ТЕХНОАТОМ

# Лекция 10. Работа с файлами

МИФИ, 2016

Роман Кузнецов

ТЕХНОАТОМ

# Просьба отметиться на портале!

# План лекции

1. Потоки ввода/вывода.
2. Чтение и запись двоичных и текстовых файлов.
3. Формат JSON и работа с библиотекой jsoncpp.
4. Формат XML и работа с библиотекой tinyxml.

# План лекции

1. Потоки ввода/вывода.
2. Чтение и запись двоичных и текстовых файлов.
3. Формат JSON и работа с библиотекой jsoncpp.
4. Формат XML и работа с библиотекой tinyxml.

# Поток

- некоторая абстракция, обозначающая «поток данных» в самом широком смысле. Поток имеет направление: ввод данных и/или вывод данных.

Поток может вводить и выводить данные в командную строку, в блок памяти, в файл. Можно определить схожий интерфейс для работы с сетью.

# Класс ios (input output system)

- данный класс определяет базовый функционал почти для всех потоков в STL.

Определяет:

1) Флаги форматирования;

2) Флаги ошибок.

# Некоторые флаги форматирования

**dec** - Отображение чисел в 10-ной СС

**oct** - Отображение чисел в 8-ной СС

**hex** - Отображение чисел в 16-ной СС

**showbase** - Выводить индикатор СС

**showpoint** - Показывать десятичную точку

**scientific** - Отображение вещественных чисел в экспоненциальной форме

**fixed** - Фиксированный вывод вещественных чисел

# Установка флагов: пример

```
int main(int argc, char ** argv)
{
  std::cout.setf(std::ios::hex, std::ios::basefield);
  std::cout.setf(std::ios::showbase);
  std::cout << 15 << std::endl; // 0xf
  std::cout.unsetf(std::ios::hex);
  std::cout.unsetf(std::ios::showbase);
  std::cout << 15 << std::endl;
  return 0;
}
```

# Манипуляторы

- методы, встраиваемые в поток, которые позволяют воздействовать на поток тем или иным образом.

STL содержит достаточно большое число манипуляторов. Вот некоторые:

- **std::endl** — вставка в поток перевода строки;
- **std::dec** — переход в 10-ую СС;
- **std::oct** — переход в 8-ую СС;
- **std::hex** — переход в 16-ую СС;
- **std::setw(int)** — устанавливает ширину числа;
- **std::setfill(char)** — устанавливает символ заполнения.

# Манипуляторы: пример

```
#include <iomanip>

int main(int argc, char ** argv)
{
  std::cout << std::setw(5) << std::setfill('*') << 15 << std::endl;
  return 0;
}
```

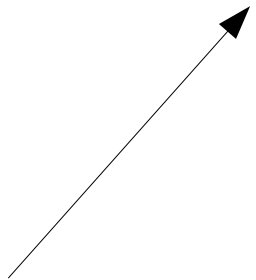
Результат:

***15

# Поддержка манипуляторов

```
class Logger
{
public:
  template<typename T>
  Logger & operator << (T const & val)
  {
    std::cout << val;
    return *this;
  }
};
```

```
int main(int argc, char ** argv)
{
  Logger logger;
  logger << 5 << std::endl;
  return 0;
}
```



Ошибка

# Поддержка манипуляторов

```
Logger & operator << (std::ostream & (*manip)(std::ostream &))
{
  manip(std::cout);
  return *this;
}
```

Указатель на функцию

# Создание собственных манипуляторов

```
std::ios_base & floatnormal(std::ios_base & io)
{
   io.setf(0, std::ios_base::floatfield);
   return io;
}

int main(int argc, char ** argv)
{
  std::ios_base::fmtflags flags =  std::cout.flags(); // Save old flags.
  double const val = 220.0 / 7.0;
  std::cout << "val = " << std::scientific << val << std::endl;
  std::cout << "val = " << floatnormal << val << std::endl;
  std::cout.flags(flags);
  return 0;
}
```

**Результат:**

val = 3.142857e+01

val = 31.4286

# Класс istream

- базовый поток ввода данных.

Полезные методы:

**>>** - Форматированное извлечение данных из потока;

**get** - Извлекает символ/строку из потока (неформатированно);

**read** - Читает блок данных (неформатированно);

**tellg** - Возвращает текущее положение курсора чтения;

**seekg** - Устанавливает текущее положение курсора чтения.

# Класс ostream

- базовый поток вывода данных.

Полезные методы:

**<<** - Форматированное помещение данных в поток;

**put** - Помещает символ в поток (неформатированно);

**write** - Записывает блок данных (неформатированно);

**tellp** - Возвращает текущее положение курсора записи;

**seekp** - Устанавливает текущее положение курсора записи.

# Ошибки в потоках

**Биты**:

**goodbit** Ошибок нет (флаги не установлены)
**eofbit** Достигнут конец файла
**failbit** Операция не выполнена
**badbit** Недопустимая операция
**hardfail** Неисправимая ошибка

**Методы**:

**eof()** true если eofbit
**fail()** true если failbit | badbit | hardfail
**bad()** true если badbit | hardfail
**good()** true если goodbit

# Ввод/вывод в файлы

**ifstream** — поток для чтения файла

**ofsteam** — поток для записи файла

**fstream** — чтение и запись.

# Запись текстового файла: пример

```
std::ofstream f("file.txt");
if (f.is_open())
{
  f << "Test" << std::endl;
  f << 1 << std::endl;
  f << 15.4 << std::endl;
  f.close();
}
```

# Чтение тестового файла: пример

```
std::ifstream f("file.txt");
if (f.is_open())
{
  std::string str;
  f >> str;
  std::cout << str << std::endl;

  int ival;
  f >> ival;
  std::cout << ival << std::endl;

  float fval;
  f >> fval;
  std::cout << fval << std::endl;

  f.close();
}
```

# Запись двоичного файла: пример

```
std::ofstream f("file.bin", std::ios::binary);
if (f.is_open())
{
  int ival = 13;
  f.write(reinterpret_cast<char const *>(&ival), sizeof(ival));
  float fval = 15.4f;
  f.write(reinterpret_cast<char const *>(&fval), sizeof(fval));
  f.close();
}
```

# Чтение двоичного файла: пример

```
std::ifstream f("file.bin", std::ios::binary);
if (f.is_open())
{
  int ival;
  f.read(reinterpret_cast<char *>(&ival), sizeof(ival));
  std::cout << ival << std::endl;

  float fval;
  f.read(reinterpret_cast<char *>(&fval), sizeof(fval));
  std::cout << fval << std::endl;

  f.close();
}
```

# Формат JSON

JavaScript Object Notation.

Текстовый формат обмена данными, основанный на JavaScript.

Формат независим от языка и может использоваться практически с любым языком программирования.

Для многих языков существует библиотеки для создания и обработки данных в формате JSON.

# Синтаксис JSON

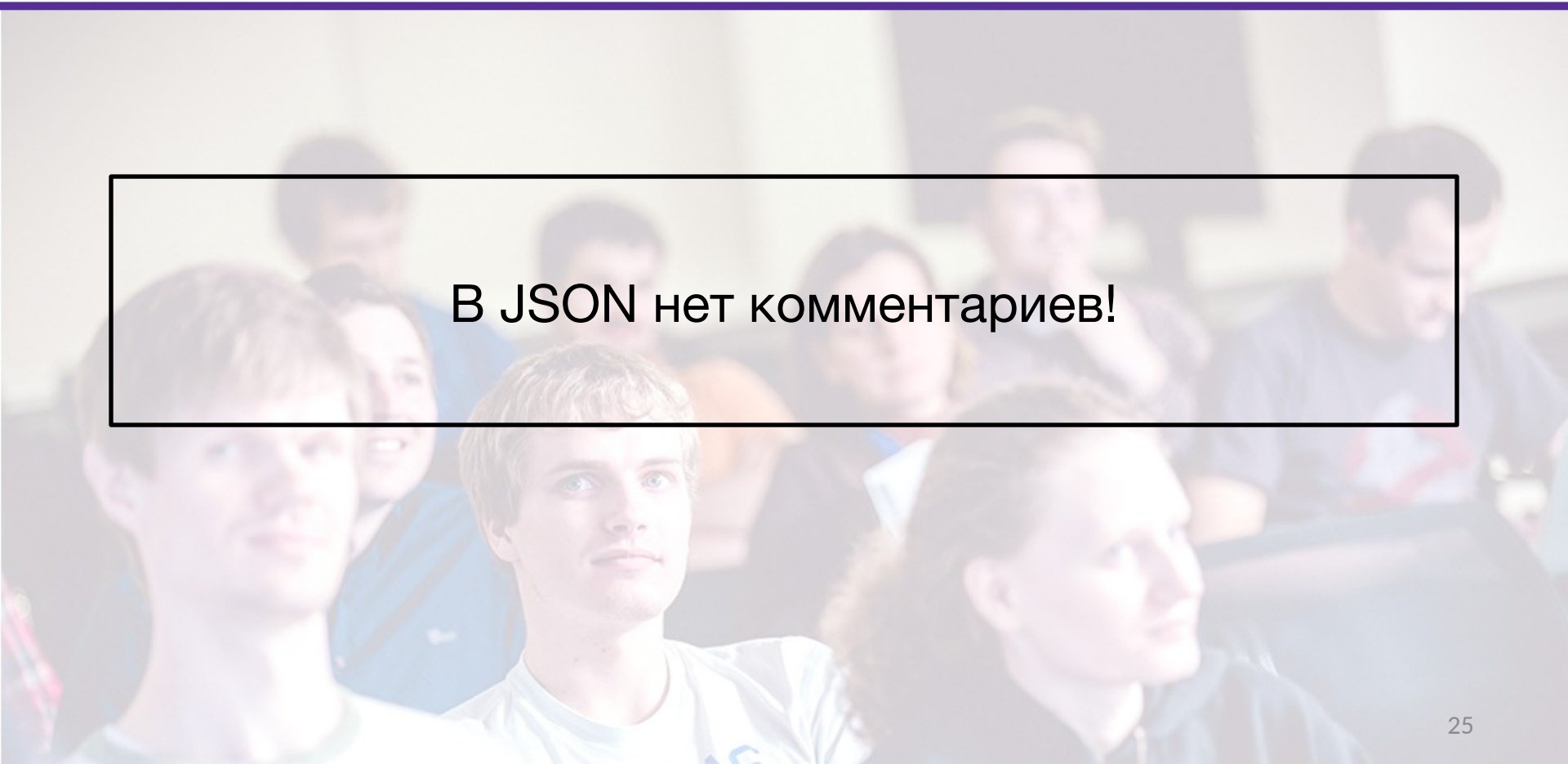
- Пара “ключ-значение”. Ключом может быть только строка (регистрозависимая), значением — любая форма.
- Набор значений.

В качестве значений в JSON могут быть:

- **Объект** — это множество пар “ключ:значение”, заключенное в фигурные скобки «{ }». Между ключом и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.
- **Массив** (одномерный) — это множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми.
- **Число**.
- **Литералы** true, false и null.
- **Строка** — это множество из нуля или более символов юникода, заключенное в двойные кавычки.

# JSON: пример

```
{
   "firstName": "Иван",
   "lastName": "Иванов",
   “age”: 28,
   “unemployed”: true,
   "address": {
       "streetAddress": "Московское ш., 101, кв.101",
       "city": "Ленинград",
       "postalCode": 101101
   },
   "phoneNumbers": [
       "812 123-1234",
       "916 123-4567"
   ]
}
```



В JSON нет комментариев!

# JSON5

JSON5 — предложенное расширение формата json в соответствии с синтаксисом ECMAScript 5, вызванное тем, что json используется не только для общения между программами, но и создается/редактируется вручную.

JSON5 является корректным кодом ECMAScript 5. Для некоторых языков программирования уже существуют парсеры json5.

Некоторые нововведения:

- Поддерживаются как однострочные //, так и многострочные /* */ комментарии.
- Объекты и списки могут иметь запятую после последнего элемента.
- Ключи объекта могут быть без кавычек, если они являются валидными идентификаторами ECMAScript 5.
- Строки могут заключаться как в одинарные, так и в двойные кавычки.
- Числа могут быть в шестнадцатеричном виде, начинаться или заканчиваться десятичной точкой, включать Infinity, -Infinity, NaN и -NaN, начинаться со знака +.

# Достоинства и недостатки JSON

**Достоинства**
- Поддерживает Unicode
- Лаконичен
- Удобен для работы на разных языках
- Большая часть бэкендов отдает именно json

**Недостатки**
- Менее распространен как формат хранения
- Нет комментариев

# Добавление зависимостей

git submodule add https://github.com/zeux/pugixml.git 3party/pugixml
git submodule add https://github.com/open-source-parsers/jsoncpp.git 3party/jsoncpp

**В CMake-скрипт:**

```
add_subdirectory(3party/jsoncpp)
include_directories(3party/jsoncpp/include)
aux_source_directory(3party/jsoncpp/src/lib_json JSONCPP_SRC)

add_subdirectory(3party/pugixml)
include_directories(3party/pugixml/src)
aux_source_directory(3party/pugixml/src PUGIXML_SRC)

# Create executable by SRC_LIST.
add_executable(${PROJECT_NAME} ${SRC_LIST} ${QT_WRAPPED_SRC} ${JSONCPP_SRC} $
{PUGIXML_SRC})
```

# Запись Json-файла

```
void WriteJson()
{
  Json::Value settings;
  Json::Value resolutions(Json::arrayValue);
  resolutions.append(Json::Value("800x600"));
  resolutions.append(Json::Value("1024x768"));
  resolutions.append(Json::Value("1280x1024"));

  auto & root = settings["settings"];
  root["resolution"] = resolutions;
  root["aliensCount"] = 100;
  root["bulletsCount"] = 200;
  root["entities"]["gun"]["health"] = 50;
  root["entities"]["alien"]["health"] = 20;
  root["entities"]["obstacle"]["health"] = 15;

  std::ofstream settingsFile;
  settingsFile.open("settings.json");
  if (settingsFile.is_open())
  {
    Json::StyledWriter styledWriter;
    settingsFile << styledWriter.write(settings);
    settingsFile.close();
  }
}
```

# Запись Json-файла: результат

```
{
   "settings" : {
      "aliensCount" : 100,
      "bulletsCount" : 200,
      "entities" : {
         "alien" : {
            "health" : 20
         },
         "gun" : {
            "health" : 50
         },
         "obstacle" : {
            "health" : 15
         }
      },
      "resolution" : [ "800x600", "1024x768", "1280x1024" ]
   }
}
```

# Чтение Json-файла

```
void ReadJson()
{
  Json::Value settings;
  std::ifstream file("settings.json");
  if (file.is_open())
  {
    file >> settings;
    file.close();
  }

  Json::Value & resolutions = settings["settings"]["resolution"];
  if (resolutions.isArray())
  {
    for (Json::Value::ArrayIndex i = 0; i < resolutions.size(); i++)
      std::cout << resolutions[i].asString() << std::endl;
  }
```

# Чтение Json-файла (продолжение)

```
  std::cout << settings["settings"]["aliensCount"].asInt() << std::endl;
  std::cout << settings["settings"]["bulletsCount"].asInt() << std::endl;

  Json::Value & entities = settings["settings"]["entities"];
  std::cout << entities["gun"]["health"].asInt() << std::endl;
  std::cout << entities["alien"]["health"].asInt() << std::endl;
  std::cout << entities["obstacle"]["health"].asInt() << std::endl;
}
```

# Формат XML

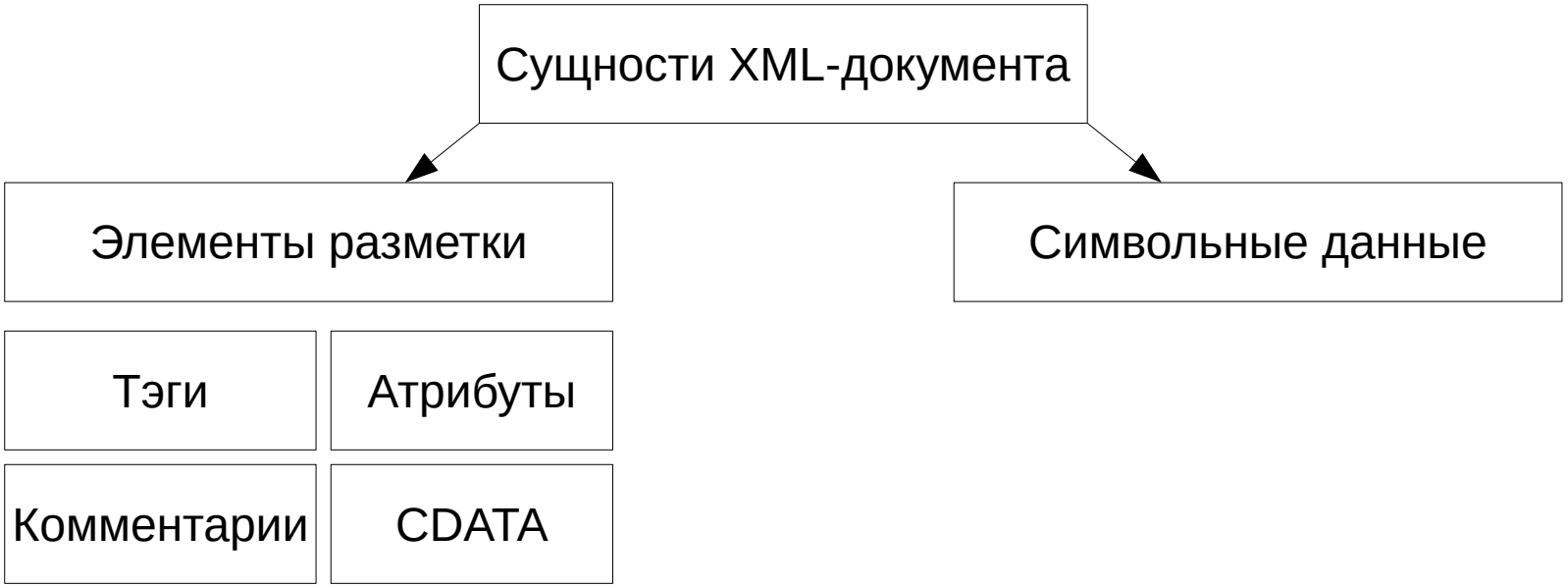
eXtensible Markup Language — расширяемый язык разметки

XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, был нацелен на использование в Интернете.

Язык называется расширяемым, поскольку он не фиксирует разметку, используемую в документах: разработчик волен создать разметку в соответствии с потребностями к конкретной области, будучи ограниченным лишь синтаксическими правилами языка.

Расширение XML — это конкретная грамматика, представленная словарём тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и элементы могут входить в состав других элементов.

# Состав XML-документа



# Структура XML-документа

1) Заголовок
   - Объявления
   - Инструкции обработки
   - Комментарии

2) Корневой элемент
   - Вложенные элементы
   - Символьные данные
   - Комментарии

**Элемент**, начинающийся внутри другого элемента, должен заканчиваться внутри элемента, в котором он начался.

**Символьные данные** могут встречаться внутри элементов как непосредственно так и в специальных секциях «CDATA».

**Объявления, инструкции обработки и элементы** могут иметь связанные с ними атрибуты.

**Атрибуты** используются для связывания с логической единицей текста пар имя-значение.

# Объявление XML

Может быть первой строкой документа, необязательное. Определяет документ как файл XML, что может помочь распознавать файл как XML, а не SGML или другой способ разметки.

```
<?xml version="1.1" encoding="UTF-8" ?>

<?xml version="1.0" encoding="windows-1251"?>
```

# Объявление типа документа

Для объявления типа документа существует специальная инструкция ! DOCTYPE. Она позволяет задать при помощи языка DTD (Document Type Definition), какие в документ входят элементы, каковы их атрибуты, какие сущности могут использоваться и т.д.

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

Здесь «SYSTEM "hello.dtd"» — внешний идентификатор: адрес «hello.dtd» позволяет задействовать данные в документе «hello.dtd» как объявления разметки.

# Секция CDATA

Секция может встречаться в любом месте документа, где синтаксис позволяет размещать символьные данные.

<![CDATA[ Данные здесь ]]>.

Между этой разметкой находятся символьные данные, символьные данные при этом включают символы < > & в их непосредственной форме. Служит для того, чтобы указать парсеру, что не надо анализировать то, что находится внутри этого блока.

# Пример XML-документа

```
<?xml version="1.0" encoding="UTF-8"?>
<addresses>
  <address>
    <phone type='mobile'>+74953456789</phone>
    <owner>Petrov</owner>
    <documents>
      <document type='passport'>5011851917</document>
    </documents>
  </address>
  <address>
    <phone type='mobile'>+7495987654</phone>
    <owner>Sidorov</owner>
    <documents>
      <document type='passport'>5012536917</document>
    </documents>
  </address>
</addresses>
```

# Достоинства XML

- Поддерживает Unicode
- Имеет строго заданный синтаксис
- Стандартизован W3C
- Платформонезависим
- Есть язык преобразования XSLT (можно, например, сделать из XML валидный HTML)

# Недостатки XML

- Синтаксис избыточен
- Размер существенно больше, чем у других форматов (JSON, YAML)
- Нет поддержки типов данных

# Запись Xml-файла

```
void WriteXml()
{
  pugi::xml_document doc;
  auto declarationNode = doc.append_child(pugi::node_declaration);
  declarationNode.append_attribute("version") = "1.0";
  declarationNode.append_attribute("encoding") = "UTF-8";

  auto root = doc.append_child("settings");
  pugi::xml_node resolutions = root.append_child("resolutions");
  resolutions.append_child("resolution").append_child(pugi::node_pcdata).set_value("800x600");
  resolutions.append_child("resolution").append_child(pugi::node_pcdata).set_value("1024x768");
  resolutions.append_child("resolution").append_child(pugi::node_pcdata).set_value("1280x1024");
```

# Запись XML-файла (продолжение)

```
  root.append_child("aliensCount").append_attribute("value").set_value(100);
  root.append_child("bulletsCount").append_attribute("value").set_value(200);

  pugi::xml_node entities = root.append_child("entities");
  entities.append_child("gun").append_attribute("health").set_value(50);
  entities.append_child("alien").append_attribute("health").set_value(20);
  entities.append_child("obstacle").append_attribute("health").set_value(15);

  doc.save_file("settings.xml", PUGIXML_TEXT("  "));
}
```

# Запись Xml-файла: результат

```
<?xml version="1.0" encoding="UTF-8"?>
<settings>
  <resolutions>
    <resolution>800x600</resolution>
    <resolution>1024x768</resolution>
    <resolution>1280x1024</resolution>
  </resolutions>
  <aliensCount value="100" />
  <bulletsCount value="200" />
  <entities>
    <gun health="50" />
    <alien health="20" />
    <obstacle health="15" />
  </entities>
</settings>
```

# Чтение Xml-файла

```
void ReadXml()
{
  pugi::xml_document doc;
  pugi::xml_parse_result result = doc.load_file("settings.xml", pugi::parse_default | pugi::parse_declaration);
  if (result)
  {
    pugi::xml_node root = doc.document_element();
    pugi::xml_node resolutions = root.child("resolutions");
    if (!resolutions.empty())
    {
      for (pugi::xml_node const & resolution : resolutions.children())
        std::cout << resolution.child_value() << std::endl;
    }

    pugi::xml_node aliensCount = root.child("aliensCount");
    if (!aliensCount.empty())
      std::cout << aliensCount.attribute("value").as_int() << std::endl;
```

# Чтение Xml-файла (продолжение)

```
    pugi::xml_node bulletsCount = root.child("bulletsCount");
    if (!bulletsCount.empty())
      std::cout << bulletsCount.attribute("value").as_int() << std::endl;

    pugi::xml_node entities = root.child("entities");
    if (!entities.empty())
    {
      pugi::xml_node gun = entities.child("gun");
      if (!gun.empty()) std::cout << gun.attribute("health").as_int() << std::endl;

      pugi::xml_node alien = entities.child("alien");
      if (!alien.empty()) std::cout << alien.attribute("health").as_int() << std::endl;

      pugi::xml_node obstacle = entities.child("obstacle");
      if (!obstacle.empty()) std::cout << obstacle.attribute("health").as_int() << std::endl;
    }
  }
}
```

# Пример кода

https://github.com/rokuz/template/tree/files-demo

# Домашнее задание

1) Добавить в логгер запись лога в файл;
2) Добавить сохранение состояния окна настроек в Json или Xml (формат на выбор).

**Срок сдачи:** 24.11.2016 23:59:59

# Просьба оставить отзыв о данном занятии на портале!


# Спасибо за внимание!

Роман Кузнецов

r.kuznetsov@mapswithme.com