

---

---

# Работа с файлами.

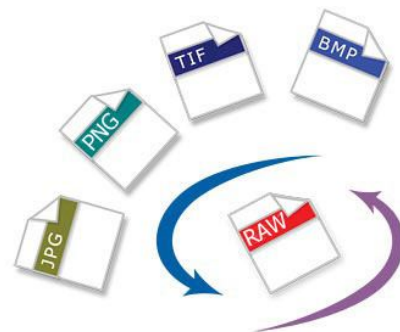
C++.

---

# Что такое файл?

**Файл** - коллекция данных, которая храниться на постоянном носителе информации, таком как жесткий диск, флеш-накопитель, CD-ROM и др. Файлы используются для хранения информации различного типа, включая:

- ❖ текст
- ❖ изображения
- ❖ звуковые файлы
- ❖ программы
- ❖ и др.

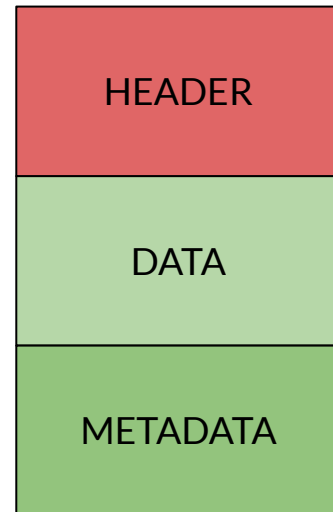


# Структура файла;

**Заголовок (Header):** Заголовок файла содержит информацию о его формате, структуре и другие метаданные, которые могут быть необходимы для его интерпретации. Например, текстовый файл может содержать информацию о кодировке символов, а бинарный файл - о структуре данных.

**Данные (Data):** Это основная часть файла, которая содержит фактические данные, которые пользователь или программа хочет хранить или обрабатывать. В текстовом файле данные представлены в виде символов, а в двоичном файле - в виде байтов.

**Метаданные (Metadata):** Это информация, связанная с файлом, которая может включать в себя различные атрибуты файла, такие как его размер, дата создания, дата последнего доступа и дата последнего изменения. Кроме того, метаданные могут включать разрешения доступа к файлу, владельца файла и другие системные атрибуты.



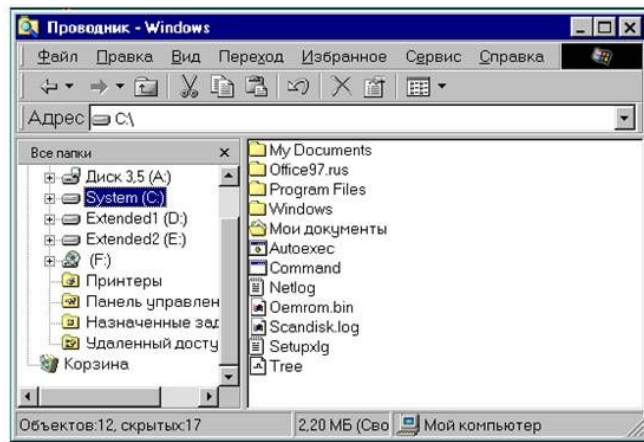
# Кодировки файла;

Кодировка файла определяет способ представления символов и текста в файле. Вот некоторые из наиболее распространенных кодировок файлов:

- ❖ **UTF-8:** Это переменная длина, многобайтовая кодировка Unicode. Она использует от одного до четырех байтов для представления символов Unicode. UTF-8 является наиболее распространенной кодировкой в Интернете.
- ❖ UTF-16: Это также переменная длина, многобайтовая кодировка Unicode. Она использует два или четыре байта для представления символов Unicode. UTF-16 часто используется в Windows-системах.
- ❖ UTF-32: Это фиксированная длина, многобайтовая кодировка Unicode. Она использует четыре байта для представления каждого символа Unicode.
- ❖ **ASCII:** Это семибитная кодировка, которая представляет основной набор символов на английском языке. Она использует один байт для каждого символа.
- ❖ ISO-8859-1 (Latin-1): Это европейская однобайтовая кодировка, которая включает в себя символы для большинства западноевропейских языков.
- ❖ Windows-1251: Это однобайтовая кодировка, распространенная в операционных системах Microsoft Windows, которая поддерживает символы кириллицы.
- ❖ KOI8-R: Это стандартная семибитная кодировка, используемая для представления русских букв в операционных системах Unix и Linux.

# Работа с файлами;

Работа с файлами является важной частью программирования на C++. Файлы используются для хранения данных на постоянном носителе и для обмена информацией между программами. В этой лекции мы рассмотрим основные аспекты работы с файлами в C++, включая открытие, чтение, запись и закрытие файлов.



# std::fstream;

В C++ для работы с файлами используется класс **std::fstream**, который предоставляет возможности как для чтения, так и для записи данных в файлы. Для открытия файла используется метод **open()**:

```
std::fstream file;
```

# Методы `std::fstream`;

- ❖ `open()`: Открывает файл для чтения и/или записи.
- ❖ `is_open()`: Проверяет, открыт ли файл.
- ❖ `close()`: Закрывает файл.
- ❖ `clear()`: Очищает состояние потока, обычно используется после ошибки чтения/записи.
- ❖ `eof()`: Проверяет, достигнут ли конец файла.
- ❖ `good()`: Проверяет, что состояние потока не содержит ошибок.
- ❖ `bad()`: Проверяет, что произошла серьезная ошибка в потоке.
- ❖ `fail()`: Проверяет, что произошла неудачная операция ввода/вывода.

# Методы `std::fstream`;

- ❖ `peek()`: Возвращает следующий символ без его извлечения из потока.
- ❖ `tellg()`: Возвращает текущую позицию в потоке ввода.
- ❖ `tellp()`: Возвращает текущую позицию в потоке вывода.
- ❖ `seekg()`: Перемещает указатель ввода на определенную позицию в файле.
- ❖ `seekp()`: Перемещает указатель вывода на определенную позицию в файле.
- ❖ `write()`: Записывает данные в файл.
- ❖ `read()`: Читает данные из файла.
- ❖ `getline()`: Считывает строку из файла.



# Режимы открытия;

**Режимы открытия файлов** в C++ определяются с помощью флагов, определенных в перечислении **std::ios**. Вот некоторые из наиболее часто используемых режимов открытия файлов:

- ❖ `std::ios::in`: Открыть файл для чтения.
- ❖ `std::ios::out`: Открыть файл для записи.
- ❖ `std::ios::binary`: Открыть файл в бинарном режиме (для работы с двоичными данными).
- ❖ `std::ios::app`: Добавить новые данные в конец файла вместо перезаписи.
- ❖ `std::ios::ate`: Установить указатель в конец файла после открытия.
- ❖ `std::ios::trunc`: Усечь файл при открытии (если файл существует, его содержимое будет удалено).

*Эти флаги можно комбинировать, используя оператор побитового **ИЛИ (|)**, чтобы указать несколько режимов открытия файлов одновременно. Например, **std::ios::in | std::ios::out** открывает файл как для чтения, так и для записи.*

# Пример открытия файла;

```
std::fstream file;  
file.open("example.txt", std::ios::out); // Открытие файла для записи  
if (!file.is_open()) {  
    std::cout << "Ошибка открытия файла!" << std::endl;  
    return 1;  
}  
  
std::cout << "Файл успешно открыт для записи." << std::endl;  
// `todo: операции с файлом  
file.close(); // Заккрытие файла
```

# Пример записи1 файла;

Для записи данных в файл используются операторы <<, такие же, как и для стандартного вывода:

```
file << "Это строка, которую мы записываем в файл." << std::endl;
```

# std::ofstream;

**std::ofstream** является частью стандартной библиотеки C++ (STL) и представляет собой класс, который используется для записи данных в файл. Он является частью иерархии классов файлового ввода-вывода, предоставляемой в C++.

В частности, **std::ofstream** является частным случаем класса **std::fstream**, который позволяет открывать файлы для записи (и только для записи), в отличие от **std::fstream**, который позволяет открывать файлы как для чтения, так и для записи. Таким образом, **std::ofstream** предоставляет интерфейс, оптимизированный исключительно для операций записи.

# Методы `std::ofstream`;

- ❖ `open(const char* filename, ios_base::openmode mode = ios_base::out):`  
Открывает файл для записи.
- ❖ `is_open() const:` Проверяет, открыт ли файл.
- ❖ `close():` Закрывает файл.
- ❖ `operator<<():` Позволяет записывать данные в файл с использованием синтаксиса оператора `<<`.
- ❖ `write(const char* s, streamsize n):` Записывает блок данных в файл.
- ❖ `flush():` Сбрасывает буфер, записывая все данные в файл немедленно.
- ❖ `good() const:` Проверяет, что состояние потока не содержит ошибок.
- ❖ `bad() const:` Проверяет, что произошла серьезная ошибка в потоке.
- ❖ `fail() const:` Проверяет, что произошла неудачная операция ввода/вывода.

# Пример запись2 файла;

Для записи массива данных в файл в C++ вы можете воспользоваться методом **write()** объекта класса `std::fstream`. Вот пример, как это можно сделать:

```
// Создаем массив данных для записи
int data[] = {1, 2, 3, 4, 5};

// Открываем файл для записи в бинарном режиме
std::ofstream outFile("data.bin", std::ios::binary);

// Проверяем, открыт ли файл
if (!outFile.is_open()) {
    std::cerr << "Ошибка открытия файла!" << std::endl;
    return 1;
}
```

```
// Записываем массив в файл
outFile.write(reinterpret_cast<const char*>(data), sizeof(data))

// Проверяем, прошла ли запись успешно
if (!outFile.good()) {
    std::cerr << "Ошибка записи данных в файл!" << std::endl;
    outFile.close();
    return 1;
}

std::cout << "Массив данных успешно записан в файл." << std::endl;

// Закрываем файл
outFile.close();
```

# Пример чтение файла;

```
std::string line;
while (std::getline(file, line)) {
    std::cout << line << std::endl;
}
```

Этот код читает файл построчно и выводит каждую строку на экран.

- ❖ Создается переменная **line** типа `std::string` для хранения текущей считанной строки.
- ❖ В цикле **while** используется функция `std::getline()`, которая считывает строку из файла (**file**) и сохраняет ее в переменную **line**. Цикл продолжается, пока успешно удастся считать новую строку.
- ❖ Внутри цикла каждая строка выводится на экран с помощью `std::cout`.
- ❖ Процесс повторяется, пока не будет достигнут конец файла или не произойдет ошибка ввода/вывода.

---

# Работа с CSV

C++

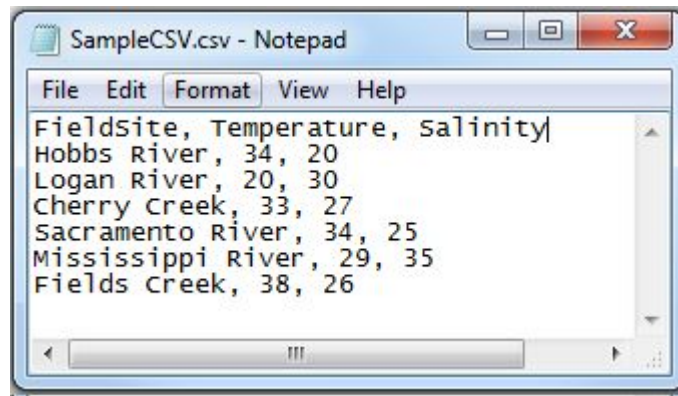
---



---

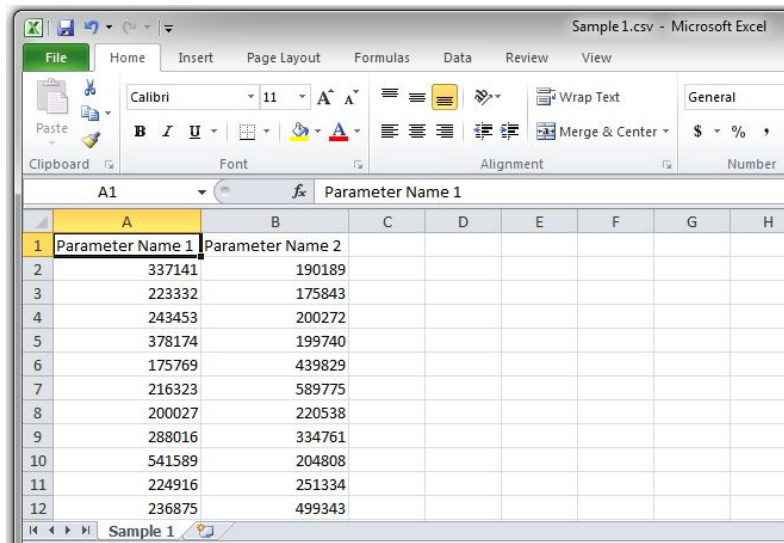
## <csv file>

**CSV (Comma-Separated Values)** - это формат текстовых файлов, используемый для представления табличных данных. Каждая строка файла представляет собой одну запись таблицы, а значения разных столбцов разделены запятыми или другими разделителями (например, точкой с запятой). CSV файлы обычно используются для обмена данными между различными программами, поскольку они легко читаемы как людьми, так и компьютерами. Чтение и запись CSV файлов в C++ обычно выполняется с помощью стандартных средств языка, таких как потоки ввода-вывода (ifstream и ofstream).



# <table data>

**Table Data** - таблица данных, которая содержит информацию представленную в виде специализированной таблицы (строки и столбцы)



Sample 1.csv - Microsoft Excel

	A	B	C	D	E	F	G	H
1	Parameter Name 1	Parameter Name 2						
2	337141	190189						
3	223332	175843						
4	243453	200272						
5	378174	199740						
6	175769	439829						
7	216323	589775						
8	200027	220538						
9	288016	334761						
10	541589	204808						
11	224916	251334						
12	236875	499343						

---

# Чтение данных.

Предположим есть таблица данных:

```
struct Row {  
    string name;  
    int age;  
};
```

row1  
row2  
...  
rowN



Имя	Возраст
Алексей	22
Петр	34
Катя	41
Вера	25
...	...

---

---

# Код чтение данных

```
Std::vector<Row> readCSV(const string& filename) {  
    Std::vector<Row> data;  
    ifstream file(filename);  
    if (!file.is_open()) {  
        cerr << "Unable to open file: " << filename << endl;  
        return data;  
    }  
    string line;  
    while (getline(file, line)) {  
        stringstream ss(line);  
        Row row;  
        getline(ss, row.name, ',');  
        ss >> row.age;  
        data.push_back(row);  
    }  
    file.close();  
    return data;  
}
```

---

---

# <main>

```
int main() {  
    Std::vector<Row> data = readCSV("data.csv");  
  
    // Вывод данных  
    for (const auto& row : data) {  
        cout << "Name: " << row.name << ", Age: " << row.age  
<< endl;  
    }  
    return 0;  
}
```

---

---

# Запись данных

```
// Функция для записи данных в CSV файл
void writeCSV(const string& filename, const Std::vector<Row>& data) {
    ofstream file(filename);

    if (!file.is_open()) {
        cerr << "Unable to open file: " << filename << endl;
        return;
    }

    // Записываем заголовки столбцов (если нужно)
    file << "Name, Age" << endl;
    // Записываем данные из вектора структур в файл
    for (const auto& row : data) {
        file << row.name << "," << row.age << endl;
    }
    file.close();
}
```

---

---

# Объединение двух csv файлов в один;

```
void mergeCSV(const string& input1, const string& input2, const string& output) {  
    // Читаем данные из CSV файла  
    Std::vector<Row> data1 = readCSV(input1);  
    Std::vector<Row> data2 = readCSV(input2);  
  
    // Объединяем данные  
    for (const auto& row : data2) {  
        data1.push_back(row);  
    }  
  
    // Записываем объединенные данные в новый файл  
    writeCSV(output, data1);  
}
```

---