НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Алгоритмизация и программирование

Программирование на C/C++ и Kotlin

(ч.4 – файлы)

Беркунский Е.Ю., кафедра ИУСТ, НУК
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Файлы

- Что такое файл?
- Файл – именованный набор байтов, который может быть сохранен на некотором накопителе.
- Другими словами, под файлом понимается последовательность байтов, записанных на диск, которая имеет своё, уникальное имя, например **`file.txt`**.

Файлы

- В одном каталоге не могут находиться файлы с одинаковыми именами.
- Под именем файла понимается не только его название, но и расширение, например: file.txt и file.dat - разные файлы, хоть и имеют одинаковые названия.
- Существует такое понятие, как полное имя файлов – это полный путь к каталогу файла с указанием имени файла, например: **`D:\docs\file.txt`**.

Файлы

- 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.

Файлы

- 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 тому, как использовались cin и cout.

Файлы

- Например, необходимо создать текстовый файл и записать в него строку ”Работа с файлами в C++”.
- 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.

Файлы

```
// создаём объект для записи в файл
ofstream /*имя объекта*/; // объект класса ofstream
```

Назовём объект – fout:

```
ofstream fout;
```

Для чего нужен 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("example.txt");
// связываем объект с файлом
```

Файлы

```
ofstream fout;
fout.open("example.txt");
```

- 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.

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

Используя операцию << с объектом fout, строка “Работа с файлами в C++” записывается в файл.

Файлы

```
ofstream fout;
fout.open("example.txt");
fout << "Работа с файлами в C++";
```

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 C++”.

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

```
ofstream fout("example.txt");
```

Файлы

В итоге получим такую программу:

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

int main()
{
    ofstream fout("example.txt");
    fout << "Работа с файлами в C++";
    fout.close();
    return 0;
}
```

Осталось проверить правильность работы программы, а для этого открываем файл example.txt:

```
Работа с файлами в C++
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Демонстрация

Файлы

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

- создать объект класса **ifstream** и связать его с файлом, в который будет производиться запись;
- прочитать файл;
- закрыть файл.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Файлы

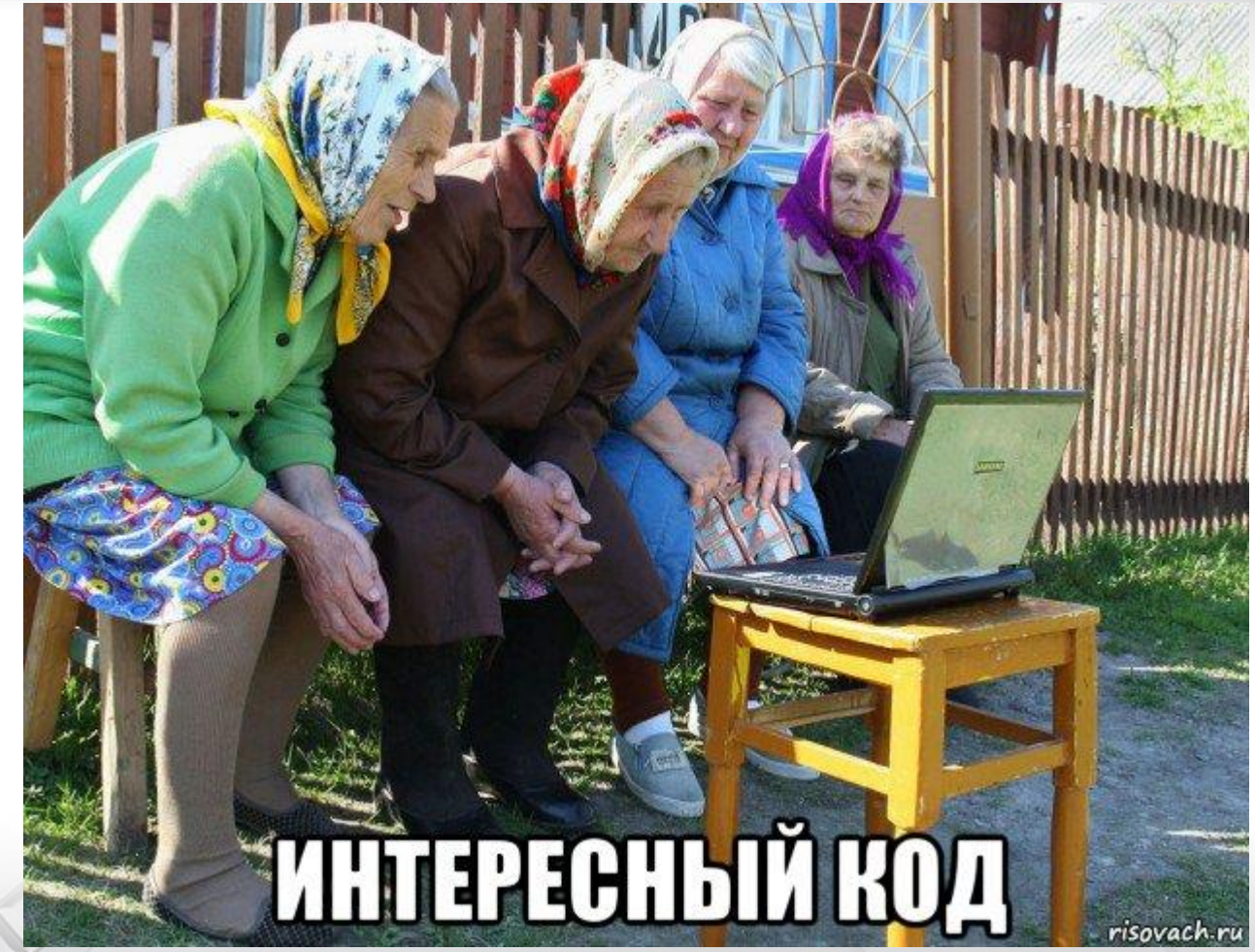
```
#include <fstream>
#include <iostream>
#include <windows.h>

using namespace std;

int main()
{
    SetConsoleCP(1251); SetConsoleOutputCP(1251);
    char buff[50];
    ifstream fin("example.txt");
    fin >> buff;
    cout << buff << endl;
    fin.getline(buff, 50);
    fin.close();
    cout << buff << endl;
    return 0;
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Демонстрация



Файлы

- 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.
- 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 на экране ничего показано не будет.

Файлы

- Простому пользователю не будет понятно, в чём дело и почему на экране не появилась строка из файла.
- Чтобы отреагировать в такой ситуации, в C++ предусмотрена специальная 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, что если файл не открыт выводилось соответствующее сообщение.

Файлы

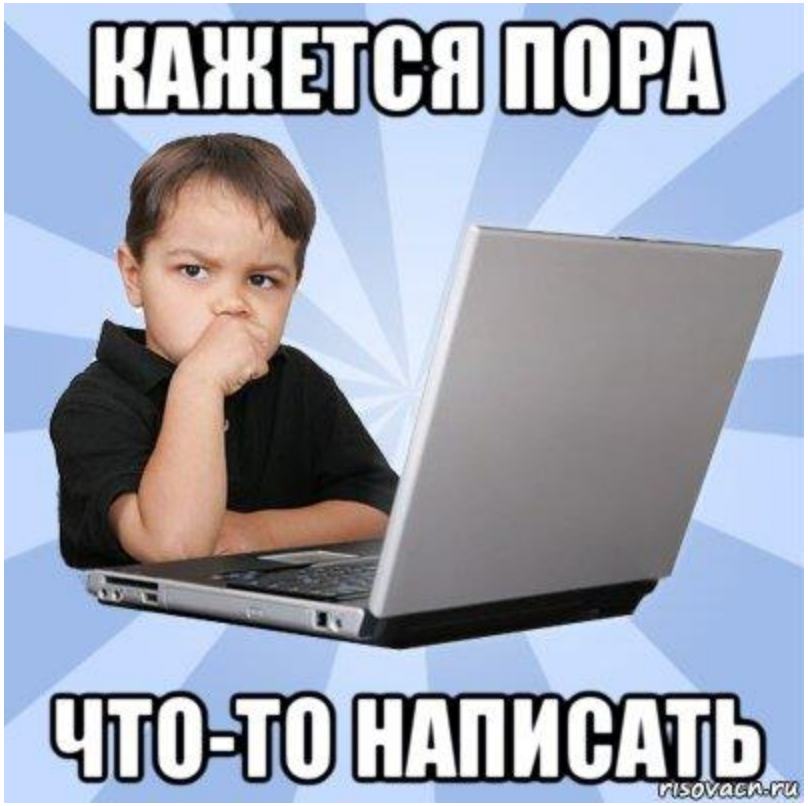
```
#include <fstream>
#include <iostream>

using namespace std;

int main()
{
    char buff[50];
    ifstream fin("example.txt");
    if (!fin.is_open()) // если файл не открыт
        cout << "Файл не может быть открыт!\n";
    else {
        fin >> buff;
        cout << buff << endl;
        fin.getline(buff, 50);
        fin.close();
        cout << buff << endl;
    }
    return 0;
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Демонстрация



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

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

```
// открываем файл для добавления информации
// в конец файла
ofstream fout("example.txt", ios_base::app);
fout.open("example.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.

Режимы по умолчанию

- 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”.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Бонус: обработка текстовых файлов в Kotlin

Есть несколько способов записывать текстовые файлы в Kotlin:

- Писать прямо:
  - *writeText*
  - *writeBytes*
- *Писать, используя Writers:*
  - *printWriter*
  - *bufferedWriter*

writeText

Вероятно, самый простой метод расширения: **writeText** *принимает содержимое в качестве аргумента String и записывает его непосредственно в указанный файл.* Данное содержимое является текстом, закодированным в UTF-8 (по умолчанию) или любым другим указанным

```
File(fileName).writeText(fileContent)
```

writeBytes

- Аналогично, мы можем использовать байты в качестве входных данных.
- **Метод *writeBytes* принимает ByteArray в качестве аргумента и напрямую записывает его в указанный файл.**
- Это полезно, когда у нас есть содержимое в виде массива байтов, а не простой текст

```
File(fileName).writeBytes(fileContentAsArray)
```

printWriter

- Если мы хотим использовать Java *PrintWriter* , Kotlin предоставляет метод *printWriter* именно для этой цели.
- С его помощью мы можем печатать отформатированные представления объектов в выходной поток:

```
File(fileName).printWriter()
```

Этот метод возвращает новый экземпляр *PrintWriter* .
Далее мы можем воспользоваться методом *use* , чтобы его использовать

```
File(fileName).printWriter().use {
    out -> out.println(fileContent)
}
```

Ресурс закрывается независимо от того, была ли функция выполнена успешно или вызвала исключение

bufferedWriter

- Аналогично, Kotlin также предоставляет функцию *bufferedWriter* , которая предоставлена нам из Java
- Затем с его помощью мы можем более эффективно записывать текст в поток вывода символов

```
File(fileName).bufferedWriter()
```

Подобно *PrintWriter* , эта функция возвращает новый экземпляр *BufferedWriter* , который позже мы можем использовать для записи содержимого файла

```
File(fileName).bufferedWriter().use {
    out -> out.write(fileContent)
}
```

Бонус: обработка текстовых файлов в Kotlin

Есть несколько способов читать и обрабатывать текстовые файлы в Kotlin:

- *forEachLine*
- *useLines*
- *bufferedReader*
- *readLines*
- *inputStream*
- readText

forEachLine

- Читает файл построчно, используя указанный charset (по умолчанию UTF-8) и вызывает действие для каждой строки:

```
fun readFileLineByLineUsingForEachLine(fileName: String)
        = File(fileName).forEachLine { println(it) }
```

useLines

- Вызывает обратный вызов заданного блока, давая ему последовательность всех строк в файле.
- По завершении обработки файл закрывается:

```
fun readFileAsLinesUsingUseLines(fileName: String): List<String>
        = File(fileName).useLines { it.toList() }
```

bufferedReader

- Возвращает новый *BufferedReader* для чтения содержимого файла.
- Когда у нас есть *BufferedReader* , мы можем прочитать все строки в нем:

```
fun readFileAsLinesUsingBufferedReader(fileName: String): List<String>
        = File(fileName).bufferedReader().readLines()
```

readLines

- Непосредственно читает содержимое файла в виде списка строк:

```
fun readFileAsLinesUsingReadLines(fileName: String): List<String>
        = File(fileName).readLines()
```

Этот метод не рекомендуется использовать для больших файлов.

inputStream

- Создает новый *FileInputStream* для файла и возвращает его в результате.
- Получив входной поток, мы можем преобразовать его в байты, а затем в полную строку *String*

```
fun readFileAsTextUsingInputStream(fileName: String) =
    File(fileName)
        .inputStream()
        .readBytes()
        .toString(Charsets.UTF_8)
```

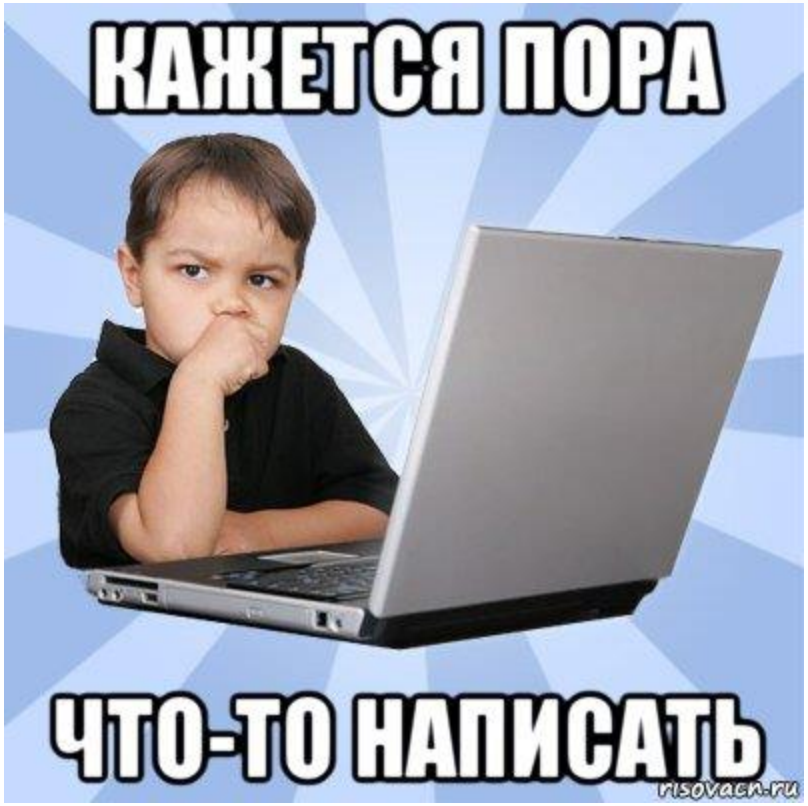
readText

- Читает все содержимое файла как *String* указанный charset (по умолчанию UTF-8)

```
fun readFileDirectlyAsText(fileName: String): String
        = File(fileName).readText(Charsets.UTF_8)
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Демонстрация



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Алгоритмизация и программирование

Программирование на C/C++ и Kotlin

(ч.4 – файлы)

Беркунский Е.Ю., кафедра ИУСТ, НУК
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua