

Работа с файлами

Работа с файлами

- Для работы с файлами необходимо подключить заголовочный файл `<fstream>`.
- Файловый ввод/вывод аналогичен стандартному вводу/выводу, единственное отличие – это то, что ввод/вывод выполнятся не на экран, а в файл.
- Если ввод/вывод на стандартные устройства выполняется с помощью объектов `cin` и `cout`, то для организации файлового ввода/вывода достаточно создать **собственные объекты**, которые можно использовать аналогично операторам `cin` и `cout`.

Запись в файл

Запись в файл

Для того, чтобы записать строку в текстовый файл нужно выполнить:

1. создать объект класса ofstream;
2. связать объект класса с файлом, в который будет производиться запись;
3. записать строку в файл;
4. закрыть файл.

Запись в файл

```
#include <fstream>
using namespace std;
void main() {

    // создаём объект класса ofstream для записи
    ofstream file;

    // связываем его с файлом test.txt
    file.open("test.txt");

    // запись строки в файл
    file << "Запись строки в текстовый файл.";

    // закрываем файл
    file.close();
}
```

Запись в файл. Оператор <<

Перенаправляет форматированный вывод в файл

.

Принцип тот же, что и у аналога из `iostream`.

```
// Строка
string s = "Привет мир";

// Вещественное
double d = 123.456;

// Запись
file << s << endl << d << endl;
```

Предназначен для вывода в текстовые файлы.

Управляется операциями форматирования такими как **width()** или **setf()**.

Запись в файл

```
#include <fstream>
using namespace std;

void main()
{
    // создаём объект класса ofstream для записи и связываем
    // его с файлом test.txt
    ofstream file("test.txt");

    // запись строки в файл
    file << "Запись строки в текстовый файл.";

    // закрываем файл
    file.close();
}
```

Запись в файл. Метод write

Используется в **бинарных** файлах для записи блока памяти (массива байт) в файл как они есть.

```
// Запись побайтно
```

```
// Строки
```

```
char *sc = "Строка текста\n";  
file.write(sc, strlen(sc));
```

```
// Целого в машинном представлении
```

```
int k = 123;  
file.write((char*)&k, sizeof(k));
```

```
// Вещественного в машинном представлении
```

```
double dd = 456.789;  
file.write((char*)&dd, sizeof(dd));
```

метод **write()** не выводит данные в текстовом представлении

Запись в файл. Метод `close`

- Закрывает файл метод `close()`.
- Для файлов, открытых на запись, в отличие от файлов на чтение, закрытие файла – обязательный ритуал.
- Незакрытый файл может не получить данные.
- Происходить такой эффект может из-за буферизации самой операционной системы, когда данные, сбрасываемые в файл, хранятся на самом деле в памяти и сразу в файл не поступают.
- Операционная система сама решает, когда пора записывать данные.

Запись в файл. Метод `close`

- Такая “отложенная” запись называется “Коммитом” (от латинского `commit`).
- Этим эффектом весьма удачно пользуются системы управления базами данных, где вставляемые записи попадают в хранилище в памяти (называемой транзакцией).
- И только после специальной команды скопом пишутся в сам файл базы.
- Метод `close()` как раз пример такой команды закрывающей транзакцию вместе с файлом.

Запись в файл. Методы `close` и `flush`

если нужно произвести коммит данных без закрытия самого файла, нужно применить метод `flush()`:

```
file.flush();
```

отложенные на запись данные поступят в файл, но он еще будет открыт для записи.

Запись в файл. Методы форматирования

Для красивой разметки данных в файле могут применяться методы форматирования данных для вывода оператором `<<` .

- **width()** указывает ширину в символах, в которое будет укладываться выводимое значение,
- **precision()** количество знаков дробной части вещественного.

Запись в файл. Методы форматирования. Пример. Часть 1

```
#include <fstream>
using namespace std;

int main()
{
    // открытие файла в конструкторе
    ofstream file("test.txt");

    // Форматирование вывода
    double d = 1; int i;

    for (i = 0; i < 10; i++) {
        d += sin(i / d);
        // Указываем ширину ячейки для целого и выводим целое
        file.width(4);
        // Выводим целое
        file << i;
    }
}
```

Запись в файл. Методы форматирования. Пример

```
// Указываем ширину ячейки для вещественного и
// кол-во знаков после запятой максимум
    file.width(12);
    file.precision(5);

// выводим вещественное
    file << d;

// Переводим каретку на новую строку
    file << endl;
}
file.close();
}
```

Чтение из файла

Чтение из файла

Для того, чтобы прочитать строку из текстового файла нужно выполнить:

1. создать объект класса **ifstream** и связать его с файлом, из которого будет производиться считывание;
2. прочитать файл;
3. закрыть файл.

Чтение из файла. Открытие с помощью метода open

```
#include <iostream>
// подключение библиотеки
#include <fstream>
using namespace std;

int main()
{
    // создание объекта класса ifstream
    ifstream file;

    // открытие файла
    file.open("d:\\my\\test.txt");
}
```

Чтение из файла. Открытие в конструкторе

```
#include <iostream>
// подключение библиотеки
#include <fstream>
using namespace std;

int main()
{
    // открытие файла в конструкторе
    ifstream file("d:\\my\\test.txt");
}
```

Чтение из файла. Оператор считывания >>

Так же как и в `iostream` считывание можно организовать оператором `>>`, который указывает в какую переменную будет произведено считывание:

```
double d;  
int i;  
string s;  
  
file >> d >> i >> s;
```

Считает вещественное, целое и строку. Считывание строки закончится, если появится пробел или конец строки.

Оператор `>>` применяется к текстовым файлам.

Чтение из файла. Метод eof()

Оператор `>>` удобен, если стоит задача разделить файл на слова.

Метод `eof` проверяет не достигнут ли конец файла. Т.е. можно ли из него продолжать чтение.

```
// Считывание слов из файла
for (file >> s; !file.eof(); file >> s)
{
    cout << s << endl;
}
```

Чтение из файла. Методы `getline()` и `get()`

- Считывание целой строки до перевода каретки производится так же как и в `iostream` методом `getline()`.
- Причем рекомендуется использовать его переопределенную версию в виде **функции**, если считывается строка типа `string`:

```
//Считывание строки из текстового файла  
string s;  
getline(file, s);  
cout << s << endl;
```

Чтение из файла. Методы `getline()` и `get()`

- Если же читать нужно в массив символов `char[]`, то нужно использовать либо `get()` либо `getline()` именно как методы:

```
int n = 10;
//Создаем буффер для чтения
char* buffer = new char[n + 1];
buffer[n] = 0;

//n – максимальное кол-во считываемых символов
file.get(buffer, n);

//пробел – разделитель до которого производится считывание
file.getline(buffer, n, ' ');

//выводим считанное
cout << buffer;
//Освобождаем буффер
delete[] buffer;
```

Чтение из файла. Чтение строки

```
#include <fstream>
using namespace std;
void main()
{
// буфер промежуточного хранения считываемого из файла текста
char buff[80];

// открыли файл для чтения
ifstream file("test.txt");

// считали строку из файла
file.getline(buff, 80);

// закрываем файл
file.close();

// напечатали эту строку ...
}
```

Чтение из файла. Чтение слов

```
#include <iostream>
#include <fstream>
using namespace std;
void main() {
// буфер промежуточного хранения считываемого из файла текста
char buff[80];

// открыли файл для чтения
ifstream file("test.txt");

// считали первое слово из файла
file >> buff;
// напечатали это слово
cout << buff << endl;

// считали второе слово из файла ...
// закрываем файл
file.close();
}
```


Чтение из файла. Метод read

```
//Считывание из файла N байт
int n = 10;

//Создаем буффер
char* buffer = new char[n + 1];
buffer[n] = 0;

//Читаем в него байты
file.read(buffer, n);

//выводим их на экран
cout << buffer;

delete[] buffer;
```

- Метод **read** обычно применяется для бинарных файлов.
- В этом методе нельзя указать разделитель (как в `getline`).
- **read** применяется для неформатированного ввода.
- метод **read** можно применять и к текстовому файлу.

Работа с файлами. Функция `is_open`

```
#include <iostream>
#include <fstream>
using namespace std;
void main() {

char buff[80];
ifstream fin("test.txt");

if (!fin.is_open()) // если файл не открыт
    cout << "Файл не может быть открыт!\n";
else
{
    // считали строку из файла
    fin.getline(buff, 80);
    // вывели строку на экран
    cout << buff << endl;
    // закрываем файл
    fin.close();
}}
```

Режимы открытия файлов

Режимы открытия файлов устанавливают характер использования файлов.

Для установки режима в классе `ios_base` предусмотрены константы, которые определяют режим открытия файлов.

Константа	Описание
<code>ios_base::in</code>	открыть файл для чтения
<code>ios_base::out</code>	открыть файл для записи
<code>ios_base::ate</code>	при открытии переместить указатель в конец файла
<code>ios_base::app</code>	открыть файл для записи в конец файла
<code>ios_base::trunc</code>	удалить содержимое файла, если он существует
<code>ios_base::binary</code>	открытие файла в двоичном режиме

Режим открытия файла называют флагом.

Режимы открытия файлов. Установка режима

```
#include <fstream>
using namespace std;

void main()
{
    setlocale(0, "");

    // связываем объект с файлом,
    // при этом файл открываем в режиме записи,
    // предварительно удаляя все данные из него

    ofstream file("data.txt", ios_base::out | ios_base::trunc);

    // запись строки в файл
    file << "Установка режима открытия файла.";

    // закрываем файл
    file.close();
}
```

Работа с файлами. Чтение всех строк из файла

```
char str[80];  
  
ifstream file("data.txt");  
  
while (true) {  
    if (file.eof()) break;  
    file.getline(str, sizeof(str));  
  
    cout << str << endl;  
}
```

Работа с файлами. Чтение всех строк из файла

```
char str[80];  
ifstream file("data.txt");  
  
while (!file.eof()) {  
    file.getline(str, sizeof(str));  
    cout << str << endl;  
}
```

Работа с файлами. Произвольный доступ к файлу

Система ввода-вывода C++ позволяет осуществлять произвольный доступ с использованием методов

seekg() и **seekp()**.

Система ввода-вывода C++ обрабатывает **два указателя**, ассоциированные с каждым файлом:

- **get pointer g** - определяет, где именно в файле будет производиться следующая операция **ввода**;
- **put pointer p** - определяет, где именно в файле будет производиться следующая операция **вывода**.

Работа с файлами. Произвольный доступ к файлу

Позиция смещения определяется как

Позиция	Значение
<code>ios::beg</code>	Начало файла
<code>ios::cur</code>	Текущее положение
<code>ios::end</code>	Конец файла

Всякий раз, когда осуществляются операции ввода или вывода, соответствующий указатель автоматически перемещается.

С помощью методов `seekg()` и `seekp()` можно получить доступ к файлу в произвольном месте.

Произвольный доступ к файлу. Метод `seekg()`

Метод `seekg()` производит установку текущей позиции в нужную, указываемую числом.

В этот метод так же передается способ позиционирования:

- `ios_base::end` – отсчитать новую позицию с конца файла
- `ios_base::beg` – отсчитать новую позицию с начала файла (абсолютное позиционирование)
- `ios_base::cur` – перескочить на `n` байт начиная от текущей позиции в файле (по умолчанию)

Произвольный доступ к файлу. Метод seekg()

```
//Стать в конец файла
file.seekg(0, ios_base::end);

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

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

//перепрыгнуть через 3 байта
file.seekg(3, ios_base::cur);
//перепрыгнуть через 3 байта
file.seekg(3);
```

Работа с файлами. Запись с указанной позиции

```
#include <iostream>
#include <fstream>
using namespace std;
void main() {

    char str[80];

    fstream file;
    file.open("test.txt", ios::out);
    file << "0123456789" << endl;

    // нашли в файле позицию 5
    file.seekp(5, ios::beg);

    // дописали в файл с найденной позиции
    file << "abcdefgh";
    file.close(); }
```

Работа с файлами. Чтение с указанной позиции

```
file.open("test.txt", ios::in);  
  
// поиск 8 позиции с конца файла  
file.seekg(-8, ios::end);  
  
// прочитали информацию с найденной позиции  
file >> str;  
  
file.close();  
cout << str;
```

Работа с файлами. Метод `tellg()`

Иногда нужно получать информацию о том, сколько уже прочитано.

```
cout << "Считано байт: " << file.tellg();
```

метод `tellg()` возвращает значение типа `int`, которое показывает сколько уже пройдено в байтах.

Определение размера файла:

```
//переход в конец файла
```

```
file.seekg(0, ios_base::end);
```

```
cout << "Размер файла (в байтах): " << file.tellg();
```

Работа с двоичными файлами. Пример. Часть 1

Принцип работы с двоичными файлами аналогичен работы с текстовыми файлами.

Если необходимо записать, а потом считать данные, представленные в виде структуры, то считывание необходимо производить в переменную этого же структурного типа.

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

struct Notes { // структура данных записной книжки
    char Name[60]; // Ф.И.О.
    char Phone[16]; // телефон
    int Age; // возраст
};
```

Работа с двоичными файлами. Пример. Часть 3

```
ifstream ifile("Notebook.dat", ios::binary);

Notes Note;    // структурированная переменная

char str[80];  // статический буфер строки

// Считывать и отображать строки в цикле, пока не eof
while (!ifile.read((char*)&Note, sizeof(Notes)).eof())
{
    cout << setw(27) << Note.Name
    << setw(13) << Note.Phone
    << setw(3) << Note.Age
    << endl;
}
}
```

Работа с двоичными файлами. Пример. Часть 2

```
void main() {  
    setlocale(0, "");
```

```
Notes Note1 = { "Иванов Иван Иванович", "111-222-333", 15 };  
Notes Note2 = { "Петров Петр Петрович", "444-555-666", 16 };  
Notes Note3 = { "Долгов Игорь Иванович", "777-888-999", 17 };
```

```
ofstream ofile("Notebook.dat", ios::binary);
```

```
ofile.write((char*)&Note1, sizeof(Notes)); // 1-й блок  
ofile.write((char*)&Note2, sizeof(Notes)); // 2-й блок  
ofile.write((char*)&Note3, sizeof(Notes)); // 3-й блок
```

```
ofile.close(); // закрыть записанный файл
```


Форматирование кода. Правила

Правильное форматирование кода действительно важно, если хотите чтобы код был легко-читаемым и был понятен не только вам.

```
1 #include<iostream>
2 usingnamespace std;
3 intmain(){setlocale(LC_ALL, "rus");intboxWithFruit=15;intamountBox
4 cout<<"Сейчас на складе " << boxWithFruit<<" ящиков с яблоками.\n";
5 for(int i=1;;i++){cout<<"Сколько ящиков загрузить в " <<i<<"-ю маш
6 cin>> amountBoxForSale;if (amountBoxForSale>
7 boxWithFruit){cout<<"\nНа складе недостаточно товара!";
8 cout<<"Осталось только " << boxWithFruit<<" ящиков\n\n";i--;}
9 else{boxWithFruit-= amountBoxForSale;
10 cout<<"Осталось " << boxWithFruit<<" ящиков.\n";}
11 if(boxWithFruit==0){cout<<"Яблоки закончились!Досвидания!\n";
12 break;}}return 0;}

11 cout << "Сейчас на складе " << boxWithFruit<< " ящиков с яблоками
12 for (int i = 1;; i++) // счетчик i будет считать количество машин
13 {
14     cout << "Сколько ящиков загрузить в " << i << "-ю машину? ";
15     cin >> amountBoxForSale;
16
17     if (amountBoxForSale > boxWithFruit)
18     {
19         cout << "\nНа складе недостаточно товара!";
20         cout << "Осталось только " << boxWithFruit<< " ящиков\n";
21         i--; // уменьшить счетчик на единицу
22     }
23     else
24     {
25         boxWithFruit-= amountBoxForSale; // перезаписываем значение
26         cout << "Осталось " << boxWithFruit<< " ящиков.\n";
27     }
28
29     if (boxWithFruit== 0)// если ящиков больше нет - выйти из цикла
30     {
31         cout << "Фрукты закончились! До свидания!\n";
32         break;
33     }
34 }
```