Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра мультимедійних систем

Курсова робота

освітній ступінь – бакалавр

на тему: «**Властивість збережності в об'єктно-орієнтованому програмуванні: серіалізація об'єктів**»

Виконав: студент 3-го року навчання,
освітньо-наукової програми
«Інженерія програмного
забезпечення», 121

Загорулько Андрій Костянтинович

Керівник Бублик В.В.,_____
кандидат фіз.-мат. наук, доцент

Курсова робота захищена
з оцінкою _____

Київ – 2023

Зміст

Вступ

Серіалізація – операція приведення об’єкту або групи об’єктів у послідовність байтів з можливістю подальшого відтворення.

Серіалізація є однією з фундаментальних операцій у програмуванні, який більшість застосунків у певному вигляді потребують. У зв'язку з цим з’явилося багато бібліотек для виконання цього завдання за користувача як Boost.Serialization, Boost.JSON, Cereal. Більшість подібних реалізацій спрямовані на одну мову програмуванні і не можуть бути використані поза нею. Подібні обмеження систем ускладнюють їх використання при побудові мікросервних архітектур та клієнт-серверних застосунків, де можуть бути використані декілька мов програмування.

Виходячи з популярності подібних застосуків за мету даної роботи було обрано створення системи серіалізації, яку можна легко адаптувати до більшості об’єктно-орієнтованих мов.

Мета даної роботи зумовила наступне наукове завдання:

1. Виявити необхідні види абстракції прийнятих у відомих бібліотеках серіалізації.
2. На основі розглянутих абстракції побудувати власну систему серіалізації загального призначення та реалізувати її у мові C++ стандарту 20, яку можна адаптувати для збереження об’єктів у об’єктно орієнтованих мовах як C++, Java, C#.
3. Порівняти розроблену модель даних з іншими бібліотеками серіалізації.

Робота розділена на 3 частини.

У першій розглядаються види абстракцій серіалізації та вплив стилів програмування на можливості системи серіалізації.

У другій описана система Serene та її реалізація у мові C++ стандарту 20.

У третій задокументовано результати використання системи серіалізації для побудови HTTP серверу та порівняння Serene з іншими бібліотеками.

Розділ 1: Загальні концепції

1.1 Категоризація систем серіалізації

Маршал Клайн розрізнив техніки серіалізації на читабельні та нечитабельні, проте подібна категоризація занадто спрощує питання.[5] Boost.Serialization підтримує обидві техніки серіалізації, тобто за цим припущенням її можна використовувати у всіх випадках, коли необхідна серіалізація. Насправді, припущення не справджується і Boost.Serialization досить обмежений у своїх можливостях.

Іншим варіантом категоризації може слугувати категоризації систем серіалізації на формати, але дане рішення не пояснює причини вибору одного формату замість іншого. Бажано категоризувати серіалізацію за намірами, які намагаються вирішити користувачі, що наддасть можливість визначити базові необхідності до кожного завдання та дозволить виміряти ефективність абстракції.

1.2 Намір користувача

Існують різні причини для серіалізації та десеріалізації. Деякими з них є:

1. Зберігання інформації між сесіями застосунку.
2. Кешування.
3. Комунікація між застосунками.
4. Завантаження тестових даних.
5. Налаштування застосунку.

Кожна з причин має базові вимоги до розміру даних, читабельності. Також, деякі з них можуть з’являтися лише після зміни вимог, наприклад, необхідність кешування даних.

В першу чергу, наміри користувача обмежують вибір можливих серіалізаційних бібліотек. Boost.Serialization призначена для збереження та відтворення даних у мові C++, що унеможливлює використання бібліотеки при комунікації з застосунками створеними іншими мовами програмування.[4] З іншого боку, бібліотеки як Protocol Buffers призначенні для комунікації застосунків, але вони не підтримують ієрархій класів, обмежуючи випадки, коли їх можна використовувати.[7] Cereal, яка порівняно

з Boost.Serialization додає нові формати як JSON, XML та YAML, і, теоретично, підтримує різні наміри користувача вимагає особливостей мови C++, і через це не може бути використана в інших мовах програмування.

Подібні ускладнення в C++ можна порівняти з Serde – фреймворком для серіалізації у мові програмування Rust. На момент написання 26718 бібліотек на crates.io посилаються на неї.[11] Рівень інтеграції можна побачити у бібліотеці Rocket, яка дозволяє надіслати довільний об’єкт за допомогою Serde.[9] Основними рисами даного фреймворку є:

1. Простота використання
2. Підтримка багатьох форматів
3. Відсутність особливої поведінки.
4. Можливість реалізувати власну серіалізацію за необхідності.

Популярність Serde можна аргументувати підтримкою більшості намірів користувачів. Як вже було зазначено, фреймворк можна використовувати для комунікації з користувачем, але Serde, додатково, підтримує оптимізовані формати для збереження даних: CBOR, MessagePack.Також, фреймворк може бути використаний для конфігурації застосунку за допомогою бібліотеки confy.[6]

Serde доводить можливість створення абстракції, яка зможе задовільнити серіалізацію для більшості намірів користувача, але з’являється питання вирішення подібного питання для об’єктно-орієнтованих мов.

Незважаючи на те, що напрямок нашої системи орієнтований на комунікацію між застосунками, подібне обмеження значно ускладнює роботу користувача. Якщо розроблена система підтримуватиме лише один намір, користувач буде змушений використовувати додаткові бібліотеки серіалізації, що створюватиме дуплікацію коду. У зв’язку з подібними недоліками спеціалізації, система повинна підтримувати різні наміри користувача.

1.3 Проблеми простої реалізації

Спершу, варто розглянути просту реалізацію на можливість задовільнити різні наміри користувача і проблеми, які відсутність абстракцій може створити.

Для можливості зберігати звичайний об’єкт нам потрібен лише потік байтів та інструкції приведення об’єкту. Використаємо приклад простої серіалізації з C++ Cookbook.[1] [Рисунок 1, Приклад простої серіалізації]

```
class Employee {
  friend ostream& operator<< // These have to be friends
    (ostream& out, const Employee& emp); // so they can access
  friend istream& operator>> // nonpublic members
    (istream& in, Employee& emp);
public:
  Employee() {}
  ~Employee() {}
private:
  string firstName_;
  string lastName_;
};
// Send an Employee object to an ostream...
ostream& operator<<(ostream& out, const Employee& emp) {
  out << emp.firstName_ << endl;
  out << emp.lastName_ << endl;
  return(out);
}
// Read an Employee object from a stream
istream& operator>>(istream& in, Employee& emp) {
  in >> emp.firstName_;
  in >> emp.lastName_;
  return(in);
}
```

Рисунок 1, Приклад простої серіалізації

Оператори << та >> виконують функції серіалізації та десеріалізації відповідно. Як параметри приймаються потоки за допомогою яких відбувається збереження

об’єкту. Даної імплементації може бути достатньо при малій кількості об’єктів необхідних для збереження стану. В інших випадках вона створюватиме проблеми:

1. Відсутність визначеної форми.

   При необхідності передавання даних застосунку на іншій мові програмування поведінку серіалізації потрібно буде відтворювати. У результаті методи серіалізації можуть категорично відрізнятися, що ускладнить подальші зміни застосунку. Якщо розробник бажатиме використовувати вже існуючий формат як CBOR або MessagePack, то відсутність абстракцій може привести до логічних помилок та великій кількості повторювань у коді.

2. Успадкування та поліморфізм.

   Реалізація не передбачає можливості успадкувати клас. Наприклад, якщо клас Student буде успадковувати клас Person, то потрібно буде додати віртуальний метод, що наддасть можливість обирати стратегію серіалізації. При відтворенні з’являється проблема залежності класу Person від Student, адже під час десеріалізації потрібно розуміти, який об’єкт був збережений: базовий чи похідний. У результаті користувач створюватиме самостійно систему серіалізації.

3. Складність підтримки коду.

   Даний приклад може породжувати помилки. Наприклад, якщо ім’я або фамілія міститимуть пробіли, то об’єкт буде зчитаний неправильно, що спотворить подальшу десеріалізацію.

Подібна серіалізація, теоретично, може задовільнити довільний намір користувача, але занадто сильно ускладнює роботу та вимагатиме завелику кількість коду для підтримки декількох намірів.

1.4 Абстрагування серіалізації

Минуле рішення можна розглянути, як об’єкт з можливістю привести до послідовності байтів та зчитати з неї за допомогою інструкцій користувача. [Рисунок 2, структура простої серіалізації]

Рисунок 2, структура простої серіалізації

Незважаючи на те, що подібна реалізація надає великий рівень свободи, в більшості випадках об’єкт відокремлюють від потоку байтів задля спрощення. З’являється певна абстракція поверх потоку байтів, яка бере декотру частину відповідальності на себе. Можна виявити три види абстракцій: серіалізація поверх формату, серіалізація поверх моделі та шаблонна серіалізація.

1.4.1 Формат

При серіалізації поверх формату розробник обирає певний стандарт або визначає власний для інтерпретації послідовності байтів. Ціллю даного підходу є створення класів, роль яких є приведення формату до потоку байтів та відтворення формату з потоку байтів. Завдання користувача складається у написанні інструкцій, що будуть приводити клас до формату. [Рисунок 3, структура серіалізації поверх формату]

Рисунок 3, структура серіалізації поверх формату

Прикладами бібліотек, які використовують подібну структуру є msgpack-c, Boost.JSON. Також, можна виділити бібліотеки, де формат створюється користувачем за допомогою схеми: бази даних, Protocol Buffers, Cap'n Proto.

Розглянемо серіалізацію за допомогою бібліотеки Boost.JSON для класу Employee, який був визначений раніше.[2] Метод save та функція read виконують приведення до формату, який бібліотека підтримує. [Рисунок 4, приклад серіалізації поверх формату]

```
boost::json::value save() {
  boost::json::object obj;
  obj["first_name"] = _firstName;
  obj["second_name"] = _lastName;
  return obj;
}

static Person read(boost::json::value& v) {
  boost::json::object& obj = v.as_object();
  return Person(
    std::string(obj["first_name"].as_string().operator boost::json::string_view()),
    std::string(obj["second_name"].as_string().operator boost::json::string_view()));
}
```

Рисунок 4, приклад серіалізації поверх формату

Порівняно з простою серіалізацією, дана реалізація безпечніша: методи не можуть спотворювати десеріалізацію при певних значеннях даних, але вибір формату складно змінити. У даному випадку була використаний JSON, який зазвичай використовують під час комунікації у веб-застосунках, але, якщо дані необхідно буде не лише серіалізувати для надсилання даних іншому застосунку, а і кешувати, то користувач буде змушений використати досить неефективний формат при кешуванні або додати іншу бібліотеку з функціями, що будуть робити ті самі операції для нового формату.

Дана абстракція надає найвищий рівень контролю, якщо не враховувати просту серіалізацію, адже користувач може використовувати всі можливості формату для збереження об'єкту.

1.4.2 Модель даних

Серіалізація поверх моделі додає рівень абстракції між об’єктом та форматом. Причиною для цього є завелика кількість форматів серіалізації та умовна схожість між ними. Різні формати використовують подібну структуру і мають подібні можливості. У зв’язку з цим серіалізацію можна виконувати поверх «Моделі даних», яка в свою чергу може привести об’єкт до певного формату. [Рисунок 5, структура серіалізації поверх моделі даних]

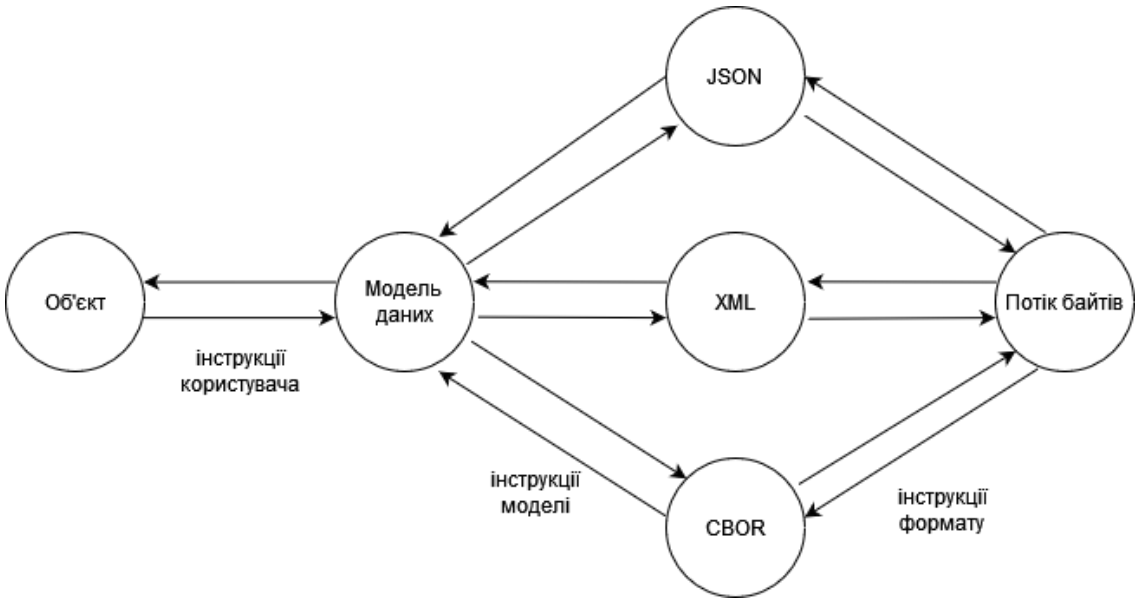

Рисунок 5, структура серіалізації поверх моделі даних

Подібна реалізація надає можливість користувачу не залежати від обраного формату, що полегшує перехід між ними. Наприклад, для комунікації з іншим застосунком користувач використовує JSON, а для кешування вживається більш оптимальний формат як CBOR. Користувач може визначити функції серіалізації один раз і зберігати об’єкт в декількох форматах.

Розглянемо подібну абстракцію в бібліотеці Serde. Інтерфейс Serializer виступає моделлю даних, яку формат повинен підтримувати.[10] [Рисунок 6, часткове визначення моделі даних Serde]

```rust
pub trait Serializer {
    type Ok;
    type Error: Error;
    type SerializeSeq: SerializeSeq<Ok = Self::Ok, Error = Self::Error>;
    type SerializeMap: SerializeMap<Ok = Self::Ok, Error = Self::Error>;
    type SerializeStruct: SerializeStruct<Ok = Self::Ok, Error = Self::Error>;

    // Required methods
    fn serialize_bool(self, v: bool) -> Result<Self::Ok, Self::Error>;
    fn serialize_i64(self, v: i64) -> Result<Self::Ok, Self::Error>;
    fn serialize_u64(self, v: u64) -> Result<Self::Ok, Self::Error>;
    fn serialize_char(self, v: char) -> Result<Self::Ok, Self::Error>;
    fn serialize_str(self, v: &str) -> Result<Self::Ok, Self::Error>;

    fn serialize_seq(self, len: Option<usize>) -> Result<Self::SerializeSeq, Self::Error>;
    fn serialize_map(self, len: Option<usize>) -> Result<Self::SerializeMap, Self::Error>;
    fn serialize_struct(self, name: &'static str,len: usize) -> Result<Self::SerializeStruct, Self::Error>;
}
```

Рисунок 6, часткове визначення моделі даних Serde

Реалізація серіалізації для класу Person відбуватиметься за допомогою інтерфейсу Serialize. Користувач повинен вказати інструкції для збереження об’єкту за допомогою Моделі даних у методі serialize. [Рисунок 7, приклад серіалізації поверх моделі даних]

```rust
struct Person {
    first_name: String,
    last_name: String,
}

impl Serialize for Person {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: Serializer,
    {
        let mut state = serializer.serialize_struct("Person", 2)?;
        state.serialize_field("first_name", &self.first_name);
        state.serialize_field("last_name", &self.last_name);
        state.end()
    }
}
```

Рисунок 7, приклад серіалізації поверх моделі даних

Даних інструкцій достатньо для збереження об'єкту Person. Користувач зможе вибрати один з підтримуваних моделлю даних форматів та зберегти екземпляр. У прикладі ми використовуємо бібліотеки rmp_serde та serde_json задля збереження у форматах MessagePack та JSON відповідно. [Рисунок 8, приклад збереження об'єкту бібліотекою Serde]

```
let person = Person {
    first_name: "Andrii".to_string(),
    last_name: "Zahorulko".to_string(),
};
{
    let mut msgpack_byte_buffer = Vec::<u8>::new();
    person
        .serialize(&mut rmp_serde::Serializer::new(&mut msgpack_byte_buffer))
        .unwrap();
}

{
    let mut json_byte_buffer = Vec::<u8>::new();
    person
        .serialize(&mut serde_json::Serializer::new(&mut json_byte_buffer))
        .unwrap();
}
```

Рисунок 8, приклад збереження об'єкту бібліотекою Serde

Порівняно з серіалізацією поверх формату ми можемо використовувати методи для серіалізації у декілька форматів. Якщо повернутися до теоретичної ситуації кешування даних, то користувачу не потрібно буде додавати методи серіалізації для нового формату.

З іншого боку, модель даних обмежує можливості серіалізації, адже вона узагальнює можливості різних форматів. У результаті рівень свободи користувача зменшується порівняно з серіалізацією поверх формату.

1.4.3 Шаблонна серіалізація

Шаблонна серіалізація заснована на спробі з’єднати інструкції користувача з форматом за допомогою «функцій серіалізації», які призначенні до збереження певного типу об’єкту до певного формату. Даний підхід дозволяє реалізувати

збереження у різні формати без необхідності конкретизації модель даних. У зв’язку з відсутністю форми порівняно з минулими варіантами інструкції користувача не можуть бути складними, через неструктурованість функцій серіалізації. [Рисунок 9, структура шаблонної серіалізації]

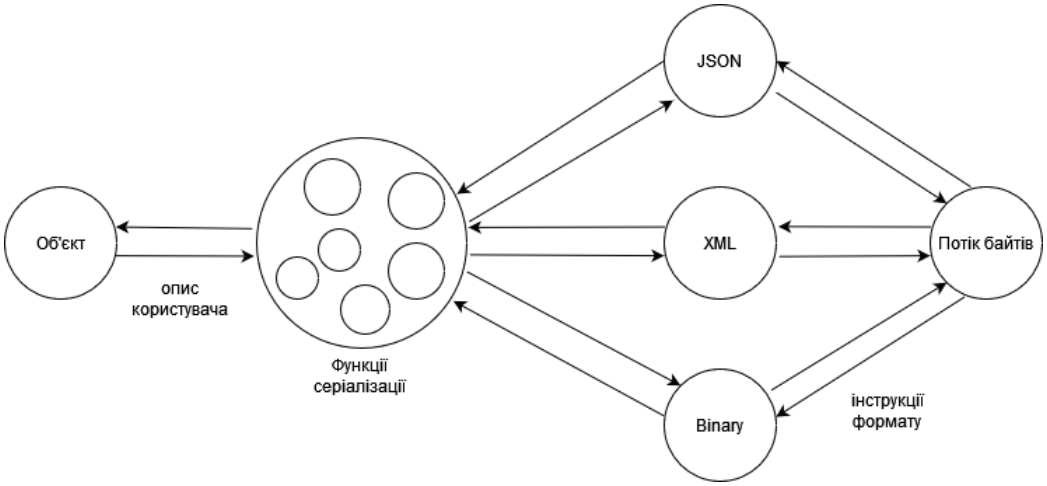

Рисунок 9, структура шаблонної серіалізації

Розглянемо приклад серіалізації у C++ з бібліотекою Boost та Cereal, які використовують шаблону серіалізацію. У Boost аргументами є архіватор та версія об’єкту, у Cereal лише архіватор.[3,12] Головна ідея – спростити серіалізацію до вказування атрибутів необхідних для збереження та відтворення. Коли користувач буде намагатися зберегти об'єкт за допомого обраного архіватору і будуть існувати функції для збереження типів кожного з атрибутів, то код буде скомпільований. [Рисунок 10, приклад шаблонної серіалізації]

```
template<class Archive>
void serialize(Archive& ar, unsigned int version) {
  ar& BOOST_SERIALIZATION_NVP(_firstName);
  ar& BOOST_SERIALIZATION_NVP(_lastName);
}

template<class Archive>
void serialize(Archive& ar) {
  ar(_firstName, _lastName);
}
```

Рисунок 10, приклад шаблонної серіалізації

Дана реалізація має деякі покращення порівняно з моделлю даних. Так як відсутня структура, функції можуть використовувати особливості певного формату у своїй реалізації. Проте, система неможлива в більшості мов програмування без значних ускладнень. Спеціалізація функцій у цій абстракції є необхідністю, але об’єктно-орієнтовані мови програмування як C# та Java не надають таких можливостей.

Порівняно з іншими абстракціями користувач має найменшу кількість свободи при використанні даної абстракції. Причиною є загальна складність узагальненого програмування в C++ та цілі які намагаються досягти бібліотеки. Boost.Serialization намагається максимально спростити серіалізацію, і майже повністю контролює процес збереження об’єктів. У результаті якщо бібліотека не підтримує певну операцію, в користувача може не бути варіанту його вирішити.

1.5 Стиль інструкцій

При дослідженні способів реалізації серіалізації було виявлено два основних стилі визначення інструкцій для серіалізації: імперативний та декларативний.

Імперативний стиль вимагає від користувача інструкцій для приведення об’єкту до формату. Минулі приклади серіалізації поверх формату та моделі даних використовували імперативний стиль. Форма даних при збереженні залежить від

інструкцій користувача, а не від об’єкту. За необхідності, користувач може замість структури зберегти число, або зберегти результат у список.

Декларативний стиль використовує опис об’єкту для серіалізації. Залежно від реалізації описом може бути сформований рефлексією, спеціальними генераторами коду або користувачем. Бібліотеки як Boost.Serialization та Cereal виконують серіалізацію за допомогою прив’язування атрибутів об’єкту до серіалізатора; шаблонні функції serialize повністю описують об’єкт.

Даний підхід має декотрі покращення порівняно з імперативним стилем:

1. Кількість коду необхідна для реалізації простого збереження значно менша і може додатково зменшитися, якщо використовується рефлексія або генерація коду.
2. Зменшується кількість місць для можливих помилок.

Проте подібний підхід до серіалізації не підходить для реалізації збереження складних об’єктів. Доцільність імперативної серіалізації проявляє себе, коли вигляд об’єкту в збереженому форматі відрізняється від вигляду об’єкту у коді.

Розглянемо клас Library, який містить список книжок та авторів. Книжки містять назву та список авторів, які її написали. Якщо автори посилаються на книжки, які написали, то декларативна серіалізація або унеможливлюється або вимагає створення певних особливостей серіалізації. Наприклад, Cereal зберігає реєстр всіх збережених вказівників, щоб розуміти, чи треба об’єкт серіалізовувати.

Стаття «C++ FAQ Lite» Маршала Клайна розрізняє 5 рівнів складності об’єктів для серіалізації. Найскладнішим вважається циклічним графом.[5] Якщо розглядати алгоритм запропонований Клайном для серіалізації подібного об’єкту, то при використанні імперативного стилю користувач матиме можливість одразу зберігати вершини. На противагу, при серіалізації в декларативному стилі користувачу потрібно спочатку визначити список об’єктів, які потрібно зберегти, і лише потім виконати серіалізацію.

Важливо зауважити, що бібліотеки можуть підтримувати обидва стилі серіалізації. Наприклад, бібліотека Serde підтримує декларативну серіалізацію за допомогою генерації коду, але дозволяє за необхідності самостійно визначати інструкції збереження.

Розділ 2. Опис системи та реалізація її у C++ 20

2.1 Вимоги до системи

Проаналізувавши способи абстрагування серіалізації та базові наміри користувача, було сформульовано подібні вимоги до системи:

1. Модель даних.

   Модель даних надає певний рівень незалежності від форматів, що дозволяє за допомогою однієї бібліотеки підтримувати більшість намірів користувачів.

2. Імперативний стиль

   Рівень свободи, який надається користувачу імперативним стилем серіалізації значно більший порівняно з декларативною серіалізацією. Реалізація обов'язково повинна підтримувати імперативний стиль, але може, додатково, підтримувати декларативний.

3. Підтримка ієрархії класу та поліморфізму.

   Система намагається задовольнити вимоги більшості користувачів, що вимагає підтримки збереження ієрархій класів.

4. Мовна агностичність.

   Система серіалізації не повинна використовувати особливостей певної мови для ядра реалізації і повинна мати можливість бути реалізованою в більшості сучасних об’єктно-орієнтованих мов. Це вирішує намір користувача комунікації між різними системами, що наддасть можливість використовувати дану систему в мікросервісній архітектурі.

5. Відсутність особливих правил.

   Система не повинна додавати власних правил до серіалізації та десеріалізації. Наприклад, збереження циклічних посилань відбувається ідентично до збереження звичайних посилань. Система не повинна виконувати перевірку на

попередню серіалізацію об’єкту, дане завдання лежить на користувачеві. Єдина особистість яку система матиме – збереження поліморфних типів.

Порівняно з цілями, які Boost.Serialization реалізовує, дані вимоги суттєво відрізняються. З перелічених у Boost.Serialization є належне відновлення вказівників до спільних об’єктів та можливість зберігати класи без доповнення.[12] Наша реалізація порушуватиме ці вимоги, адже вона не спрямована для роботу лише в одній мові та не намагається повністю замінити подібні бібліотеки.

Причина вибору подібних вимог – бажання задовольнити різні наміри серіалізації за допомогою одної системи. Більшість бібліотек лише спеціалізуються на одному намірі, що спрощує роботу, але фрагментує середовище. Наша реалізація намагатиметься задовільнити потреби користувача у більшості випадках, хоч і ускладнить роботу загалом.

Під час написання цієї роботи були розглянуті Boost.JSON, Boost.Serialization, Serde, msgpack-c, Cereal, kotlinx.serialization, Jackson, з яких жодна не реалізує всі вимоги.

1. Boost.JSON, Msgpack-c та інші бібліотеки поверх формату порушують вимогу 1. Також, в більшості випадках, дані бібліотеки не підтримують збереження ієрархії класів та залишають це завдання для користувача.
2. Boost.Serialization, Cereal порушують 1, 2, 4, 5. Обидві бібліотеки намагаються якомога спростити збереження об’єктів, що, загалом, є гарною ідеєю. З іншого боку, подібні спрощення можуть працювати лише за рахунок невиконання зазначених вимог.
3. Serde порушує 3, 4. Так як бібліотека була створена для мови програмування Rust, вона активно використовує можливості, які відсутні в інших мовах програмування. Також, вона не підтримує серіалізацію ієрархії класів.
4. kotlinx-serialization порушує 4 та частково 3. Хоч, бібліотека використовує модель даних і дозволяє користувачу створювати власні класи для серіалізації, вона не містить пояснень, щодо збереження об’єктів ієрархії класів. В результаті, користувач змушений обрати між серіалізацією з успадкуванням та

імперативним стилем. Варто зауважити, що навіть якщо бібліотека виконуватиме вимогу 3, її напрям залишатиметься серіалізація об’єктів у мові програмування Kotlin.

5. Jackson порушує 4. Бібліотека активно використовує рефлексію, яку мови як C++ не підтримують.

2.2 Опис системи

Назва: Serene.

Імплементація системи була написана для C++ 20. При створенні реалізації була використана бібліотека Boost.JSON для десеріалізації об'єктів з формату JSON та система керування пакунками vcpkg.[2, 13]

2.3 Модель даних

Задля можливості підтримки моделі даних у об’єктно орієнтованих мовах були використаний інтерфейси AbstractSerializer та AbstractDeserializer. Похідні класи слугують адаптерами або реалізаторами, що можуть перетворювати запити «зберегти число» на правильний виклик методу або на інструкції, які за певним форматом приводять об’єкт до послідовності байтів.

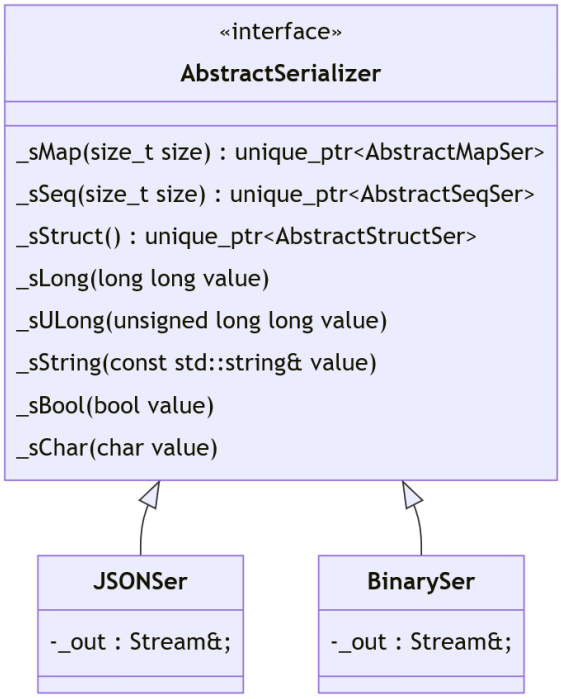

Рисунок 11, структура серіалізаторів

Модель даних визначає 3 складених структури, які може зберегти: асоціативний масив ключів стрічок та значень, послідовність, структура. У даному контексті структура є впорядкованою послідовністю елементів з відомим розміром. [Рисунок 12, складені серіалізатори]

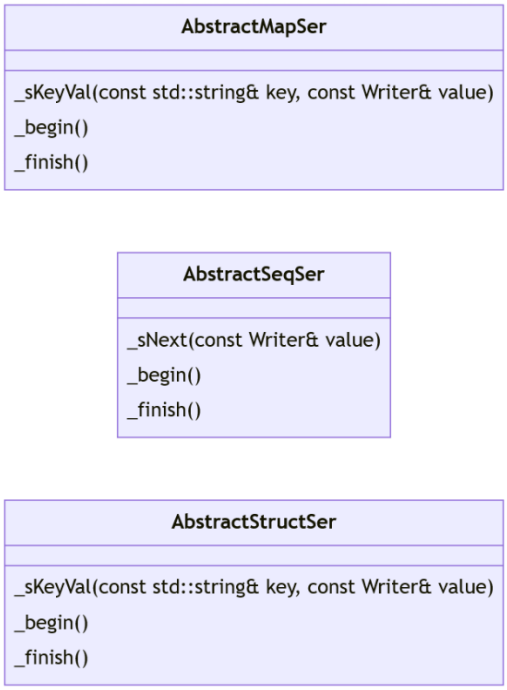

Рисунок 12, складені серіалізатори

- AbstarctMapSer, AbstractMapDe –збереження пар ключів-значення, де ключем виступає std::string.
- AbstractSeqSer, AbstractSeqDe –збереження послідовностей.
- AbstractStructSer, AbstractStructDe –збереження впорядкованої послідовності елементів з передвизначеною кількістю.

Десеріалізатор має подібну структуру до серіалізатора. Відмінності наявні лише в серіалізаторах послідовностей та множин ключів-значень. Кількість елементів десеріалізатор не обов’язково знатиме, тому користувач може дізнатись лише наявність незчитаних елементів. [Рисунок 13, структура десеріалізаторів] [Рисунок 14, структура складених десеріалізаторів]

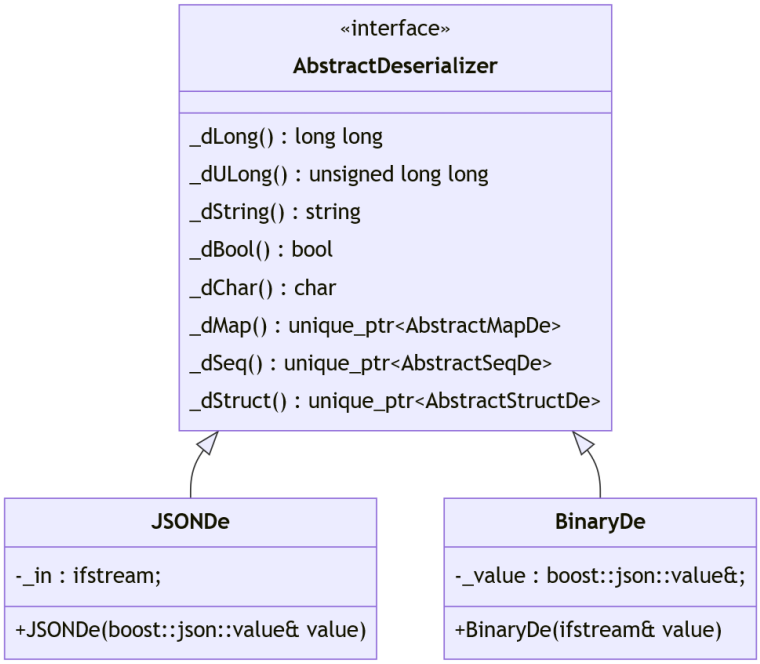

Рисунок 13, структура десеріалізаторів

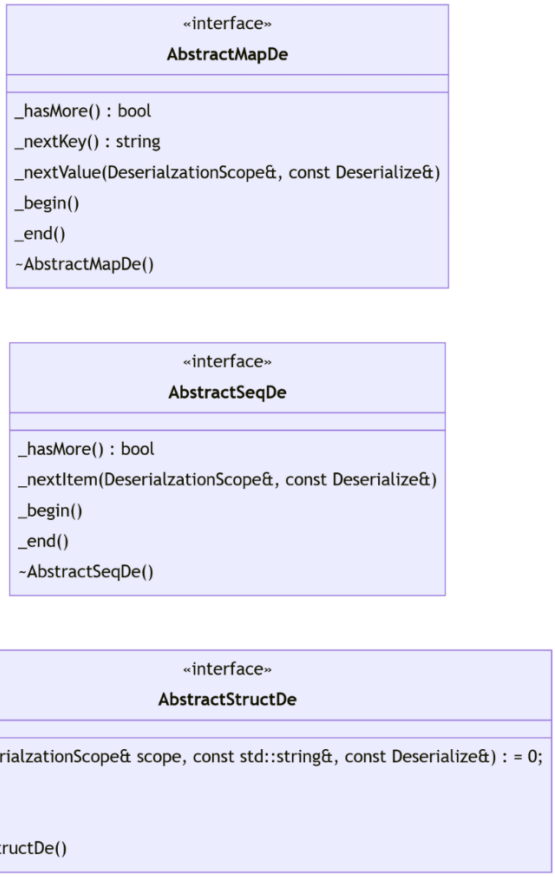

Рисунок 14, структура складених десеріалізаторів

Модель підтримує збереження об’єктів до бінарного формату та JSON. Варто зауважити, що система спрощена, і у результаті не підтримує всі можливості JSON.

Для десеріалізації об’єкту з JSON був використана бібліотека Boost.JSON та створений клас JSONDe, який слугує адаптером моделі даних з бібліотекою серіалізації формату.

2.4 Збереження

Серіалізація відбувається за допомогою спеціалізацій функції write з використанням концептів. У C++ дана функція є стратегією серіалізації, яка повертає лямбда-функцію з інструкціями для збереження об’єкту. У інших мовах можна уникнути спеціалізації функцій за допомогою класів або делегатів. При збереженні складених об’єктів як списків класи прийматимуть фабричну функцію, яка створюватиме зберігачі кожного окремого об’єкту списку. Причиною вибору подібної стратегії є необхідність делегувати серіалізацію при збереженні послідовностей, структур, асоціативних масивів. [Рисунок 15, вигляд функції серіалізації]

```
using Writer = std::function<void(AbstractSerializer&)>;

inline Writer write(std::same_as<long long> auto v) {
  return [v](AbstractSerializer& s) {s.sLong(v); };
}
```

Рисунок 15, вигляд функції серіалізації

Для збереження об’єктів можна, також, використовувати спеціалізацію write, але задля можливості посилатися на атрибути та збереження ієрархії класів об’єкт повинен успадковувати інтерфейс SerializableObject, який містить метод _serialize

для створення зберігача та _getObjectName для визначення типу об’єкту при поліморфізмі.

Для можливості зберігати об’єкти за допомогою функції write необхідна спеціалізація, яка вимагатиме успадкування інтерфейсу. [Рисунок 16, функція серіалізації об’єкту, який успадковує SerializableObject]

```
template<typename T>
  requires std::derived_from<T, SerializableObject>
inline Writer write(const T& v) {
  return v.serialize();
}
```

Рисунок 16, функція серіалізації об’єкту, який успадковує SerializableObject

Також, збереження типів з невизначеною формою вимагає іншої реалізації функції write, яка буде додавати певне маркування про тип об’єкту, адже під час десеріалізації потрібно розуміти, який саме десеріалізатор треба використовувати. У нашій реалізації це завдання виконує метод _getObjectName, у якому об’єкти вказують назву класу, яка буде використана при серіалізації вказівника. [Рисунок 17, серіалізація поліморфічних типів]

```
template<typename T>
requires std::derived_from<T, SerializableObject>
inline Writer write(const T* v) {
  return [v](AbstractSerializer& s) {
    auto structSer = s.sStruct();
    structSer->begin();
    structSer->sKeyVal("type", write<std::string>(v->getObjectName()));
    structSer->sKeyVal("value", write<T>(*v));
    structSer->finish();
  };
}
```

Рисунок 17, серіалізація поліморфічних типів

Варто зауважити, що розрізнення шаблонів за посиланням та вказівником спрямовано на спрощення роботи. Якщо користувач бажає зберегти атрибут, який може бути поліморфічним, то потрібно буде використати оператор адресування «&».

2.5 Зв’язок із серіалізатором

При збереженні об’єкту користувач може обрати між прямим викликом методів моделі даних та використанням існуючих функцій серіалізації. У результаті збереження об’єкту може мати різні вигляди зі схожим об’єктним кодом. Наприклад, клас для серіалізації класу Color користувач може або викликати функцію write або викликати метод моделі даних напряму. [Рисунок 18, серіалізація за допомогою спеціалізації функції] [Рисунок 19, серіалізація за допомогою методів моделі даних]

```
Writer Color::serilaizeCurrent() const
{
  return [this](AbstractSerializer& se) {
    unsigned long long value =
      (unsigned long long) _red << 16
      | (unsigned long long) _green << 8
      | (unsigned long long) _blue;

    write(value)(se);
  };
}
```

Рисунок 18, серіалізація за допомогою спеціалізації функції

```
Writer Color::serilaizeCurrent() const
{
  return [this](AbstractSerializer& se) {
    se.sULong((unsigned long long) _red << 16
      | (unsigned long long) _green << 8
      | (unsigned long long) _blue);
  };
}
```

Рисунок 19, серіалізація за допомогою методів моделі даних

Для збереження декількох атрибутів варто використовувати серіалізатор структури, асоціативного масиву або послідовності. У даному випадку ми використовуємо серіалізатор структури. [Рисунок 20, серіалізація складеного об'єкту]

```
  return [this](AbstractSerializer& se) {
    auto structSer = se.sStruct();
    structSer->begin();
    structSer->sKeyVal("red", write(_red));
    structSer->sKeyVal("green", write(_green));
    structSer->sKeyVal("blue", write(_blue));
    structSer->finish();
  };
```

Рисунок 20, серіалізація складеного об'єкту

2.6 Відтворення типів

Десеріалізація, загалом, подібна до серіалізації, але до базових вимог додається питання розташування об’єкту в пам’яті. Відтворення об’єкту може відбуватися за допомогою виділення динамічної пам’яті і більшість об’єктно-орієнтованих мов матиме можливість реалізувати подібну систему. Проте, у випадку C++, об'єкти

можна створювати і на стеці, і у динамічній пам’яті. У зв'язку з подібною свободою реалізація намагається надати користувачу можливість десеріалізувати об'єкт у довільно визначене місце в пам'яті. Для виконання цього завдання використовується інтерфейс BufferAlloc, завдання якого виділити місце в пам’яті для об’єкту.
[Рисунок 21, структура алокаторів пам’яті]

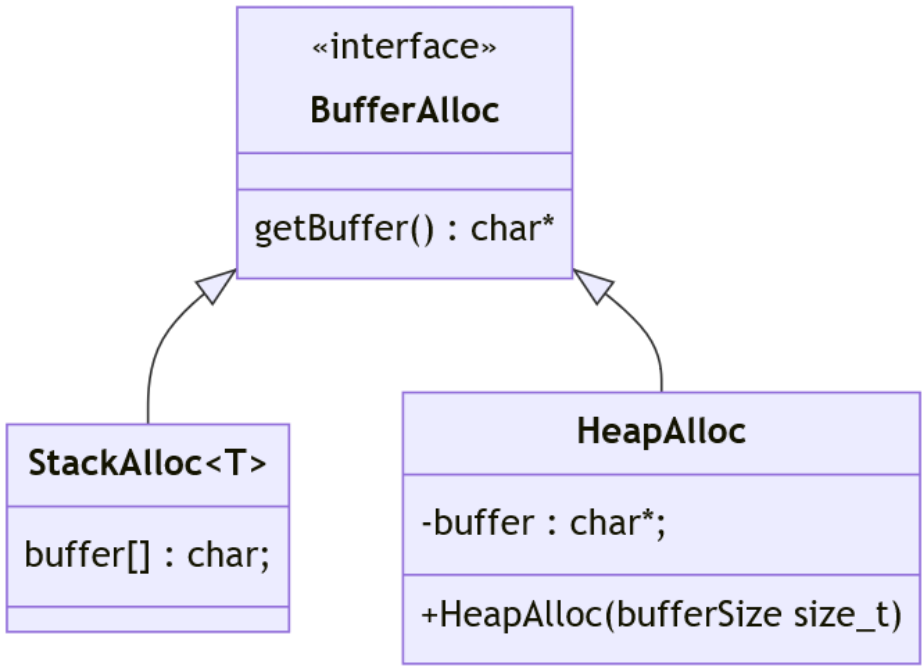

Рисунок 21, структура алокаторів пам’яті

«getBuffer» – метод, який повертає вже зарезервоване місце в пам’яті, де можна створити об’єкт. У результаті залежно від стратегії об’єкт можна створювати і на стеці, і на купі.

Відтворення об’єкту відбувається за допомогою спеціалізацій функції read. Лямбда-функція захоплює місце в пам’яті, яке далі буде вказане при створенні об’єкту за допомогою ключового слова new. Параметрами лямбда-функції слугують десеріалізатор та область десеріалізації. Область призначена для можливості передавання об’єкту посилань необхідних для створення об’єкту, адже функція не повинна приймати додаткових параметрів. Подібне ускладнення існує через необхідність десеріалізувати поліморфні типи.

```
using Deserialize =
    std::function<void(AbstractDeserializer&, DeserialzationScope&)>;

template<typename T>
  requires std::same_as<T, long long>
Deserialize read(BufferAlloc& v) {
  return [&v](AbstractDeserializer& de, DeserialzationScope&) {
    new (v.getBuffer()) long long(de.dLong());
  };
}
```

Рисунок 22, відтворення типу

Для відтворення об’єкт повинен мати статичну містити статичну функцію read або визначати спеціалізацію функції read. Визначення функції, яка буде використана для відтворення, відбувається за допомогою внутрішньої реєстрації класів. З боку користувача необхідно лише використати макрос RegisterSerializeClass(Type), який додає статичний атрибут typeName, перевизначає метод, що вказує тип об’єкту, та додає клас до реєстру.

Шаблона функція registerClassDeserialize, додає десеріалізатор класу до TypeAccumulator, який містить асоціативний масив назв об’єктів та функцій, які можна використати при десеріалізації.

Для можливості розрізняти нормальну десеріалізацію об’єкту та поліморфну була додана шаблона функція deserialize. У випадку, коли функції наданий вказівник об'єкту відбудеться поліморфна десеріалізація: буде зчитаний тип та обрана правильна функція з TypeAccumulator, яка зчитає дані та створить об’єкт. Якщо функції буде переданий буфер, відбудеться нормальна серіалізація.

Статична ініціалізація була обрана у зв’язку з простотою використання її зі сторони користувача та підтримкою в інших об’єктно-орієнтованих мовах, як Java, C#. Варто

зауважити, що у даних мовах необхідно буде використати клас перед десеріалізацію або самостійно завантажити клас, так як статична ініціалізація відбувається ліниво.

2.7 Зв’язок з десеріалізатором

Зі сторони користувача великих відмін між десеріалізацією та серіалізацією нема. Можна використовувати або спеціалізації функції deserialize або напряму викликати методи моделі даних. У даному прикладі відбувається відтворення кольору зі структури. Користувач виділяє місце для десеріалізації за допомогою DeserializeBuffer, зчитує структуру з моделі даних і створює об’єкт з отриманими даними. [Рисунок 23, приклад відтворення кольору]

```
Deserialize Color::read(BufferAlloc& buffer)
{
  return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
    DeserializeBuffer<char> red;
    DeserializeBuffer<char> green;
    DeserializeBuffer<char> blue;
    auto structSer = de.dStruct();
    structSer->begin();
    structSer->next(scope, "red", deserialize(red.buffer()));
    structSer->next(scope, "green", deserialize(green.buffer()));
    structSer->next(scope, "blue", deserialize(blue.buffer()));
    structSer->end();
    new (buffer.getBuffer()) Color(*red, *green, *blue);
  };
}
```

Рисунок 23, приклад відтворення кольору

Розділ 3. Демонстрація роботи

3.1 Опис застосунку

Для прикладу було обрано створити RESTful сервіс каталогу меблів. Застосунок підтримує три категорії меблів: стіл, стілець та ліжко. Кожна категорія містить особливі характеристики, але всі мають назву для ідентифікації. [Рисунок 24, ієрархія меблів]

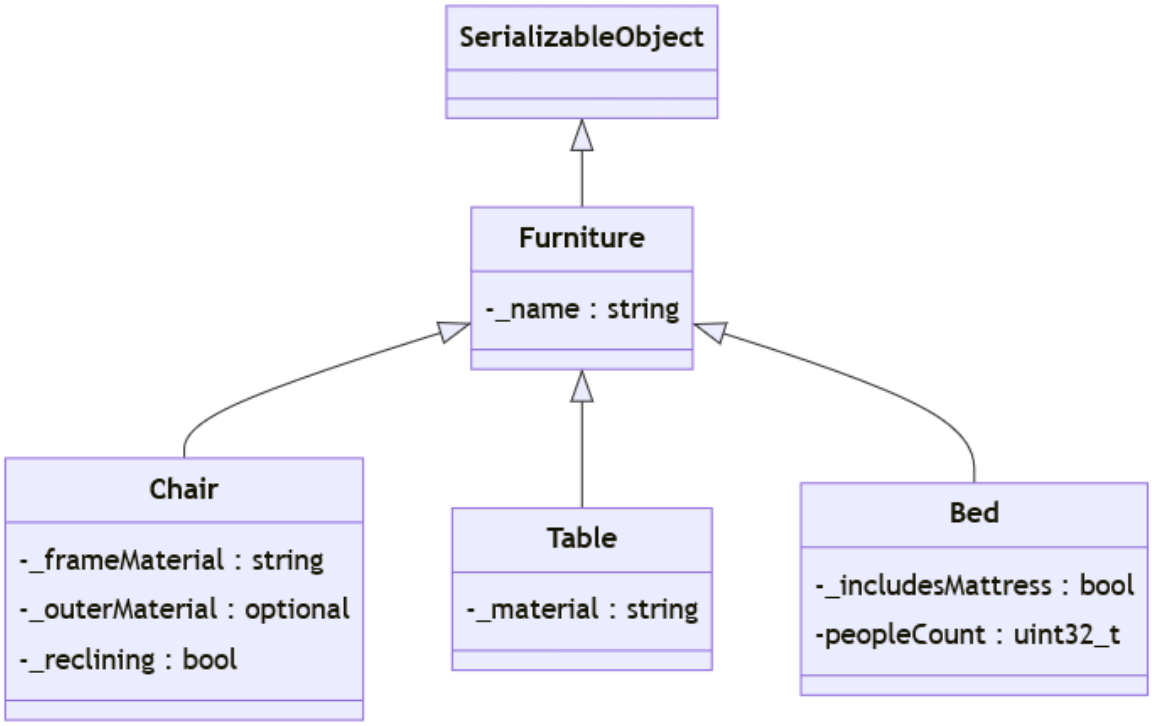

Рисунок 24, ієрархія меблів

Застосунок має дві кінцеві точки:

- «/furniture/(.+)» з методом GET дозволяє отримати меблі за назвою.
- «/furniture» з методом PUT дозволяє додати або змінити меблі.

Даний функціонал спрямований на демонстрацію реалізації наміру комунікації з користувачем. Формат, який використовується для комунікації – JSON.

Сервер зберігає дані у файловій системі, кешує дані для уникання зчитування файлів. Для даних операцій використаний бінарний формат. Функціонал демонструє намір збереження даних між сесіями та кешування відповідно.

Для тестування сервер має можливість завантажити макетні дані з JSON файлу, що спрощує розробку.

На останок, сервер використовує JSON файл при налаштуванні порту, на якому сервер буде очікувати запити, та моду застосунку. Так як у файлі можна уникати поля необхідні для налаштування, десеріалізатор повинен бути готовим до можливості відсутності значень. Наприклад, якщо користувач не вказав порт, застосунок використовуватиме 8080.

3.2 Перевірка працездатності

Налаштування застосунку містить об’єкт лише з вказаним портом. [Рисунок 25, файл налаштування застосунку]

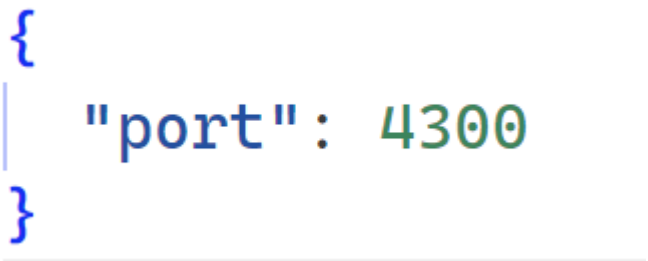

Рисунок 25, файл налаштування застосунку

Сервер завантажується зі значенням порту 4300 та модом розробника. Зчитуються дані з mockData.json файлу для тестування застосунку. Меблі додаються до папки furnitures.

3.2.1 Запити

Для перевірки RESTful сервіса був використаний застосунок Postman.

HTTP запит	**Консоль/HTTP відповідь**

GET localhost:4300/furniture/Brimnes	Using value from file
	{
	"hasValue": true,
	"value": {
	"type": "Bed",
	"value": {
	"name": "Brimnes",

	"includesMattress": false, "peopleCount": 1 } } }

GET localhost:4300/furniture/Noone	Couldn't read file
	{
	"hasValue": false
	}

GET localhost:4300/furniture/
Eames%20Lounge%20Chair
	Using value from file
	{
	"hasValue": true,
	"value": {
	"type": "Table",
	"value": {
	"name": "Eames Lounge Chair",
	"material": "wood"
	}
	}
	}

10 секунд після минулого запиту
GET localhost:4300/furniture/
Eames%20Lounge%20Chair
	Using value from cache: ETableTEames Lounge ChairДwood
	{
	"hasValue": true,
	"value": {
	"type": "Table",
	"value": {
	"name": "Eames Lounge Chair",
	"material": "wood"

	} } }

GET localhost:4300/furniture/
The%20Sitwell
	Using value from file
	{
	"hasValue": true,
	"value": {
	"type": "Chair",
	"value": {
	"name": "The Sitwell",
	"frameMaterial": "wood",
	"outerMaterial": {
	"hasValue": true,
	"value": "cotton"
	},
	"reclining": true
	}
	}
	}

PUT localhost:4300/furniture
{
"type": "Table",
"value": {
"name": "Lolounge",
"material": "metal"
}
}
	Status: 200 Ok

GET localhost:4300/furniture/Lolounge	Using value from file {

	"hasValue": true,
	"value": {
	"type": "Table",
	"value": {
	"name": "Lolounge",
	"material": "metal"
	}
	}
	}

3.3 Порівняння з іншими реалізаціями

Дану систему можна порівняти з двома іншими варіантами: використання двох бібліотек серіалізації поверх формату або використання бібліотеки Cereal.

Якщо користувач вибере використовувати бібліотеки для збереження поверх формату, то він зіткнеться з необхідністю дублювати операції збереження та відтворення для всіх об’єктів. Дана проблема буде ускладнена через необхідність самостійно реалізувати збереження ієрархії класів, адже бібліотеки Boost.JSON та msgpack-c не підтримують успадкування.

Порівняння з Cereal є складнішим, адже дана бібліотека має зовсім інший підхід до збереження. Наша система може бути адаптована до інших об’єктно-орієнтованих мов програмування, бо вона не використовує особливостей мови C++ у ядрі системи. У результаті систему можна адаптувати на стороні клієнту на мовах як Java та C#. Cereal активно використовує шаблонні функції та параметричні класи для збереження об’єктів. Незважаючи на те, що це спрощує роботу розробникам у C++, її не можна буде реплікувати інших мовах, де узагальнене програмування реалізовано по-іншому.

Для найпростішого збереження об'єкту за допомогою Cereal необхідно значно менша кількість коду. Наприклад, збереження класу Chair потребувало 5 стрічок

коду порівняно з 29 стрічок коду. [Рисунок 26, порівняння з серіалізацією Cereal]

```
// Cereal version
template<class Archive>
void serialize(Archive& ar)
{
  ar(cereal::base_class<Furniture>(this), _frameMaterial, _outerMaterial, _reclining);
}

// Our version
Writer Chair::serializeCurrent() const
{
  return [this](AbstractSerializer& serializer) {
    auto stSer = serializer.sStruct();
    stSer->begin();
    stSer->sKeyVal("name", write(_name));
    stSer->sKeyVal("frameMaterial", write(_frameMaterial));
    stSer->sKeyVal("outerMaterial", write(_outerMaterial));
    stSer->sKeyVal("reclining", write(_reclining));
    stSer->finish();
  };
}

Deserialize Chair::read(BufferAlloc& buffer)
{
  return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
    DeserializeBuffer<string> name;
    DeserializeBuffer<string> frameMaterial;
    DeserializeBuffer<optional<string>> outerMaterial;
    DeserializeBuffer<bool> reclining;
    auto stSer = de.dStruct();
    stSer->begin();
    stSer->next(scope, "name", deserialize(name.buffer()));
    stSer->next(scope, "frameMaterial", deserialize(frameMaterial.buffer()));
    stSer->next(scope, "outerMaterial", deserialize(outerMaterial.buffer()));
    stSer->next(scope, "reclining", deserialize(reclining.buffer()));
    stSer->end();
    new (buffer.getBuffer()) Chair(*name, *frameMaterial, *outerMaterial, *reclining);
  };
}
```

Рисунок 26, порівняння з серіалізацією Cereal

В іншому випадку спрощення, які робить Cereal ускладнюють десеріалізацію. Файл для конфігурації застосунку не зобов’язаний зберігати всю інформацію, що у випадку Cereal породжує проблему, адже бібліотека не підтримує подібних можливостей. У випадку нашої системи можна використати десеріалізатор ключів-значень. [Рисунок 27, десеріалізація налаштування застосунку]

```
Deserialize Configuration::read(BufferAlloc& buffer)
{
 return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
   DeserializeBuffer<int> port;
   bool hasPort = false;
   DeserializeBuffer<bool> production;
   bool hasProduction = false;
   auto stSer = de.dMap();
   stSer->begin();
   while (stSer->hasMore())
   {
    auto next = stSer->nextKey();
    if (next == "port") {
     stSer->nextValue(scope, deserialize(port.buffer()));
     hasPort = true;
    }
    else if (next == "production") {
     stSer->nextValue(scope, deserialize(production.buffer()));
     hasProduction = true;
    }
   }
   int rPort = hasPort ? *port : 8080;
   bool rProduction = hasProduction ? *production : false;
   stSer->end();
   new (buffer.getBuffer()) Configuration(rPort, rProduction);
 };
}
```

Рисунок 27, десеріалізація налаштування застосунку

3.4 Аналіз результатів роботи

Система надала можливість реалізувати серіалізацію для різних намірів: зберігання між сесіями застосунку, кешування, комунікація, тестування та налаштування. Модель даних надала рівень свободи, який зробив можливим десеріалізацію об’єктів у випадках, коли не всі значення можуть бути присутніми, а також автоматичну підтримку декількох форматів серіалізації.

З іншого боку, кількість коду необхідна для реалізації простої серіалізації є завеликою. Порівняно із системою спеціалізованою під мову C++ наша реалізація містить завелику кількість ускладнень.

Також варто зауважити обмеженість операцій у моделі даних та відсутність можливості визначити тип поля. Наприклад, збереження std::optional може складатися зі збереження null у JSON форматі, але подібна реалізація буде неможлива через відсутність можливості визначення зчитуваного типу під час десеріалізації.

Висновки

У даній роботі проаналізовано необхідність абстракцій серіалізації та їх види. На основі аналізу було створено власну систему серіалізації Serene, яка не залежить від особливостей певної мови програмування.

Створену систему вже можна використовувати замість таких бібліотек, як Cereal, через більшу свободу надану моделлю даних.

Подальші дослідження у цій сфері можуть бути зумовлені спрощенням користування моделі, розширення її та реалізацію системи іншими мовами програмування. Модель не підтримує базові типи даних як числа з рухомою крапкою, нульові значення та бінарні масиви, що суттєво обмежує її можливості. Також, варто було б створити ієрархію помилок, які можуть виникати під час серіалізації, адже система повертатиме різні типи помилок залежно від використаного серіалізатора.

Література

1. Діггінс К., Стівенс Д. Р., Турканіс Дж. C++ cookbook : навчальний посібник. O'Reilly Media Inc., 2005.
2. Boost.JSON - overview. Boost C++ Libraries. URL: https://www.boost.org/doc/libs/1_75_0/libs/json/doc/html/json/overview.html (date of access: 13.05.2023).
3. Cereal docs - main. GitHub Pages. URL: https://uscilab.github.io/cereal/ (date of access: 13.05.2023).
4. Chapter 64. Boost.Serialization. *The Boost C++ Libraries*. URL: https://theboostcpplibraries.com/boost.serialization (date of access: 13.05.2023).
5. Cline M. [36] Serialization and Unserialization, C++ FAQ Lite. *Dietmar Kühl*. URL: http://www.dietmar-kuehl.de/mirror/c++-faq/serialization.html (date of access: 13.05.2023).
6. Confy - rust. *Docs.rs*. URL: https://docs.rs/confy/0.3.1/confy (date of access: 13.05.2023).
7. Google. Protocol Buffer Basics: C++. *Protocol Buffers Documentation*. URL: https://protobuf.dev/getting-started/cpptutorial (date of access: 13.05.2023).
8. Overview. *Serde*. URL: https://serde.rs (date of access: 13.05.2023).
9. Responses - Rocket programming guide. *Rocket - Simple, Fast, Type-Safe Web Framework for Rust*. URL: https://rocket.rs/v0.5-rc/guide/responses (date of access: 13.05.2023).
10. Serde data model. Serde. URL: https://serde.rs/data-model.html (date of access: 13.05.2023).
11. Serde reverse dependencies. *crates.io*. URL: https://crates.io/crates/serde/reverse_dependencies (date of access: 13.05.2023).
12. Serialization overview. *Boost C++ Libraries*. URL: https://www.boost.org/doc/libs/1_82_0/libs/serialization/doc/index.html (date of access: 13.05.2023).

13. vcpkg - Open source C/C++ dependency manager from Microsoft. vcpcg. URL: https://vcpkg.io/en/ (date of access: 13.05.2023).

Додаток А. Ілюстраційний матеріал для доповіді

```
class Employee {
  friend ostream& operator<< // These have to be friends
    (ostream& out, const Employee& emp); // so they can access
  friend istream& operator>> // nonpublic members
    (istream& in, Employee& emp);
public:
  Employee() {}
  ~Employee() {}
private:
  string firstName_;
  string lastName_;
};
// Send an Employee object to an ostream...
ostream& operator<<(ostream& out, const Employee& emp) {
  out << emp.firstName_ << endl;
  out << emp.lastName_ << endl;
  return(out);
}
// Read an Employee object from a stream
istream& operator>>(istream& in, Employee& emp) {
  in >> emp.firstName_;
  in >> emp.lastName_;
  return(in);
}
```

Рисунок 28, Приклад простої серіалізації

Рисунок 29, структура простої серіалізації

Рисунок 30, структура серіалізації поверх формату

```
boost::json::value save() {
  boost::json::object obj;
  obj["first_name"] = _firstName;
  obj["second_name"] = _lastName;
  return obj;
}

static Person read(boost::json::value& v) {
  boost::json::object& obj = v.as_object();
  return Person(
    std::string(obj["first_name"].as_string().operator boost::json::string_view()),
    std::string(obj["second_name"].as_string().operator boost::json::string_view()));
}
```

Рисунок 31, приклад серіалізації поверх формату

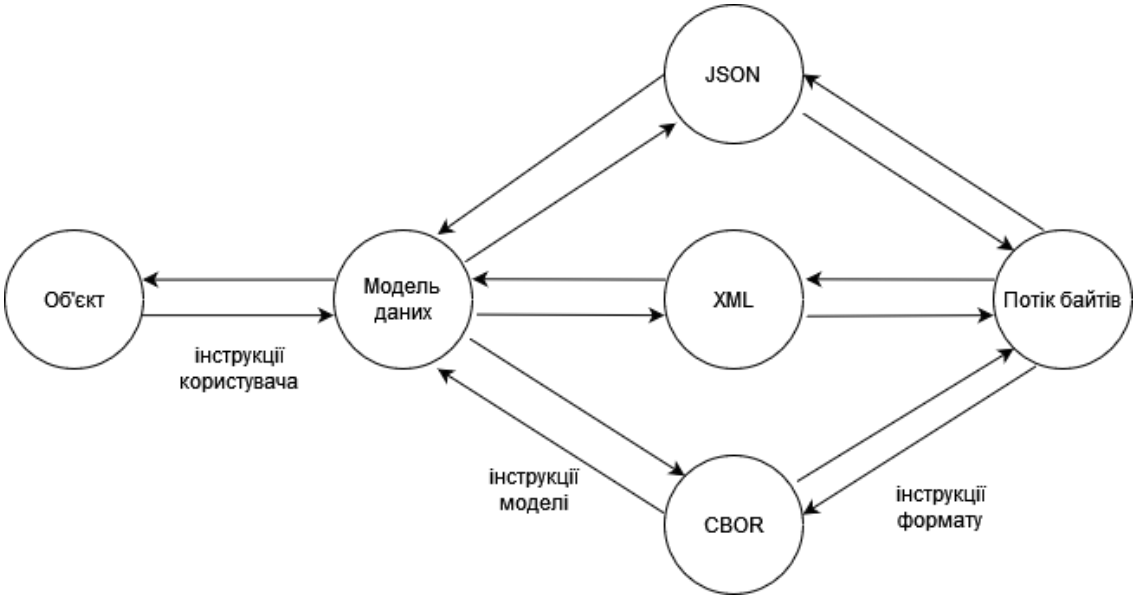

Рисунок 32, структура серіалізації поверх моделі даних

```rust
pub trait Serializer {
    type Ok;
    type Error: Error;
    type SerializeSeq: SerializeSeq<Ok = Self::Ok, Error = Self::Error>;
    type SerializeMap: SerializeMap<Ok = Self::Ok, Error = Self::Error>;
    type SerializeStruct: SerializeStruct<Ok = Self::Ok, Error = Self::Error>;

    // Required methods
    fn serialize_bool(self, v: bool) -> Result<Self::Ok, Self::Error>;
    fn serialize_i64(self, v: i64) -> Result<Self::Ok, Self::Error>;
    fn serialize_u64(self, v: u64) -> Result<Self::Ok, Self::Error>;
    fn serialize_char(self, v: char) -> Result<Self::Ok, Self::Error>;
    fn serialize_str(self, v: &str) -> Result<Self::Ok, Self::Error>;

    fn serialize_seq(self, len: Option<usize>) -> Result<Self::SerializeSeq, Self::Error>;
    fn serialize_map(self, len: Option<usize>) -> Result<Self::SerializeMap, Self::Error>;
    fn serialize_struct(self, name: &'static str,len: usize) -> Result<Self::SerializeStruct, Self::Error>;
}
```

Рисунок 33, часткове визначення моделі даних Serde

```rust
struct Person {
    first_name: String,
    last_name: String,
}

impl Serialize for Person {
    fn serialize<S>(&self, serializer: S) -> Result<S::Ok, S::Error>
    where
        S: Serializer,
    {
        let mut state = serializer.serialize_struct("Person", 2)?;
        state.serialize_field("first_name", &self.first_name);
        state.serialize_field("last_name", &self.last_name);
        state.end()
    }
}
```

Рисунок 34, приклад серіалізації поверх моделі даних

```rust
let person = Person {
    first_name: "Andrii".to_string(),
    last_name: "Zahorulko".to_string(),
};
{
    let mut msgpack_byte_buffer = Vec::<u8>::new();
    person
        .serialize(&mut rmp_serde::Serializer::new(&mut msgpack_byte_buffer))
        .unwrap();
}

{
    let mut json_byte_buffer = Vec::<u8>::new();
    person
        .serialize(&mut serde_json::Serializer::new(&mut json_byte_buffer))
        .unwrap();
}
```

Рисунок 35, приклад збереження об'єкту бібліотекою Serde

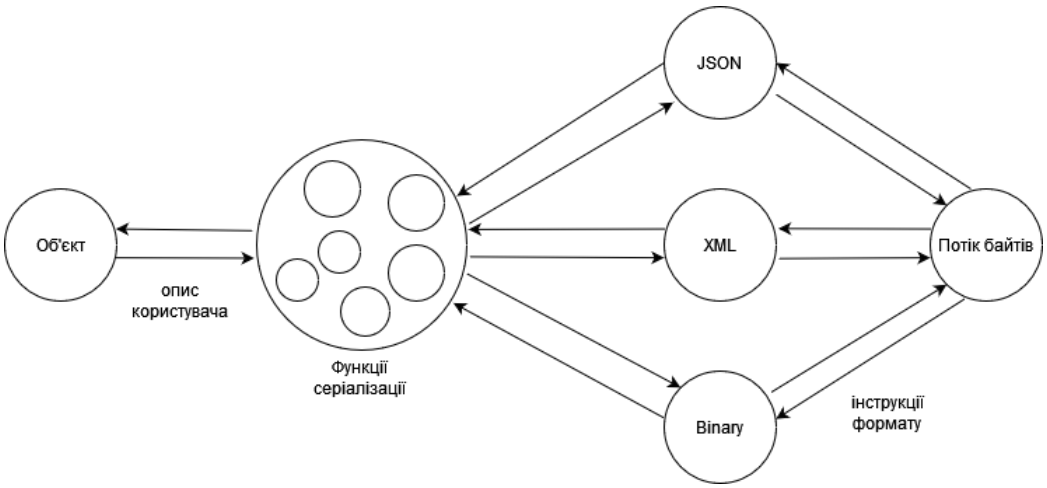

Рисунок 36, структура шаблонної серіалізації

```
template<class Archive>
void serialize(Archive& ar, unsigned int version) {
  ar& BOOST_SERIALIZATION_NVP(_firstName);
  ar& BOOST_SERIALIZATION_NVP(_lastName);
}

template<class Archive>
void serialize(Archive& ar) {
  ar(_firstName, _lastName);
}
```

Рисунок 37, приклад шаблонної серіалізації

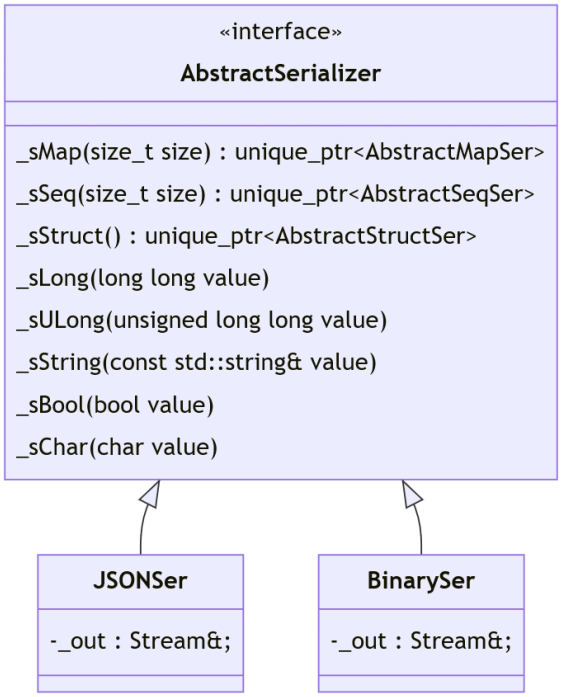

Рисунок 38, структура серіалізаторів

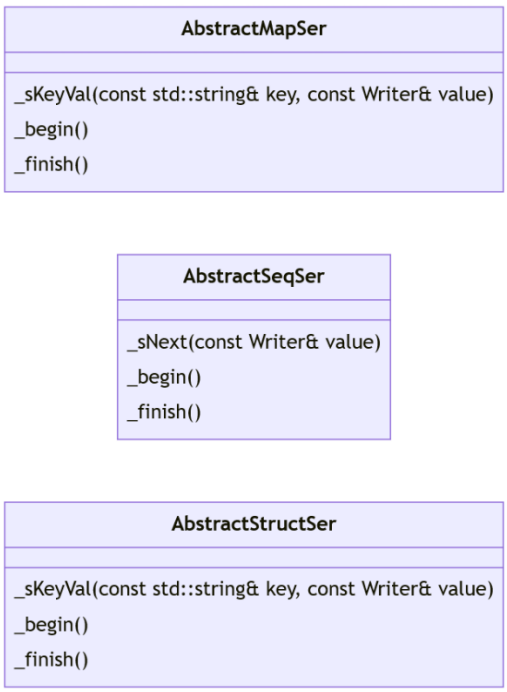

Рисунок 39, складені серіалізатори

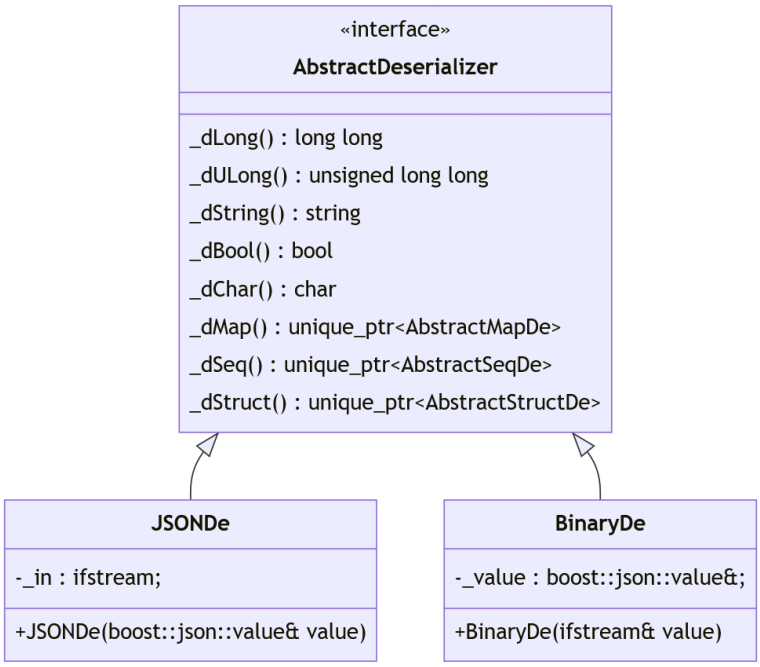

Рисунок 40, структура десеріалізаторів

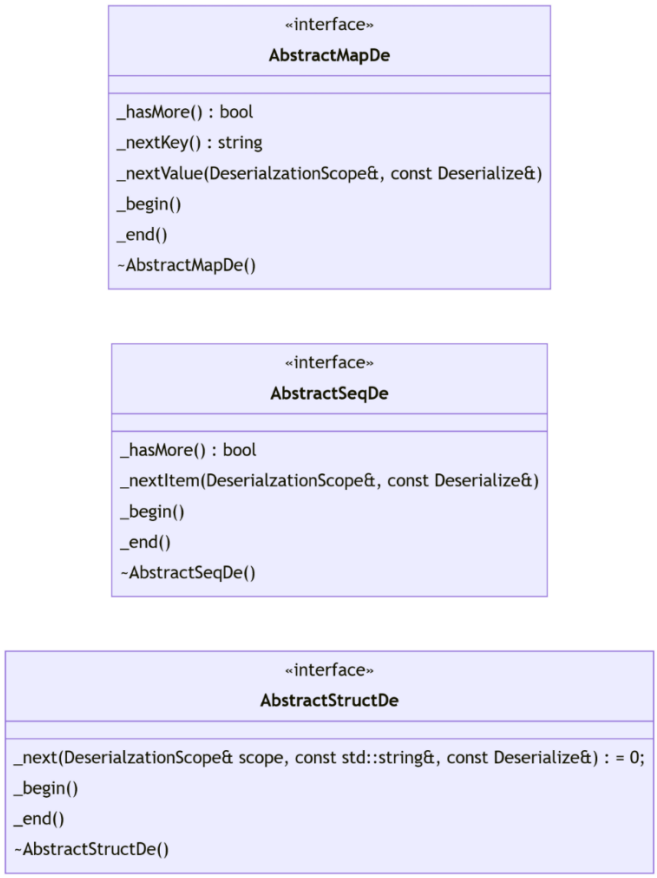

Рисунок 41, структура складених десеріалізаторів

```
using Writer = std::function<void(AbstractSerializer&)>;

inline Writer write(std::same_as<long long> auto v) {
  return [v](AbstractSerializer& s) {s.sLong(v); };
}
```

Рисунок 42, вигляд функції серіалізації

```
template<typename T>
  requires std::derived_from<T, SerializableObject>
inline Writer write(const T& v) {
  return v.serialize();
}
```

Рисунок 43, функція серіалізації об’єкту, який успадковує SerializableObject

```
template<typename T>
requires std::derived_from<T, SerializableObject>
inline Writer write(const T* v) {
  return [v](AbstractSerializer& s) {
    auto structSer = s.sStruct();
    structSer->begin();
    structSer->sKeyVal("type", write<std::string>(v->getObjectName()));
    structSer->sKeyVal("value", write<T>(*v));
    structSer->finish();
  };
}
```

Рисунок 44, серіалізація поліморфічних типів

```
Writer Color::serilaizeCurrent() const
{
  return [this](AbstractSerializer& se) {
    unsigned long long value =
      (unsigned long long) _red << 16
      | (unsigned long long) _green << 8
      | (unsigned long long) _blue;

    write(value)(se);
  };
}
```

Рисунок 45, серіалізація за допомогою спеціалізації функції

```
Writer Color::serilaizeCurrent() const
{
  return [this](AbstractSerializer& se) {
    se.sULong((unsigned long long) _red << 16
      | (unsigned long long) _green << 8
      | (unsigned long long) _blue);
  };
}
```

Рисунок 46, серіалізація за допомогою методів моделі даних

```
  return [this](AbstractSerializer& se) {
    auto structSer = se.sStruct();
    structSer->begin();
    structSer->sKeyVal("red", write(_red));
    structSer->sKeyVal("green", write(_green));
    structSer->sKeyVal("blue", write(_blue));
    structSer->finish();
  };
```

Рисунок 47, серіалізація складеного об'єкту

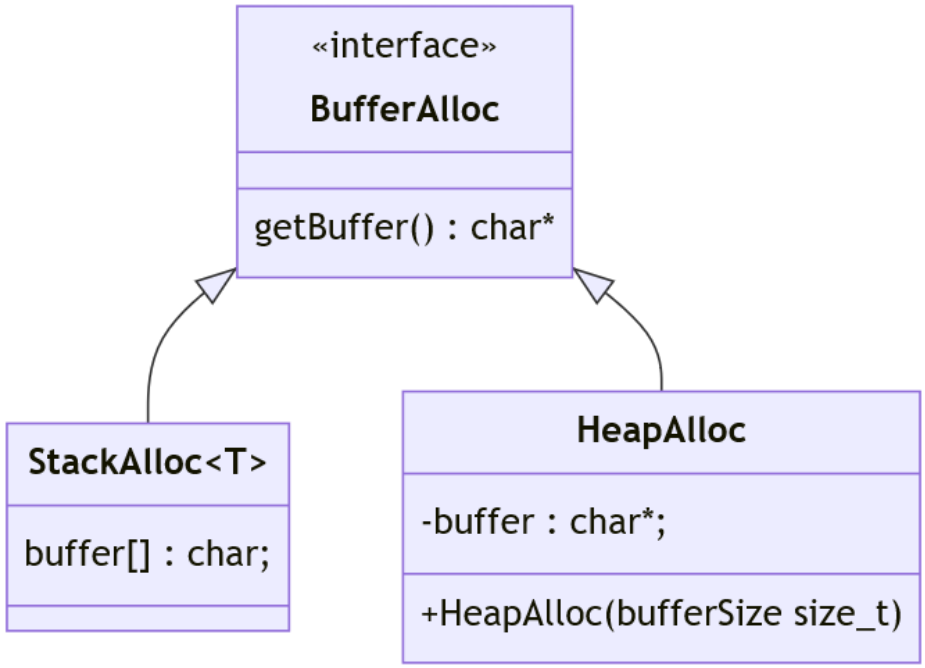

Рисунок 48, структура алокаторів пам'яті

```
using Deserialize =
    std::function<void(AbstractDeserializer&, DeserialzationScope&)>;

template<typename T>
  requires std::same_as<T, long long>
Deserialize read(BufferAlloc& v) {
  return [&v](AbstractDeserializer& de, DeserialzationScope&) {
    new (v.getBuffer()) long long(de.dLong());
  };
}
```

Рисунок 49, відтворення типу

```
Deserialize Color::read(BufferAlloc& buffer)
{
  return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
    DeserializeBuffer<char> red;
    DeserializeBuffer<char> green;
    DeserializeBuffer<char> blue;
    auto structSer = de.dStruct();
    structSer->begin();
    structSer->next(scope, "red", deserialize(red.buffer()));
    structSer->next(scope, "green", deserialize(green.buffer()));
    structSer->next(scope, "blue", deserialize(blue.buffer()));
    structSer->end();
    new (buffer.getBuffer()) Color(*red, *green, *blue);
  };
}
```

Рисунок 50, приклад відтворення кольору

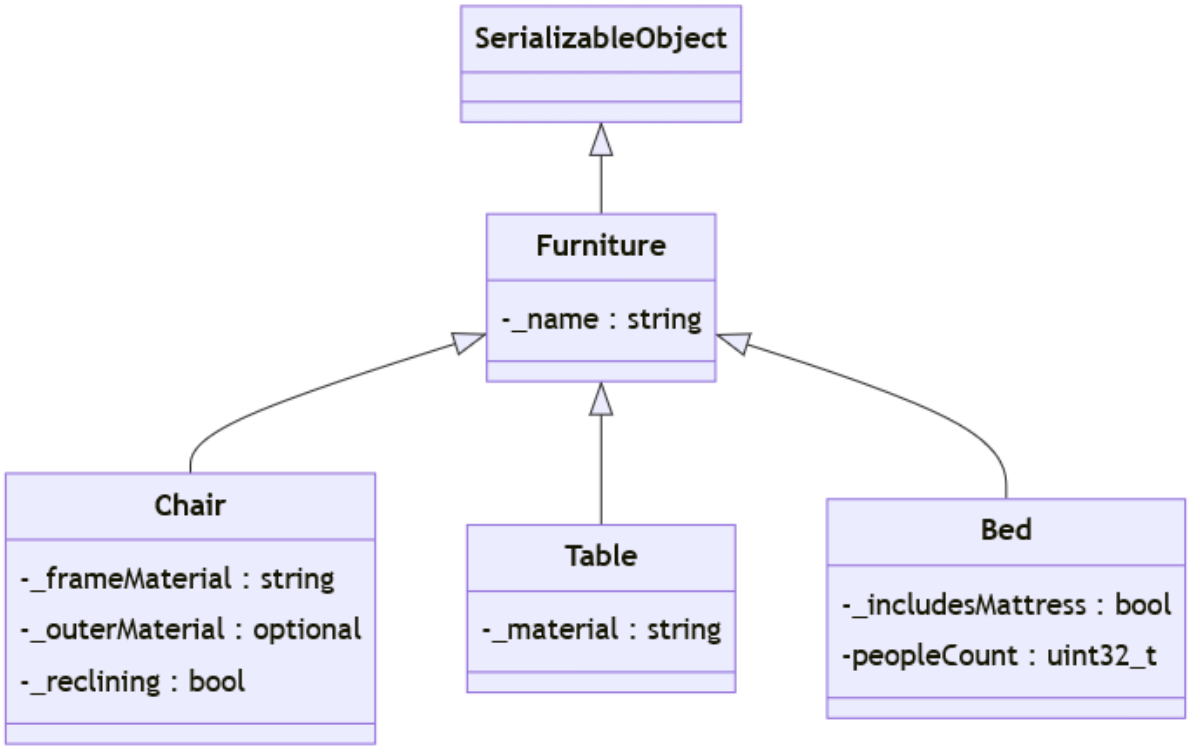

Рисунок 51, ієрархія меблів

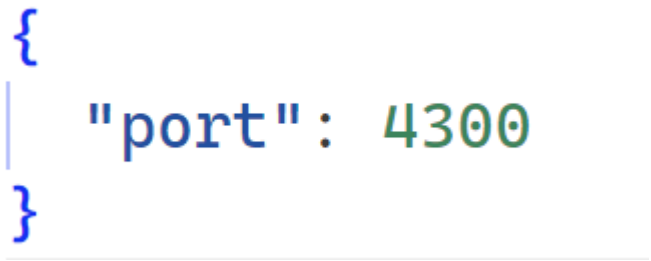

Рисунок 52, файл налаштування застосунку

HTTP запит	**Консоль/HTTP відповідь**

GET localhost:4300/furniture/Brimnes	Using value from file
	{
	"hasValue": true,
	"value": {
	"type": "Bed",
	"value": {
	"name": "Brimnes",
	"includesMattress": false,
	"peopleCount": 1
	}
	}
	}

GET localhost:4300/furniture/Noone	Couldn't read file
	{
	"hasValue": false
	}

GET localhost:4300/furniture/
Eames%20Lounge%20Chair
	Using value from file
	{
	"hasValue": true,
	"value": {
	"type": "Table",
	"value": {

	"name": "Eames Lounge Chair", "material": "wood" } } }

10 секунд після минулого запиту
GET localhost:4300/furniture/
Eames%20Lounge%20Chair
	Using value from cache: ETableTEames Lounge ChairДwood
	{
	"hasValue": true,
	"value": {
	"type": "Table",
	"value": {
	"name": "Eames Lounge Chair",
	"material": "wood"
	}
	}
	}

GET localhost:4300/furniture/
The%20Sitwell
	Using value from file
	{
	"hasValue": true,
	"value": {
	"type": "Chair",
	"value": {
	"name": "The Sitwell",
	"frameMaterial": "wood",
	"outerMaterial": {
	"hasValue": true,
	"value": "cotton"
	},

	"reclining": true } } }

PUT localhost:4300/furniture
{
"type": "Table",
"value": {
"name": "Loloungе",
"material": "metal"
}
}
	Status: 200 Ok

GET localhost:4300/furniture/Loloungе	Using value from file
	{
	"hasValue": true,
	"value": {
	"type": "Table",
	"value": {
	"name": "Loloungе",
	"material": "metal"
	}
	}
	}

```
// Cereal version
template<class Archive>
void serialize(Archive& ar)
{
  ar(cereal::base_class<Furniture>(this), _frameMaterial, _outerMaterial, _reclining);
}

// Our version
Writer Chair::serializeCurrent() const
{
  return [this](AbstractSerializer& serializer) {
    auto stSer = serializer.sStruct();
    stSer->begin();
    stSer->sKeyVal("name", write(_name));
    stSer->sKeyVal("frameMaterial", write(_frameMaterial));
    stSer->sKeyVal("outerMaterial", write(_outerMaterial));
    stSer->sKeyVal("reclining", write(_reclining));
    stSer->finish();
  };
}

Deserialize Chair::read(BufferAlloc& buffer)
{
  return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
    DeserializeBuffer<string> name;
    DeserializeBuffer<string> frameMaterial;
    DeserializeBuffer<optional<string>> outerMaterial;
    DeserializeBuffer<bool> reclining;
    auto stSer = de.dStruct();
    stSer->begin();
    stSer->next(scope, "name", deserialize(name.buffer()));
    stSer->next(scope, "frameMaterial", deserialize(frameMaterial.buffer()));
    stSer->next(scope, "outerMaterial", deserialize(outerMaterial.buffer()));
    stSer->next(scope, "reclining", deserialize(reclining.buffer()));
    stSer->end();
    new (buffer.getBuffer()) Chair(*name, *frameMaterial, *outerMaterial, *reclining);
  };
}
```

Рисунок 53, порівняння з серіалізацією Cereal

```
Deserialize Configuration::read(BufferAlloc& buffer)
{
  return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
    DeserializeBuffer<int> port;
    bool hasPort = false;
    DeserializeBuffer<bool> production;
    bool hasProduction = false;
    auto stSer = de.dMap();
    stSer->begin();
    while (stSer->hasMore())
    {
      auto next = stSer->nextKey();
      if (next == "port") {
        stSer->nextValue(scope, deserialize(port.buffer()));
        hasPort = true;
      }
      else if (next == "production") {
        stSer->nextValue(scope, deserialize(production.buffer()));
        hasProduction = true;
      }
    }
    int rPort = hasPort ? *port : 8080;
    bool rProduction = hasProduction ? *production : false;
    stSer->end();
    new (buffer.getBuffer()) Configuration(rPort, rProduction);
  };
}
```

Рисунок 54, десеріалізація налаштування застосунку

Додаток Б, Програмні коди реалізації системи на C++ 20

access.cpp

```
#include "access.h"

#include <fstream>

#include <format>

#include "BinaryDe.h"

#include "BinarySer.h"

#include "DeserializeExt.h"

#include <sstream>

#include <boost/json.hpp>

#include "JSONDe.h"




std::mutex cacheMutex;

std::map<std::string, std::string> cashedFurniture;


Configuration loadConfig()

{

	DeserializeBuffer<Configuration> configurationBuffer;

	std::ifstream configReader("./furnitureOther/config.json");

	boost::json::value v(boost::json::parse(configReader));
```

```
        JSONDe de(v);
        DeserialzationScope scope;
        deserialize(configurationBuffer.buffer())(de, scope);

        return *configurationBuffer;
}

void updateCache()
{
        cacheMutex.lock();
        cashedFurniture.clear();
        cacheMutex.unlock();
}

std::shared_ptr<Furniture> loadFromCache(const std::string& name) {
        std::cout << "Using value from cache: " << cashedFurniture.at(name) << std::endl;
        DeserializeBuffer<Furniture*> furniture;
        std::istringstream stream(cashedFurniture.at(name));
        BinaryDe de(stream);
        DeserialzationScope scope;
        deserialize(furniture.buffer())(de, scope);
        return std::shared_ptr<Furniture>(*furniture);
```

```
}

void addToCache(const Furniture& furniture) {
	cacheMutex.lock();
	{
		std::ostringstream stream;
		BinarySer ser(stream);
		write(&furniture)(ser);
		//auto str = stream.str();
		//std::ofstream test("furnitureOther/test.txt");
		//test << str;
		cashedFurniture.emplace(furniture.name(), stream.str());
	}
	cacheMutex.unlock();

}

optional<std::shared_ptr<Furniture>> getFurniture(const std::string& name)
{
	cacheMutex.lock();

	if (cashedFurniture.contains(name)) {
```

```
        auto v = loadFromCache(name);
        cacheMutex.unlock();
        return v;
}
else {
        cacheMutex.unlock();
}



string path = std::format("furnitures/{}", name);
std::ifstream file(path);

if (!file.is_open()) {
        std::cout << "Couldn't read file" << std::endl;
        return optional<std::shared_ptr<Furniture>>();
}

std::cout << "Using value from file" << std::endl;
BinaryDe de(file);
DeserialzationScope scope;
DeserializeBuffer<Furniture*> furniture;
```

```
        deserialize(furniture.buffer())(de, scope);

        std::shared_ptr<Furniture> result(*furniture);

        addToCache(*result);

        return result;
}

void insertFurniture(const Furniture& f)
{
        cacheMutex.lock();
        cashedFurniture.erase(f.name());
        cacheMutex.unlock();

        string path = std::format("furnitures/{}", f.name());
        std::ofstream file(path);
        file.clear();
        BinarySer ser(file);
        write(&f)(ser);
}
```

```
void loadTestData()
{

	DeserializeBuffer<std::vector<std::unique_ptr<Furniture>>> furnitures;
	std::ifstream stream("./furnitureOther/mockData.json");
	boost::json::value v(boost::json::parse(stream));
	JSONDe de(v);
	DeserialzationScope scope;
	deserialize(furnitures.buffer())(de, scope);

	auto items = *furnitures;
	for (auto& v : items) {
		insertFurniture(*v);
	}
}



Writer Configuration::serializeCurrent() const
{
	return [this](AbstractSerializer& serializer) {
		auto stSer = serializer.sMap(2);
```

```
            stSer->begin();
            stSer->sKeyVal("port", write(_port));
            stSer->sKeyVal("production", write(_production));
            stSer->finish();
        };
}

Deserialize Configuration::read(BufferAlloc& buffer)
{
        return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
            DeserializeBuffer<int> port;
            bool hasPort = false;
            DeserializeBuffer<bool> production;
            bool hasProduction = false;
            auto stSer = de.dMap();
            stSer->begin();
            while (stSer->hasMore())
            {
                auto next = stSer->nextKey();
                if (next == "port") {
                    stSer->nextValue(scope, deserialize(port.buffer()));
                    hasPort = true;
```

```
                }
                else if (next == "production") {
                        stSer->nextValue(scope, deserialize(production.buffer()));
                        hasProduction = true;
                }
        }
        int rPort = hasPort ? *port : 8080;
        bool rProduction = hasProduction ? *production : false;
        stSer->end();
        new (buffer.getBuffer()) Configuration(rPort, rProduction);
    };
}
```

access.h

```
#pragma once
#include <memory>
#include "furniture.h"


class Configuration : public SerializableObject {
    RegisterSerializeClass(Configuration);
private:
```

```
        int _port;
        bool _production;
        virtual Writer _serialize() const { return serializeCurrent(); }
public:
        Configuration(int port, bool production) : _port(port), _production(production) {}

        Writer serializeCurrent() const;

        static Deserialize read(BufferAlloc&);
        int port() const { return _port; }
        bool inProduction() const { return _production; }


};

Configuration loadConfig();

void updateCache();

optional<std::shared_ptr<Furniture>> getFurniture(const std::string& name);

void insertFurniture(const Furniture& f);
```

```
void loadTestData();
```

BinaryDe.cpp

```
#include "BinaryDe.h"

long long BinaryDe::_dLong()
{
	//long long v = 0;

	//for (size_t i = 0; i < 8; i++)
	//{
	//	char next;
	//	_in >> next;
	//	v = (v + next) << 8;
	//}
	return (long long)_dULong();
}

unsigned long long BinaryDe::_dULong()
```

```
{
	unsigned long long v = 0;
	char next;
	_in >> next;
	int currentPos = 0;
	while ((next & 0x80) == 0) {
		v |= ((unsigned long long)(next) << currentPos);
		currentPos += 7;
		_in >> next;
	}
	v |= ((unsigned long long)(next & 0x7F) << currentPos);
	return v;
}

std::string BinaryDe::_dString()
{

	unsigned long long size = dULong();
	char* c = new char[size];
	_in.read(c, size);
	std::string s(c, size);
	delete[] c;
```

```
    return s;
}

bool BinaryDe::_dBool()
{
    bool b;
    _in >> b;
    return b;
}

char BinaryDe::_dChar()
{
    char c;
    _in >> c;
    return c;
}

std::unique_ptr<AbstractMapDe> BinaryDe::_dMap()
{
    return std::unique_ptr<AbstractMapDe>(new BinaryMapDe(_in, *this));
}
```

```
std::unique_ptr<AbstractSeqDe> BinaryDe::_dSeq()
{
	return std::unique_ptr<AbstractSeqDe>(new BinarySeqDe(_in, *this));
}

std::unique_ptr<AbstractStructDe> BinaryDe::_dStruct()
{
	return std::unique_ptr<AbstractStructDe>(new BinaryStructDe(*this));
}

bool BinaryMapDe::_hasMore()
{
	return _size > 0;
}

std::string BinaryMapDe::_nextKey()
{
	return _de.dString();
}

void BinaryMapDe::_nextValue(DeserialzationScope& scope, const Deserialize&
deserialize)
{
```

```
        deserialize(_de, scope);
        _size--;
}

void BinaryMapDe::_begin()
{
        _size = _de.dULong();
}

void BinaryMapDe::_end()
{

}

bool BinarySeqDe::_hasMore()
{
        return _size > 0;
}

void BinarySeqDe::_nextItem(DeserialzationScope& scope, const Deserialize&
deserialize)
{
        deserialize(_de, scope);
```

```
        --_size;
}

void BinarySeqDe::_begin()
{
        _size = _de.dULong();
}

void BinarySeqDe::_end()
{
}

void BinaryStructDe::_next(DeserialzationScope& scope, const std::string&, const
Deserialize& deserialize)
{
        deserialize(_de, scope);
}

void BinaryStructDe::_begin()
{
}

void BinaryStructDe::_end()
```

```
{

}
```

BinaryDe.h

```
#pragma once
#include "Deserializer.h"
#include <fstream>
#include "types.h"

class BinaryDe final : public AbstractDeserializer {
private:
	InStream& _in;
	virtual long long _dLong() override;
	virtual unsigned long long _dULong() override;
	virtual std::string _dString() override;
	virtual bool _dBool() override;
	virtual char _dChar() override;
	virtual std::unique_ptr<AbstractMapDe> _dMap() override;
	virtual std::unique_ptr<AbstractSeqDe> _dSeq() override;
	virtual std::unique_ptr<AbstractStructDe> _dStruct() override;

public:
```

```
    BinaryDe(InStream& in) : _in(in) {}
};

class BinaryMapDe final : public AbstractMapDe {
private:
    InStream& _in;
    BinaryDe& _de;
    size_t _size = 0;
    virtual bool _hasMore() override;
    virtual std::string _nextKey() override;
    virtual void _nextValue(DeserialzationScope&, const Deserialize&) override;
    virtual void _begin() override;
    virtual void _end() override;
public:
    BinaryMapDe(InStream& in, BinaryDe& de) : _in(in), _de(de) {}
};

class BinarySeqDe final : public AbstractSeqDe {
private:
    InStream& _in;
    BinaryDe& _de;
```

```
    size_t _size = 0;
    virtual bool _hasMore() override;
    virtual void _nextItem(DeserialzationScope&, const Deserialize&) override;
    virtual void _begin() override;
    virtual void _end() override;
public:
    BinarySeqDe(InStream& in, BinaryDe& de) : _in(in), _de(de) {}
};

class BinaryStructDe final : public AbstractStructDe {
private:
    BinaryDe& _de;
    virtual void _next(DeserialzationScope& scope, const std::string&, const
Deserialize&) override;
    virtual void _begin() override;
    virtual void _end() override;
public:
    BinaryStructDe(BinaryDe& de) : _de(de) {}

};
```

BinarySer.cpp

```
#include "BinarySer.h"
```

```
#include <bitset>



std::unique_ptr<AbstractMapSer> BinarySer::_sMap(size_t size)
{
	return std::unique_ptr<AbstractMapSer>(new BinaryMapSer(size, *this, _out));
}

std::unique_ptr<AbstractSeqSer> BinarySer::_sSeq(size_t size)
{
	return std::unique_ptr<AbstractSeqSer>(new BinarySeqSer(size, *this, _out));
}

std::unique_ptr<AbstractStructSer> BinarySer::_sStruct()
{
	return std::unique_ptr<AbstractStructSer>(new BinaryStructSer(*this, _out));
}



void BinarySer::_sLong(long long v)
{
	_sULong((unsigned long long) v);
```

```
	//_out
	//	<< char(v >> 56) << char(v >> 48) << char(v >> 40) << char(v >> 32) << char(v >> 24) << char(v >> 16) << char(v >> 8) << char(v);
}

void BinarySer::_sULong(unsigned long long v)
{
	long long vCur = v;
	while (vCur > 0x7F) {
		char next = vCur & 0x7F;
		_out << next;
		vCur >>= 7;
	}
	_out << char(0x80 | vCur);
	_out.flush();
	//_out
	//	<< char(v >> 56) << char(v >> 48) << char(v >> 40) << char(v >> 32) << char(v >> 24) << char(v >> 16) << char(v >> 8) << char(v);
}

void BinarySer::_sString(const std::string& v)
```

```
{
	sULong(unsigned long long(v.size()));
	_out << v;
}

void BinarySer::_sBool(bool v)
{
	_out << v;
}

void BinarySer::_sChar(char v)
{
	_out << v;
}


BinaryMapSer::BinaryMapSer(size_t size, BinarySer& ser, OutStream& out) : _size(size), _ser(ser), _out(out) {}


void BinaryMapSer::_sKeyVal(const std::string& k, const Writer& f)
{
	if (_size-- == 0)
```

```
            throw ExceededSize();
        _ser.sString(k);
        f(_ser);
}

void BinaryMapSer::_begin() {
        _ser.sULong(_size);
}

void BinaryMapSer::_finish()
{
        if (_size != 0)
                throw _size;
}

BinarySeqSer::BinarySeqSer(size_t size, BinarySer& ser, OutStream& out)
        : _size(size), _ser(ser), _out(out) {}

BinarySeqSer::~BinarySeqSer()
{
        if (_size != 0)
                throw _size;
```

```
}

void BinarySeqSer::_begin()
{
        _ser.sULong((unsigned long)_size);
}

void BinarySeqSer::_finish()
{
        if (_size != 0)
                throw _size;
}

void BinarySeqSer::_sNext(const Writer& t)
{
        if (_size-- == 0)
                throw ExceededSize();
        t(_ser);
}

void BinaryStructSer::_finish() {}
```

```
void BinaryStructSer::_begin() {}

inline void BinaryStructSer::_sKeyVal(const std::string&, const Writer& t)
{
	t(_ser);
}

BinaryStructSer::BinaryStructSer(BinarySer& ser, OutStream& out) : _ser(ser), _out(out)
{}
```

BinarySer.h

```
#pragma once
#include "Serializer.h"
#include <iostream>
#include <fstream>
#include "types.h"

class BinarySer;
class BinaryMapSer;
class BinarySeqSer;
class BinaryStructSer;
```

```
class BinarySer final : public AbstractSerializer {
private:
	OutStream& _out;

	virtual std::unique_ptr<AbstractMapSer> _sMap(size_t);
	virtual std::unique_ptr<AbstractSeqSer> _sSeq(size_t);
	virtual std::unique_ptr<AbstractStructSer> _sStruct();

	virtual void _sLong(long long);
	virtual void _sULong(unsigned long long);
	//void sWString(std::wstring);
	virtual void _sString(const std::string&);
	virtual void _sBool(bool);
	virtual void _sChar(char);
public:
	BinarySer(OutStream& out) : _out(out) {};
};

class BinaryMapSer final : public AbstractMapSer {
private:
	size_t _size;
```

```
    BinarySer& _ser;
    OutStream& _out;

    virtual void _sKeyVal(const std::string&, const Writer&);
    virtual void _begin();
    virtual void _finish();
public:
    BinaryMapSer(size_t size, BinarySer& ser, OutStream& out);

};

class BinarySeqSer final : public AbstractSeqSer {
private:
    size_t _size;
    OutStream& _out;
    BinarySer& _ser;

    virtual void _sNext(const Writer& v);
    virtual void _begin();
    virtual void _finish();
public:
    BinarySeqSer(size_t size, BinarySer& ser, OutStream& out);
```

```
        ~BinarySeqSer();

};

class BinaryStructSer final : public AbstractStructSer {
private:
        OutStream& _out;
        BinarySer& _ser;
        virtual void _sKeyVal(const std::string&, const Writer&);
        virtual void _begin();
        virtual void _finish();
public:
        BinaryStructSer(BinarySer& ser, OutStream& out);

};
```

BufferAlloc.h

```
#pragma once
#include <concepts>
```

```
class BufferAlloc {
private:
	virtual char* _getBuffer() = 0;
public:
	virtual ~BufferAlloc() {};
	inline char* getBuffer() { return _getBuffer(); }
};

template<typename T>
class StackAllocBuffer : public BufferAlloc {
private:
	char _buffer[sizeof(T)];
	bool _wasInitialized = false;

	virtual char* _getBuffer() {
		return _buffer;
	}

public:
	virtual ~StackAllocBuffer() {
```

```
        if (_wasInitialized)
            reinterpret_cast<T*>(_buffer)->~T();
    };

    inline T* getValue() {
        return reinterpret_cast<T*>(getBuffer());
    }

    T&& operator*() {
        return std::move(*reinterpret_cast<T*>(getBuffer()));
    }

    void setWasInitialized(bool wasInitialized) { _wasInitialized = wasInitialized; }
};

class HeapAllocBuffer : public BufferAlloc {
private:
    char* _buffer;

    virtual char* _getBuffer() {
        return _buffer;
    }
```

```
public:
        HeapAllocBuffer(size_t bufferSize) : _buffer(new char[bufferSize]) {}
};
```

ClassDiagram.cd

```
<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
  <Class Name="BinarySer" Collapsed="true">
    <Position X="5.5" Y="4.75" Width="1.5" />
    <TypeIdentifier>
<HashCode>BwAAAAAAAAAAAABQAEABAAAAAAgAAAIAAAAAAAAAAIAA=</HashCode>
      <FileName>BinarySer.h</FileName>
    </TypeIdentifier>
  </Class>
  <Class Name="BinaryMapSer" Collapsed="true">
    <Position X="5.5" Y="1.75" Width="1.5" />
    <TypeIdentifier>
<HashCode>AQAABAAAAAAABAAAAAAAACAQAAAAAAAAAAAAACAIAAAA=</HashCode>
      <FileName>BinarySer.h</FileName>
```

</TypeIdentifier>

</Class>

<Class Name="BinarySeqSer" Collapsed="true">

<Position X="0.5" Y="4.75" Width="1.5" />

<TypeIdentifier>

<HashCode>AQAAAAAAAAAAJAAAAAAAAAAQAAAAAAAAAAIAGAIAAAA=</HashCode>

<FileName>BinarySer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="BinaryStructSer" Collapsed="true">

<Position X="2.75" Y="1.75" Width="1.5" />

<TypeIdentifier>

<HashCode>AQAAAAAAAAAABAAAAAAAACAQAAAAAAAAAIAACAAAAAA=</HashCode>

<FileName>BinarySer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="AbstractDeVisitor<T>" Collapsed="true">

<Position X="12" Y="0.5" Width="1.5" />

<TypeIdentifier>

<HashCode>AAAAIAAAAAAAAAQAAAAAAAAACAAAACQABAAAAAAACA=
</HashCode>

<FileName>Deserializer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="AbstractSeqDe" Collapsed="true">

<Position X="10.25" Y="1.5" Width="1.5" />

<TypeIdentifier>

<HashCode>AAAAAAAAABAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAA
=</HashCode>

<FileName>Deserializer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="AbstractMapDe<MapValue>" Collapsed="true">

<Position X="3.5" Y="6.5" Width="1.5" />

<TypeIdentifier>

<HashCode>AAAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAA
=</HashCode>

<FileName>Deserializer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="AbstractDeAccess" Collapsed="true">

<Position X="10.25" Y="0.5" Width="1.5" />

<TypeIdentifier>

<HashCode>IAAAABAAAAAAAAAAAAACAAAAAAAAAAAAIAAAAAAAAA=
</HashCode>

<FileName>Deserializer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="AbstractDeserializer" Collapsed="true">

<Position X="0.5" Y="6.5" Width="1.5" />

<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAAAAAA
=</HashCode>

<FileName>Deserializer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="JSONMapDe" Collapsed="true">

<Position X="3.5" Y="7.75" Width="1.5" />

<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
=</HashCode>

<FileName>JSONDe.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="JSONDe" Collapsed="true">

<Position X="0.5" Y="7.75" Width="1.5" />

<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAABAAAAAAAAAAAAAAAQAAAAA=</HashCode>

<FileName>JSONDe.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="JSONSer" Collapsed="true">

<Position X="7.75" Y="4.75" Width="1.5" />

<TypeIdentifier>

<HashCode>BwAAAAQAAAAAABAAEABAAAAAAgAAAIAAAAAAAAAAIAA=</HashCode>

<FileName>JSONSer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="JSONMapSer" Collapsed="true">

<Position X="7.75" Y="1.75" Width="1.5" />

<TypeIdentifier>

<HashCode>AQAAAAAAAAAABCAgAAAAACAAAAAAAAAAAAAAACAAIAIA=<
/HashCode>

<FileName>JSONSer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="JSONSeqSer" Collapsed="true">

<Position X="2.75" Y="4.75" Width="1.5" />

<TypeIdentifier>

<HashCode>AQAAAAAAAAAEBAAAAAAAAAQAAAAAAAAAAIACAAIAQA=
</HashCode>

<FileName>JSONSer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="JSONStructSer" Collapsed="true">

<Position X="0.5" Y="1.75" Width="1.5" />

<TypeIdentifier>

<HashCode>AQAAAAAAAAAABAAQAAAAACAQAAAAgAAAAAAAACAAIAAA=
</HashCode>

<FileName>JSONSer.h</FileName>

</TypeIdentifier>

```
  </Class>
  <Class Name="Node&lt;T&gt;" Collapsed="true">
    <Position X="12" Y="1.5" Width="1.5" />
    <TypeIdentifier>

<HashCode>ABAAAAAAAACAAAAAgAAAAAAAAAAAAAAgAAAAAAEAAAA=
</HashCode>
      <FileName>Node.h</FileName>
    </TypeIdentifier>
  </Class>
  <Class Name="AbstractSerializer&lt;MapSer, SeqSer, StructSer&gt;" Collapsed="true">
    <Position X="6.75" Y="3.5" Width="1.5" />
    <TypeIdentifier>

<HashCode>BgCAAAAAAAAAABAAEABAAAAAgAAAIAAAAAAAAAIAA=</
HashCode>
      <FileName>Serializer.h</FileName>
    </TypeIdentifier>
  </Class>
  <Class Name="AbstractMapSer&lt;S&gt;" Collapsed="true">
    <Position X="6.75" Y="0.5" Width="1.5" />
    <TypeIdentifier>
```

<HashCode>AAAAAAAAAAAAAAAAAAAACAQQAAAAAAAAAAACAAAAAA
=</HashCode>

<FileName>Serializer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="AbstractSeqSer<S>" Collapsed="true">

<Position X="1.75" Y="3.5" Width="1.5" />

<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAQAQAAAAAAAAAAIACAAAAAA
=</HashCode>

<FileName>Serializer.h</FileName>

</TypeIdentifier>

</Class>

<Class Name="AbstractStructSer<S>" Collapsed="true">

<Position X="1.75" Y="0.5" Width="1.5" />

<TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAACAQAAAAAAAAAAgAACAAAAAA
=</HashCode>

<FileName>Serializer.h</FileName>

</TypeIdentifier>

</Class>

```
  <Struct Name="Person" Collapsed="true">
    <Position X="12" Y="2.75" Width="1.5" />
    <TypeIdentifier>

<HashCode>AAAAMAAAAAAAAAgAAAAAAAAAAAAAAAgAAAAAAAAAAAAA
=</HashCode>
      <FileName>Example.h</FileName>
    </TypeIdentifier>
  </Struct>
  <Struct Name="ExceededSize" Collapsed="true">
    <Position X="10.25" Y="2.75" Width="1.5" />
    <TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
=</HashCode>
      <FileName>Serializer.h</FileName>
    </TypeIdentifier>
  </Struct>
  <Font Name="Segoe UI" Size="9" />
</ClassDiagram>
```

ClassDiagram1.cd

```
<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram />
```

cpp.hint

```
// Hint files help the Visual Studio IDE interpret Visual C++ identifiers
// such as names of functions and macros.
// For more information see https://go.microsoft.com/fwlink/?linkid=865984
#define __cpp_threadsafe_static_init
#define RegisterSerializeClass(type) static char type##Name[] = #type; StaticInit<registerClass<type##Name, type>>;
```

DeserializeExt.h

```
#pragma once
#include "Deserializer.h"
#include <list>

template<typename T>
concept PushDeserialize = requires(T t, const typename T::value_type & g) {
	Deserializable<typename T::value_type>;
	{t.push_back(g) };
};

template<PushDeserialize T>
	requires (!std::same_as<std::string, T>) && (!std::derived_from<T, SerializableObject>)
Deserialize read(BufferAlloc& buffer) {
```

```
        return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
                auto seq = new (buffer.getBuffer()) T();
                auto seqDe = de.dSeq();
                seqDe->begin();
                while (seqDe->hasMore()) {
                        DeserializeBuffer<typename T::value_type> buffer;
                        seqDe->nextItem(scope, deserialize(buffer.buffer()));
                        seq->push_back(*buffer);

                }
                seqDe->end();
        };
}


template<typename V>
        requires std::same_as<std::optional<typename V::value_type>, V>
Deserialize read(BufferAlloc& buffer) {
        return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
                DeserializeBuffer<bool> hasValue;
                auto st = de.dStruct();
                st->next(scope, "hasValue", deserialize(hasValue.buffer()));
```

```
            if (*hasValue) {
                DeserializeBuffer<typename V::value_type> value;
                st->next(scope, "value", deserialize(value.buffer()));
                new (buffer.getBuffer()) V(*value);
            }
            else {
                new (buffer.getBuffer()) V();
            }
        };
}


template<typename T>
    requires std::same_as<T, std::unique_ptr<typename T::element_type>>
Deserialize read(BufferAlloc& v) {
    return [&v](AbstractDeserializer& de, DeserialzationScope& scope) {
        DeserializeBuffer<typename T::element_type*> value;
        deserialize(value.buffer())(de, scope);
        new (v.getBuffer()) T(*value);
    };
}
```

```
template<typename T>
	requires std::same_as<T, std::shared_ptr<typename T::element_type>>
Deserialize read(BufferAlloc& v) {
	return [&v](AbstractDeserializer& de, DeserialzationScope& scope) {
		DeserializeBuffer<typename T::element_type*> value;
		deserialize(value.buffer())(de, scope);
		new (v.getBuffer()) T(*value);
	};
}
```

Deserializer.cpp

```
#include "Deserializer.h"

 void* DeserialzationScope::read(const std::string& k) const
{

	for (size_t i = scope.size(); i >= 1; i--) {
		if (scope[i - 1].contains(k)) {
			return scope[i - 1].at(k);
		}
	}
```

```
	return nullptr;
}
```

Deserializer.h

```
#pragma once
#include <concepts>
#include <string>
#include <vector>
#include <iostream>
#include <iterator>
#include <map>
#include <functional>
#include <optional>
#include "Serializer.h"
#include "BufferAlloc.h"
#include "StaticInit.h"

class AbstractDeserializer;

class DeserialzationScope {
private:
	std::vector<std::map<std::string, void*>> scope;
```

```
public:
        inline void push() { scope.push_back({}); };
        inline void pop() { scope.pop_back(); };
        void* read(const std::string& k) const;
        inline void write(const std::string& k, void* v) { scope.back().insert(std::pair(k, v));
};


};

using Deserialize = std::function<void(AbstractDeserializer&, DeserialzationScope&)>;



class AbstractMapDe {
private:
        virtual bool _hasMore() = 0;
        virtual std::string _nextKey() = 0;
        virtual void _nextValue(DeserialzationScope&, const Deserialize&) = 0;
        virtual void _begin() = 0;
        virtual void _end() = 0;
public:
        virtual ~AbstractMapDe() {};
        bool hasMore() { return _hasMore(); }
        std::string nextKey() { return _nextKey(); }
```

```
        void nextValue(DeserialzationScope& scope, const Deserialize& de) {
_nextValue(scope, de); }
        void begin() { _begin(); }
        void end() { _end(); }
};

class AbstractSeqDe {
private:
        virtual bool _hasMore() = 0;
        virtual void _nextItem(DeserialzationScope&, const Deserialize&) = 0;
        virtual void _begin() = 0;
        virtual void _end() = 0;
public:
        virtual ~AbstractSeqDe() {};
        bool hasMore() { return _hasMore(); }
        void nextItem(DeserialzationScope& scope, const Deserialize& de) {
_nextItem(scope, de); }
        void begin() { _begin(); }
        void end() { _end(); }
};

class AbstractStructDe {
private:
```

```
    virtual void _next(DeserialzationScope& scope, const std::string&, const
Deserialize&) = 0;

    virtual void _begin() = 0;

    virtual void _end() = 0;

public:

    void next(DeserialzationScope& scope, const std::string& key, const Deserialize&
de) { _next(scope, key, de); }

    void begin() { _begin(); }

    void end() { _end(); }

};



class AbstractDeserializer {

private:

    virtual long long _dLong() = 0;

    virtual unsigned long long _dULong() = 0;

    virtual std::string _dString() = 0;

    virtual bool _dBool() = 0;

    virtual char _dChar() = 0;

    virtual std::unique_ptr<AbstractMapDe> _dMap() = 0;

    virtual std::unique_ptr<AbstractSeqDe> _dSeq() = 0;

    virtual std::unique_ptr<AbstractStructDe> _dStruct() = 0;
```

```
public:
    virtual ~AbstractDeserializer() {};
    long long dLong() { return _dLong(); }
    unsigned long long dULong() { return _dULong(); }
    std::string dString() { return _dString(); }
    bool dBool() { return _dBool(); }
    char dChar() { return _dChar(); }
    std::unique_ptr<AbstractMapDe> dMap() { return _dMap(); }
    std::unique_ptr<AbstractSeqDe> dSeq() { return _dSeq(); }
    std::unique_ptr<AbstractStructDe> dStruct() { return _dStruct(); }
};



class DeserializationException : public std::exception {
private:
    std::string message;

public:
    explicit DeserializationException(const std::string& msg) : message(msg) {}

    const char* what() const noexcept override {
```

```
            return message.c_str();
        }

};

template<typename T>
concept Deserializable = requires(T t, BufferAlloc & buffer) {
        {T::read(buffer)};
};

using PointerDeserialize = Deserialize(*)(HeapAllocBuffer& heap);

class DeserializationDescription {
private:
        PointerDeserialize _deserialize;
        unsigned long long _objectSize;
public:
        DeserializationDescription(unsigned long long objectSize, PointerDeserialize&
deserialize) : _objectSize(objectSize), _deserialize(deserialize) {}
        const PointerDeserialize& getDeserializer() const { return _deserialize; }
        const size_t getObjectSize() const { return _objectSize; }
};
```

```
class TypeAccumulator {
private:
	std::map<std::string, DeserializationDescription> _map;
public:
	static TypeAccumulator& getInstance() {
		static TypeAccumulator instance;
		return instance;
	}

	inline void add(const std::string& s, const DeserializationDescription& d) { _map.insert(std::make_pair(s, d)); };
	inline const DeserializationDescription& find(const std::string& s) const { return _map.at(s); };
};

template<typename T>
Deserialize read(BufferAlloc&);

template<typename T>
	requires std::same_as<T, long long>
Deserialize read(BufferAlloc& v) {
	return [&v](AbstractDeserializer& de, DeserialzationScope&) {
		new (v.getBuffer()) long long(de.dLong());
```

```
        };
}

template<typename T>
        requires std::same_as<T, int>
Deserialize read(BufferAlloc& v) {
        return [&v](AbstractDeserializer& de, DeserialzationScope&) {
                new (v.getBuffer()) int(de.dLong());
        };
}

template<typename T>
        requires std::same_as<T, unsigned long long>
Deserialize read(BufferAlloc& v) {
        return [&v](AbstractDeserializer& de, DeserialzationScope&) {
                new (v.getBuffer()) unsigned long long(de.dLong());
        };
}

template<typename T>
        requires std::same_as<T, unsigned int>
Deserialize read(BufferAlloc& v) {
```

```
        return [&v](AbstractDeserializer& de, DeserialzationScope&) {
                new (v.getBuffer()) unsigned int(de.dULong());
        };
}



template<typename T>
        requires std::same_as<T, std::string>
Deserialize read(BufferAlloc& v) {
        return [&v](AbstractDeserializer& de, DeserialzationScope&) {
                new (v.getBuffer()) std::string(de.dString());
        };
}

template<typename T>
        requires std::same_as<T, char>
Deserialize read(BufferAlloc& v) {
        return [&v](AbstractDeserializer& de, DeserialzationScope&) {
                new (v.getBuffer()) char(de.dChar());
        };
}
```

```
template<typename T>
	requires std::same_as<T, bool>
Deserialize read(BufferAlloc& v) {
	return [&v](AbstractDeserializer& de, DeserialzationScope&) {
		new (v.getBuffer()) bool(de.dBool());
	};
}



template<typename T>
	requires Deserializable<T>
Deserialize read(BufferAlloc& v) {
	return T::read(v);
}

template<typename T>
Deserialize deserialize(StackAllocBuffer<T>& v) {
	return [&v](AbstractDeserializer& de, DeserialzationScope& scope) {
		read<T>(v)(de, scope);
		v.setWasInitialized(true);
```

```
        };
}

template<typename T>
Deserialize deserialize(HeapAllocBuffer& v) {
        return [&v](AbstractDeserializer& de, DeserialzationScope& scope) {
                read<T>(v)(de, scope);
        };
}

template<typename T>
        requires std::derived_from<T, SerializableObject>
Deserialize deserialize(T*& v) {
        return [&v](AbstractDeserializer& de, DeserialzationScope& scope) {

                StackAllocBuffer<std::string> type;
                auto str = de.dStruct();
                str->next(scope, "type", deserialize(type));

                const DeserializationDescription& description =
TypeAccumulator::getInstance().find(*type);
                HeapAllocBuffer obj(description.getObjectSize());
                str->next(scope, "value", description.getDeserializer()(obj));
```

```
            v = reinterpret_cast<T*>(obj.getBuffer());
        };
}

template<typename T>
concept HasRead = requires(T t, HeapAllocBuffer & buffer) {
        {read<T>(buffer) };
};

template<HasRead T>
        requires (!std::derived_from<T, SerializableObject>)
Deserialize deserialize(T*& v) {
        return [&v](AbstractDeserializer& de, DeserialzationScope& scope) {
                HeapAllocBuffer obj(sizeof(T));
                read<T>(obj);
                v = reinterpret_cast<T*>(obj.getBuffer());
        };
}

template<typename T>
Deserialize heapDeserialize(HeapAllocBuffer& buffer) {
        return read<T>(buffer);
```

```
}
template<char const* name, typename T>
void registerClassDeserialize() {

    auto de = &heapDeserialize<T>;
    TypeAccumulator::getInstance().add(name, DeserializationDescription(sizeof(T), de));
}

#define RegisterSerializeClass(type) \
private: \
virtual const char* _getObjectName() const { return typeName; } \
public: \
static constexpr char typeName[] = #type; \
static inline StaticInit<registerClassDeserialize<typeName, type>> init; \




template<typename T>
class DeserializeBuffer;
```

```
template<typename T>
	requires std::is_pointer<T>::value&&
std::derived_from<std::remove_pointer_t<T>, SerializableObject>
class DeserializeBuffer<T> {
private:
	T _value = nullptr;
public:
	inline T& buffer() { return _value; }
	T& operator*() {
		return _value;
	}
};

template<typename T>
	requires (!std::is_pointer<T>::value ||
!std::derived_from<std::remove_pointer_t<T>, SerializableObject>)
class DeserializeBuffer<T> {
private:
	StackAllocBuffer<T> _buffer;
public:
	inline StackAllocBuffer<T>& buffer() { return _buffer; }
	T&& operator*() {
		return  *_buffer;
```

```
        }
};
```

Example.cpp

```
#include "Example.h"

Writer Person::serializeCurrent() const {
        return [this](AbstractSerializer& s) {
                auto structSer = s.sStruct();
                structSer->begin();
                structSer->sKeyVal("name", write(_name));
                structSer->sKeyVal("age", write(_age));
                structSer->sKeyVal("sex", write(_sex));
                structSer->finish();
        };
}

Deserialize Person::read(BufferAlloc& buffer)
{
        return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
                DeserializeBuffer<std::string> name;
```

```
            DeserializeBuffer<unsigned int> age;
            DeserializeBuffer<Sex> sex;
            auto structDe = de.dStruct();
            structDe->begin();
            structDe->next(scope, "name", deserialize(name.buffer()));
            structDe->next(scope, "age", deserialize(age.buffer()));
            structDe->next(scope, "sex", deserialize(sex.buffer()));
            structDe->end();
            new (buffer.getBuffer()) Person(*name, *age, *sex);
        };
}

void Person::show(std::ostream& o)
{
        o << _name << ", " << _age << ", " << _sex;
}

Writer Student::serializeCurrent() const
{
        return [this](AbstractSerializer& s) {
            auto structSer = s.sStruct();
            structSer->begin();
```

```
            structSer->sKeyVal("Person", this->Person::serializeCurrent());
            structSer->sKeyVal("languages", write(_languages));
            structSer->finish();
        };
}

Deserialize Student::read(BufferAlloc& buffer)
{
        return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
            DeserializeBuffer<Person> person;
            DeserializeBuffer<std::vector<std::string>> languages;
            auto structDe = de.dStruct();
            structDe->begin();
            structDe->next(scope, "Person", deserialize(person.buffer()));
            structDe->next(scope, "languages", deserialize(languages.buffer()));
            structDe->end();
            new (buffer.getBuffer()) Student(*person, *languages);
        };
}

void Student::show(std::ostream& o)
{
```

```
	o << (Person&)(*this) << ' ';
	for (size_t i = 0; i < _languages.size(); i++)
	{
		o << _languages[i] << ", ";
	}
}


Writer Color::serilaizeCurrent() const
{
	return [this](AbstractSerializer& se) {
		auto structSer = se.sStruct();
		structSer->begin();
		structSer->sKeyVal("red", write(_red));
		structSer->sKeyVal("green", write(_green));
		structSer->sKeyVal("blue", write(_blue));
		structSer->finish();
	};
}

Deserialize Color::read(BufferAlloc& buffer)
{
```

```
        return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
                DeserializeBuffer<char> red;
                DeserializeBuffer<char> green;
                DeserializeBuffer<char> blue;
                auto structSer = de.dStruct();
                structSer->begin();
                structSer->next(scope, "red", deserialize(red.buffer()));
                structSer->next(scope, "green", deserialize(green.buffer()));
                structSer->next(scope, "blue", deserialize(blue.buffer()));
                structSer->end();
                new (buffer.getBuffer()) Color(*red, *green, *blue);
        };
}
```

Example.h

```
#pragma once
#include <string>
#include "Serializer.h"
#include "Deserializer.h"
#include "DeserializeExt.h"

enum Sex {
```

```
        male, female
};



Writer write(const std::same_as<Sex> auto& v) {
        return [&v](AbstractSerializer& s) {
                switch (v)
                {
                case male:
                        s.sString("male");
                        break;
                case female:
                        s.sString("female");
                        break;
                }
        };
}

template<typename T>
        requires std::same_as<T, Sex>
Deserialize read(BufferAlloc& v) {
        return [&v](AbstractDeserializer& s, DeserialzationScope&) {
```

```
            std::string value = s.dString();


            if (value == "male") {
                new (v.getBuffer()) Sex(male);
            }
            else if (value == "female") {
                new (v.getBuffer()) Sex(female);
            }
            else {
                throw DeserializationException("Expected male or female string");
            }
        };
}




class Person : public SerializableObject {
private:
        virtual const char* _getObjectName() const {
            return typeName;
        }
public:
```

```
    static constexpr char typeName[] = "Person";
    static inline StaticInit<registerClassDeserialize<typeName, Person>> init;
private:
    std::string _name;
    unsigned int _age;
    Sex _sex;
private:
    Writer _serialize() const override { return serializeCurrent(); }
public:
    Person(const std::string& name, const unsigned int& age, const Sex& sex)
        : _name(name), _age(age), _sex(sex) {
    }


    Writer serializeCurrent() const;
    static Deserialize read(BufferAlloc& buffer);

    void show(std::ostream&);

    inline const std::string& getName() const { return _name; }
    inline unsigned int getAge() const { return _age; }
    inline Sex getSex() const { return _sex; }
```

```
};



inline std::ostream& operator<<(std::ostream& o, Person& p) {
	p.show(o);
	return o;
}

class Student : public Person {
	RegisterSerializeClass(Student);
private:
	std::vector<std::string> _languages;

	Writer _serialize() const override { return serializeCurrent(); }

public:
	Student(const Person& person, const std::vector<std::string>& languages)
		: Person(person), _languages(languages) {
	}

	Student(const std::string& name, const unsigned int& age, const Sex& sex, const std::vector<std::string>& languages)
		: Person(name, age, sex), _languages(languages) {
```

```
    }

    Writer serializeCurrent() const;
    static Deserialize read(BufferAlloc& buffer);
    void show(std::ostream&);

};

inline std::ostream& operator<<(std::ostream& o, Student& p) {
    p.show(o);
    return o;
}

class Color : public SerializableObject {
    RegisterSerializeClass(Color);
private:
    char _red;
    char _green;
    char _blue;
    Writer _serialize() const override { return serilaizeCurrent(); }
```

```
public:
    Color(char red, char green, char blue) : _red(red), _green(green), _blue(blue) {}
    static Deserialize read(BufferAlloc& buffer);
    Writer serilaizeCurrent() const;
};



template<Serializable T>
class BinaryTree {
private:
    struct Node {
        T _value;
        Node* _left;
        Node* _right;
        Node(const T& value, Node* _left = nullptr, Node* _right = nullptr) :
_value(value), _left(_left), _right(_right) {}
    };
    Node* _root;

public:
```

```
    BinaryTree() : _root(nullptr) {};

    void add(const T&);
};

template<Serializable T>
inline void BinaryTree<T>::add(const T& v)
{
    if (_root == nullptr)
        _root = new Node(v);
    Node* current;
    while (true) {
        if (v < current->_value && current->_left) {
            current = current->_left;
        }
        else if (v > current->_value && current->_right) {
            current = current->_right;
        }
        else {
            return;
        }
    }
```

```
        if (v < current->_value) {
                current->_left = new Node(v);
        }
        else if (v > current->_value) {
                current->_right = new Node(v);
        }
}
```

furniture.cpp

```
#include "furniture.h"
#include "SerializeExt.h"
#include "DeserializeExt.h"


Writer Chair::_serialize() const
{
        return serializeCurrent();
}


//template<class Archive>
```

```
//void serialize(Archive& ar)
//{
//	ar(cereal::base_class<Furniture>(this), _frameMaterial, _outerMaterial, _reclining);
//}

Writer Chair::serializeCurrent() const
{
	return [this](AbstractSerializer& serializer) {
		auto stSer = serializer.sStruct();
		stSer->begin();
		stSer->sKeyVal("name", write(_name));
		stSer->sKeyVal("frameMaterial", write(_frameMaterial));
		stSer->sKeyVal("outerMaterial", write(_outerMaterial));
		stSer->sKeyVal("reclining", write(_reclining));
		stSer->finish();
	};
}

Deserialize Chair::read(BufferAlloc& buffer)
{
	return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
		DeserializeBuffer<string> name;
```

```
            DeserializeBuffer<string> frameMaterial;
            DeserializeBuffer<optional<string>> outerMaterial;
            DeserializeBuffer<bool> reclining;
            auto stSer = de.dStruct();
            stSer->begin();
            stSer->next(scope, "name", deserialize(name.buffer()));
            stSer->next(scope, "frameMaterial", deserialize(frameMaterial.buffer()));
            stSer->next(scope, "outerMaterial", deserialize(outerMaterial.buffer()));
            stSer->next(scope, "reclining", deserialize(reclining.buffer()));
            stSer->end();
            new (buffer.getBuffer()) Chair(*name, *frameMaterial, *outerMaterial,
*reclining);
        };
}



Writer Table::_serialize() const
{
    return serializeCurrent();
}

Writer Table::serializeCurrent() const
{
```

```
    return [this](AbstractSerializer& serializer) {
        auto stSer = serializer.sStruct();
        stSer->begin();
        stSer->sKeyVal("name", write(_name));
        stSer->sKeyVal("material", write(_material));
        stSer->finish();
    };
}

Deserialize Table::read(BufferAlloc& buffer)
{
    return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
        DeserializeBuffer<string> name;
        DeserializeBuffer<string> material;
        auto stSer = de.dStruct();
        stSer->begin();
        stSer->next(scope, "name", deserialize(name.buffer()));
        stSer->next(scope, "material", deserialize(material.buffer()));
        stSer->end();
        new (buffer.getBuffer()) Table(*name, *material);
    };
}
```

```
Writer Bed::_serialize() const
{
	return serializeCurrent();
}

Writer Bed::serializeCurrent() const
{
	return [this](AbstractSerializer& serializer) {
		auto stSer = serializer.sStruct();
		stSer->begin();
		stSer->sKeyVal("name", write(_name));
		stSer->sKeyVal("includesMattress", write(_includesMattress));
		stSer->sKeyVal("peopleCount", write(_peopleCount));
		stSer->finish();
	};
}

Deserialize Bed::read(BufferAlloc& buffer)
{
	return [&buffer](AbstractDeserializer& de, DeserialzationScope& scope) {
		DeserializeBuffer<string> name;
```

```
            DeserializeBuffer<bool> includesMattress;
            DeserializeBuffer<uint32_t> peopleCount;
            auto stSer = de.dStruct();
            stSer->begin();
            stSer->next(scope, "name", deserialize(name.buffer()));
            stSer->next(scope, "includesMattress",
deserialize(includesMattress.buffer()));
            stSer->next(scope, "peopleCount", deserialize(peopleCount.buffer()));
            stSer->end();
            new (buffer.getBuffer()) Bed(*name, *includesMattress, *peopleCount);
        };
}
```

furniture.h

```
#pragma once
#include <string>
#include "Serializer.h"
#include <optional>
#include "Deserializer.h"
//#include <cereal/types/base_class.hpp>
//#include <cereal/types/polymorphic.hpp>

using std::string;
```

```
using std::optional;



class Furniture : public SerializableObject {
protected:
	string _name;
public:
	Furniture(string name) : _name(name) {}
	Furniture() {}
	virtual ~Furniture() {}

	inline const string& name() const { return _name; }

	//template<class Archive>
	//void serialize(Archive& ar)
	//{
	//	ar(_name);
	//}
};

class Chair : public Furniture {
	RegisterSerializeClass(Chair);
```

```
private:
    string _frameMaterial;
    optional<string> _outerMaterial;
    bool _reclining;
    virtual Writer _serialize() const;

public:
    Chair(string name, string frameMaterial, optional<string> outerMaterial, bool reclining) :
        Furniture(name), _frameMaterial(frameMaterial), _outerMaterial(outerMaterial), _reclining(reclining) {}

    //template<class Archive>
    //void serialize(Archive& ar)
    //{
    //    ar(cereal::base_class<Furniture>(this), _frameMaterial, _outerMaterial, _reclining);
    //}

    Writer serializeCurrent() const;
    static Deserialize read(BufferAlloc&);

};
```

```
class Table : public Furniture {
        RegisterSerializeClass(Table);
private:
        string _material;
        virtual Writer _serialize() const;
public:
        Table(string name, string material) : Furniture(name), _material(material) {}



        //template<class Archive>
        //void serialize(Archive& ar)
        //{
        //      ar(cereal::base_class<Furniture>(this), _material);
        //}

        Writer serializeCurrent() const;
        static Deserialize read(BufferAlloc&);
};

class Bed : public Furniture {
        RegisterSerializeClass(Bed);
```

```
private:
	bool _includesMattress;
	uint32_t _peopleCount;
	virtual Writer _serialize() const;
public:
	Bed(string name, bool includesMattresse, uint32_t peopleCount) :
		Furniture(name), _includesMattress(includesMattresse),
_peopleCount(peopleCount) {}

	//template<class Archive>
	//void serialize(Archive& ar)
	//{
	//	ar(cereal::base_class<Furniture>(this), _includesMattress, _peopleCount);
	//}

	Writer serializeCurrent() const;
	static Deserialize read(BufferAlloc&);
};

//CEREAL_REGISTER_TYPE(Bed);
//CEREAL_REGISTER_TYPE(Chair);
//CEREAL_REGISTER_TYPE(Table);
//CEREAL_REGISTER_POLYMORPHIC_RELATION(Furniture, Bed);
```

```
//CEREAL_REGISTER_POLYMORPHIC_RELATION(Furniture, Chair);
//CEREAL_REGISTER_POLYMORPHIC_RELATION(Furniture, Table);
```

httpExample.cpp

```
#include <iostream>
#include <boost/asio.hpp>
#include <httplib.h>
#include "Deserializer.h"
#include "furniture.h"
#include <boost/json.hpp>
#include "JSONDe.h"
#include "JSONSer.h"
#include "access.h"
#include "SerializeExt.h"
#include "DeserializeExt.h"
#include "httpExample.h"

boost::asio::io_service io_service;
boost::posix_time::seconds interval(10); // 10 seconds
boost::asio::deadline_timer timer(io_service, interval);
```

```
void task(const boost::system::error_code&)
{
	// do something
	updateCache();

	// reschedule the timer for the next interval
	timer.expires_at(timer.expires_at() + interval);
	timer.async_wait(task);
}



void runHttpExample()
{
	// load configuration for the server
	Configuration config(loadConfig());



	loadTestData();

	std::thread run_thread([&] {
```

```
        timer.async_wait(task);
        io_service.run();
        });




    httplib::Server svr;

    svr.Get(R"(/furniture/(.+))", [](const httplib::Request& req, httplib::Response& res)
{
        auto name = req.matches[1];
        DeserializeBuffer<Furniture*> furniture;
        try {
            auto furniture = getFurniture(name);
            std::ostringstream stream;
            JSONSer ser(stream);
            write(furniture)(ser);

            res.set_content(req.body, "application/json");
            res.body = stream.str();

        }
```

```
        catch (...) {
        }

        });

svr.Put("/furniture", [](const httplib::Request& req, httplib::Response& res) {

        DeserializeBuffer<Furniture*> furniture;
        try {
                boost::json::value v(boost::json::parse(req.body));
                JSONDe de(v);
                DeserialzationScope scope;
                deserialize(furniture.buffer())(de, scope);
                insertFurniture(**furniture);
                delete* furniture;
        }
        catch (...) {

        }

        });
```

```
    std::cout << "Port for server is: " << config.port() << std::endl;
    svr.listen("localhost", config.port());


}
```

httpExample.h

```
#pragma once
#include <boost/beast/core.hpp>
#include <boost/beast/http.hpp>
#include <boost/beast/version.hpp>
#include <boost/asio/connect.hpp>
#include <boost/asio/ip/tcp.hpp>
#include <iostream>
#include <string>



void runHttpExample();
```

JSONDe.cpp

```
#include "JSONDe.h"
#include <iostream>

long long JSONDe::_dLong()
{
	return _value.as_int64();
}

unsigned long long JSONDe::_dULong()
{
	//std::cout << std::boolalpha << _value.is_int64() << _value.is_uint64() << _value.as_int64();
	if (_value.is_int64())
		return _value.as_int64();
	else
		return _value.as_uint64();
}

std::string JSONDe::_dString()
{
	return std::string(_value.as_string().operator std::string_view());
}
```

```
bool JSONDe::_dBool()
{
	return _value.as_bool();
}

char JSONDe::_dChar()
{
	return _value.as_string()[0];
}

std::unique_ptr<AbstractMapDe> JSONDe::_dMap()
{
	return std::unique_ptr<AbstractMapDe>(new JSONMapDe(_value.as_object()));
}

std::unique_ptr<AbstractSeqDe> JSONDe::_dSeq()
{
	return std::unique_ptr<AbstractSeqDe>(new JSONSeqDe(_value.as_array()));
}

std::unique_ptr<AbstractStructDe> JSONDe::_dStruct()
```

```
{
	return std::unique_ptr<AbstractStructDe>(new JSONStructDe(_value.as_object()));
}

bool JSONMapDe::_hasMore()
{
	return _current != _last;
}

std::string JSONMapDe::_nextKey()
{
	boost::json::string_view s = _current->key();
	_currentValue = &_current->value();
	return s;
}

void JSONMapDe::_nextValue(DeserialzationScope& scope, const Deserialize& d)
{
	if (_currentValue == nullptr) {
		throw DeserializationException("Key/Value pair hasn't been read yet");
```

```
        }
        JSONDe deserializer = JSONDe(*_currentValue);
        d(deserializer, scope);
        ++_current;
}

void JSONMapDe::_begin()
{
}

void JSONMapDe::_end()
{
}

bool JSONSeqDe::_hasMore()
{
        return _current != _last;
}

void JSONSeqDe::_nextItem(DeserialzationScope& scope, const Deserialize& d)
{
        JSONDe des(*_current);
```

```
        d(des, scope);
        ++_current;
}

void JSONSeqDe::_begin()
{
}

void JSONSeqDe::_end()
{
}

void JSONStructDe::_next(DeserialzationScope& scope, const std::string& key, const Deserialize& d)
{
        JSONDe des(_obj.at(key));
        d(des, scope);
}

void JSONStructDe::_begin()
{
}
```

```
void JSONStructDe::_end()
{
}
```

JSONDe.h

```
#pragma once
#include "Deserializer.h"
#include <fstream>
#include <optional>
#include <boost/json.hpp>
#include <boost/json/detail/value_from.hpp>

class JSONDe final : public AbstractDeserializer {

private:
	boost::json::value& _value;

	virtual long long _dLong();
	virtual unsigned long long _dULong();
	virtual std::string _dString();
	virtual bool _dBool();
	virtual char _dChar();
```

```
        virtual std::unique_ptr<AbstractMapDe> _dMap();
        virtual std::unique_ptr<AbstractSeqDe> _dSeq();
        virtual std::unique_ptr<AbstractStructDe> _dStruct();
public:
        JSONDe( boost::json::value& value) : _value(value) {};
};

class JSONMapDe final : public AbstractMapDe {
private:
        boost::json::object::iterator _current;
        boost::json::object::iterator _last;

         boost::json::value* _currentValue = nullptr;

        virtual bool _hasMore() override;
        virtual std::string _nextKey() override;
        virtual void _nextValue(DeserialzationScope& scope, const Deserialize&) override;
        virtual void _begin() override;
        virtual void _end() override;
public:
        JSONMapDe(boost::json::object& map) : _current(map.begin()), _last(map.end())
{}
```

```
};


class JSONSeqDe final : public AbstractSeqDe {
private:
        boost::json::array::iterator _current;
        boost::json::array::iterator _last;

        virtual bool _hasMore();
        virtual void _nextItem(DeserialzationScope&, const Deserialize&);
        virtual void _begin() override;
        virtual void _end() override;
public:
        JSONSeqDe(boost::json::array& arr) : _current(arr.begin()), _last(arr.end()) {}

};

class JSONStructDe final : public AbstractStructDe {
private:
        boost::json::object& _obj;
        virtual void _next(DeserialzationScope& scope, const std::string&, const
Deserialize&);
        virtual void _begin() override;
```

```
        virtual void _end() override;
public:
        JSONStructDe(boost::json::object& obj) : _obj(obj) {}

};
```

JSONSer.cpp

```
#include "JSONSer.h"
#include <boost/json.hpp>



void JSONMapSer::_begin()
{
        _out << '{';
}

void JSONMapSer::_finish()
{

        _out << '}';
}
```

```
std::unique_ptr<AbstractMapSer> JSONSer::_sMap(size_t)
{
    return  std::unique_ptr<AbstractMapSer>(new JSONMapSer(_out, *this));
}

std::unique_ptr<AbstractSeqSer> JSONSer::_sSeq(size_t)
{
    return std::unique_ptr<AbstractSeqSer>(new JSONSeqSer(_out, *this));
}

std::unique_ptr<AbstractStructSer> JSONSer::_sStruct()
{
    return std::unique_ptr<AbstractStructSer>(new JSONStructSer(_out, *this));
}

void JSONSer::_sLong(long long v)
{
    _out << v;
}


void JSONSer::_sULong(unsigned long long v)
```

```
{
	_out << v;
}

void JSONSer::_sString(const std::string& v)
{
	_out << boost::json::serialize(boost::json::string(v));
}

void JSONSer::_sBool(bool v)
{
	_out << v;
}

void JSONSer::_sChar(char v)
{
	_out << "\"" << v << "\"";
}

void JSONSeqSer::_begin()
{
	_out << '[';
```

```
}

void JSONSeqSer::_finish()
{
	_out << ']';
}

void JSONStructSer::_begin()
{
	_out << '{';
}

void JSONStructSer::_finish()
{
	_out << '}';
}


void JSONMapSer::_sKeyVal(const std::string& k, const Writer& v)
{
	if (addComma)
		_out << ',';
```

```
        _ser.sString(k);
        _out << ':';
        v(_ser);
        addComma = true;
}

void JSONStructSer::_sKeyVal(const std::string& k, const Writer& v)
{
        if (addComma)
                _out << ',';
        _ser.sString(k);
        _out << ':';
        v(_ser);
        addComma = true;
}

void JSONSeqSer::_sNext(const Writer& v)
{
        if (addComma)
                _out << ',';
        v(_ser);
        addComma = true;
```

```
}
```

JSONSer.h

```
#pragma once
#include "Serializer.h"
#include <fstream>
#include <memory>
#include "types.h"

class JSONSer;
class JSONMapSer;
class JSONSeqSer;
class JSONStructSer;

class JSONSer final
	: public AbstractSerializer {
private:
	OutStream& _out;

	virtual		std::unique_ptr<AbstractMapSer> _sMap(size_t);
	virtual std::unique_ptr<AbstractSeqSer> _sSeq(size_t);
	virtual std::unique_ptr<AbstractStructSer> _sStruct();
```

```
        virtual void _sLong(long long);
        virtual void _sULong(unsigned long long);
        virtual void _sString(const std::string& v);
        virtual void _sBool(bool);
        virtual void _sChar(char);
public:
        JSONSer(OutStream& out) : _out(out) {
                out << std::boolalpha;
        };
};

class JSONMapSer final : public AbstractMapSer {
private:
        OutStream& _out;
        JSONSer& _ser;
        bool addComma = false;

        virtual void _sKeyVal(const std::string&, const Writer& v);
        virtual void _begin();
        virtual void _finish();
public:
        JSONMapSer(OutStream& out, JSONSer& ser) : _out(out), _ser(ser) { }
```

```
        ~JSONMapSer() { }


};

class JSONSeqSer final : public AbstractSeqSer {
private:
        OutStream& _out;
        JSONSer& _ser;
        bool addComma = false;
        virtual void _sNext(const Writer& v);
        virtual void _begin();
        virtual void _finish();

public:
        JSONSeqSer(OutStream& out, JSONSer& ser) : _out(out), _ser(ser) { }
        ~JSONSeqSer() { }

};

class JSONStructSer final : public AbstractStructSer {
private:
```

```
        OutStream& _out;
        JSONSer& _ser;
        bool addComma = false;
        virtual void _sKeyVal(const std::string&, const Writer&);
        virtual void _begin();
        virtual void _finish();
public:
        JSONStructSer(OutStream& out, JSONSer& ser) : _out(out), _ser(ser) { }
        ~JSONStructSer() { }

};
```

main.cpp

```
#include "Serializer.h"
#include <vector>
#include <iostream>
#include <io.h>
#include <fstream>
#include "Deserializer.h"
#include "Example.h"
#include "Test.h"
```

```
#include "httpExample.h"



using namespace std;



int main() {
	//serTest1();
	runHttpExample();
}
```

Node.h

```
#pragma once
#include "Serializer.h"

template<typename T>
class Node {
private:
	T _value;
	std::unique_ptr<Node<T>> _next;
public:
	Node(const T& value) : _value(value), _next() {}
	Node& connect(const T& nextValue);
```

```
        template<Serializer S>
        void save(S& ser);
};

template<typename T>
inline Node<T>& Node<T>::connect(const T& nextValue)
{
        _next.reset(new Node(nextValue));
        return *_next;
}

//template<typename T>
//template<Serializer S>
//inline void Node<T>::save(S& ser)
//{
//        auto seqSer = ser.sSeq();
//        auto* current = this;
//
//}
```

resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by SerTest.rc

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        101
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1001
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

SerializeExt.h

```
#pragma once
#include "Serializer.h"

template<typename T>
        requires std::same_as<std::optional<typename T::value_type>, T>
inline Writer write(const T& v) {
```

```
	return[&v](AbstractSerializer& ser) {
		auto st = ser.sStruct();
		st->begin();
		st->sKeyVal("hasValue", write(v.has_value()));
		if (v.has_value()) {
			st->sKeyVal("value", write(*v));
		}

		st->finish();
	};
}

template<typename T>
	requires std::same_as<std::unique_ptr<typename T::element_type>, T>
inline Writer write(const T& v) {
	return [&v](AbstractSerializer& s) {
		write(v.get())(s);
	};
}

template<typename T>
```

```
        requires std::same_as<std::shared_ptr<typename T::element_type>, T>
inline Writer write(const T& v) {
        return [&v](AbstractSerializer& s) {
                write(v.get())(s);
        };
}
```

Serializer.h

```
#pragma once
#include <concepts>
#include <string>
#include <vector>
#include <iostream>
#include <iterator>
#include <map>
#include <functional>
#include <optional>

class AbstractSerializer;
class AbstractMapSer;
class AbstractSeqSer;
class AbstractStructSer;
```

```
using Writer = std::function<void(AbstractSerializer&)>;


class AbstractSerializer {
protected:
	virtual ~AbstractSerializer() {}
private:
	virtual std::unique_ptr<AbstractMapSer> _sMap(size_t) = 0;
	virtual std::unique_ptr<AbstractSeqSer> _sSeq(size_t) = 0;
	virtual std::unique_ptr<AbstractStructSer> _sStruct() = 0;

	virtual void _sLong(long long) = 0;
	virtual void _sULong(unsigned long long) = 0;
	virtual void _sString(const std::string&) = 0;
	virtual void _sBool(bool) = 0;
	virtual void _sChar(char) = 0;
public:

	inline std::unique_ptr<AbstractMapSer> sMap(size_t s) { return _sMap(s); }
	inline std::unique_ptr<AbstractSeqSer> sSeq(size_t s) { return _sSeq(s); };
	inline std::unique_ptr<AbstractStructSer> sStruct() { return _sStruct(); };
```

```
        inline void sLong(long long v) { return _sLong(v); };
        inline void sULong(unsigned long long v) { return _sULong(v); };
        inline void sString(std::string v) { return _sString(v); };
        inline void sBool(bool v) { return _sBool(v); };
        inline void sChar(char v) { return _sChar(v); };

};



template<typename T>
concept Serializable = requires(T t) {
        {write(t)};
};

class AbstractMapSer {
private:
        virtual void _sKeyVal(const std::string&, const Writer&) = 0;
        virtual void _begin() = 0;
        virtual void _finish() = 0;

public:
```

```
        virtual ~AbstractMapSer() {}
        inline void sKeyVal(const std::string& k, const Writer& v) { return _sKeyVal(k, v);
};
        void begin() { return _begin(); };
        void finish() { return _finish(); };

};


class AbstractSeqSer {
private:

        virtual void _sNext(const Writer& v) = 0;
        virtual void _begin() = 0;
        virtual void _finish() = 0;
public:
        virtual ~AbstractSeqSer() {}

        inline void sNext(const Writer& v) { return _sNext(v); }
        inline void begin() { return _begin(); };
        inline void finish() { return _finish(); };
};
```

```
class AbstractStructSer {
private:
	virtual void _sKeyVal(const std::string&, const Writer&) = 0;
	virtual void _begin() = 0;
	virtual void _finish() = 0;

public:
	virtual ~AbstractStructSer() {}

	inline void sKeyVal(const std::string& k, const Writer& v) { return _sKeyVal(k, v);
};
	inline void begin() { return _begin(); };
	inline void finish() { return _finish(); };
};


inline Writer write(std::same_as<long long> auto v) {
	return [v](AbstractSerializer& s) {s.sLong(v); };
}

inline Writer write(std::same_as<unsigned long long> auto v) {
	return [v](AbstractSerializer& s) {s.sULong(v); };
}
```

```
inline Writer write(std::same_as<int> auto v) {
	return write((long long)v);
}

inline Writer write(std::same_as<unsigned int> auto v) {
	return write((unsigned long long) v);
}



inline Writer write(const std::same_as<std::string> auto& v) {
	return [&v](AbstractSerializer& s) {s.sString(v); };
}

inline Writer write(const std::same_as<const char* const> auto& v) {
	return [&v](AbstractSerializer& s) {s.sString(v); };
}


inline Writer write(std::same_as<bool> auto v) {
```

```
	return [v](AbstractSerializer& s) {s.sBool(v); };
}

inline Writer write(std::same_as<char> auto v) {
	return [v](AbstractSerializer& s) {s.sChar(v); };
}



class SerializableObject {
private:
	virtual Writer _serialize() const = 0;
	virtual const char* _getObjectName() const = 0;
public:
	inline Writer serialize() const { return _serialize(); }
	inline const char* getObjectName() const { return _getObjectName(); }
};

template<typename T>
requires std::derived_from<T, SerializableObject>
inline Writer write(const T* v) {
	return [v](AbstractSerializer& s) {
		auto structSer = s.sStruct();
```

```
            structSer->begin();
            structSer->sKeyVal("type", write<std::string>(v->getObjectName()));
            structSer->sKeyVal("value", write<T>(*v));
            structSer->finish();
        };
}


template<typename T>
requires (!std::derived_from<T, SerializableObject>)
inline Writer write(const T* v) {
        return [v](AbstractSerializer& s) {
            auto structSer = s.sStruct();
            structSer->begin();
            structSer->sKeyVal("value", write<T>(*v));
            structSer->finish();
        };
}
```

```
template<typename T>
concept SerializableIterator = requires(T t) {
        T::reference;

        Serializable<typename T::reference>;
        { ++t } -> std::same_as<T>;
        { *t } -> std::same_as<typename T::reference>;
};

template<typename T>
concept SerializableSeq = requires(T t) {
        SerializableIterator<decltype(t.begin())>;
        SerializableIterator<decltype(t.end())>;
        {t.size()} -> std::same_as<size_t>;
};

template<typename T>
concept SerializableMap = requires(T t) {
        SerializableIterator<decltype(t.begin())>;
        SerializableIterator<decltype(t.end())>;
        {t.size()} -> std::same_as<size_t>;
```

```
        typename T::key_type;
        typename T::value_type;
        {*t.begin()} -> std::same_as < std::pair<typename T::key_type, typename T::value_type>>;

};

template<typename T>
        requires  (!std::same_as<T, std::string>) && SerializableSeq<T> && (!std::derived_from<T, SerializableObject>)
inline Writer write(const T& v) {
        return [&v](AbstractSerializer& s) {
                auto seqSer = s.sSeq(v.size());
                seqSer->begin();
                for (auto iter = v.begin(); iter != v.end(); ++iter) {
                        auto& current = *iter;
                        seqSer->sNext(write(current));
                }
                seqSer->finish();
        };
}

template<typename T>
```

```
        requires  (!std::same_as<T, std::string>) && SerializableMap<T> &&
(!std::derived_from<T, SerializableObject>)
inline Writer write(const T& v) {
        return [&v](AbstractSerializer& s) {
                auto seqSer = s.sSeq(v.size());
                seqSer->begin();
                for (auto iter = v.begin(); iter != v.end(); ++iter) {
                        auto& current = *iter;
                        seqSer->sNext(write(current));
                }
                seqSer->finish();
        };
}

template<typename T>
        requires std::derived_from<T, SerializableObject>
inline Writer write(const T& v) {
        return v.serialize();
}
```

```
struct ExceededSize {};
```

SerTest.vcxproj

```
<?xml version="1.0" encoding="utf-8"?>
<Project DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <ItemGroup Label="ProjectConfigurations">
    <ProjectConfiguration Include="Debug|Win32">
      <Configuration>Debug</Configuration>
      <Platform>Win32</Platform>
    </ProjectConfiguration>
    <ProjectConfiguration Include="Release|Win32">
      <Configuration>Release</Configuration>
      <Platform>Win32</Platform>
    </ProjectConfiguration>
    <ProjectConfiguration Include="Debug|x64">
      <Configuration>Debug</Configuration>
      <Platform>x64</Platform>
    </ProjectConfiguration>
    <ProjectConfiguration Include="Release|x64">
```

```
    <Configuration>Release</Configuration>
    <Platform>x64</Platform>
  </ProjectConfiguration>
 </ItemGroup>
 <PropertyGroup Label="Globals">
  <VCProjectVersion>16.0</VCProjectVersion>
  <Keyword>Win32Proj</Keyword>
  <ProjectGuid>{34bfeb50-6034-4ce7-9d85-a1ce52a20522}</ProjectGuid>
  <RootNamespace>SerTest</RootNamespace>
  <WindowsTargetPlatformVersion>10.0</WindowsTargetPlatformVersion>
 </PropertyGroup>
 <Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />
 <PropertyGroup Condition="'$(Configuration)|$(Platform)'=='Debug|Win32'"
Label="Configuration">
  <ConfigurationType>Application</ConfigurationType>
  <UseDebugLibraries>true</UseDebugLibraries>
  <PlatformToolset>v143</PlatformToolset>
  <CharacterSet>Unicode</CharacterSet>
 </PropertyGroup>
 <PropertyGroup Condition="'$(Configuration)|$(Platform)'=='Release|Win32'"
Label="Configuration">
  <ConfigurationType>Application</ConfigurationType>
  <UseDebugLibraries>false</UseDebugLibraries>
```

```
    <PlatformToolset>v143</PlatformToolset>
    <WholeProgramOptimization>true</WholeProgramOptimization>
    <CharacterSet>Unicode</CharacterSet>
  </PropertyGroup>
  <PropertyGroup Condition="'$(Configuration)|$(Platform)'=='Debug|x64'"
Label="Configuration">
    <ConfigurationType>Application</ConfigurationType>
    <UseDebugLibraries>true</UseDebugLibraries>
    <PlatformToolset>v143</PlatformToolset>
    <CharacterSet>Unicode</CharacterSet>
  </PropertyGroup>
  <PropertyGroup Condition="'$(Configuration)|$(Platform)'=='Release|x64'"
Label="Configuration">
    <ConfigurationType>Application</ConfigurationType>
    <UseDebugLibraries>false</UseDebugLibraries>
    <PlatformToolset>v143</PlatformToolset>
    <WholeProgramOptimization>true</WholeProgramOptimization>
    <CharacterSet>Unicode</CharacterSet>
  </PropertyGroup>
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.props" />
  <ImportGroup Label="ExtensionSettings">
  </ImportGroup>
  <ImportGroup Label="Shared">
```

```
  </ImportGroup>
  <ImportGroup Label="PropertySheets"
Condition="'$(Configuration)|$(Platform)'=='Debug|Win32'">
    <Import Project="$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props"
Condition="exists('$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props')"
Label="LocalAppDataPlatform" />
  </ImportGroup>
  <ImportGroup Label="PropertySheets"
Condition="'$(Configuration)|$(Platform)'=='Release|Win32'">
    <Import Project="$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props"
Condition="exists('$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props')"
Label="LocalAppDataPlatform" />
  </ImportGroup>
  <ImportGroup Label="PropertySheets"
Condition="'$(Configuration)|$(Platform)'=='Debug|x64'">
    <Import Project="$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props"
Condition="exists('$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props')"
Label="LocalAppDataPlatform" />
  </ImportGroup>
  <ImportGroup Label="PropertySheets"
Condition="'$(Configuration)|$(Platform)'=='Release|x64'">
    <Import Project="$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props"
Condition="exists('$(UserRootDir)\Microsoft.Cpp.$(Platform).user.props')"
Label="LocalAppDataPlatform" />
  </ImportGroup>
```

```
  <PropertyGroup Label="UserMacros" />
  <ItemDefinitionGroup Condition="'$(Configuration)|$(Platform)'=='Debug|Win32'">
    <ClCompile>
      <WarningLevel>Level3</WarningLevel>
      <SDLCheck>true</SDLCheck>

<PreprocessorDefinitions>WIN32;_DEBUG;_CONSOLE;%(PreprocessorDefinitions)</P
reprocessorDefinitions>
      <ConformanceMode>true</ConformanceMode>
    </ClCompile>
    <Link>
      <SubSystem>Console</SubSystem>
      <GenerateDebugInformation>true</GenerateDebugInformation>
    </Link>
  </ItemDefinitionGroup>
  <ItemDefinitionGroup Condition="'$(Configuration)|$(Platform)'=='Release|Win32'">
    <ClCompile>
      <WarningLevel>Level3</WarningLevel>
      <FunctionLevelLinking>true</FunctionLevelLinking>
      <IntrinsicFunctions>true</IntrinsicFunctions>
      <SDLCheck>true</SDLCheck>
```

<PreprocessorDefinitions>WIN32;NDEBUG;_CONSOLE;%(PreprocessorDefinitions)</PreprocessorDefinitions>

<ConformanceMode>true</ConformanceMode>

</ClCompile>

<Link>

<SubSystem>Console</SubSystem>

<EnableCOMDATFolding>true</EnableCOMDATFolding>

<OptimizeReferences>true</OptimizeReferences>

<GenerateDebugInformation>true</GenerateDebugInformation>

</Link>

</ItemDefinitionGroup>

<ItemDefinitionGroup Condition="'$(Configuration)|$(Platform)'=='Debug|x64'">

<ClCompile>

<WarningLevel>Level3</WarningLevel>

<SDLCheck>true</SDLCheck>

<PreprocessorDefinitions>_DEBUG;_CONSOLE;%(PreprocessorDefinitions)</PreprocessorDefinitions>

<ConformanceMode>true</ConformanceMode>

<LanguageStandard>stdcpp20</LanguageStandard>

</ClCompile>

<Link>

```
      <SubSystem>Console</SubSystem>
      <GenerateDebugInformation>true</GenerateDebugInformation>
    </Link>
  </ItemDefinitionGroup>
  <ItemDefinitionGroup Condition="'$(Configuration)|$(Platform)'=='Release|x64'">
    <ClCompile>
      <WarningLevel>Level3</WarningLevel>
      <FunctionLevelLinking>true</FunctionLevelLinking>
      <IntrinsicFunctions>true</IntrinsicFunctions>
      <SDLCheck>true</SDLCheck>

<PreprocessorDefinitions>NDEBUG;_CONSOLE;%(PreprocessorDefinitions)</Preproce
ssorDefinitions>
      <ConformanceMode>true</ConformanceMode>
    </ClCompile>
    <Link>
      <SubSystem>Console</SubSystem>
      <EnableCOMDATFolding>true</EnableCOMDATFolding>
      <OptimizeReferences>true</OptimizeReferences>
      <GenerateDebugInformation>true</GenerateDebugInformation>
    </Link>
  </ItemDefinitionGroup>
  <ItemGroup>
```

```
    <ClCompile Include="access.cpp" />
    <ClCompile Include="BinaryDe.cpp" />
    <ClCompile Include="BinarySer.cpp" />
    <ClCompile Include="Deserializer.cpp" />
    <ClCompile Include="Example.cpp" />
    <ClCompile Include="furniture.cpp" />
    <ClCompile Include="httpExample.cpp" />
    <ClCompile Include="JSONDe.cpp" />
    <ClCompile Include="JSONSer.cpp" />
    <ClCompile Include="main.cpp" />
  </ItemGroup>
  <ItemGroup>
    <ClInclude Include="access.h" />
    <ClInclude Include="BinaryDe.h" />
    <ClInclude Include="BinarySer.h" />
    <ClInclude Include="BufferAlloc.h" />
    <ClInclude Include="DeserializeExt.h" />
    <ClInclude Include="Deserializer.h" />
    <ClInclude Include="Example.h" />
    <ClInclude Include="furniture.h" />
    <ClInclude Include="httpExample.h" />
    <ClInclude Include="JSONDe.h" />
```

```
    <ClInclude Include="JSONSer.h" />
    <ClInclude Include="Node.h" />
    <ClInclude Include="SerializeExt.h" />
    <ClInclude Include="Serializer.h" />
    <ClInclude Include="StaticInit.h" />
    <ClInclude Include="Test.h" />
    <ClInclude Include="types.h" />
  </ItemGroup>
  <ItemGroup>
    <None Include="ClassDiagram.cd" />
    <None Include="ClassDiagram1.cd" />
    <None Include="cpp.hint" />
  </ItemGroup>
  <Import Project="$(VCTargetsPath)\Microsoft.Cpp.targets" />
  <ImportGroup Label="ExtensionTargets">
  </ImportGroup>
</Project>
```

SerTest.vcxproj.filters

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <ItemGroup>
    <Filter Include="Source Files">
```

```
    <UniqueIdentifier>{4FC737F1-C7A5-4376-A066-
2A32D752A2FF}</UniqueIdentifier>
    <Extensions>cpp;c;cc;cxx;c++;cppm;ixx;def;odl;idl;hpj;bat;asm;asmx</Extensions>
  </Filter>
  <Filter Include="Header Files">
    <UniqueIdentifier>{93995380-89BD-4b04-88EB-
625FBE52EBFB}</UniqueIdentifier>
    <Extensions>h;hh;hpp;hxx;h++;hm;inl;inc;ipp;xsd</Extensions>
  </Filter>
  <Filter Include="Resource Files">
    <UniqueIdentifier>{67DA6AB6-F800-4c08-8B7A-
83BB121AAD01}</UniqueIdentifier>

<Extensions>rc;ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe;resx;tiff;tif;png;wav;mfcri
bbon-ms</Extensions>
  </Filter>
  <Filter Include="Header Files\httpExample">
    <UniqueIdentifier>{2421e27b-e228-4119-a7a7-4bb2365307e9}</UniqueIdentifier>
  </Filter>
  <Filter Include="Source Files\httpExample">
    <UniqueIdentifier>{287c7eec-cf31-4f48-8194-5c5e268418ea}</UniqueIdentifier>
  </Filter>
 </ItemGroup>
 <ItemGroup>
```

```
<ClCompile Include="main.cpp">
  <Filter>Source Files</Filter>
</ClCompile>
<ClCompile Include="BinarySer.cpp">
  <Filter>Source Files</Filter>
</ClCompile>
<ClCompile Include="JSONSer.cpp">
  <Filter>Source Files</Filter>
</ClCompile>
<ClCompile Include="Example.cpp">
  <Filter>Source Files</Filter>
</ClCompile>
<ClCompile Include="JSONDe.cpp">
  <Filter>Source Files</Filter>
</ClCompile>
<ClCompile Include="Deserializer.cpp">
  <Filter>Source Files</Filter>
</ClCompile>
<ClCompile Include="BinaryDe.cpp">
  <Filter>Source Files</Filter>
</ClCompile>
<ClCompile Include="httpExample.cpp">
```

```
    <Filter>Source Files\httpExample</Filter>
  </ClCompile>
  <ClCompile Include="furniture.cpp">
    <Filter>Source Files\httpExample</Filter>
  </ClCompile>
  <ClCompile Include="access.cpp">
    <Filter>Source Files\httpExample</Filter>
  </ClCompile>
 </ItemGroup>
 <ItemGroup>
  <ClInclude Include="Serializer.h">
    <Filter>Header Files</Filter>
  </ClInclude>
  <ClInclude Include="BinarySer.h">
    <Filter>Header Files</Filter>
  </ClInclude>
  <ClInclude Include="Deserializer.h">
    <Filter>Header Files</Filter>
  </ClInclude>
  <ClInclude Include="Example.h">
    <Filter>Header Files</Filter>
  </ClInclude>
```

```
<ClInclude Include="Test.h">
  <Filter>Header Files</Filter>
</ClInclude>
<ClInclude Include="JSONSer.h">
  <Filter>Header Files</Filter>
</ClInclude>
<ClInclude Include="JSONDe.h">
  <Filter>Header Files</Filter>
</ClInclude>
<ClInclude Include="Node.h">
  <Filter>Header Files</Filter>
</ClInclude>
<ClInclude Include="DeserializeExt.h">
  <Filter>Header Files</Filter>
</ClInclude>
<ClInclude Include="BinaryDe.h">
  <Filter>Header Files</Filter>
</ClInclude>
<ClInclude Include="BufferAlloc.h">
  <Filter>Header Files</Filter>
</ClInclude>
<ClInclude Include="StaticInit.h">
```

```
    <Filter>Header Files</Filter>
  </ClInclude>
  <ClInclude Include="httpExample.h">
    <Filter>Header Files\httpExample</Filter>
  </ClInclude>
  <ClInclude Include="furniture.h">
    <Filter>Header Files\httpExample</Filter>
  </ClInclude>
  <ClInclude Include="access.h">
    <Filter>Header Files\httpExample</Filter>
  </ClInclude>
  <ClInclude Include="types.h">
    <Filter>Header Files</Filter>
  </ClInclude>
  <ClInclude Include="SerializeExt.h">
    <Filter>Header Files</Filter>
  </ClInclude>
</ItemGroup>
<ItemGroup>
  <None Include="ClassDiagram.cd" />
  <None Include="cpp.hint" />
  <None Include="ClassDiagram1.cd" />
```

```
  </ItemGroup>
</Project>
```

SerTest.vcxproj.user

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="Current"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup />
</Project>
```

StaticInit.h

```
#pragma once

template<void Func()>
class StaticInit {
public:
	StaticInit() {
		Func();
	}
};
```

Test.h

```
#pragma once
#include <string>
#include <vector>
```

```
#include "BinarySer.h"
#include "Deserializer.h"
#include "Example.h"
#include <boost/json.hpp>
#include "JSONSer.h"
#include "JSONDe.h"
#include "BinaryDe.h"

template<typename T>
inline void serTest(std::string path, const T& value) {
	{
		std::ofstream outBinary(path + ".txt");
		outBinary.clear();
		BinarySer s(outBinary);
		write(value)(s);
		outBinary.close();
	}

	{
		std::ofstream outJSON(path + ".json");
		outJSON.clear();
		JSONSer s(outJSON);
```

```
        write(value)(s);
        outJSON.close();
}


{
        std::ifstream inBinary(path + ".txt");
        BinaryDe de(inBinary);
        DeserialzationScope scope;
        DeserializeBuffer<T> value;
        deserialize(value.buffer())(de, scope);

        auto result = value.operator*();
        std::cout << result << std::endl;
}


{
        std::ifstream inJson(path + ".json");
        boost::json::value v(boost::json::parse(inJson));

        JSONDe de(v);
        DeserialzationScope scope;
```

```
            DeserializeBuffer<T> value;
            deserialize(value.buffer())(de, scope);

            auto result = value.operator*();
            std::cout << result << std::endl;
        }

}

void serTest1() {

        Person p("Andrii Zahorulko", 400, Sex::male);
        serTest("./out1", p);

        Student s("Andrii Zahorulko", 19, Sex::male, { "dart", "c#", "rust" });
        serTest("./out2", s);

}
```

types.h

```
#pragma once
```

```
#include <fstream>



using OutStream = std::ostream;

using InStream = std::istream;
```