Министерство образования Республики Беларусь

ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра технологий программирования

**Методические указания  
 к лабораторной работе № 2\_5  
 по курсу «Основы алгоритмизации   
и программирования»**

«Файлы»

Преподаватель: Войтехович   
Агния Витольдовна

Составитель: Войтехович   
Агния Витольдовна

Полоцк, 2017

# **ЦЕЛЬ РАБОТЫ**

Изучить библиотечные функции для работы с файлами. Освоить основные методы обработки файлов.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

# 1 Потоки и файлы

Перед тем как начать изучение файловой системы языка С++, необходимо уяснить, в чем разница между потоками и файлами. В системе ввода/вывода С++ для программ поддерживается единый интерфейс, не зависящий от того, к какому конкретному устройству осуществляется доступ. То есть в этой системе между программой и устройством находится нечто более общее, чем само устройство. Такое обобщенное устройство ввода или вывода (устройство более высокого уровня абстракции) называется потоком, в то время как конкретное устройство называется файлом. (Впрочем, файл — тоже понятие абстрактное.) Очень важно понимать, каким образом происходит взаимодействие потоков и файлов.

## 1.1 Потоки

Файловая система языка С++ предназначена для работы с самыми разными устройствами, в том числе терминалами, дисководами и накопителями на магнитной ленте. Даже если какое-то устройство сильно отличается от других, буферизованная файловая система все равно представит его в виде логического устройства, которое называется потоком. Все потоки ведут себя похожим образом. И так как они в основном не зависят от физических устройств, то та же функция, которая выполняет запись в дисковый файл, может ту же операцию выполнять и на другом устройстве, например, на консоли. Потоки бывают двух видов: текстовые и двоичные.

### 1.1.1 Текстовые потоки

**Текстовый поток** — это последовательность символов. В стандарте С++ считается, что текстовый поток организован в виде строк, каждая из которых заканчивается символом новой строки. Однако в конце последней строки этот символ не является обязательным. В текстовом потоке по требованию базовой среды могут происходить определенные преобразования символов. Например, символ новой строки может быть заменен парой символов — возврата каретки и перевода строки. Поэтому может и не быть однозначного соответствия между символами, которые пишутся (читаются), и теми, которые хранятся во внешнем устройстве. Кроме того, количество тех символов, которые пишутся (читаются), и тех, которые хранятся во внешнем устройстве, может также не совпадать из-за возможных преобразований.

### 1.1.2 Двоичные потоки

Двоичный поток — это последовательность байтов, которая взаимно однозначно соответствует байтам на внешнем устройстве, причем никакого преобразования символов не происходит. Кроме того, количество тех байтов, которые пишутся (читаются), и тех, которые хранятся на внешнем устройстве, одинаково. Однако в конце двоичного потока может добавляться определяемое приложением количество нулевых байтов. Такие нулевые байты, например, могут использоваться для заполнения свободного места в блоке памяти незначащей информацией, чтобы она в точности заполнила сектор на диске.

## 1.2 Файлы

В языке С++ файлом может быть все что угодно, начиная с дискового файла и заканчивая терминалом или принтером. Поток связывают с определенным файлом, выполняя операцию открытия. Как только файл открыт, можно проводить обмен информацией между ним и программой.

Но не у всех файлов одинаковые возможности. Например, к дисковому файлу прямой доступ возможен, в то время как к некоторым принтерам — нет. Таким образом, мы пришли к одному важному принципу, относящемуся к системе ввода/вывода языка С++: все потоки одинаковы, а файлы — нет.

Если файл может поддерживать запросы на местоположение (указатель текущей позиции), то при открытии такого файла указатель текущей позиции в файле устанавливается в начало. При чтении из файла (или записи в него) каждого символа указатель текущей позиции увеличивается, обеспечивая тем самым продвижение по файлу.

Файл отсоединяется от определенного потока (т.е. разрывается связь между файлом и потоком) с помощью операции закрытия. При закрытии файла, открытого с целью вывода, содержимое (если оно есть) связанного с ним потока записывается на внешнее устройство. Этот процесс, который обычно называют дозаписью потока (или принудительным освобождением содержимого буфера), гарантирует, что никакая информация случайно не останется в буфере диска. Если программа завершает работу нормально, т.е. либо main() возвращает управление операционной системе, либо вызывается exit(), то все файлы закрываются автоматически. В случае аварийного завершения работы программы, например, в случае краха или завершения путем вызова abort(), файлы не закрываются.

Если вы новичок в программировании, то разграничение потоков и файлов может показаться излишним или даже "заумным". Однако надо помнить, что основная цель такого разграничения — это обеспечить единый интерфейс. Для выполнения всех операций ввода/вывода следует использовать только понятия потоков и применять всего лишь одну файловую систему. Ввод или вывод от каждого устройства автоматически преобразуется системой ввода/вывода в легко управляемый поток.

# 2 Основы работы с файлами

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**.

Например, необходимо создать текстовый файл и записать в него строку «Работа с файлами в С++». Для этого необходимо проделать следующие шаги:

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

Почему необходимо создавать объект класса **ofstream**, а не класса **ifstream**? Потому, что нужно сделать запись в файл, а если бы нужно было считать данные из файла, то создавался бы объект класса **ifstream**.

// создаём объект для записи в файл

ofstream /\*имя объекта\*/; // объект класса ofstream

Назовём объект – fout, Вот что получится:

ofstream fout;

Для чего нам объект? Объект необходим, чтобы можно было выполнять запись в файл. Уже объект создан, но не связан с файлом, в который нужно записать строку.

fout.open("example2\_5.txt"); // связываем объект с файлом

Через операцию точка получаем доступ к методу класса **open()**, в круглых скобочках которого указываем имя файла. Указанный файл будет создан в текущей директории с программой. Если файл с таким именем существует, то существующий файл будет заменен новым. Итак, файл открыт, осталось записать в него нужную строку. Делается это так:

fout << "Работа с файлами в С++"; // запись строки в файл

Используя операцию передачи в поток совместно с объектом **fout** строка «Работа с файлами в С++» записывается в файл. Так как больше нет необходимости изменять содержимое файла, его нужно закрыть, то есть отделить объект от файла.

fout.close(); // закрываем файл

Итог – создан файл со строкой «Работа с файлами в С++».

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

ofstream fout("example2\_5.txt "); // создаём объект класса ofstream и связываем его с файлом example2\_5.txt

Объединим весь код и получим следующую программу.

#include <fstream>

using namespace std;

int main(int argc, char\* argv[])

{

// создаём объект класса ofstream для записи и связываем его

//с файлом example2\_5.txt

ofstream fout("example2\_5.txt ");

fout << "Работа с файлами в С++"; // запись строки в файл

fout.close(); // закрываем файл

system("pause");

return 0;

}

Осталось проверить правильность работы программы, а для этого открываем файл example2\_5.txt и смотрим его содержимое (рисунок 2.1).

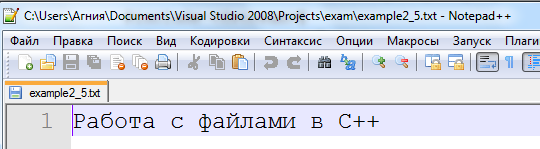


Рисунок 2.1 – Скриншот содержимого файла example2\_5.txt

Для того чтобы прочитать файл понадобится выполнить те же шаги, что и при записи в файл с небольшими изменениями:

1. создать объект класса **ifstream** и связать его с файлом, из которого будет производиться считывание;
2. прочитать файл;
3. закрыть файл.
4. #include <fstream>
5. #include <iostream>
6. using namespace std;
7. int main(int argc, char\* argv[])
8. {
9. // корректное отображение Кириллицы
10. setlocale(LC\_ALL, "rus");
11. // буфер промежуточного хранения считываемого из файла текста
12. char buff[50];
13. // открыли файл для чтения
14. ifstream fin("example2\_5.txt");
15. fin >> buff; // считали первое слово из файла
16. cout << buff << endl; // напечатали это слово
17. fin.getline(buff, 50); // считали строку из файла
18. fin.close(); // закрываем файл
19. cout << buff << endl; // напечатали эту строку
20. system("pause");
21. return 0;
22. }

В программе показаны два способа чтения из файла, первый – используя операцию передачи в поток, второй – используя функцию **getline()**. В первом случае считывается только первое слово, а во втором случае считывается строка, длинной 50 символов. Но так как в файле осталось меньше 50 символов, то считываются символы включительно до последнего. Обратите внимание на то, что считывание во второй раз (строка 14) продолжилось, после первого слова, а не с начала, так как первое слово было прочитано в строке 12. Результат работы программы показан на рисунке 2.2.

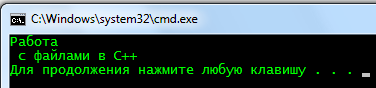


Рисунок 2.2 – Скриншот работы приложения, печатающего в консоль содержимое файла example2\_5.txt

Программа сработала правильно, но не всегда так бывает, даже в том случае, если с кодом всё в порядке. Например, в программу передано имя несуществующего файла или в имени допущена ошибка. Что тогда? В этом случае ничего не произойдёт вообще. Файл не будет найден, а значит и прочитать его не возможно. Поэтому компилятор проигнорирует строки, где выполняется работа с файлом. В результате корректно завершится работа программы, но ничего, на экране показано не будет.

Казалось бы это вполне нормальная реакция на такую ситуацию. Но простому пользователю не будет понятно, в чём дело и почему на экране не появилась строка из файла. Так вот, чтобы всё было предельно понятно в С++ предусмотрена такая функция — **is\_open()**, которая возвращает целые значения:   
1 — если файл был успешно открыт, 0 — если файл открыт не был. Доработаем программу с открытием файла, таким образом, чтобы если файл не открыт, то выводилось соответствующее сообщение.

#include <fstream>

#include <iostream>

using namespace std;

int main(int argc, char\* argv[])

{

// корректное отображение Кириллицы

setlocale(LC\_ALL, "rus");

// буфер промежуточного хранения считываемого из файла текста

char buff[50];

// (ВВЕЛИ НЕ КОРРЕКТНОЕ ИМЯ ФАЙЛА)

ifstream fin("example2\_5.doc");

if (!fin.is\_open()) // если файл не открыт

// сообщить об этом

cout << "Файл не может быть открыт!\n";

else

{

fin >> buff; // считали первое слово из файла

cout << buff << endl; // напечатали это слово

fin.getline(buff, 50); // считали строку из файла

fin.close(); // закрываем файл

cout << buff << endl; // напечатали эту строку

}

system("pause");

return 0;

}

Результат работы программы показан на рисунке 2.3.

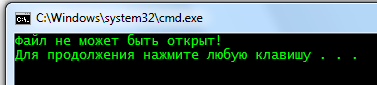


Рисунок 2.3 – Результат попытки чтения из файла example2\_5.doc

Как видно из рисунка 2.3 программа сообщила о невозможности открыть файл. Поэтому, если программа работает с файлами, рекомендуется использовать функцию **is\_open()** даже если уверены, что файл существует.

# 3 Режимы открытия файлов

Режимы открытия файлов устанавливают характер использования файлов. Для установки режима в классе **ios\_base** предусмотрены константы, которые определяют режим открытия файлов. Их список представлен в таблице 3.1.

Таблица 3.1 — 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 (бинарном) режиме |

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

// открываем файл для добавления информации к концу файла

ofstream fout("example2\_5.txt", ios\_base::app);

// открываем файл для добавления информации к концу файла

fout.open("example2\_5.txt", ios\_base::app);

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. В таблице 3.1 перечислены далеко не все флаги, но для начала этих должно хватить.

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.

Разработаем программу, которая, используя операцию **sizeof()**, будет вычислять характеристики основных типов данных в С++ и записывать их в файл. Характеристики:

1. число байт, отводимое под тип данных
2. максимальное значение, которое может хранить определённый тип данных.

Запись в файл должна выполняться в таком формате:

/\* data type byte max value

bool = 1 255.00

char = 1 255.00

short int = 2 32767.00

unsigned short int = 2 65535.00

int = 4 2147483647.00

unsigned int = 4 4294967295.00

long int = 4 2147483647.00

unsigned long int = 4 4294967295.00

float = 4 2147483647.00

long float = 8 9223372036854775800.00

double = 8 9223372036854775800.00 \*/

Для решения поставленной задачи необходимо открыть файл в режиме записи, с предварительным усечением текущей информации файла (желтая строка). Как только файл создан и успешно открыт (голубые строки), используем объект **fout**. Таким образом, вся информация о типах данных запишется в файл.

#include <fstream>

#include <iostream>

#include <iomanip> // манипуляторы ввода/вывода

#include <cmath>

using namespace std;

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "rus");

// связываем объект с файлом, при этом файл открываем в

//режиме записи, предварительно удаляя все данные из него

ofstream fout("data\_types.txt", ios\_base::out | ios\_base::trunc);

if (!fout.is\_open()) // если файл небыл открыт

{

// напечатать соответствующее сообщение

cout << "Файл не может быть открыт или создан\n";

return 1; // выполнить выход из программы

}

// заголовки столбцов

fout << " data type byte max value " << endl ;

fout << "bool = " << sizeof(bool)<< " "

<< fixed << setprecision(2)

//вычисляем максимальное значение для типа данных bool

<< (pow(2,sizeof(bool) \* 8.0) - 1)<< endl;

fout << "char = " << sizeof(char)<< " "

<< fixed << setprecision(2)

//вычисляем максимальное значение для типа данных char

<< (pow(2,sizeof(char) \* 8.0) - 1)<< endl;

fout << "short int = " << sizeof(short int)<< " "

<< fixed << setprecision(2)

//вычисляем максимальное значение для типа данных short int

<< (pow(2,sizeof(short int) \* 8.0 - 1) - 1)<< endl;

fout << "unsigned short int = " << sizeof(unsigned short int)

<< " " << fixed << setprecision(2)

//вычисляем максимальное значение для типа данных unsigned short int

<< (pow(2,sizeof(unsigned short int) \* 8.0) - 1) << endl;

fout << "int = " << sizeof(int)<< " "

<< fixed << setprecision(2)

//вычисляем максимальное значение для типа данных int

<< (pow(2,sizeof(int) \* 8.0 - 1) - 1)<< endl;

fout << "unsigned int = " << sizeof(unsigned int)

<< " " << fixed << setprecision(2)

//вычисляем максимальное значение для типа данных unsigned int

<< (pow(2,sizeof(unsigned int) \* 8.0) - 1)<< endl;

fout << "long int = " << sizeof(long int)<< " "

<< fixed << setprecision(2)

//вычисляем максимальное значение для типа данных long int

<< (pow(2,sizeof(long int) \* 8.0 - 1) - 1)<< endl;

fout << "unsigned long int = " << sizeof(unsigned long int)

<< " " << fixed << setprecision(2)

//вычисляем максимальное значение для типа данных undigned long int

<< (pow(2,sizeof(unsigned long int) \* 8.0) - 1) << endl;

fout << "float = " << sizeof(float)<< " "

<< fixed << setprecision(2)

//вычисляем максимальное значение для типа данных float

<< (pow(2,sizeof(float) \* 8.0 - 1) - 1)<< endl;

fout << "long float = " << sizeof(long float)<< " "

<< fixed << setprecision(2)

//вычисляем максимальное значение для типа данных long float

<< (pow(2,sizeof(long float) \* 8.0 - 1) - 1)<< endl;

fout << "double = " << sizeof(double)<< " "

<< fixed << setprecision(2)

//вычисляем максимальное значение для типа данных double

<< (pow(2,sizeof(double) \* 8.0 - 1) - 1) << endl;

fout.close(); // программа больше не использует файл,

//поэтому его нужно закрыть

cout << "Данные успешно записаны в файл data\_types.txt\n";

system("pause");

return 0;

}

В конце программы, в красной строке явно закрываем файл, хотя это и не обязательно, но считается хорошим тоном программирования. Стоит отметить, что все функции и манипуляторы используемые для форматирования стандартного ввода/вывода актуальны и для файлового ввода/вывода. Поэтому не возникает никаких ошибок, когда оператор **cout** заменяется объектом **fout**.

Результат работы представленной выше программы показан на рисунках 3.1 и 3.2.

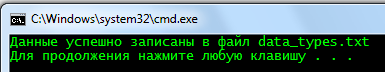


Рисунок 3.1 – Скриншот работы приложения, печатающего в файл характеристики основных типов данных в С++

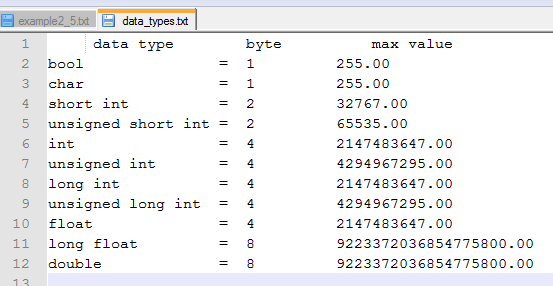


Рисунок 3.2 – Скриншот содержимого файла data\_types.txt

ЗАДАНИЯ ДЛЯ ЛАБОРАТОРНОЙ РАБОТЫ №14

**Дополнить проект, созданный в предыдущей лабораторной работе, следующими функциями и добавьте новые пункты в меню программы:**

**Задание 1.** Функция **new\_DB()**, в которой будет создаваться новый файл базы данных и открываться для записи. Функция должна возвращать объект потока ввода для нового файла.

**Задание 2.** Функция **remove\_DB()**, удаляющая файл по названию. В функции должна быть реализована обработка ошибок.

**Задание 3.** Функция **clear()**, которая будет очищать динамическую память, выделенную для структур базы данных.

**Задание 4.** Функция **save\_DB()**, которая будет сохранять данные из динамического массива структур в указанный файл.

**Задание 5.** Функция **load\_DB()**, которая будет выгружать данные из указанного файла в динамический массив структур.