МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра Автоматизированных систем управления

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6
ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ»
«Использование контейнеров STL. Исключения»

Факультет: АВТ
Группа: ДТ-460а
Студент: Пантюхин Артём Евгеньевич

Преподаватель:
Лауферман Ольга Викторовна

Новосибирск, 2025 г.

Оглавление

Постановка задачи

Необходимо разработать ещё один или два класса, а затем в один из них добавить поле с набором (коллекцией) объектов второго. Хотя все коллекции и предназначены для хранения набора объектов, они все отличаются. Какие-то коллекции накладывают определённые ограничения на хранящиеся в коллекции элементы, другие отличаются скоростью доступа к элементам, третьи отличаются интерфейсом и т.д.

Необходимо выбрать STL-контейнер (коллекцию), наиболее подходящий для решения поставленной задачи и обосновать выбор.

При возникновении ошибок методы должны генерировать исключения при помощи throw, исключения обязательно должны быть обработаны выше по стеку вызовов. Задача: разработать класс Раздел на накопителе данных с полями: название, номер байта начала раздела, номер байта конца раздела, тип файловой системы, объём занятой памяти в байтах. Название у раздела может быть не задано. Файловая система может отсутствовать. Начальный и конечный номера байтов задаются при создании объекта Раздел и далее не меняются. Объём занятой памяти, название и файловую систему можно изменять во время жизни объекта. При изменении файловой системы все данные стираются с раздела.

Добавить в класс Накопитель список разделов на нём. Разделы не могут пересекаться. Между разделами может быть свободное место. Удалите методы для установки количества разделов и их суммарного объёма. Добавьте методы для добавления разделов на накопитель и удаления разделов с накопителя по их номеру.

Раздел не может существовать вне накопителя. Поместим разрабатываемый класс в закрытую часть пространства имён класса Накопитель, в классе Накопитель реализуем методы получения доступа к полям Раздела по ID в словаре.

В силу отсутствия класса Раздел в открытой части пространства имён класса Накопитель, нам нет нужды реализовывать множество конструкторов класса. Достаточно реализации конструктора со всеми полями класса, частичное же заполнение обеспечит промежуточный метод из класса Накопитель, подставив значения по умолчанию.

Для создания разделов, необходимо реализовать функцию проверки занятости

произвольного адресного пространства в классе НакопительДанных.

Также, поскольку мы разрабатываем исключительно модель организации данных, заранее подумаем о разработке интерфейса и взаимодействии с моделью. Может случиться так, что разработчик решит сохранить когда-то полученные ID разделов, удалить раздел, а после создать новый, но обратиться к нему по старому ID с ожиданием получения старых же данных. Во избежание подобного, добавим в класс НакопительДанных поле, отслеживающее состояние базы ID (в нашем случае, ID просто увеличивается на 1 с каждым успешным созданием, поэтому тип переменной – unsigned long), благодаря чему при попытке обращения к устаревшим данным разработчик интерфейса получит исключение вместо, возможно, совершенно другого экземпляра класса Раздел.

Известно, что некоторые системы не позволяют использовать спецсимволы в названиях разделов. Названия файловых систем также не должны содержать спецсимволов. Для проверок корректности ввода данных полей в классе Раздел реализуем отдельный метод IsValidString. Он не будет конфликтовать с одноимённым методом класса НакопительДанных ввиду разной области видимости.

Тестовые данные:

Проверка функционала класса НакопительДанных проводилась в лабораторной работе №5. Данные тестовые данные нацелены исключительно на проверку функционала класса Раздел и новых разработанных методов класса НакопительДанных.

Функции Get*() класса НакопительДанных

Функция получения	Значение из конструктора	Ожидаемое значение
Производитель	Seagate	Seagate
Модель	ID-K-Some-4455-thing	ID-K-Some-4455-thing
Серийный номер	887665	887665
Вид накопителя	Task::StorageDrive::DriveTypes::DT_SSD	SSD
Тип таблицы разделов	Task::StorageDrive::PartitionTypes::PT_GPT	GPT
Ёмкость	16391	16391
Свободное место	Не указано	16391
Количество разделов	Не указано	0
Суммарный объём разделов	Не указано	0
Наличие таблицы разделов	Не указано	1 (true)

На данных значениях:

Функции класса Раздел

Функция	Входные значения	Ожидаемый результат
Создание раздела	0, 512, «Volume1», «ext4»	Успешное создание со всеми полями
Создание раздела	0, 4097	Исключение: Раздел с такими адресами не может быть расположен на накопителе.
Создание раздела	2566, 4000	Успешное создание с пустыми полями имени и файловой системы.
Удаление раздела по ID	3	Успешное удаление
Удаление раздела по ID	3	Удаления не произошло, исключения нет
Создание раздела	0, 500, Нет таблицы разделов	Исключение: Невозможно создать раздел на неразмеченном накопителе

Получение информации для несуществующего раздела	3, любая GetVolume*()	Исключение unordered_map::at
Установка поля «Имя»	«vali-d»	Исключение: некорректное имя раздела
Установка поля «Файловая система»	«fsy’stem»	Исключение: некорректный тип файловой системы
Установка поля «Занятый объём»	Из конструктора: 0-16000 Входные данные: 18000	Исключение: невозможно занять больше пространства, чем размер раздела
Установка поля «Имя»	«valid»	Значение поля установлено
Установка поля «Файловая система»	«fsystem»	Значение поля установлено. Поле «Занятый объём» == 0
Установка поля «Занятый объём»	Из конструктора: 0-16000 Входные данные: 16000	1 свободный байт на разделе.

Описание разработанных классов

Volume			
Поля	Примечание	Методы	Примечание
std::string name;	Закрытое Название раздела, может быть пустым	unsigned long long GetStartByte() const;	Открытый Возвращает адрес начала раздела.
std::string filesystem;	Закрытое Тип файловой системы, может быть пустым	unsigned long long GetEndByte() const;	Открытый Возвращает адрес конца раздела.
unsigned long long startByte;	Закрытое Адрес первого байта раздела	std::string GetName() const;	Открытый Возвращает имя раздела.
unsigned long long endByte;	Закрытое Адрес последнего байта раздела	std::string GetFilesystem() const;	Открытый Возвращает тип файловой системы.
unsigned long long usedSpace;	Закрытое Объём занятого полезными данными пространства	unsigned long long GetSize() const;	Открытый Возвращает вычисленное значение размера раздела (end-start+1)
		unsigned long long GetUsedSpace() const;	Открытый Возвращает значение поля "занятое место"
		void SetName(const std::string& newName);	Открытый. Валидирует и устанавливает значение поля «имя раздела» Строка передаётся как ссылка на константную строку и копируется в поле.
		void SetFilesystem(const	Открытый. Валидирует и

		std::string& fSysName);	устанавливает значение поля «Файловая система» Устанавливает «занятое место» в 0. Строка передаётся как ссылка на константную строку и копируется в поле.
		void SetUsedSpace(unsigned long long newSpace);	Открытый. Устанавливает объём занятого пространства. Объём не может быть больше размера раздела.
		void _SetBoundary(unsigned long long start, unsigned long long end);	Закрытый. Вызывается в конструкторе, устанавливает значения адреса для первого и последнего байта. Адрес последнего байта не может быть меньше адреса первого.
		static bool _IsValidString(const std::string& sample);	Закрытый. Статический. Вызывается при установке строковых значений для валидации.

Выбор типа контейнера

В качестве типа контейнера был выбран словарь std::unordered_map с типом ключа unsigned long, типом значения – Volume. Данный контейнер хранит данные в формате {ключ;значение}. В отличие от std::map контейнер не сортирует значения по ключу, что полезно для нашей задачи, ввиду возможности отслеживать хронологию добавления/удаления разделов. Кроме того, условие задачи гласит, что доступ к данным накопителя должен быть возможен по присвоенному номеру, что убирает смысл в использовании последовательных контейнеров – время доступа станет больше, часть контейнеров будет тратить кратно больше времени на реаллокацию при добавлении значений, и в нашем случае нет таких позитивных факторов, чтобы перевесить данные негативные. Помимо прочего, выбранный вариант вызывает исключение при попытке доступа к отсутствующему элементу, что полезно в рамках очерчивания чётких границ дозволенного в работе с нашим классом.

Проектирование функции main() для тестирования класса:

Переиспользуем данные из main() для класса НакопительДанных без класса Раздел для проверки сохранения функциональности методов класса НакопительДанных.
На базе этих данных, проведём проверку работы промежуточных методов для получения и установки данных в классе Раздел. Моя реализация предполагает вызов исключений и проверку входных значений только в самих методах класса Раздел, что позволяет нам говорить и о проверке работоспособности разработанного класса в целом.
Помимо проверок базовых функций получения/установки значений класса раздел на корректных и некорректных значениях, уделяется внимание работоспособности промежуточного метода-конструктора класса Раздел, проверяется создание экземпляра класса в условиях, должных вызвать исключение при попытке создания класса.
Для наглядности проверки реализованы также две вспомогательные функции вывода значений, не являющиеся частью задания.
Работа в main() ведётся над единичным экземпляром класса НакопительДанных с множеством объектов класса Раздел в составе.
Для достижения полной уверенности в готовности продукта исполняемый файл проверяется на утечки памяти при помощи утилиты valgrind с аргументом --leak-check=full.

Тестирование программы

В ходе тестирования все методы показали ожидаемый результат. Отклонений не выявлено. Вывод результатов работы main() в терминал вынесен в Приложение 2.

Выводы

В ходе выполнения данной лабораторной работы были углублены навыки работы с классами в языке C++, в процессе решения задач реализации получен опыт использования деструкторов, различных видов STL-контейнеров, изучены принципы работы хэш-таблиц, алгоритмов бинарного поиска в рамках функционала хэш-таблиц, логика аллокации памяти стандатных последовательных контейнеров.
Собственную реализацию считаю не идеальной, но хорошим выбором между времязатратами и функционалом.

Приложение. Исходные коды всех разработанных файлов.

StorageDrive.hpp:

```
#ifndef STORAGE_DRIVE_HPP
#define STORAGE_DRIVE_HPP

#include <string>
#include <unordered_map>
#include <vector>

namespace Task
{
   class StorageDrive
   {
      public:
         enum class DriveTypes //перечисление видов накопителя
         {
            DT_SSD = 1 << 0,
            DT_HDD = 1 << 1,
         };

         enum class PartitionTypes //перечисление типов таблицы разделов
         {
            PT_UNPARTITIONED = 0,
            PT_MBR = 1 << 0,
            PT_GPT = 1 << 1,
         };

         StorageDrive(); //конструктор по умолчанию
         StorageDrive( //конструктор информационной составляющей
            const std::string& Vendor, //производитель
            const std::string& Model, //модель
            const std::string& Serial, //серийный номер
            DriveTypes DriveType //вид накопителя
         );

         StorageDrive( //конструктор технической составляющей
            DriveTypes DriveType, //вид накопителя
            PartitionTypes PartitionType, //тип таблицы разделов
            unsigned long long StorageSpace //ёмкость
         );

         StorageDrive( //полный конструктор
            const std::string& Vendor, //производитель
            const std::string& Model,//модель
            const std::string& Serial,//серийный номер
            DriveTypes DriveType,//вид накопителя
            PartitionTypes PartitionType,//тип таблицы разделов
```

```
            unsigned long long StorageSpace //ёмкость
        );

        std::string GetVendor() const; //получить значение поля "производитель"
        std::string GetModel() const;//получить значение поля "модель"
        std::string GetSerialNumber() const;//получить значение поля "серийный номер"
        DriveTypes GetDriveType() const;//получить значение поля "вид накопителя"
        std::string GetDriveTypeName() const;//получить значение поля "вид накопителя" в виде
строки
        PartitionTypes GetPTableType() const;//получить значение поля "тип таблицы разделов"
        std::string GetPTableName() const;//получить значение поля "тип таблицы разделов" в виде
строки

        unsigned long long GetStorageSpace() const;//получить значение поля "ёмкость"

        void SetVendor(const std::string& newVendor); //установить значение поля "производитель"
        void SetModel(const std::string& newModel); //установить значение поля "модель"
        void SetSerial(const std::string& newSerial); //установить значение поля "серийный номер"
        void SetDriveType(DriveTypes newDriveType); //установить значение поля "вид накопителя"
        void SetStorageSpace(unsigned long long newStorageSpace); //установить значение поля
"ёмкость

        bool IsPartitioned() const; //получить информацию о наличии таблицы разделов

        void SetPTableType(PartitionTypes type);
        unsigned long long GetFreeSpace() const;//получить информацию о незанятом месте на
накопителе
        unsigned long GetVolumeAmount() const;//получить значение поля "количество разделов"
        unsigned long long GetVolumeSizeSum() const;//получить значение поля "суммарный объём
разделов"
        void CreateVolume( //"конструктор" раздела
            unsigned long long startByte,
            unsigned long long endByte,
            const std::string& name = "",
            const std::string& filesystem = "");
        std::vector<unsigned long> GetVolumeIDs() const; //получение вектора ID существующих
разделов
        bool DeleteVolume(unsigned long ID); //удаление раздела
        unsigned long long GetVolumeStartByte(unsigned long ID) const; //получение адреса начала
раздела
        unsigned long long GetVolumeEndByte(unsigned long ID) const; //получение адреса конца
раздела
        std::string GetVolumeName(unsigned long ID) const; //получение имени раздела
        std::string GetVolumeFilesystem(unsigned long ID) const; //получение файловой системы
раздела
        unsigned long long GetVolumeSize(unsigned long ID) const; //получение размера раздела
        unsigned long long GetVolumeUsedSpace(unsigned long ID) const; //получение занятого
пространства раздела
        void SetVolumeName(unsigned long ID, const std::string& name); //установка нового имени
раздела
```

```
        void SetVolumeFSystem(unsigned long ID, const std::string& fSysName); //установка новой файловой системы раздела
        void SetVolumeUsedSpace(unsigned long ID, unsigned long long newSpace); //обновление занятого места раздела

    private:
        class Volume
        {
            public:
                /*
                Конструктор со всеми полями.
                Поскольку класс находится в закрытой части StorageDrive,
                Мы используем метод-прокладку в публичной части StorageDrive,
                Поэтому дополнительные конструкторы не нужны.
                */
                Volume(
                    const std::string& volName,
                    const std::string& filesystem,
                    unsigned long long startByte,
                    unsigned long long endByte
                );

                unsigned long long GetStartByte() const; //Возвращает адрес начала раздела.
                unsigned long long GetEndByte() const; //Возвращает адрес конца раздела.
                std::string GetName() const; //Возвращает имя раздела.
                std::string GetFilesystem() const; //Возвращает тип файловой системы.
                unsigned long long GetSize() const; //возвращает вычисленное значение размера раздела (end-start+1)
                unsigned long long GetUsedSpace() const; //возвращает значение поля "занятое место"

                //Устанавливает название раздела.
                void SetName(const std::string& newName);
                //Устанавливает тип файловой системы. При установке происходит очистка занятого пространства.
                void SetFilesystem(const std::string& fSysName);
                //устанавливает объём занятого пространства. Не может быть больше размера раздела.
                void SetUsedSpace(unsigned long long newSpace);

            private:
                std::string name; //название раздела, может быть пустым
                std::string filesystem; //тип файловой системы, может быть пустым
                unsigned long long startByte; //адрес первого байта раздела
                unsigned long long endByte; //адрес последнего байта раздела
                unsigned long long usedSpace; //объём занятого полезными данными пространства

                void _SetBoundary(unsigned long long start, unsigned long long end); //вызывается в конструкторе
                static bool _IsValidString(const std::string& sample); //внутренний метод, определяет корректность ввода строк name и filesystem.
        };
```

```
        std::string vendor; //производитель
        std::string model; //модель
        std::string serial; //серийный номер
        DriveTypes driveType; //вид накопителя
        PartitionTypes partitionType; //тип таблицы разделов
        unsigned long long storageSpace; //ёмкость

        std::unordered_map<unsigned long, Volume> volumes; //словарь разделов
        //служебная функция проверки, свободен ли указанный диапазон адресов
        bool _doIFit(unsigned long long startByte, unsigned long long endByte) const;
        //служебный счётчик ID разделов, чтобы не допустить повторения.
        unsigned long lastUsedID;
        static bool _IsValidString(const std::string& sample); //служебная функция проверки
корректности входной строки
    };
}
#endif
```

StorageDrive.cpp

```
#include <exception>
#include <stdexcept>
#include <string>
#include <vector>
#include <unordered_map>

#include "StorageDrive.hpp"

namespace Task
{
    StorageDrive::StorageDrive()
    {
        //Я был неправ в ЛР5, дело в том, что я не константы принимал.
        SetVendor("Unknown Vendor");
        SetModel("Generic Storage Drive");
        SetSerial("Unknown");
        SetDriveType(DriveTypes::DT_HDD);
        SetStorageSpace(0);
        SetPTableType(PartitionTypes::PT_UNPARTITIONED);
        volumes.clear();
        lastUsedID = 0;
    }

    StorageDrive::StorageDrive(
        const std::string& Vendor,
        const std::string& Model,
        const std::string& Serial,
```

```
    DriveTypes DriveType
)
{
    SetVendor(Vendor);
    SetModel(Model);
    SetSerial(Serial);
    SetDriveType(DriveType);
    SetStorageSpace(0);
    SetPTableType(PartitionTypes::PT_UNPARTITIONED);
    volumes.clear();
    lastUsedID = 0;
}

StorageDrive::StorageDrive(
    DriveTypes DriveType,
    PartitionTypes PartitionType,
    unsigned long long StorageSpace
)
{
    //Я был неправ в ЛР5, дело в том, что я не константы принимал.
    SetVendor("Unknown Vendor");
    SetModel("Generic Storage Drive");
    SetSerial("Unknown");
    SetDriveType(DriveType);
    SetStorageSpace(StorageSpace);
    SetPTableType(PartitionType);
    volumes.clear();
    lastUsedID = 0;
}

StorageDrive::StorageDrive(
    const std::string& Vendor,
    const std::string& Model,
    const std::string& Serial,
    DriveTypes DriveType,
    PartitionTypes PartitionType,
    unsigned long long StorageSpace
)
{
    SetVendor(Vendor);
    SetModel(Model);
    SetSerial(Serial);
    SetDriveType(DriveType);
    SetStorageSpace(StorageSpace);
    SetPTableType(PartitionType);
    volumes.clear();
    lastUsedID = 0;
}

std::string StorageDrive::GetVendor() const
```

```
{
    return vendor;
}

std::string StorageDrive::GetModel() const
{
    return model;
}

std::string StorageDrive::GetSerialNumber() const
{
    return serial;
}

StorageDrive::DriveTypes StorageDrive::GetDriveType() const
{
    return driveType;
}

std::string StorageDrive::GetDriveTypeName() const
{
    //тернарный оператор не имеет особого смысла кроме лёгкой экономии места.
    return driveType == DriveTypes::DT_HDD ? std::string("HDD") : std::string("SSD");
}

StorageDrive::PartitionTypes StorageDrive::GetPTableType() const
{
    return partitionType;
}

std::string StorageDrive::GetPTableName() const
{
    std::string out("Таблица отсутствует");
    if (partitionType == PartitionTypes::PT_GPT) {
        out = std::string("GPT");
    }
    if (partitionType == PartitionTypes::PT_MBR) {
        out = std::string("MBR");
    }
    return out;
}

unsigned long long StorageDrive::GetFreeSpace() const
{
    if (!IsPartitioned()) return storageSpace;

    return storageSpace-GetVolumeSizeSum();
}

void StorageDrive::SetVendor(const std::string& newVendor)
```

```
{
    if (!_IsValidString(newVendor))
    {
        throw std::invalid_argument("Поле \"Производитель\": некорректный ввод!");
    };

    vendor = newVendor;
}


void StorageDrive::SetModel(const std::string& newModel)
{
    if (!_IsValidString(newModel))
    {
        throw std::invalid_argument("Поле \"Модель\": некорректный ввод!");
    }

    model = newModel;
}


void StorageDrive::SetSerial(const std::string& newSerial)
{
    if (!_IsValidString(newSerial))
    {
        throw std::invalid_argument("Поле \"Серийный номер\": некорректный ввод!");
    }

    serial = newSerial;
}

void StorageDrive::SetDriveType(StorageDrive::DriveTypes newDriveType)
{
    driveType = newDriveType;
}

void StorageDrive::SetStorageSpace(unsigned long long newStorageSpace)
{
    storageSpace = newStorageSpace;
    SetPTableType(PartitionTypes::PT_UNPARTITIONED);
}

bool StorageDrive::IsPartitioned() const
{
    return !(partitionType == PartitionTypes::PT_UNPARTITIONED);
}

bool StorageDrive::_IsValidString(const std::string& sample) {
    int isNotSpecial = true;
    //данная функция не учитывает конец таблицы ASCII, а так же кириллические символы
```

```
        //кириллица не учитывается по причине необходимости переработки строки под "широкие" символы
        for (auto ch {sample.begin()}; ch != sample.end() && isNotSpecial; ch++) {
            isNotSpecial = (*ch >= 'a' && *ch <= 'z')
                || (*ch >= 'A' && *ch <= 'Z')
                || (*ch >= ' ' && *ch < '@');
        }
        //также проверим "непустоту" строки.
        return !sample.empty() && isNotSpecial;
    }

    unsigned long StorageDrive::GetVolumeAmount() const
    {
        return volumes.size();
    }

    unsigned long long StorageDrive::GetVolumeSizeSum() const
    {
        unsigned long long sum = 0;
        for (auto vol{volumes.begin()}; vol != volumes.end(); vol++)
        {
            sum += vol->second.GetSize();
        }
        return sum;
    }

    unsigned long long StorageDrive::GetStorageSpace() const
    {
        return storageSpace;
    }

    void StorageDrive::SetPTableType(PartitionTypes type)
    {
        partitionType = type;
        volumes.clear();
    }

    bool StorageDrive::_doIFit(unsigned long long startByte, unsigned long long endByte) const
    {
        bool fit = endByte < storageSpace;
        unsigned long long tempStart, tempEnd;
        for (auto vol{volumes.begin()}; vol != volumes.end() && fit; vol++)
        {
            tempStart = vol->second.GetStartByte();
            tempEnd = vol->second.GetEndByte();
            //если хоть один из адресов оказался в пределах запрошенного - невозможно создать раздел.
            fit &= !((startByte <= tempStart) && (tempStart <= endByte));
            fit &= !((startByte <= tempEnd) && (tempEnd <= endByte));
        }
```

```
    return fit;
  }

  void StorageDrive::CreateVolume(unsigned long long startByte, unsigned long long endByte, const
std::string& name, const std::string& filesystem)
  {
    Volume* vol = 0;
    if (partitionType == PartitionTypes::PT_UNPARTITIONED)
    {
      throw std::domain_error("Невозможно создать раздел на неразмеченном накопителе");
    }
    if (!_doIFit(startByte, endByte))
    {
      throw std::out_of_range("Раздел с такими адресами не может быть расположен на
накопителе.");
    }
    try
    {
      vol = new Volume(name, filesystem, startByte, endByte);
    }
    catch (const std::logic_error& ex)
    {
      delete(vol);
      throw ex;
    }
    volumes.insert({lastUsedID++, *vol});
    delete(vol);
  }

  std::vector<unsigned long> StorageDrive::GetVolumeIDs() const
  {
    std::vector<unsigned long> out;
    out.reserve(volumes.size());
    for (auto vol{volumes.begin()}; vol != volumes.end(); vol++)
    {
      out.push_back(vol->first);
    }
    return out;
  }

  bool StorageDrive::DeleteVolume(unsigned long ID)
  {
    //В качестве информации передадим дальше bool = true, если удаление успешно.
    return volumes.erase(ID) != 0;
    //Можно было бы вызвать исключение если такого ID не было в нашем словаре
    //Но глобально - если его не было, ничего не изменилось, критической ошибки не произошло.
  }

  unsigned long long StorageDrive::GetVolumeStartByte(unsigned long ID) const
  {
```

```
    return volumes.at(ID).GetStartByte();
  }

  unsigned long long StorageDrive::GetVolumeEndByte(unsigned long ID) const
  {
    return volumes.at(ID).GetEndByte();
  }

  std::string StorageDrive::GetVolumeName(unsigned long ID) const
  {
    return volumes.at(ID).GetName();
  }

  std::string StorageDrive::GetVolumeFilesystem(unsigned long ID) const
  {
    return volumes.at(ID).GetFilesystem();
  }

  unsigned long long StorageDrive::GetVolumeSize(unsigned long ID) const
  {
    return volumes.at(ID).GetSize();
  }

  unsigned long long StorageDrive::GetVolumeUsedSpace(unsigned long ID) const
  {
    return volumes.at(ID).GetUsedSpace();
  }

  void StorageDrive::SetVolumeName(unsigned long ID, const std::string& name)
  {
    volumes.at(ID).SetName(name);
  }

  void StorageDrive::SetVolumeFSystem(unsigned long ID, const std::string& fSysName)
  {
    volumes.at(ID).SetFilesystem(fSysName);
  }

  void StorageDrive::SetVolumeUsedSpace(unsigned long ID, unsigned long long newSpace)
  {
    volumes.at(ID).SetUsedSpace(newSpace);
  }
}
```

Volume.cpp

```
#include "StorageDrive.hpp"

#include <stdexcept>
```

```
namespace Task
{
   StorageDrive::Volume::Volume(
      const std::string& volName,
      const std::string& filesystem,
      unsigned long long startAddress,
      unsigned long long endAddress
   )
   {
      _SetBoundary(startAddress, endAddress);
      SetName(volName);
      SetFilesystem(filesystem);
      SetUsedSpace(0);
   }

   unsigned long long StorageDrive::Volume::GetStartByte() const
   {
      return startByte;
   }
   unsigned long long StorageDrive::Volume::GetEndByte() const
   {
      return endByte;
   }
   std::string StorageDrive::Volume::GetName() const
   {
      return name;
   }
   std::string StorageDrive::Volume::GetFilesystem() const
   {
      return filesystem;
   }
   unsigned long long StorageDrive::Volume::GetUsedSpace() const
   {
      return usedSpace;
   }

   unsigned long long StorageDrive::Volume::GetSize() const
   {
      return GetEndByte()-GetStartByte()+1;
   }

   void StorageDrive::Volume::SetName(const std::string& newName)
   {
      if (!_IsValidString(newName))
      {
         throw std::invalid_argument("Некорректное имя раздела.");
      }
      name = newName;
   }
```

```
    void StorageDrive::Volume::SetFilesystem(const std::string& fSysName)
    {
        if (!_IsValidString(fSysName))
        {
            throw std::invalid_argument("Некорректный тип файловой системы.");
        }
        SetUsedSpace(0);
        filesystem = fSysName;
    }

    void StorageDrive::Volume::SetUsedSpace(unsigned long long newSpace)
    {
        if (newSpace > GetSize())
        {
            throw std::out_of_range("Невозможно занять больше пространства, чем размер раздела.");
        }
        usedSpace = newSpace;
    }


    void StorageDrive::Volume::_SetBoundary(unsigned long long start, unsigned long long end)
    {
        if (end < start)
        {
            throw std::invalid_argument("Начало раздела не может быть в памяти дальше его конца!");
        }
        startByte = start;
        endByte = end;
    }

    bool StorageDrive::Volume::_IsValidString(const std::string& sample)
    {
        bool isValid = true;
        for (auto ch{sample.begin()}; ch != sample.end() && isValid; ch++)
        {
            isValid = (*ch >= 'a' && *ch <= 'z')
                || (*ch >='0' && *ch <='9')
                || (*ch >='A' && *ch <= 'Z');
        }
        return isValid;
    }
}
```

Main.cpp

```
#include <exception>
#include <iostream>
#include <stdexcept>
```

```
#include <vector>

#include "StorageDrive.hpp"

void PrintDriveStorageInfo(const Task::StorageDrive* drive)
{
   if (!drive) return;
   std::cout << "Тип таблицы разделов: " << drive->GetPTableName() << std::endl;
   std::cout << "Ёмкость накопителя: " << drive->GetStorageSpace() << std::endl;
   std::cout << "Свободное место: " << drive->GetFreeSpace() << std::endl;
   std::cout << "Количество разделов: " << drive->GetVolumeAmount() << std::endl;
   std::cout << "Суммарный объём разделов: " << drive->GetVolumeSizeSum() << std::endl;
   std::cout << "Наличие таблицы разделов: " << drive->IsPartitioned() << std::endl;
}

void PrintDriveFullInfo(const Task::StorageDrive* drive)
{
   if(!drive) return;
   std::cout << "Производитель: " << drive->GetVendor() << std::endl;
   std::cout << "Модель: " << drive->GetModel() << std::endl;
   std::cout << "Серийный номер: " << drive->GetSerialNumber() << std::endl;
   std::cout << "Вид накопителя: " << drive->GetDriveTypeName() << std::endl;
   PrintDriveStorageInfo(drive);
}

void PrintVolumeInfo(const Task::StorageDrive* drive, unsigned long id)
{
   if (!drive) return;
   std::cout << "ID: " << id << std::endl;
   std::cout << "\tОбъём: " << drive->GetVolumeSize(id) << std::endl;
   std::cout << "\tПервый байт: " << drive->GetVolumeStartByte(id) << std::endl;
   std::cout << "\tПоследний байт: " << drive->GetVolumeEndByte(id) << std::endl;
   std::cout << "\tИмя раздела: " << drive->GetVolumeName(id) << std::endl;
   std::cout << "\tФайловая система: " << drive->GetVolumeFilesystem(id) << std::endl;
   std::cout << "\tЗанятое место: " << drive->GetVolumeUsedSpace(id) << std::endl;
}

void PrintStorageVolumes(const Task::StorageDrive* drive)
{
   if (!drive) return;
   std::vector<unsigned long> ids = drive->GetVolumeIDs();
   for (unsigned long i = 0; i < ids.size(); i++)
   {
      PrintVolumeInfo(drive, ids[i]);
      std::cout << std::endl;
   }
}

void PrintException(const char* ex)
```

```
{
    std::cout << "Исключение: " << ex << std::endl;
}

int main(void)
{
    Task::StorageDrive* drive;
    drive = new Task::StorageDrive(
        "Seagate",
        "ID-K-Some-4455-thing",
        "887665",
        Task::StorageDrive::DriveTypes::DT_SSD,
        Task::StorageDrive::PartitionTypes::PT_GPT,
        16391
    );
    std::cout << "Исходный накопитель:" << std::endl;
    PrintDriveFullInfo(drive);
    std::cout << "\nСоздадим разделы на весь объём накопителя:" << std::endl;
    drive->CreateVolume(0, 512, "Volume1", "ext4");
    drive->CreateVolume(513, 1024, "Volume2", "FAT32");
    drive->CreateVolume(1025, 2048, "Volume3", "NTFS");
    drive->CreateVolume(2049, 4096, "Volume4");
    drive->CreateVolume(4097,8192);
    drive->CreateVolume(8193, 16390);
    PrintDriveFullInfo(drive);
    PrintStorageVolumes(drive);
    std::cout << "Удалим раздел: с ID = 3" << std::endl;
    std::cout << "drive->DeleteVolume(3);" << std::endl;
    std::cout << "Результат: " << drive->DeleteVolume(3) << std::endl;
    PrintDriveFullInfo(drive);
    PrintStorageVolumes(drive);
    std::cout << "Попробуем добавить на его место не вмещающийся раздел:" << std::endl;
    std::cout << "drive->CreateVolume(0, 4097);" << std::endl;
    try
    {
        drive->CreateVolume(0, 4097);
    }
    catch (const std::out_of_range &ex)
    {
        PrintException(ex.what());
    }
    PrintStorageVolumes(drive);
    std::cout << "Добавим точно вмещающийся раздел:" << std::endl;
    std::cout << "drive->CreateVolume(2566, 4000);" << std::endl;
    drive->CreateVolume(2566, 4000);
    PrintStorageVolumes(drive);
    std::cout << "Заметим, что ID = 3 в списке уже нет, попробуем вновь удалить ID = 3" << std::endl;
    std::cout << "drive->DeleteVolume(3);" << std::endl;
    std::cout << "Результат: " << drive->DeleteVolume(3) << std::endl;
    PrintStorageVolumes(drive);
```

```
   std::cout << "С существующего раздела данные собрать удалось. Попробуем с несуществующего:"
<< std::endl;
   std::cout << "drive->GetVolumeFilesystem(3);" << std::endl;
   try
   {
      drive->GetVolumeFilesystem(3);
   }
   catch(const std::logic_error& ex)
   {
      PrintException(ex.what());
   }
   std::cout << "Исключение не наше, контейнер справился сам." << std::endl;
   std::cout << "\nУдалим разметку, попробуем создать раздел" << std::endl;
   std::cout << "drive->SetPTableType(Task::StorageDrive::PartitionTypes::PT_UNPARTITIONED);" <<
std::endl;
   std::cout << "drive->CreateVolume(0, 500);" << std::endl;
   drive->SetPTableType(Task::StorageDrive::PartitionTypes::PT_UNPARTITIONED);
   PrintDriveStorageInfo(drive);
   try
   {
      drive->CreateVolume(0, 500);
   }
   catch (std::exception& ex)
   {
      PrintException(ex.what());
   }
   std::cout << "\nПопробуем получить вектор ID разделов:" << std::endl;
   std::cout <<"drive->GetVolumeIDs().empty() == " << drive->GetVolumeIDs().empty() << std::endl;
   std::cout << "\nВернём разметку, удостоверимся что ID остались:" << std::endl;
   drive->SetPTableType(Task::StorageDrive::PartitionTypes::PT_MBR);
   drive->CreateVolume(0, 16000);
   std::cout << "drive->SetPTableType(Task::StorageDrive::PartitionTypes::PT_MBR);" << std::endl;
   std::cout << "drive->CreateVolume(0, 16000);" << std::endl;
   PrintDriveStorageInfo(drive);
   PrintStorageVolumes(drive);
   std::cout << "Разделы удалились, но отсчёт всё равно идёт с 7, как и должно быть." << std::endl;
   std::cout << "\nВ завершение тестирования, проверим работу установки значений для раздела:"
<< std::endl;
   std::cout << "Корректные:" << std::endl;
   std::cout << "drive->SetVolumeName(7, \"valid\");" << std::endl;
   std::cout << "drive->SetVolumeFSystem(7, \"fsystem\");" << std::endl;
   std::cout << "drive->SetVolumeUsedSpace(7, 16000);" << std::endl;
   drive->SetVolumeName(7, "valid");
   drive->SetVolumeFSystem(7, "fsystem");
   drive->SetVolumeUsedSpace(7, 16000);
   PrintStorageVolumes(drive);
   std::cout << "Некорректные:" << std::endl;
   std::cout << "drive->SetVolumeName(7, \"vali-d\");" << std::endl;
   try
   {
```

```
        drive->SetVolumeName(7, "vali-d");
    }
    catch (const std::exception& ex)
    {
        PrintException(ex.what());
    }
    std::cout << "drive->SetVolumeFSystem(7, \"fsy`stem\");" << std::endl;
    try
    {
        drive->SetVolumeFSystem(7, "fsy`stem");
    }
    catch (const std::exception& ex)
    {
        PrintException(ex.what());
    }
    std::cout << "drive->SetVolumeUsedSpace(7, 18000);" << std::endl;
    try
    {
        drive->SetVolumeUsedSpace(7, 18000);
    }
    catch (const std::exception& ex)
    {
        PrintException(ex.what());
    }
    std::cout << "\nОчистка занятого места при смене файловой системы:" << std::endl;
    PrintVolumeInfo(drive, 7);
    std::cout << "drive->SetVolumeFSystem(7, \"123\");" << std::endl;
    drive->SetVolumeFSystem(7, "123");
    PrintVolumeInfo(drive, 7);
    delete(drive);
    return 0;
}
```

Приложение 2. Вывод функции main()

Исходный накопитель:
Производитель: Seagate
Модель: ID-K-Some-4455-thing
Серийный номер: 887665
Вид накопителя: SSD
Тип таблицы разделов: GPT
Ёмкость накопителя: 16391
Свободное место: 16391
Количество разделов: 0
Суммарный объём разделов: 0
Наличие таблицы разделов: 1

Создадим разделы на весь объём накопителя:
Производитель: Seagate
Модель: ID-K-Some-4455-thing
Серийный номер: 887665
Вид накопителя: SSD
Тип таблицы разделов: GPT
Ёмкость накопителя: 16391
Свободное место: 0
Количество разделов: 6
Суммарный объём разделов: 16391
Наличие таблицы разделов: 1
ID: 5
    Объём: 8198
    Первый байт: 8193
    Последний байт: 16390
    Имя раздела:
    Файловая система:
    Занятое место: 0

ID: 4
    Объём: 4096
    Первый байт: 4097
    Последний байт: 8192
    Имя раздела:
    Файловая система:
    Занятое место: 0

ID: 3
    Объём: 2048
    Первый байт: 2049
    Последний байт: 4096
    Имя раздела: Volume4
    Файловая система:
    Занятое место: 0

ID: 2

Объём: 1024
Первый байт: 1025
Последний байт: 2048
Имя раздела: Volume3
Файловая система: NTFS
Занятое место: 0

ID: 1
Объём: 512
Первый байт: 513
Последний байт: 1024
Имя раздела: Volume2
Файловая система: FAT32
Занятое место: 0

ID: 0
Объём: 513
Первый байт: 0
Последний байт: 512
Имя раздела: Volume1
Файловая система: ext4
Занятое место: 0

Удалим раздел: с ID = 3
drive->DeleteVolume(3);
Результат: 1
Производитель: Seagate
Модель: ID-K-Some-4455-thing
Серийный номер: 887665
Вид накопителя: SSD
Тип таблицы разделов: GPT
Ёмкость накопителя: 16391
Свободное место: 2048
Количество разделов: 5
Суммарный объём разделов: 14343
Наличие таблицы разделов: 1
ID: 5
Объём: 8198
Первый байт: 8193
Последний байт: 16390
Имя раздела:
Файловая система:
Занятое место: 0

ID: 4
Объём: 4096
Первый байт: 4097
Последний байт: 8192
Имя раздела:
Файловая система:

Занятое место: 0

ID: 2
Объём: 1024
Первый байт: 1025
Последний байт: 2048
Имя раздела: Volume3
Файловая система: NTFS
Занятое место: 0

ID: 1
Объём: 512
Первый байт: 513
Последний байт: 1024
Имя раздела: Volume2
Файловая система: FAT32
Занятое место: 0

ID: 0
Объём: 513
Первый байт: 0
Последний байт: 512
Имя раздела: Volume1
Файловая система: ext4
Занятое место: 0

Попробуем добавить на его место не вмещающийся раздел:
drive->CreateVolume(0, 4097);
Исключение: Раздел с такими адресами не может быть расположен на накопителе.
ID: 5
Объём: 8198
Первый байт: 8193
Последний байт: 16390
Имя раздела:
Файловая система:
Занятое место: 0

ID: 4
Объём: 4096
Первый байт: 4097
Последний байт: 8192
Имя раздела:
Файловая система:
Занятое место: 0

ID: 2
Объём: 1024
Первый байт: 1025
Последний байт: 2048
Имя раздела: Volume3

Файловая система: NTFS
Занятое место: 0

ID: 1
Объём: 512
Первый байт: 513
Последний байт: 1024
Имя раздела: Volume2
Файловая система: FAT32
Занятое место: 0

ID: 0
Объём: 513
Первый байт: 0
Последний байт: 512
Имя раздела: Volume1
Файловая система: ext4
Занятое место: 0

Добавим точно вмещающийся раздел:
drive->CreateVolume(2566, 4000);
ID: 6
Объём: 1435
Первый байт: 2566
Последний байт: 4000
Имя раздела:
Файловая система:
Занятое место: 0

ID: 5
Объём: 8198
Первый байт: 8193
Последний байт: 16390
Имя раздела:
Файловая система:
Занятое место: 0

ID: 4
Объём: 4096
Первый байт: 4097
Последний байт: 8192
Имя раздела:
Файловая система:
Занятое место: 0

ID: 2
Объём: 1024
Первый байт: 1025
Последний байт: 2048
Имя раздела: Volume3

Файловая система: NTFS
Занятое место: 0

ID: 1
Объём: 512
Первый байт: 513
Последний байт: 1024
Имя раздела: Volume2
Файловая система: FAT32
Занятое место: 0

ID: 0
Объём: 513
Первый байт: 0
Последний байт: 512
Имя раздела: Volume1
Файловая система: ext4
Занятое место: 0

Заметим, что ID = 3 в списке уже нет, попробуем вновь удалить ID = 3
drive->DeleteVolume(3);
Результат: 0
ID: 6
Объём: 1435
Первый байт: 2566
Последний байт: 4000
Имя раздела:
Файловая система:
Занятое место: 0

ID: 5
Объём: 8198
Первый байт: 8193
Последний байт: 16390
Имя раздела:
Файловая система:
Занятое место: 0

ID: 4
Объём: 4096
Первый байт: 4097
Последний байт: 8192
Имя раздела:
Файловая система:
Занятое место: 0

ID: 2
Объём: 1024
Первый байт: 1025
Последний байт: 2048

Имя раздела: Volume3
Файловая система: NTFS
Занятое место: 0

ID: 1
Объём: 512
Первый байт: 513
Последний байт: 1024
Имя раздела: Volume2
Файловая система: FAT32
Занятое место: 0

ID: 0
Объём: 513
Первый байт: 0
Последний байт: 512
Имя раздела: Volume1
Файловая система: ext4
Занятое место: 0

С существующего раздела данные собрать удалось. Попробуем с несуществующего:
drive->GetVolumeFilesystem(3);
Исключение: unordered_map::at
Исключение не наше, контейнер справился сам.

Удалим разметку, попробуем создать раздел
drive->SetPTableType(Task::StorageDrive::PartitionTypes::PT_UNPARTITIONED);
drive->CreateVolume(0, 500);
Тип таблицы разделов: Таблица отсутствует
Ёмкость накопителя: 16391
Свободное место: 16391
Количество разделов: 0
Суммарный объём разделов: 0
Наличие таблицы разделов: 0
Исключение: Невозможно создать раздел на неразмеченном накопителе

Попробуем получить вектор ID разделов:
drive->GetVolumeIDs().empty() == 1

Вернём разметку, удостоверимся что ID остались:
drive->SetPTableType(Task::StorageDrive::PartitionTypes::PT_MBR);
drive->CreateVolume(0, 16000);
Тип таблицы разделов: MBR
Ёмкость накопителя: 16391
Свободное место: 390
Количество разделов: 1
Суммарный объём разделов: 16001
Наличие таблицы разделов: 1
ID: 7
Объём: 16001

Первый байт: 0
Последний байт: 16000
Имя раздела:
Файловая система:
Занятое место: 0

Разделы удалились, но отсчёт всё равно идёт с 7, как и должно быть.

В завершение тестирования, проверим работу установки значений для раздела:
Корректные:
drive->SetVolumeName(7, "valid");
drive->SetVolumeFSystem(7, "fsystem");
drive->SetVolumeUsedSpace(7, 16000);
ID: 7
Объём: 16001
Первый байт: 0
Последний байт: 16000
Имя раздела: valid
Файловая система: fsystem
Занятое место: 16000

Некорректные:
drive->SetVolumeName(7, "vali-d");
Исключение: Некорректное имя раздела.
drive->SetVolumeFSystem(7, "fsy`stem");
Исключение: Некорректный тип файловой системы.
drive->SetVolumeUsedSpace(7, 18000);
Исключение: Невозможно занять больше пространства, чем размер раздела.

Очистка занятого места при смене файловой системы:
ID: 7
Объём: 16001
Первый байт: 0
Последний байт: 16000
Имя раздела: valid
Файловая система: fsystem
Занятое место: 16000
drive->SetVolumeFSystem(7, "123");
ID: 7
Объём: 16001
Первый байт: 0
Последний байт: 16000
Имя раздела: valid
Файловая система: 123
Занятое место: 0