Запись в текстовый файл

Для работы с файлами необходимо подключить заголовочный файл **<fstream>**. В <fstream> определены несколько классов и подключены заголовочные файлы <ifstream> — файловый ввод и <ofstream> — файловый вывод.

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.

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.

Пример 1. Запись строки в текстовый файл

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

void main()
{
	// создаём объект класса ofstream для записи
	ofstream fout;
	// связываем его с файлом test.txt
    fout.open("test.txt");
	// запись строки в файл
	fout << "Запись строки в текстовый файл.";
	// закрываем файл
	fout.close();
}
```


Шаги 1 и 2 можно объединить, то есть в одной строке создать объект и связать его с файлом.

Пример 1а. Запись строки в текстовый файл

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

void main()
{
// создаём объект класса ofstream для записи и связываем его с файлом test.txt
	ofstream fout("test.txt");
	// запись строки в файл
	fout << "Запись строки в текстовый файл.";
	// закрываем файл
	fout.close();
}
```

Чтение из текстового 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.

Пример 2. Запись строки в текстовый файл, а потом чтение из файла

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

void main()
{    // корректное отображение Кириллицы
        setlocale(LC_ALL, "rus");
        ofstream fout("test.txt");
        fout << "Запись строки в текстовый файл, а потом чтение из файла.";
        fout.close();

        // буфер промежуточного хранения считываемого из файла текста
        char buff[80];
        // открыли файл для чтения
        ifstream fin("test.txt");
        // считали строку из файла
        fin.getline(buff, 80);
        // закрываем файл
        fin.close();
        // напечатали эту строку
        cout << buff << endl; // напечатали эту строку
}
```


Пример 3. Чтение слов из файла

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

void main()
{
        setlocale(0, ""); // корректное отображение Кириллицы
        // создаём объект класса ofstream для записи
        ofstream fout;
        // связываем его с файлом test.txt
   fout.open("test.txt");
        // запись строки в файл
        fout << "Запись строки в текстовый файл, а потом чтение из файла.";
        // закрываем файл
        fout.close();

        // буфер промежуточного хранения считываемого из файла текста
        char buff[80];
        // открыли файл для чтения
        ifstream fin("test.txt");

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

```
	// считали второе слово из файла
	fin >> buff;
	// напечатали это слово
	cout << buff << endl;
	// закрываем файл
	fin.close(); }
```

Использование функции is_open

Функция — is_open() возвращает 1, если файл был открыт успешно и 0 — если файл не был открыт.

Пример 4. Использование функции is_open

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

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


	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
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

Режим открытия файла ещё называют — флаг.

Пример 5. Установка режима открытия файлов

```
#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 fout("data.txt", ios_base::out | ios_base::trunc);

	// запись строки в файл
	fout << "Установка режима открытия файла.";
	// закрываем файл
	fout.close();
}
```

Режимы открытия файлов можно комбинировать с помощью поразрядной логической операции или |.

Чтение всех строк из файла

Пример 6. Чтение всех строк из файла

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

void main()
{
	setlocale(0, "");
	ofstream fout("data.txt", ios_base::out | ios_base::trunc);

	// запись строк в файл
	for (int i = 1; i <= 5;i++)
		fout<<i <<"  "<<i*100<<"\n" ;
    fout.close();

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


Пример 6а. Чтение всех строк из файла

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

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

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

        // запись строк в файл
        for (int i = 1; i <= 5;i++)
                fout<<i <<"  "<<i*100<<"\n" ;
    fout.close();

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

Произвольный доступ к файлу

Режимы открытия файлов устанавливают характер использования файлов. Для установки режима в классе

Система ввода-вывода C++ позволяет осуществлять произвольный доступ с использованием методов seekg() и seekp().

- ifstream &seekg(Смещение, Позиция);
- ofstream &seekp(Смещение, Позиция);

Смещение определяет область значений в пределах файла (long int).

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.

Позиция смещения определяется как

Позиция	Значение
ios::beg	Начало файла
ios::cur	Текущее положение
ios::end	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.

Можно определить текущую позицию файлового указателя, используя следующие функции:

- streampos tellg() - позиция для ввода
- streampos tellp() - позиция для вывода

Пример 7. Произвольный доступ к файлу. Запись в файл с указанной позиции

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

void main()
{
	setlocale(0, "");
	char str[80];
	fstream inOut;
	inOut.open("test.txt", ios::out);
	inOut << "0123456789" << endl;
	// нашли в файле позицию 5
	inOut.seekp(5, ios::beg);
	// дописали в файл с найденной позиции
	inOut << "abcdefgh";
	inOut.close();
}
```

Пример 8. Произвольный доступ к файлу. Чтение из файла с указанной позиции

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

void main()
{
	setlocale(0, "");
	char str[80];
	fstream inOut;
	inOut.open("test.txt", ios::out);
	inOut << "0123456789" << endl;
	// нашли в файле позицию 5
	inOut.seekp(5, ios::beg);
	// дописали в файл с найденной позиции
	inOut << "abcdefgh";
	inOut.close();
	inOut.open("test.txt", ios::in);
	// поиск 8 позиции с конца файла
	inOut.seekg(-8, ios::end);
	// прочитали информацию с найденной позиции
	inOut >> str;
	inOut.close();
	cout << str;
}
```


Работа с двоичными файлами

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.

Пример 9. Запись в двоичный файл. Чтение из двоичного файла

```
#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
};

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
       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;
       }
}
```


Задания

1. Написать функцию, которая записывает в файл n в диапазоне от k до m. Другая функция должна находить минимальный и максимальный элементы и записывать их в другой файл. Еще одна функция должна выполнять эти действия для заданного количества файлов - p. Последняя функция должна находить две суммы – максимальных и минимальных элементов этих файлов.
2. В файле записаны целые положительные и отрицательные числа. Функция создает два файла – в первый она записывает положительные числа из исходного файла, а во второй – отрицательные.
3. В файле записаны целые числа. Функция создает два файла – в первый она записывает элементы, стоящие на четных местах, а во второй – на нечетных.
4. В файле записано четное количество целых чисел. Функция создает новый файл, состоящий из элементов, которые равны сумме рядом стоящих элементов (1 и 2, 3 и 4, и т.д.).
5. Написать функцию, которая считывает матрицы из файла, находит их сумму и результат записывает в новый файл.