**1. Текстовый файл. Его особенности.**  
Текстовый файл хранит данные в виде последовательности символов, представленных в определённой кодировке (например, ASCII или UTF-8). Он обычно содержит символы перевода строки (например, \n или \r\n), что делает его удобным для чтения человеком. Особенность текстовых файлов в том, что они подвергаются преобразованию символов (например, перевод строк) при записи и чтении, что может отличаться на разных платформах.

**2. Отличия текстовых и двоичных файлов с точки зрения представления данных.**

* **Текстовые файлы:**  
  Хранят данные как набор символов. При записи/чтении происходит преобразование специальных символов (например, перевод строки). Они удобны для редактирования в текстовых редакторах.
* **Двоичные файлы:**  
  Хранят данные в виде «сырых» байтов без какого-либо преобразования. Нет обработки переводов строки или кодировки – данные читаются и записываются как есть. Это делает их подходящими для хранения изображений, аудио, видео и других структурированных данных.

**3. Перечислите стандартные потоки ввода-вывода.**  
Стандартными потоками являются:

* std::cin – стандартный поток ввода.
* std::cout – стандартный поток вывода.
* std::cerr – поток для вывода сообщений об ошибках (небуферизуемый).
* std::clog – поток для вывода логов (буферизуемый).  
  В C также существуют stdin, stdout и stderr.

**4. Файловый поток, виды, создание.**  
В C++ для работы с файлами используются классы из библиотеки <fstream>, которые подразделяются на:

* **std::ifstream** – поток для чтения (input file stream).
* **std::ofstream** – поток для записи (output file stream).
* **std::fstream** – поток, поддерживающий и чтение, и запись.

Файл можно открыть либо сразу при создании объекта (через конструктор с указанием имени файла и режима), либо создать объект и вызвать метод open(). Например:

cpp

CopyEdit

#include <fstream>

using namespace std;

int main() {

ofstream out("data.txt", ios::out); // открытие файла для записи

if (!out.is\_open()) {

// обработка ошибки

}

// запись данных

out << "Hello, file!";

out.close(); // закрытие файла

return 0;

}

**5. Основные алгоритмы работы с файлами (запись и чтение).**

* **Запись в файл:**
  1. Создать объект файлового потока (например, ofstream) и открыть файл в нужном режиме.
  2. Записать данные с помощью оператора <<, метода write(), или других функций.
  3. Закрыть файл (или позволить деструктору выполнить закрытие).
* **Чтение из файла:**
  1. Создать объект (например, ifstream) и открыть файл в режиме чтения.
  2. Считывать данные с помощью оператора >>, метода getline() или read(), используя цикл до достижения конца файла.
  3. Закрыть файл после завершения работы.

**6. Основные методы класса ifstream.**

* open(const char\* filename, ios::openmode mode = ios::in) – открыть файл.
* close() – закрыть файл.
* is\_open() – проверить, открыт ли файл.
* get() – считывание одного символа.
* getline() – считывание строки до символа перевода строки.
* read() – чтение указанного количества байтов (чаще используется для двоичных файлов).
* eof() – проверка достижения конца файла.
* peek() – просмотр следующего символа без его извлечения.

**7. Основные методы класса ofstream.**

* open(const char\* filename, ios::openmode mode = ios::out) – открыть файл для записи.
* close() – закрыть файл.
* is\_open() – проверить успешное открытие файла.
* put() – вывод одного символа в файл.
* write() – запись массива байтов в файл.
* flush() – принудительная запись буферизованных данных на диск.
* Перегруженный оператор << для записи данных в текстовом формате.

**8. Какие значения возвращает метод проверки открытия файла в зависимости от результата?**  
Метод is\_open() возвращает true, если файл успешно открыт, и false в противном случае. Кроме того, можно использовать поток в логическом контексте (например, if (file)), что также проверяет корректность открытия.

**9. Какие значения возвращает метод проверки достижения конца файла в зависимости от результата?**  
Метод eof() возвращает true, если достигнут конец файла, и false если ещё есть данные для чтения.

**10. Какие режимы открытия файлов существуют?**  
Основные режимы (константы из ios):

* ios::in – открыть для чтения.
* ios::out – открыть для записи.
* ios::app – открыть для добавления в конец файла (append).
* ios::ate – открыть файл и сразу переместить указатель в конец (но запись возможна в любом месте).
* ios::binary – открыть файл в двоичном режиме.
* ios::trunc – обрезать файл при открытии (удалить старое содержимое).

Режимы можно комбинировать с помощью побитового ИЛИ (|).

**11. Возможные причины ошибок при открытии файлов для чтения и записи.**

* Неверно указан путь к файлу или имя файла.
* Файл не существует (при открытии в режиме чтения).
* Отсутствие прав доступа (например, файл защищён от записи или чтения).
* Файл занят другим процессом или заблокирован.
* Недостаточно места на диске (при записи).
* Ограничения операционной системы (например, максимальное число одновременно открытых файлов).

**12. Можно ли один и тот же файл открыть несколько раз, не закрывая после каждого открытия?**  
Да, можно создать несколько потоковых объектов, открывающих один и тот же файл, однако это может привести к конфликтам, особенно при записи. Если несколько потоков одновременно читают или записывают в один файл, то нужно внимательно следить за синхронизацией и корректностью работы, так как результаты могут быть непредсказуемыми.

**13. Можно ли один и тот же файл одновременно открыть для записи и для чтения? Если да, то где будет находиться файловый указатель?**  
Да, это возможно с использованием класса std::fstream, который позволяет работать в обоих режимах, задавая режим открытия как ios::in | ios::out (и, возможно, дополнительно другие режимы, например, ios::binary). При открытии с такими флагами файловый указатель, как правило, устанавливается в начале файла, если не указан флаг ios::ate (в этом случае указатель сразу переходит в конец).

**14. Сохранится ли информация в файле, если его не закрыть в программе с помощью функции? Обоснуйте ответ.**  
В большинстве случаев при завершении работы программы деструктор объекта файлового потока автоматически вызовет метод close(), что приведёт к сбросу (flush) буферов и записи оставшихся данных на диск. Однако, если программа аварийно завершится (например, из-за ошибки или сбоя), данные, находящиеся в буфере, могут не сохраниться. Поэтому рекомендуется явно закрывать файлы после завершения работы с ними.

**15. Как осуществляется чтение символов и строк из файла?**

* **Символы:**  
  Можно использовать метод get(), оператор >> или peek() для последовательного чтения отдельных символов.
* **Строки:**  
  Чтение целых строк обычно осуществляется с помощью метода getline(), который считывает символы до символа перевода строки и автоматически добавляет завершающий нулевой символ в C-строках (или формирует объект std::string).

**16. Что такое последовательный доступ к текстовым файлам?**  
Последовательный (линейный) доступ означает чтение или запись данных в файле строго от начала к концу. При таком доступе данные обрабатываются последовательно – от первого байта до последнего, без возможности мгновенного перехода к произвольной позиции в файле.

**17. В чем особенность произвольного доступа к текстовым файлам?**  
Произвольный (случайный) доступ позволяет перемещаться по файлу и читать/записывать данные из любых позиций с помощью функций перемещения указателя (например, seekg() для чтения и seekp() для записи). Однако для текстовых файлов такой доступ может быть неточным, так как символы (в особенности символы перевода строки) могут иметь различную длину представления в зависимости от платформы, что требует осторожного использования и понимания внутреннего устройства файла.