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 №5
ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ»
«Разработка простого класса»

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

Преподаватель:
Лауферман Ольга Викторовна

Новосибирск, 2025 г.

Оглавление

Постановка задачи

Реализуйте класс, моделирующий заданный тип объектов предметной области. Не должно быть возможности создать или перевести объект в некорректное состояние (например, человек с отрицательным возрастом). Хорошо подумайте над интерфейсом вашего класса. Необходимые поля и методы заданы, но вы можете добавить свои, если в этом есть необходимость. Описание класса должно быть разбито на два файла – заголовочный и реализации. Напишите программу, демонстрирующую работоспособность всех методов. Необходимо использовать функции и объекты стандартной библиотеки C++, а не C, где это возможно. Например, хранить строки в объекте класса std::string, а не в виде массива символов, использовать std::cin и std::cout для ввода и вывода соответственно.

Необходимо разработать класс НакопительДанных для информационной системы сервисного центра, оказывающего услуги по ремонту и диагностике накопителей данных. Количество и суммарный объём разделов могут быть равны 0 только одновременно. При создании новой таблицы разделов все существующие разделы удаляются. Если на накопителе нет таблицы разделов, то на нём нет и разделов.

Поля: производитель, модель, серийный номер, тип (жёсткий диск, твердотельный накопитель), объём, количество разделов, суммарный объём разделов, тип таблицы разделов (GPT, MBR, нет).

Методы: конструктор со всеми полями; возврат производителя; возврат модели; возврат серийного номера; возврат типа; возврат объёма; возврат типа; возврат количества разделов; возврат суммарного объёма разделов; установка количества и суммарного объёма разделов; возврат оставшегося свободного места; вернуть тип таблицы разделов; создать новую таблицу разделов; есть ли таблица разделов?.

Добавим методы для установки полей «Производитель», «Модель», «Серийный номер» и «Тип жёсткого диска» – в реалиях разработки рабочего ПО итоговый пользователь нисколько не застрахован от ошибок, а перевыделять память для всего класса - нерационально.

Также добавим метод для установки поля «Объём накопителя» на случай ошибки пользователя.
Отметим, что в данном методе также будет вызываться метод установки типа таблицы разделов на «Отсутствует», поскольку мы не можем гарантировать верность данных, введённых при создании экземпляра класса с неверным объёмом.

Для полей «Вид накопителя» и «Тип таблицы разметки» считаю разумным создать соответствующие перечисления, реализовать методы получения значения и его строковой интерпретации.

Поля «Количество разделов» и «Суммарный объём разделов» непосредственно влияют друг на друга, логично реализовать единый метод установки значения с необходимыми исключениями, он же вызывается для этих полей конструктором.

Для определения корректности ввода строковых полей («Производитель», «Модель», «Серийный номер» реализуем отдельный метод IsValidString(). Он не является логической частью класса, но используется только в нём, сделаем метод приватным.

Тестовые данные:

Проверка функционала конструктора со всеми полями. В конструкторе ни одно значение не устанавливается напрямую – тестовые данные подходят и проверяют также методы установки.

Название	Se@gate
Модель	ID-K-Some-4455-thing
Серийный номер	887665
Вид накопителя	Task::StorageDrive::DriveTypes::DT_HDD
Тип таблицы разделов	Task::StorageDrive::PartitionTypes::PT_GPT
Вместимость накопителя	3e12
Количество разделов	5
Суммарный объём разделов	2e10
Ожидаемый результат	Исключение: Поле "Производитель": некорректный ввод!

Название	Seagate
Модель	ID-\nK-Some-4455-thing
Серийный номер	887665
Вид накопителя	Task::StorageDrive::DriveTypes::DT_HDD
Тип таблицы разделов	Task::StorageDrive::PartitionTypes::PT_GPT
Вместимость накопителя	3e12
Количество разделов	5
Суммарный объём разделов	2e10
Ожидаемый результат	Исключение: Поле "Модель": некорректный ввод!

Название	Seagate
Модель	ID-K-Some-4455-thing
Серийный номер	8876@65
Вид накопителя	Task::StorageDrive::DriveTypes::DT_HDD
Тип таблицы разделов	Task::StorageDrive::PartitionTypes::PT_GPT
Вместимость накопителя	3e12
Количество разделов	5
Суммарный объём разделов	2e10
Ожидаемый результат	Исключение: Поле "Серийный номер": некорректный ввод!

Название	Seagate

Модель	ID-K-Some-4455-thing
Серийный номер	887665
Вид накопителя	Task::StorageDrive::DriveTypes::DT_HDD
Тип таблицы разделов	Task::StorageDrive::PartitionTypes::PT_GPT
Вместимость накопителя	0
Количество разделов	5
Суммарный объём разделов	2e10
Ожидаемый результат	Исключение: На накопителе нет места для разделов.

Название	Seagate
Модель	ID-K-Some-4455-thing
Серийный номер	887665
Вид накопителя	Task::StorageDrive::DriveTypes::DT_HDD
Тип таблицы разделов	Task::StorageDrive::PartitionTypes::PT_GPT
Вместимость накопителя	3e12
Количество разделов	0
Суммарный объём разделов	2e10
Ожидаемый результат	Исключение: Количество разделов и их суммарный объём могут быть 0 только одновременно!

Название	Seagate
Модель	ID-K-Some-4455-thing
Серийный номер	887665
Вид накопителя	Task::StorageDrive::DriveTypes::DT_HDD
Тип таблицы разделов	Task::StorageDrive::PartitionTypes::PT_GPT
Вместимость накопителя	1
Количество разделов	5
Суммарный объём разделов	2e10
Ожидаемый результат	Исключение: Суммарный объём разделов не может быть больше ёмкости накопителя!

Методы получения данных проверялись на следующем наборе полей:

Название	Seagate
Модель	ID-K-Some-4455-thing
Серийный номер	887665
Вид накопителя	Task::StorageDrive::DriveTypes::DT_SSD

Тип таблицы разделов	Task::StorageDrive::PartitionTypes::PT_GPT
Вместимость накопителя	3e12
Количество разделов	5
Суммарный объём разделов	2e10

Ожидаемые результаты:

Название	Seagate
Модель	ID-K-Some-4455-thing
Серийный номер	887665
Вид накопителя	SSD
Тип таблицы разделов	GPT
Вместимость накопителя	3000000000000
Количество разделов	5
Суммарный объём разделов	20000000000
Свободное место	2980000000000
Наличие таблицы разделов	1

Проектирование свойств класса

Принятые следующие решения касательно структуры класса:

Вид накопителя может принимать лишь два значения, запишем в перечисление:

```
enum class DriveTypes
{
   DT_SSD = 1 << 0,
   DT_HDD = 1 << 1,
};
```

И само поле в приватной части класса:

DriveTypes driveType;

Тип таблицы разделов может принимать лишь три значения, запишем в перечисление:

```
enum class PartitionTypes
{
   PT_UNPARTITIONED = 0,
   PT_MBR = 1 << 0,
   PT_GPT = 1 << 1,
};
```

И само поле в приватной части класса:

PartitionTypes partitionType;

Поля производителя, модели и серийного номера – переменные типа std::string, в приватной части класса:

std::string vendor;

std::string model;

std::string serial;

Вопрос проверки корректности будет затронут в методах установки данных значений.

Поля общей ёмкости накопителя, количества разделов и суммарного объёма разделов:

Unsigned – ни одно из этих значений никогда не может оказаться отрицательным.

Long long для storageSpace и storageAmount – данные поля отражают байтовые размеры, по этой причине необходимо получить максимально доступное значение.

Рассматривалось использование double для хранения данных значений, но было принято решение в пользу целочисленных типов, поскольку в рабочих задачах «заказчика» может потребоваться абсолютно точная информация, как об объёме занятого пространства, так и об общей ёмкости накопителя.

Количество разделов решено сократить до просто unsigned long – разработчику неизвестно о случаях, когда количество разделов на диске превышало 4.2 миллиарда.

Если бы контекст задачи предполагал указание не количества разделов и их объёма, а количества кластеров и размера одного кластера – вероятно, ситуация была бы обратной.

unsigned long long storageSpace;

unsigned long volumeAmount;

unsigned long long volumeSizeSum;

Разработка конструкторов и деструктора

Принято решение реализации четырёх конструкторов класса:

1) Конструктор по умолчанию, создающий референсный экземпляр класса.
2) Конструктор, задающий информационные поля класса (производитель, модель, серийный номер, вид накопителя). Данный конструктор может использоваться в случаях, когда оценить технические характеристики накопителя на месте невозможно, к примеру, в рамках ПО приёма на ремонт.
3) Конструктор, задающий технические поля класса (вид накопителя, тип таблицы разделов, общая ёмкость, количество разделов и их суммарный объём) – данный конструктор предполагается использовать в рамках задач

дистанционного ремонта (если планируется оказание таковых услуг) и/или при невозможности определить информационную часть данных – к примеру, отсутствие у накопителя приложения в виде наклейки или брошюры с описанием.

4) Конструктор со всеми полями для идеального случая, когда вся информация может быть получена на месте.

Ни один из конструкторов не устанавливает значения полей напрямую – вместо этого вызываются соответствующие методы класса, в них же учтены ограничения на диапазон и корректность данных.

В классе отсутствует дополнительное выделение памяти, собственная имплементация деструктора не применяется.

Проектирование методов класса

Список открытых методов класса представлен в следующем виде:

std::string GetVendor() const; – возвращает значение поля «производитель»

std::string GetModel() const; – возвращает значение поля «модель»

std::string GetSerialNumber() const; – возвращает значение поля «модель»

DriveTypes GetDriveType() const; – возвращает тип накопителя из перечисления, для использования во внутренней логике приложения.

std::string GetDriveTypeName() const; – возвращает тип накопителя в строковом представлении, для использования в интерфейсной части приложения.

PartitionTypes GetPTableType() const; – возвращает тип таблицы разделов из перечисления, для использования во внутренней логике приложения.

std::string GetPTableName() const; – возвращает тип таблицы разделов в строковом представлении, для использования в интерфейсной части приложения.

unsigned long GetVolumeAmount() const; – возвращает значение поля «Количество разделов»

unsigned long long GetVolumeSizeSum() const; – возвращает значение поля «Суммарный объём разделов»

unsigned long long GetStorageSpace() const; – возвращает значение поля «Ёмкость»

unsigned long long GetFreeSpace() const; – возвращает вычисленное значение свободного места на диске.

void SetVendor(std::string& newVendor); – устанавливает значение поля «производитель»

Проводит проверку входных данных при помощи IsValidString(), в случае некорректного ввода – вызывает исключение.

void SetModel(std::string& newModel); – устанавливает значение поля «модель»

Проводит проверку входных данных при помощи IsValidString(), в случае некорректного ввода – вызывает исключение.

void SetSerial(std::string& newSerial); – устанавливает значение поля «серийный номер»

Проводит проверку входных данных при помощи IsValidString(), в случае некорректного ввода – вызывает исключение.

void SetDriveType(DriveTypes newDriveType); – устанавливает значение поля «Вид накопителя». Проверки не проводятся, поскольку тип не предполагает некорректных значений.

void SetStorageSpace(unsigned long long newStorageSpace); – устанавливает значение поля «Ёмкость». Проверки не проводятся, изменение поля предполагается только в случае ошибки ввода ранее. Также вызывает SetPTableType(PT_UNPARTITIONED), таким образом, сбрасывает значения полей «таблица разделов», «количество разделов» и «суммарный объём разделов»

void SetVolumeData(unsigned long newAmount, unsigned long long newSum); – устанавливает значения полей «количество разделов» и «суммарный объём разделов».

Проводит проверки:

1) На отсутствие таблицы разделов (отсутствует и входные данные не 0 == исключение)

2) На «работоспособность» накопителя – используется соглашение, что если накопитель не определяется системой, его ёмкость равна нулю. Тогда, при попытке записать ненулевые входные данные – вызывается исключение.

3) Входные данные могут быть равны нулю только одновременно

4) Входное значение суммарного объёма не может быть меньше количества разделов.

5) Входное значение суммарного объёма не может превышать ёмкость накопителя.

void SetPTableType(PartitionTypes newPTableType); – устанавливает тип таблицы разделов.

Также устанавливает поля, касающиеся разделов диска в 0.

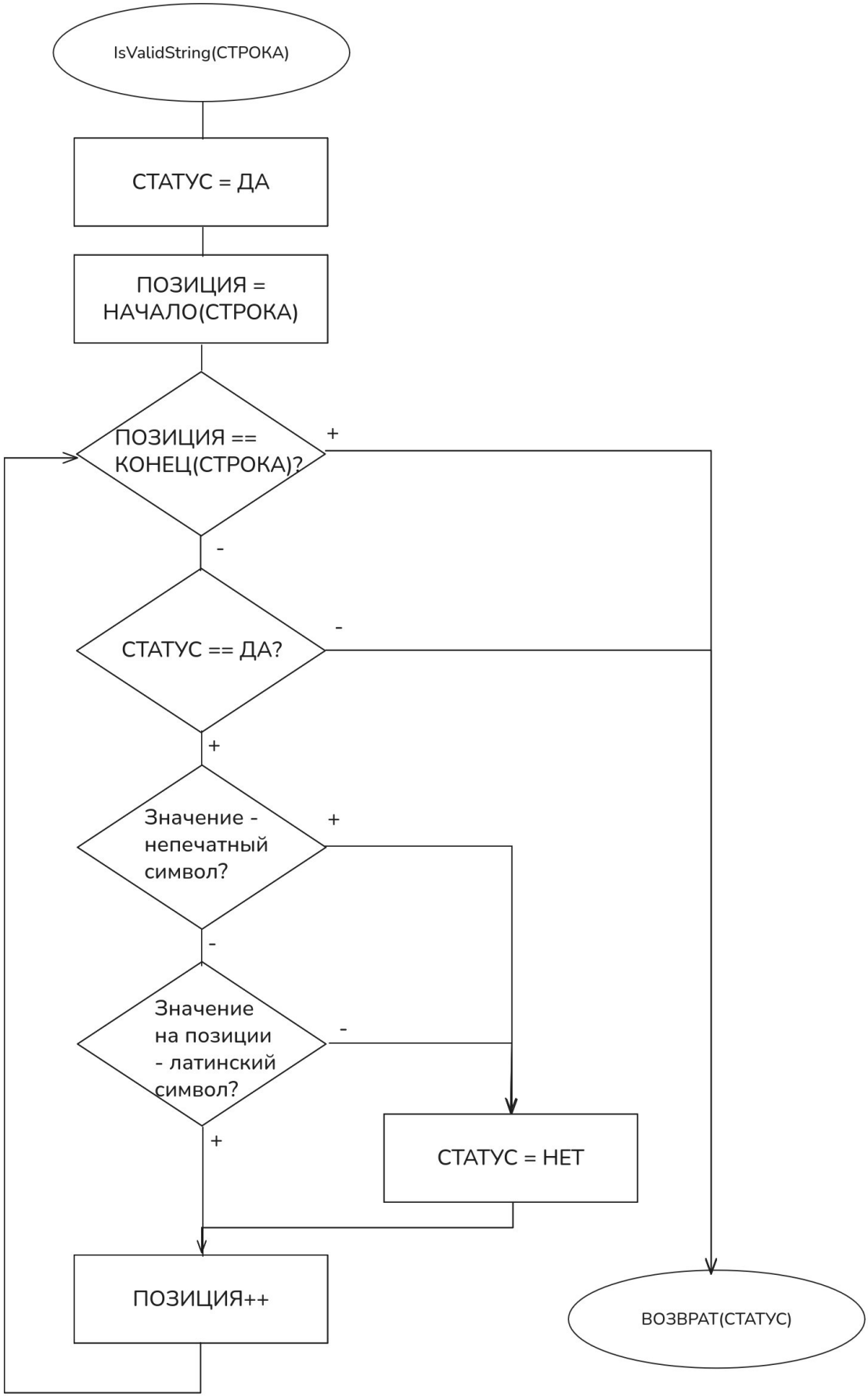
bool IsPartitioned() const; – возвращает true если тип таблицы разделов == «Отсутствует», иначе false.

В закрытых методах класса находится

bool IsValidString(std::string sample); – выполняет проверку корректности строки для использования в строковых полях. (Только символы из таблицы ASCII кроме непечатных и пробел)

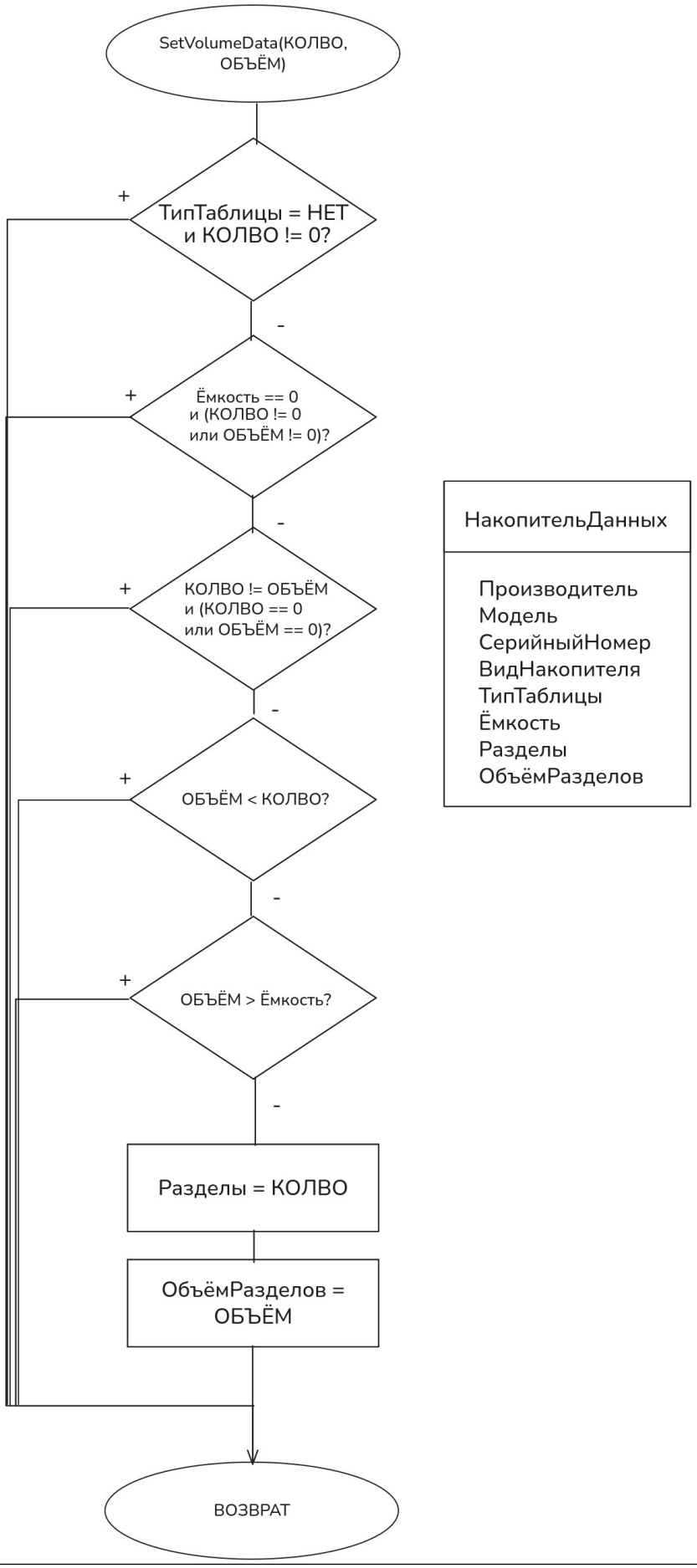
Блок-схемы

bool StorageDrive::IsValidString(std::string sample)



void StorageDrive::SetVolumeData(unsigned long newAmount, unsigned long long

newSum)


SetVolumeData(КОЛВО, ОБЪЁМ)
ТипТаблицы = НЕТ и КОЛВО != 0?
+
-
Ёмкость == 0 и (КОЛВО != 0 или ОБЪЁМ != 0)?
+
-
КОЛВО != ОБЪЁМ и (КОЛВО == 0 или ОБЪЁМ == 0)?
+
-
ОБЪЁМ < КОЛВО?
+
-
ОБЪЁМ > Ёмкость?
+
-
Разделы = КОЛВО
ОбъёмРазделов = ОБЪЁМ
ВОЗВРАТ
НакопительДанных
Производитель
Модель
СерийныйНомер
ВидНакопителя
ТипТаблицы
Ёмкость
Разделы
ОбъёмРазделов

Проектирование функции main() для тестирования класса:

В рамках main() добавим функции человекопонятного вывода информации, данная часть оставлена на реализацию пользовательского интерфейса и не является частью класса.

Будет проведено тестирование работы конструктора со всеми полями, как наиболее широко охватывающего функционал класса и наиболее часто используемого.

Будет проведена проверка корректности работы функций, устанавливающих технические характеристики экземпляра класса (SetVolumeData(), SetPTableType())

Проверка корректности работы функций, возвращающих те или иные значения экземпляра класса, будет проведена в ходе форматированного вывода значений на экран.

Тестирование программы

При запуске приложения с вышеописанным main() (код приведён в приложении), был получен следующий вывод в консоль:

```
Вызовы конструктора с некорректными данными:
Исключение: Поле "Производитель": некорректный ввод!
Исключение: Поле "Модель": некорректный ввод!
Исключение: На накопителе нет места для разделов.
Исключение: Количество разделов и их суммарный объём могут быть 0 только одновременно!
Исключение: Суммарный объём разделов не может быть больше ёмкости накопителя!

Вызов конструктора с корректными данными:
Получена информация:
Производитель: Seagate
Модель: ID-K-Some-4455-thing
Серийный номер: 887665
Вид накопителя: HDD
Тип таблицы разделов: GPT
Ёмкость накопителя: 3000000000000
Свободное место: 2980000000000
Количество разделов: 5
Суммарный объём разделов: 20000000000
Наличие таблицы разделов: 1

Удаление таблицы разделов:
Тип таблицы разделов: Таблица отсутствует
Ёмкость накопителя: 3000000000000
Свободное место: 3000000000000
Количество разделов: 0
Суммарный объём разделов: 0
Наличие таблицы разделов: 0

Попытка добавления раздела без таблицы:
Исключение: На неразмеченном накопителе не может быть разделов!
```

```
Тип таблицы разделов: Таблица отсутствует
Ёмкость накопителя: 3000000000000
Свободное место: 3000000000000
Количество разделов: 0
Суммарный объём разделов: 0
Наличие таблицы разделов: 0

Установка новой таблицы разделов:
Тип таблицы разделов: MBR
Ёмкость накопителя: 3000000000000
Свободное место: 3000000000000
Количество разделов: 0
Суммарный объём разделов: 0
Наличие таблицы разделов: 1

Попытки добавления некорректной информации о разделах:

0 разделов, объём = 3
Исключение: Количество разделов и их суммарный объём могут быть 0 только одновременно!

3 раздела, объём = 0
Исключение: Количество разделов и их суммарный объём могут быть 0 только одновременно!

Объём раздела больше ёмкости накопителя: (3e18 > 3e12)
Исключение: Суммарный объём разделов не может быть больше ёмкости накопителя!

Добавление корректной информации о разделах:
Тип таблицы разделов: MBR
Ёмкость накопителя: 3000000000000
Свободное место: 0
Количество разделов: 44
Суммарный объём разделов: 3000000000000
Наличие таблицы разделов: 1

Контроль полной информации о накопителе:
Производитель: Seagate
Модель: ID-K-Some-4455-thing
Серийный номер: 887665
Вид накопителя: HDD
Тип таблицы разделов: MBR
Ёмкость накопителя: 3000000000000
Свободное место: 0
Количество разделов: 44
Суммарный объём разделов: 3000000000000
Наличие таблицы разделов: 1
```

Тестирование показало ожидаемый результат, функционал программы признан удовлетворительным.

Выводы

В процессе выполнения данной лабораторной работы были получены навыки работы с классами C++, представление о работе конструкторов, деструкторов и итератора (в рамках работы со строковым типом), навыки работы с блоками try{}catch{}, обработки и вызова исключений.

Реализованный класс не является достоверной моделью, поскольку на практике ни ёмкость, ни объём раздела не являются просто «числом» – как правило, ёмкость накопителя неизвестна конечному пользователю, доступная ёмкость – определяется размером поля в таблице разделов,
а объём раздела кратен как размеру поля таблицы, так и размеру кластера в рамках формата файловой системы.

Для использования в реальных задачах данный класс нуждается в переработке, возможно, добавлении подкласса «Раздел» и контейнера экземпляров данного класса, а так же расширения информационной составляющей полей таблицы разделов.

Приложение. Исходные коды всех разработанных файлов.

StorageDrive.hpp:

```
#ifndef STORAGE_DRIVE_HPP
#define STORAGE_DRIVE_HPP

#include <string>

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
            std::string& Vendor, //производитель
            std::string& Model, //модель
            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
            unsigned long VolumeAmount, //количество разделов
            unsigned long long VolumeSizeSum //суммарный объём разделов
         );

         StorageDrive( //полный конструктор
            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
            unsigned long VolumeAmount,//количество разделов
            unsigned long long VolumeSizeSum//суммарный объём разделов
         );
```

```
        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 GetVolumeAmount() const;//получить значение поля "количество разделов"
        unsigned long long GetVolumeSizeSum() const;//получить значение поля "суммарный объём
разделов"


        unsigned long long GetStorageSpace() const;//получить значение поля "ёмкость"
        unsigned long long GetFreeSpace() const;//получить информацию о незанятом месте на накопителе

        void SetVendor(std::string& newVendor); //установить значение поля "производитель"
        void SetModel(std::string& newModel); //установить значение поля "модель"
        void SetSerial(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
        void SetVolumeData(unsigned long newAmount, unsigned long long newSum); //установить значения
полей "количество разделов" и "суммарный объём разделов"
        void SetPTableType(PartitionTypes newPTableType); //установить значение поля "тип таблицы
разделов"

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

    private:
        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
        unsigned long long volumeAmount; //количество разделов
        unsigned long long volumeSizeSum; //суммарный объём разделов

        bool IsValidString(std::string sample); //служебная функция проверки корректности входной строки
  };
}
#endif
```

StorageDrive.cpp:

```
#include <stdexcept>
#include <string>

#include "StorageDrive.hpp"

namespace Task
```

```
{
    StorageDrive::StorageDrive()
    {
        //в данном случае строки создаются в конструкторе,
        //поскольку в методах предполагается ссылочный тип(для экономии памяти)
        //а значит, мы не можем ссылаться на "входные" данные - это не lvalue
        //и они исчезнут как только будут получены
        std::string _vendor("Unknown Vendor");
        std::string _model("Generic Storage Drive");
        std::string _serial("Unknown");

        SetVendor(_vendor);
        SetModel(_model);
        SetSerial(_serial);
        SetDriveType(DriveTypes::DT_HDD);
        SetStorageSpace(0);
        SetPTableType(PartitionTypes::PT_UNPARTITIONED);
        SetVolumeData(0, 0);
    }

    StorageDrive::StorageDrive(
        std::string& Vendor,
        std::string& Model,
        std::string& Serial,
        DriveTypes DriveType
    )
    {
        SetVendor(Vendor);
        SetModel(Model);
        SetSerial(Serial);
        SetDriveType(DriveType);
        SetStorageSpace(0);
        SetPTableType(PartitionTypes::PT_UNPARTITIONED);
        SetVolumeData(0, 0);
    }

    StorageDrive::StorageDrive(
        DriveTypes DriveType,
        PartitionTypes PartitionType,
        unsigned long long StorageSpace,
        unsigned long VolumeAmount,
        unsigned long long VolumeSizeSum
    )
    {
        //в данном случае строки создаются в конструкторе,
        //поскольку в методах предполагается ссылочный тип(для экономии памяти)
        //а значит, мы не можем ссылаться на "входные" данные - это не lvalue
        //и они исчезнут как только будут получены
        std::string _vendor("Unknown Vendor");
        std::string _model("Generic Storage Drive");
        std::string _serial("Unknown");
```

```
    SetVendor(_vendor);
    SetModel(_model);
    SetSerial(_serial);
    SetDriveType(DriveType);
    SetStorageSpace(StorageSpace);
    SetPTableType(PartitionType);
    SetVolumeData(VolumeAmount, VolumeSizeSum);
}

StorageDrive::StorageDrive(
    std::string& Vendor,
    std::string& Model,
    std::string& Serial,
    DriveTypes DriveType,
    PartitionTypes PartitionType,
    unsigned long long StorageSpace,
    unsigned long VolumeAmount,
    unsigned long long VolumeSizeSum
)
{
    SetVendor(Vendor);
    SetModel(Model);
    SetSerial(Serial);
    SetDriveType(DriveType);
    SetStorageSpace(StorageSpace);
    SetPTableType(PartitionType);
    SetVolumeData(VolumeAmount, VolumeSizeSum);
}

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 StorageDrive::GetVolumeAmount() const
{
    return volumeAmount;
}

unsigned long long StorageDrive::GetVolumeSizeSum() const
{
    return volumeSizeSum;
}

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

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

    return storageSpace-GetVolumeSizeSum();
}

void StorageDrive::SetVendor(std::string& newVendor)
{
    if (!IsValidString(newVendor))
    {
        throw std::invalid_argument("Поле \"Производитель\": некорректный ввод!");
    };

    vendor = newVendor;
}
```

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

    model = newModel;
}


void StorageDrive::SetSerial(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);
}


void StorageDrive::SetVolumeData(unsigned long newAmount, unsigned long long newSum)
{
    //если нет таблицы, но значение ненулевое
    if (!IsPartitioned() && (newAmount != 0 || newSum!= 0))
    {
        throw std::invalid_argument("На неразмеченном накопителе не может быть разделов!");
    }
    //если ёмкость диска не определилась, а значение ненулевое
    if (storageSpace == 0 && (newAmount != 0 || newSum != 0))
    {
        throw std::invalid_argument("На накопителе нет места для разделов.");
    }
    //если только одно из значений ненулевое
    if ((newAmount && !newSum) || (newSum && !newAmount))
    {
        throw std::invalid_argument("Количество разделов и их суммарный объём могут быть 0 только
одновременно!");
    }
```

```
        //если объём меньше количества разделов
        if (newSum < newAmount)
        {
            throw std::invalid_argument("Суммарный объём разделов не может быть меньше их количества!");
        }
        //если объём больше ёмкости
        if (newSum > storageSpace)
        {
            throw std::invalid_argument("Суммарный объём разделов не может быть больше ёмкости
накопителя!");
        }

        volumeAmount = newAmount;
        volumeSizeSum = newSum;
    }

    void StorageDrive::SetPTableType(PartitionTypes newPTableType)
    {
        //очистить записи о разделах
        SetVolumeData(0, 0);
        partitionType = newPTableType;
    }

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

    bool StorageDrive::IsValidString(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;
    }

}
```

Main.cpp:

```
#include "StorageDrive.hpp"
#include <exception>
#include <iostream>

void PrintStorageInfo(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 PrintFullInfo(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;
    PrintStorageInfo(drive);
}

int main(void)
{
    Task::StorageDrive* drive;
    std::string name("Se@gate");
    std::string model("ID-K-Some-4455-thing");
    std::string serial("887665");

    std::cout << "\nВызовы конструктора с некорректными данными:" << std::endl;

    try
    {
        drive = new Task::StorageDrive(
            name,
            model,
            serial,
            Task::StorageDrive::DriveTypes::DT_HDD,
            Task::StorageDrive::PartitionTypes::PT_GPT,
            3e12,
            5,
            2e10
        );
    }
    catch (const std::exception &ex)
    {
        std::cout << "Исключение: " << ex.what() << std::endl;
        drive = 0;
    }

    name = "Seagate";
    model = "ID-\nK-Some-4455-thing";
    try
    {
```

```
    drive = new Task::StorageDrive(
        name,
        model,
        serial,
        Task::StorageDrive::DriveTypes::DT_HDD,
        Task::StorageDrive::PartitionTypes::PT_GPT,
        3e12,
        5,
        2e10
    );
}
catch (const std::exception &ex)
{
    std::cout << "Исключение: " << ex.what() << std::endl;
    drive = 0;
}
model = "ID-K-Some-4455-thing";
serial = "8876 65";

try
{
    drive = new Task::StorageDrive(
        name,
        model,
        serial,
        Task::StorageDrive::DriveTypes::DT_HDD,
        Task::StorageDrive::PartitionTypes::PT_GPT,
        3e12,
        5,
        2e10
    );
}
catch (const std::exception &ex)
{
    std::cout << "Исключение: " << ex.what() << std::endl;
    drive = 0;
}
serial = "887665";

try
{
    drive = new Task::StorageDrive(
        name,
        model,
        serial,
        Task::StorageDrive::DriveTypes::DT_HDD,
        Task::StorageDrive::PartitionTypes::PT_GPT,
        0,
        5,
        2e10
    );
}
```

```
catch (const std::exception &ex)
{
    std::cout << "Исключение: " << ex.what() << std::endl;
    drive = 0;
}
try
{
    drive = new Task::StorageDrive(
        name,
        model,
        serial,
        Task::StorageDrive::DriveTypes::DT_HDD,
        Task::StorageDrive::PartitionTypes::PT_GPT,
        3e12,
        0,
        2e10
    );
}
catch (const std::exception &ex)
{
    std::cout << "Исключение: " << ex.what() << std::endl;
    drive = 0;
}
try
{
    drive = new Task::StorageDrive(
        name,
        model,
        serial,
        Task::StorageDrive::DriveTypes::DT_HDD,
        Task::StorageDrive::PartitionTypes::PT_GPT,
        1,
        5,
        2e10
    );
}
catch (const std::exception &ex)
{
    std::cout << "Исключение: " << ex.what() << std::endl;
    drive = 0;
}

std::cout << "\nВызов конструктора с корректными данными:" << std::endl;
drive = new Task::StorageDrive(
    name,
    model,
    serial,
    Task::StorageDrive::DriveTypes::DT_HDD,
    Task::StorageDrive::PartitionTypes::PT_GPT,
    3e12,
    5,
    2e10
```

```
);
std::cout << "Получена информация:" << std::endl;
PrintFullInfo(drive);

std::cout << "\nУдаление таблицы разделов: " << std::endl;
drive->SetPTableType(Task::StorageDrive::PartitionTypes::PT_UNPARTITIONED);
PrintStorageInfo(drive);

std::cout << "\nПопытка добавления раздела без таблицы:" << std::endl;
try
{
   drive->SetVolumeData(3, 330);
}
catch (const std::exception &ex)
{
   std::cout << "Исключение: " << ex.what() << std::endl;
}
PrintStorageInfo(drive);

std::cout << "\nУстановка новой таблицы разделов:" << std::endl;
drive->SetPTableType(Task::StorageDrive::PartitionTypes::PT_MBR);
PrintStorageInfo(drive);

std::cout << "\nПопытки добавления некорректной информации о разделах:" << std::endl;
std::cout << "\n0 разделов, объём = 3" << std::endl;
try
{
   drive->SetVolumeData(0, 3);
}
catch (const std::exception &ex)
{
   std::cout << "Исключение: " << ex.what() << std::endl;
}
std::cout << "\n3 раздела, объём = 0" << std::endl;
try
{
   drive->SetVolumeData(3, 0);
}
catch (const std::exception &ex)
{
   std::cout << "Исключение: " << ex.what() << std::endl;
}
std::cout << "\nОбъём раздела больше ёмкости накопителя: (3e18 > 3e12)" << std::endl;
try
{
   drive->SetVolumeData(1, 3e18);
}
catch (const std::exception &ex)
{
   std::cout << "Исключение: " << ex.what() << std::endl;
}
```

```
    std::cout << "\nДобавление корректной информации о разделах:" << std::endl;
    drive->SetVolumeData(44, drive->GetStorageSpace());
    PrintStorageInfo(drive);

    std::cout << "\nКонтроль полной информации о накопителе:" << std::endl;
    PrintFullInfo(drive);

    delete(drive);
    return 0;
}
```