**Режимы работы с файлами.**

Режимы открытия файлов

Режимы открытия файлов устанавливают характер использования файлов. Для установки режима в классе ios\_base предусмотрены константы, которые определяют режим открытия файлов (см. Таблица 1).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | Таблица 1 — режимы открытия файлов | | | --- | --- | | **Константа** | **Описание** | | **ios\_base::in** | открыть файл для чтения | | **ios\_base::out** | открыть файл для записи | | **ios\_base::ate** | при открытии переместить указатель в конец файла | | **ios\_base::app** | открыть файл для записи в конец файла | | **ios\_base::trunc** | удалить содержимое файла, если он существует | | **ios\_base::binary** | открытие файла в двоичном режиме | |
|  |

Режимы открытия файлов можно устанавливать непосредственно при создании объекта или при вызове функции open().

|  |  |
| --- | --- |
| 1  2 | ofstream fout("cppstudio.txt", ios\_base::app); // открываем файл  для добавления информации к концу файла  fout.open("cppstudio.txt", ios\_base::app); // открываем файл  для добавления информации к концу файла |

Режимы открытия файлов можно комбинировать с помощью поразрядной логической операции или |, например: ios\_base::out | ios\_base::trunc — открытие файла для записи, предварительно очистив его.

Объекты класса ofstream, при связке с файлами по умолчанию содержат режимы открытия файлов  ios\_base::out | ios\_base::trunc. То есть файл будет создан, если не существует. Если же файл существует, то его содержимое будет удалено, а сам файл будет готов к записи. Объекты класса ifstream связываясь с файлом, имеют по умолчанию режим открытия файла   ios\_base::in  — файл открыт только для чтения. Режим открытия файла ещё называют — флаг, для удобочитаемости в дальнейшем будем использовать именно этот термин.

Обратите внимание на то, что флаги ate и app по описанию очень похожи, они оба перемещают указатель в конец файла, но флаг app позволяет производить запись, только в конец файла, а флаг ate просто переставляет флаг в конец файла и не ограничивает места записи.

В С++ нету типа byte , а вместо типа byte используют тип char. Согласно правилам языка С++, тип char оказывается наименьшей единицей информации. Из-за этого восприятие нами ситуации в целом немного искажается, ведь мы видим char\*, а на самом деле это как byte\*

В наших компьютерах используется такая модель памяти, где одна ячейка занимает 1 байт. А один символ char гарантировано, как утверждают в стандарте, должен залезать в эту ячейку. Поэтому, в зависимости от ситуации, тип char можно трактовать или как символьный тип, или как байтовый тип. Файлы, хранящие последовательность байтов, называют бинарными файлами. Это связано с тем, что байт делится на биты, а биты представляют собой двоичные цифры. Слово бинарный переводится как двоичный.

Использование двоичного режима файла приводит к тому, что данные пересылаются из памяти в файл или, наоборот, без какого-либо их преобразования. Чтобы сохранять данные в двоичном формате, при создании потокового объекта (или при открытии) необходимо указать режим ios::binary.

В отличие от текстового режима, этот режим должен быть задан явно. При явном указании режима требуется определить все режимы открытия файла, соединив их поразрядной операцией | (или).

ifstream fin(nameF, ios::in | ios::binary);

ofstream fout(nameF,ios::app |ios::binary);

fstream finout(nameF, ios::in | ios::out | ios::binary);

Для записи данных в двоичном формате используется метод write():

fout.write((char\*)&X,sizeof X);

Этот метод копирует определенное число байтов из памяти в файл. Количество байтов, которое должно быть скопировано, задается вторым параметром. Первый параметр определяет адрес, где расположены данные, которые необходимо скопировать. Особенностью метода является то, что адрес должен быть преобразован к типу «указатель на char».

Для чтения из двоичного файла используется метод read(), имеющий такой же список параметров, как и метод write():

fin.read((char\*)&Y,sizeof Y);

Данный метод возвращает значение true, если операция чтения завершилась нормально, и false — в случае возникновения ошибки, например, при достижении конца файла.

Код C++ Записать число в бинарный файл. Прочитать число из бинарного файла  
=================  
#include <iostream> //Для работы с клавиатурой  
#include <fstream> //Для работы с файловыми потоками

using namespace std;

int main()  
{ int y=0; //Y будем записывать в файл  
 int x=0;   //X будем считывать из файла

cout<<“Y = “;cin>>y;  //Вводим число, которое нужно сохранить в файл  
 ofstream out(“1.txt”,ios::binary|ios::out);   //Открываем файл в двоичном режиме для записи  
   out.write((char\*)&y,sizeof y);  //Записываем в файл число y  
 out.close();  //Закрываем файл

 cout<<“x = “<<x<<endl;  //Показываем X до его изменений

 ifstream in(“1.txt”,ios::binary|ios::in);  //Открываем файл в двоичном режиме только для чтения  
   in.read((char\*)&x,sizeof x);  //Читаем оттуда информацию и запоминаем её в X  
 in.close();  //Закрываем файл

 cout<<“ X=“<<x<<endl;  //Показываем X после изменения  
}

Чтобы мы могли записать какое-нибудь значение в бинарном представлении, нам нужно для начала вывести это бинарное представление, а чтобы записалось правильное количество байт, нужно явно указывать это количество. Это выглядит приблизительно следующим образом:

======================================================  
out.write((char\*)&X,sizeof X);  //Записываем объект X в открытый нами файл

Узнаем адрес объекта X и приводим объект Х к однобайтовому типу

=======================================================  
in.read((char\*)&Y,sizeof Y);  //Считываем информацию в объект Y  
Узнаем адрес объекта Y и кладем в объект Y каждый новый прочитанный байт.  
Инструктируем, что нам в объект Y нужно положить sizeof Y байт  
=======================================================

**Методы позиционирования.**

Поскольку при организации внешней таблицы файл состоит из записей одинакового размера, легко обеспечить доступ к записи с определенным номером.

Для этого используются методы:

seekg() — с объектами классов ifstream и fstream,

и seekp() — с объектами классов ofstream и fstream.

Произвольный доступ можно одинаково задействовать и для файлов, открытых в текстовом режиме, и для файлов, открытых в бинарном режиме.

В С++ используется две разновидности файловых курсоров: курсор чтения и курсор записи. В зависимости от действия: чтение из файла или запись в файл — выбирается seekp или seekg соответственно. В некоторых случаях они работают одинаково, но не стоит использовать их не по назначению. Кроме того, что можно принудительно установить курсор файла в любую произвольную позицию внутри файла, можно узнавать текущую позицию курсора. Здесь так же нужно Методы позиционирования разделять курсор чтения и курсор записи tellp и tellg.

Методы позиционирования различаются для входных и выходных потоков: для входных потоков имена методов заканчиваются символом 'g' (от слова get), а методы выходных потоков заканчиваются символом 'p' (от слова put).

**Seekg** установить позицию чтения

**Seekp** установить позицию записи

**Tellg** узнать позицию курсора чтения

**Tellp** узнать позицию курсора записи

Удобнее всего с позиционированием работать в бинарных файлах. В текстовых файлах сложнее отслеживать нужную позицию, если, например, в текстовом файле записаны числа. Поскольку число состоит из нескольких цифр, то сколько на какое число нужно скакнуть, расчитывать утомительно и невыгодно. В бинарном же файле более простое для компьютера представление и нам достаточно знать количество байт, занимаемых типом, чтобы сделать скачок на нужную позицию в таком файле и необходимо знать порядок следования байт.

Параметры :

1. ios\_base::beg (смещение от начала).
2. ios\_base::cur (смещение от текущей позиции).
3. ios\_base::end (смещение от конца).

file.seekg(0,ios\_base::end); //Стать в конец файла

file.seekg(10,ios\_base::end); //Стать на 10 байтов с конца

file.seekg(30,ios\_base::beg); //Стать на 31-й байт

file.seekg(3,ios\_base::cur); //перепрыгнуть через 3 байта

file.seekg(3); //перепрыгнуть через 3 байта - аналогично

Версия функции seekg() с одним параметром перемещают указатели файлов в позиции, заданные параметром . Это значение должно быть предварительно получено путем обращения либо к функции tellg(), либо к функции tellp() соответственно.

**Метод tellg()**

Иногда нужно получать информацию о том, сколько уже прочитано. В этом поможет метод tellg():

|  |  |
| --- | --- |
| 1 | cout << "Считано байт: " << file.tellg(); |

Он возвращает значение типа int, которое показывает сколько уже пройдено в байтах. Его можно использовать в паре с методом seekg(), чтоб получать размер файла:

|  |  |
| --- | --- |
| 1  2  3  4 | //становимся в конец файла  file.seekg(0,ios\_base::end);  //Получаем текущую позицию  cout << "Размер файла (в байтах): " << file.tellg(); |

**Метод eof()**

Проверяет не достигнут ли конец файла. Т.е. можно ли из него продолжать чтение. Выше пример с считкой слов оператором >> как раз использует такую проверку.