнительных ресурсов для хранения обрабатываемых данных можно решить за счет динамической памяти, однако, ее ресурс ограничен. Организовать длительное хранение информации только средствами оперативной памяти практически невозможно ввиду энергозависимости устройства. Поэтому работу с объемными данными и организацию долговременного хранения данных в языках программирования осуществляют с помощью файлов, расположенных на внешних носителях. При этом под внешними устройствами следует понимать устройства ввода-вывода данных, к которым также можно отнести и файлы. Прототипы основных функций для работы с файлами входят в стандартную библиотеку.»

Создадим файл вещественных чисел. Числа вводятся с клавиатуры и записываются ы файл.

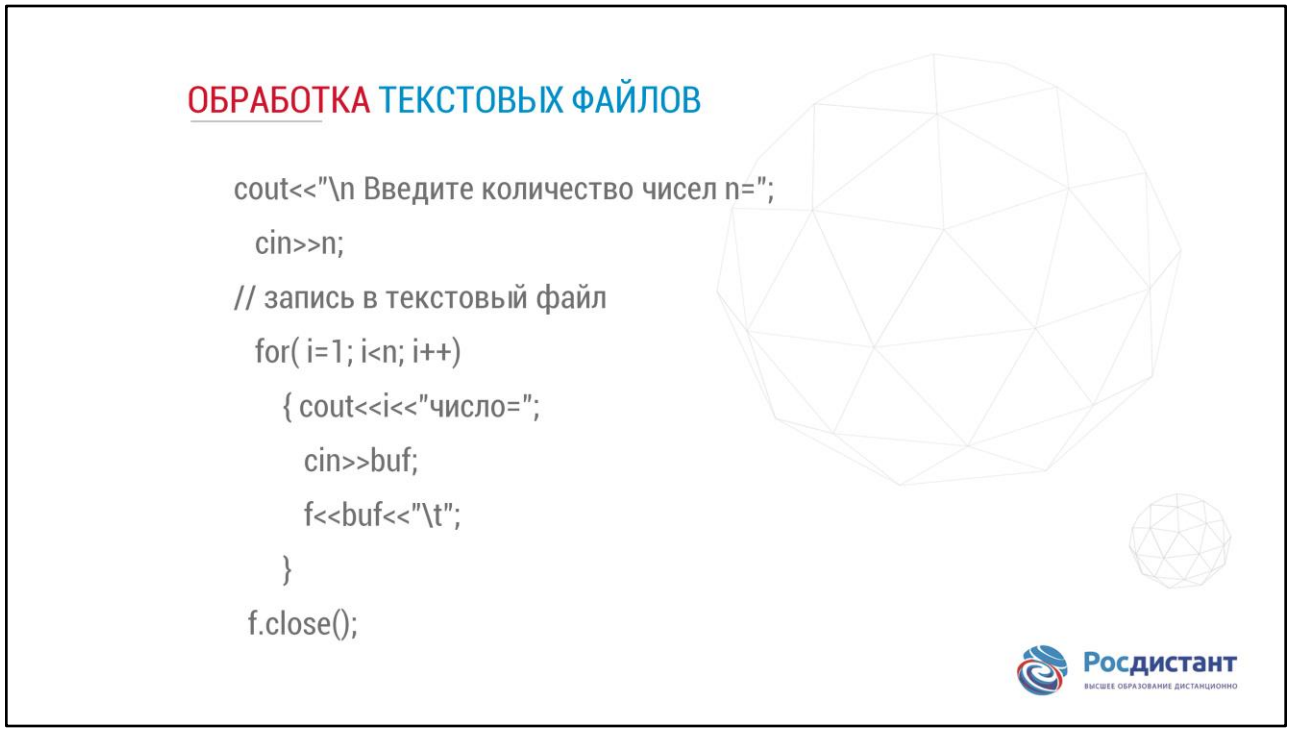
Подключены заголовочные файлы:

- **iostream** – для ввода информации с клавиатуры и вывода на экран;
- **fstream** – для обращения к функциям обработки файла;

- **conio** – для работы функции **gech** .

Введены переменные:

- **i** – параметр цикла;
- **n** – количество чисел, которые надо записать в файл;
- **buf** – переменная, буфер, через которую числа записываются в файл, тип которой определяет количество байт записываемых в файл;
- **f –** файловая переменная, связанная с именем файла для записи в файл.



Слайд 265

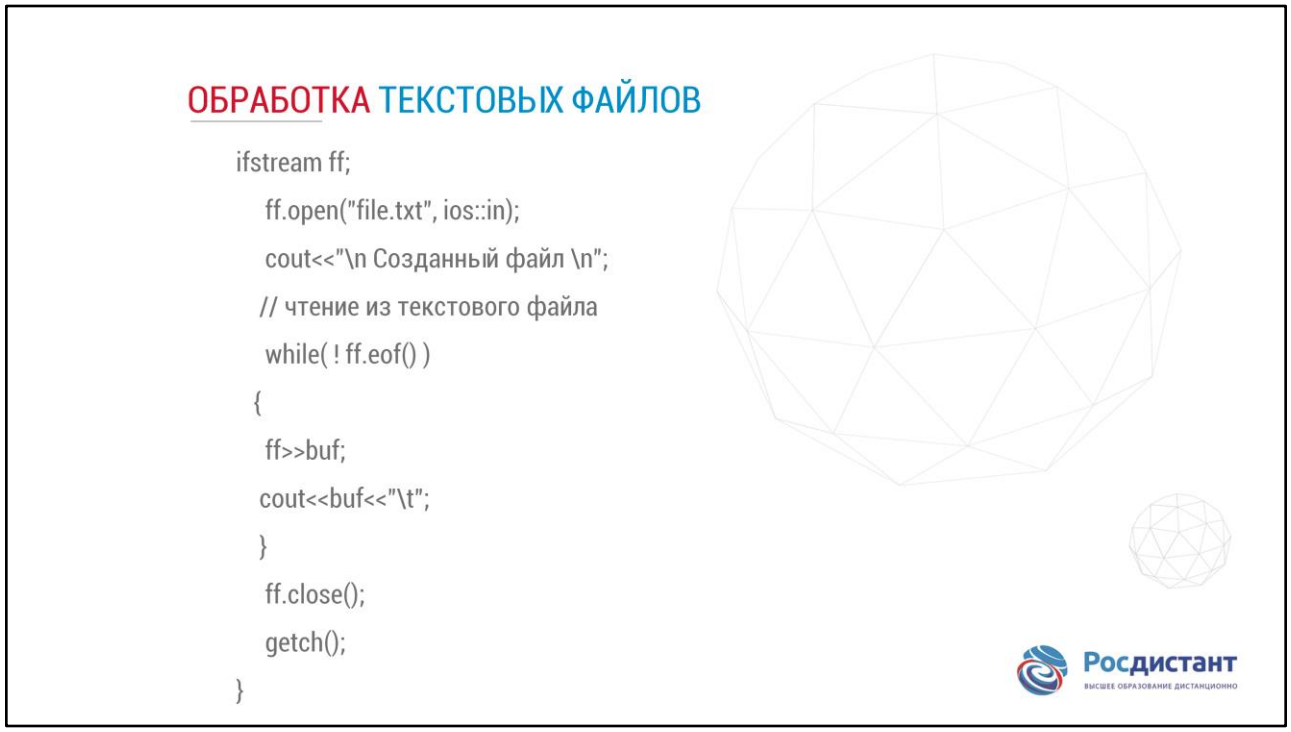
На слайде представлен следующий фрагмент программы. С клавиатуры вводится количество чисел **n** , которые необходимо записать в файл.

Организован цикл с параметром, в котором введенные с клавиатуры числа через переменную **buf** записываются в файл. Запись в текстовый файл напоминает вывод информации на экран с помощью функции **cout .**

После каждого числа в файл заносится управляющий символ **\t .** При каждом обращении к текстовому файлу, а именно чтению из файла, считывается очередная компонента файла. В нашем примере – это одно вещественное число.

Чтобы проверить содержимое созданного файла, то есть читать информацию из файла, необходимо закрыть файл.

Функция **close** закрывает файл, открытый для записи.



Слайд 266

В фрагменте программы на слайде открывается файл для чтения.

Для того чтобы прочитать информацию из текстового файла, необходимо описать файловую переменную **ff** типа **ifstream** для чтения из текстового файла. Открывается файл для чтения с помощью оператора **open** . При повторном обращению к одному и тому же файлу в пределах одной программы, вводится новая файловая переменная.

После открытия файла в режиме чтения из него можно считывать информацию точно так же, как и с клавиатуры, только вместо **cin** нужно указать имя потока, из которого будет происходить чтение данных.

В цикле с предусловием читается каждая компонента файла и выводится на экран. Цикл продолжается до тех пор, пока не будут прочитаны все компоненты файла.

Обратите внимание на условие продолжения цикла, указанное в операторе **while .** Цикл продолжается до тех пор, пока не будет достигнут конец файла.

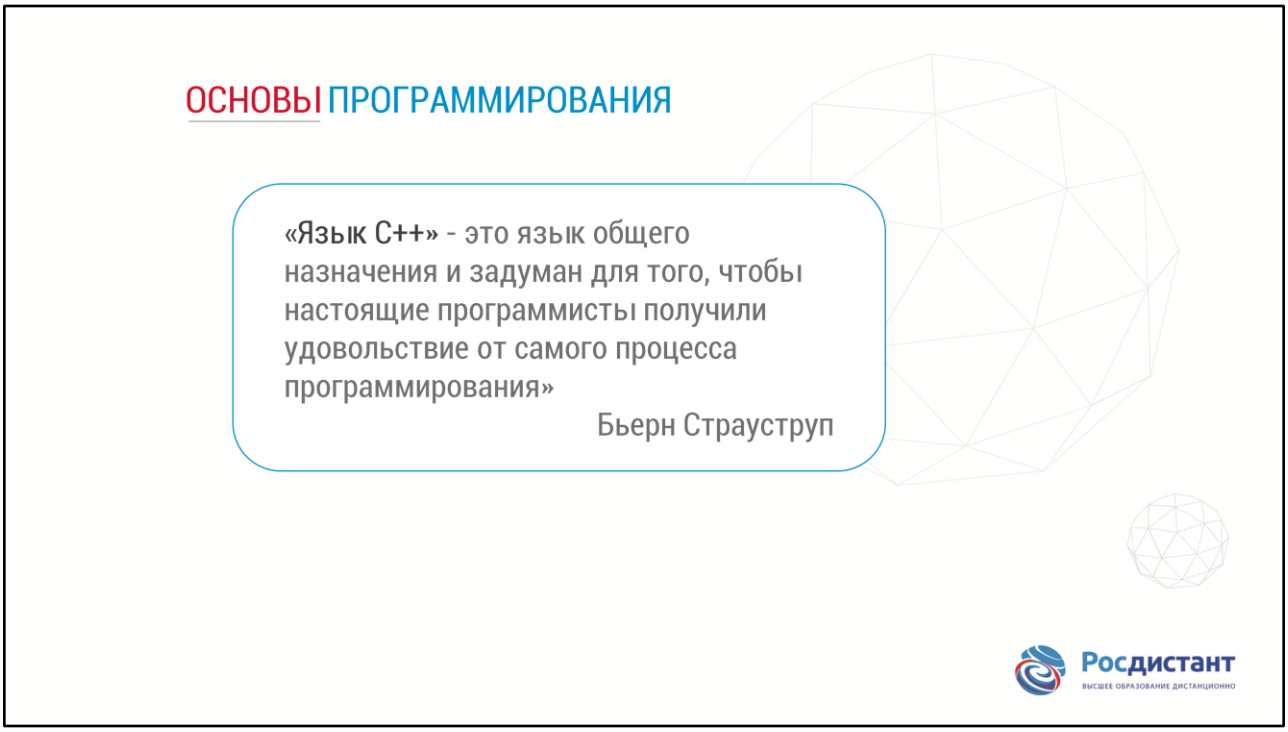
«Два числа в текстовом редакторе считаются разделенными, если между ними есть хотя бы один из символов: пробел, табуляция, символ конца строки. Хорошо, когда программисту заранее известно, сколько и какие значения хранятся в текстовом файле. Однако часто известен лишь тип значений, хранящихся в файле, при этом их количество может быть различным. Для

решения данной проблемы необходимо считывать значения из файла поочередно, а перед каждым считыванием проверять, достигнут ли конец файла.»

Функция **eof** определяет признак конца файла.

Итак, в этой теме мы подробна рассмотрели процесс создания и обработки файлов данных, рассмотрели файлы последовательного и произвольного доступа, достоинства и недостатки работы с теми и другими типами файлов.

В зависимости от конкретной задачи, выбирается принцип организации файла.



Слайд 267

Язык **C++** имеет ряд существенных особенностей, которые выделяют его среди других языков программирования. В значительной степени на формирование идеологии языка повлияла цель, которую ставили перед собой его создатели. Цель - обеспечение системного программиста удобным инструментальным языком, который мог бы заменить Ассемблер. В результате появился язык программирования высокого уровня, обеспечивающий необычайно легкий доступ к аппаратным средствам компьютера. Язык **C++** поддерживает полный набор конструкций структурного программирования, модульность, блочную структуру программы.

«Главная цель, к которой нужно стремиться — получить легко читаемую программу возможно более простой структуры. Широко известен верный в большинстве случаев девиз - лучше по-простому, чем по-умному. Смысл девиза заключается в том, что если какое-либо действие можно запрограммировать разными способами, то предпочтение должно отдаваться не наиболее компактному и даже не наиболее эффективному, а такому, который легче для понимания. Особенно это важно в том случае, когда пишут программу одни, а сопровождают другие, что является широко распространенной практикой.

Язык образует среду мышления и формирует представление о том, о чем мы думаем – говорил Бенджамин Ли Уорф.

Вы можете написать небольшую программу, используя грубую силу и нарушая все правила хорошего стиля. Для программ большего размера вы не сможете это сделать. Если структура программы, состоящей из 100 000 строк, плоха, вы обнаружите, что новые ошибки появляются с той же скоростью, с которой исправляются старые. **C++**

На этом мы завершаем изучение языка **C++** . Надеемся, что этот курс поможет Вам в освоение дисциплин специальности.

Желаем удачи!

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