



*Spring 2021*

*[compscicenter.ru](http://compscicenter.ru)*

Basharin Egor

[t.me/egorbasharin](https://t.me/egorbasharin)

# Lecture V

The Serialization, or There and Back Again

# Section 1

Motivation

# Motivation

```
struct CompositeData {  
    // ...  
};  
  
// Moscow  
void evaluator() {  
    for (const auto& query : queries) {  
        CompositeData data = evaluate(query);  
        send(data);  
    }  
}  
  
// Saint-Petersburg  
void analyzer() {  
    while (true) {  
        CompositeData data = receive();  
        auto stat = analyze(data);  
        report(stat);  
    }  
}
```

# Motivation

ToDo:

- Преобразовать объект в поток байтов
- Передать по сети
- Восстановить объект из потока байтов

# Motivation

ToDo:

- Преобразовать объект в поток байтов (*Serialization*)
- Передать по сети
- Восстановить объект из потока байтов (*Deserialization*)

# Serialization/Deserialization

```
struct CompositeData {
    int i = 42;
};

int main() {
    // Machine-1
    {
        CompositeData d{255};
        std::ofstream outf("./serialized.bin", std::ios::binary);
        outf.write(reinterpret_cast<const char*>(&d), sizeof(d));
    }

    // Machine-2
    {
        CompositeData d;
        std::ifstream inf("./serialized.bin", std::ios::binary);
        inf.read(reinterpret_cast<char*>(&d), sizeof(d));
        assert(d.i == 255);
    }

    return 0;
}
```

# Serialization/Deserialization

- Независимость от платформы
- Нарушение инкапсуляции



# Section 2

Basics

# Text vs Binary

```
{  
    int32_t i = 256;  
    std::ofstream outf("./serialized.bin", std::ios::binary);  
    outf.write(reinterpret_cast<const char*>(&i), sizeof(i));  
}  
  
{  
    int32_t i = 256;  
    std::ofstream outf("./serialized.txt", std::ios::binary);  
    outf << i;  
}
```

# Text vs Binary

Text:

- Прост для чтения человеком
- Нет проблем с порядком байтов на разных платформах
- Использование разделителей (пробельных символов,...)
- Размер сериализованного сообщения зависит от данных

Binary:

- Эффективнее (по количеству тиков CPU)
- Нет разделителей, длина обычно фиксирована

# Text vs Binary

Text:

- JSON
- XML
- ...

Binary:

- BSON
- Protocol Buffers
- FlatBuffers
- ...

# Serialize numbers & strings

Text:

- ``operator<<`, `operator>>``
- пробельные символы для разделения
- строки: кавычки, new-line или информация о длине

Binary:

- ``write`, `read``
- `endianness; `sizeof``
- `compression`
- строки: информация о длине

Both:

- ``std::ios::binary``

# Serialize simple structs

- Функции сериализации/десериализации
- Версионность!

# Example (text-serialization)

```
struct Starship {
    std::uint32_t mass;
    std::uint32_t carrying;
    std::string model;

    void serialize(std::ostream& out) {
        out << mass << " " << carrying << " " << model;
    }

    void deserialize(std::istream& in) {
        in >> mass >> carrying >> model;
    }
};

int main() {
    Starship ship{100, 200, "model-x"};
    std::stringstream ss;
    ship.serialize(ss);

    Starship ship2;
    ship2.deserialize(ss);
}
```

# Example (text serialization)

```
struct Starship {
    std::uint32_t mass;
    std::uint32_t carrying;
    std::string model;

    static constexpr std::uint32_t VERSION = 1;

    void serialize(std::ostream& out) {
        out << VERSION << " " << mass << " " << carrying << " " << model;
    }

    void deserialize(std::istream& in) {
        std::uint32_t version = 0;
        in >> version;
        in >> mass >> carrying >> model;
    }
};
```



# Example (text serialization)

```
struct Engine {
    void serialize(std::ostream& out) const {}
    void deserialize(std::istream& in) {}
};

struct Starship {
    std::uint32_t mass;
    std::uint32_t carrying;
    Engine engine;

    static constexpr std::uint32_t VERSION = 2;

    void serialize(std::ostream& out) {
        out << VERSION << " " << mass << " " << carrying << " ";
        engine.serialize(out);
    }
    void deserialize(std::istream& in) {
        std::uint32_t version = 0;
        in >> version >> mass >> carrying;
        switch (version) {
            case 1: /* skip `std::string model` */ break;
            case 2: engine.deserialize(in); break;
            default: break;
        }
    }
};
```

# Inheritance

Исходный код: [inheritance\\_serialization.cpp](#)

В исходной версии кода будут проблемы с десериализацией для случая, когда в векторе более одного животного одного типа (например, два кота)

Исправленный вариант: [inheritance\\_serialization\\_fixed.cpp](#)

---

Слайды будут приложены к презентации

# Inheritance

- Использование исключений
- Порядок инициализации статических объектов
- Null Pointers

# Serialization vectors & strings

```
std::ofstream ouf("v.bin", std::ios::binary);  
std::string s = "hello";  
ouf.write(reinterpret_cast<const char*>(&s), sizeof(  
  
std::vector<int> v = {1,2,3};  
ouf.write(reinterpret_cast<const char*>(&v), sizeof(
```