Лабораторная работа №3. Проектирование собственных классов

Использование объектно-ориентированного подхода при разработке программ. Разработка программ через тестирование.

Анализ предметной области решаемой задачи

Рассмотрим задачу: имеется стакан определенной емкости. Имеется определенное количество жидкости заданного объема. В стакан можно налить некоторое количество жидкости, а также вылить из него некоторое количество жидкости. Очевидно, что в стакан нельзя поместить жидкости больше, чем в нем есть места, также из стакана нельзя вылить жидкости больше, чем имеется в наличии.

Разработать программу, позволяющую пользователю вводить заданные команды с клавиатуры, при помощи которых пользователь может наливать в стакан жидкость и выливать ее из стакана, а также узнать количество жидкости, фактически находящейся в стакане и фактически вылитое из него.

Для решения данной задачи можно было бы воспользоваться обычными целочисленными переменными, хранящими емкость стакана и объем находящейся в нем воды, а также разработать несколько функций для выполнения основных операций над ними.

Одним из очевидных недостатков такого решения программы будет следующий - потребуется кардинальная переделка программы в случае, если задача будет иметь дело не с одним стаканом, а с произвольным их количеством. В этом случае ранее написанный код окажется попросту бесполезным.

Улучшить ситуацию можно было бы, объявив структуру **Glass**, объединяющую данные о емкости стакана и объеме содержащейся в нем жидкости, а также функции, осуществляющие операции над данной структурой.

```
struct Glass
{
    int capacity;
    int amountOfWater;
};
void CreateEmptyGlass(Glass & glass, int capacity);
int AddWater(Glass & glass, int amount);
int PourWater(Glass & glass, int amount);
```

Тем не менее, у пользователя данных классов по-прежнему сохраняется прямой доступ к полям структуры Glass в обход имеющихся функций, что позволяет умышленно или случайно задать им недопустимые значения, например, отрицательную вместимость или количество воды, превышающее вместимость стакана.

Еще одна проблема, связанная с возможностью прямого доступа к полям структур, - это привязка внешнего кода к деталям ее реализации, что существенно ограничивает свободу по их расширению. В нашем случае наличие поля **amountOfWater** в структуре Glass выйдет нам боком, если в дальнейшем понадобиться «наливать» в стакан помимо воды еще и масло. В этом случае нам придется переписать весь клиентский код, который для определения объема жидкости в стакане использует значение поля **amountOfWater**, т.к. в случае, когда стакан содержит и масло и воду, объем жидкости в нем будет складываться из объема воды и масла.

Использование классов для моделирования сущностей предметной области

Классы – как раз тот механизм объектно-ориентированных языков программирования, позволяющий не только представить сущности предметной области на нужном уровне абстракции, но и ограничить набор действий которые можно совершать над ними, а также скрыть детали их внутренней реализации от доступа извне.

Объявление класса в языке Си++ во многом схоже с объявлением структуры. Класс объявляется при помощи ключевого слова **class**. В нашем случае объявление класса CGlass может быть сделано следующим образом:

```
class CGlass
{
    // поля класса
};
```

Поля класса – это данные и методы, которыми будут обладать все **объекты**, или как их еще называют – **экземпляры**, данного класса.

> Проектирование класса следует начинать с его интерфейсной части, т.е. того, как будет представлен класс внешнему миру, нежели с его внутреннего устройства. Иными словами, интерфейс класса должен определять внутреннее устройство класса, а не наоборот. В противном случае велик риск получения в итоге класса, совершенно непригодного или неудобного для использования.

Проектирование интерфейсной части класса

Выбор необходимого уровня абстракции

Для начала попробуем абсолютно забыть о деталях внутренней реализации класса и полностью сосредоточиться на необходимом уровне абстракции сущности «стакан» **в рамках решаемой задачи**.

- Стакан с момента своего создания обладает некоторой **вместимостью**, которая остается неизменной на всем протяжении жизни стакана. Различные стаканы (различные экземпляры класса «Стакан») могут иметь различную вместимость
- **Количество жидкости** в стакане изначально равно нулю
- В стакан можно **добавить некоторое** неотрицательное **количество воды**. При этом суммарное количество жидкости в стакане не может превышать его вместимость
- Из стакана можно **вылить некоторое** неотрицательное **количество воды**, суммарно не превышающее количество воды, находившееся в стакане.

Итак, после этого небольшого осмысления предметной области мы обнаружили, что наш стакан обладает следующими **свойствами**, определяющими его состояние:

- Вместимость стакана
- Количество жидкости в стакане

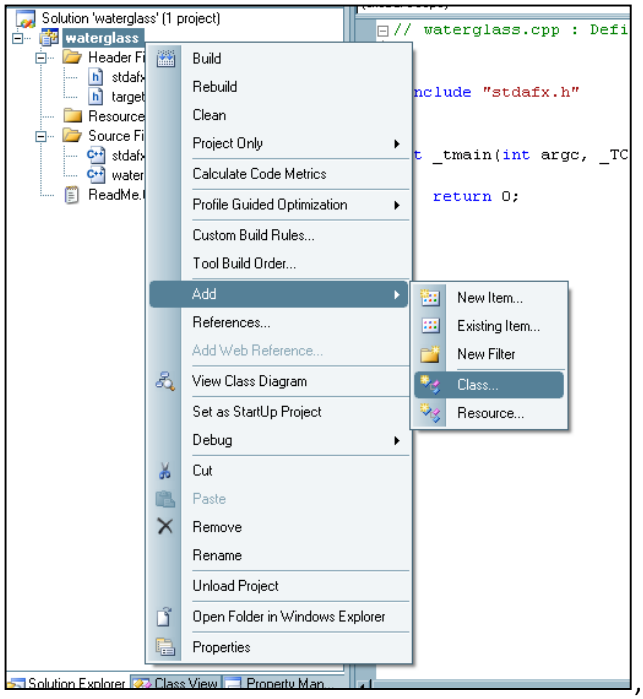
Помимо свойств, над стаканом можно осуществить ряд **действий**:

- Добавить воду
- Вылить воду

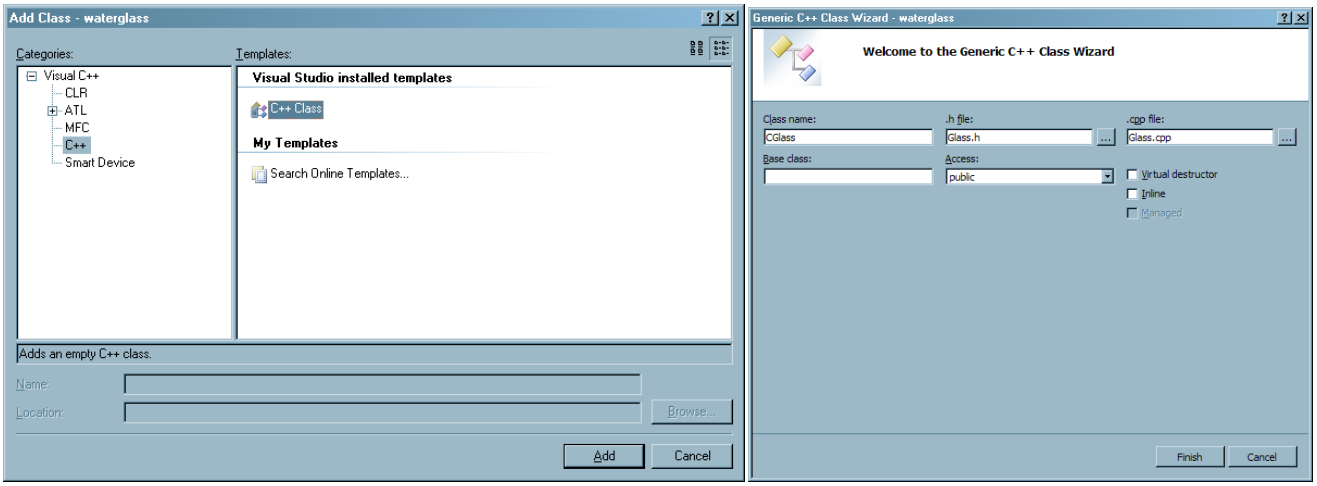
Действия, совершаемые над объектом, называются **методами**. Их вызов может приводить, а может и не приводить к изменению состояния объекта. Например, действия по выливанию и добавлению воды в стакан приводят к изменению количества жидкости в стакане, изменяя его состояние.

Вспомогательные средства Microsoft Visual Studio для добавления класса

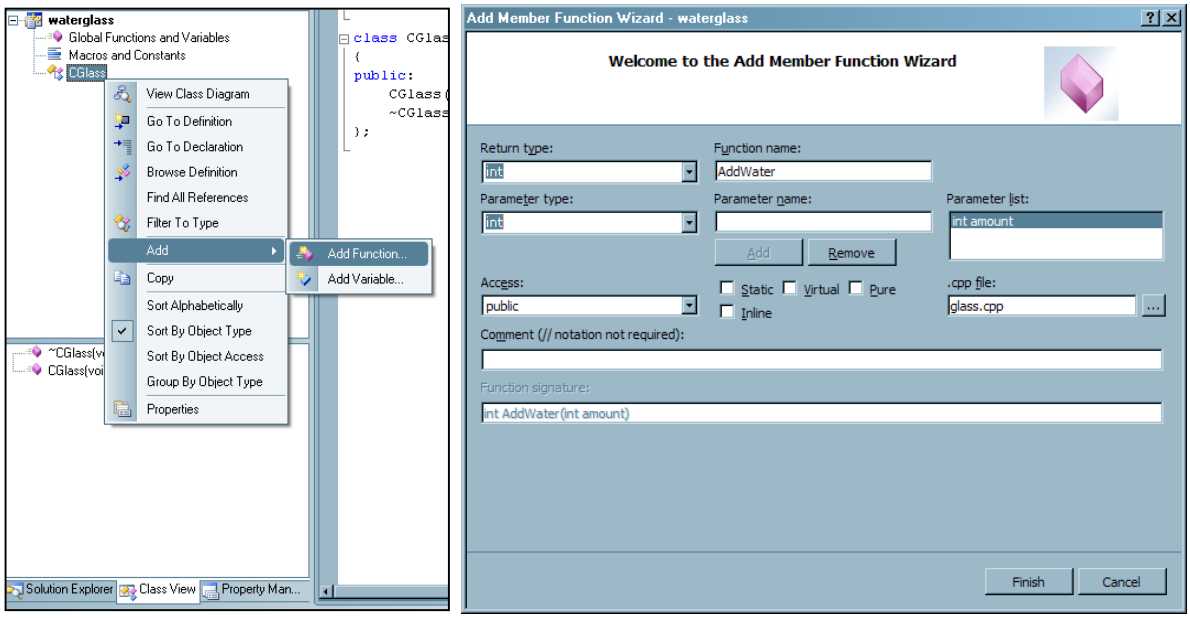
Для добавления класса в среде Microsoft Visual Studio можно воспользоваться встроенным мастером добавления нового класса. При помощи контекстного меню Solution Explorer-а выберите пункт Add→Class:



в открывшемся окне выберите C++ class и укажите имя класса и имена заголовочного файла и файла с реализацией класса:



С помощью контекстного меню панели Class View можно добавить необходимые методы к выбранному классу:



Разумеется, проделать вышеупомянутую работу можно и вручную путем создания и добавления файлов исходного кода к проекту и ручного их редактирования.

Запись интерфейсной части класса

Согласно условиям задания определение класса CGlass будет выглядеть следующим образом:

```
class CGlass
{
public:
    // создаем стакан заданной вместимости
    CGlass(int capacity);

    ~CGlass();

    // возвращаем вместимость стакана
    int GetCapacity()const;

    // возвращаем количество воды в стакане
    int GetWaterAmount()const;

    // добавляем воду в стакан, возвращая фактически добавленное количество
    int AddWater(int amount);

    // выливаем воду из стакана, возвращаем фактически вылитое количество
    int PourWater(int amount);
};
```

Упомянутые методы класса, а также его конструктор и деструктор, являются публичными и формируют его **интерфейсную часть**.

> Отметим, что методы GetCapacity() и GetWaterAmount() возвращают информацию о состоянии стакана, **не изменяя его**, поэтому они должны быть константными. В противном случае, мы не сможем их вызвать для константных экземпляров класса CGlass, а также экземпляров, доступных нам по константной ссылке или указателю на константные данные.

Разработка через тестирование

На операции добавления и выливания воды из стакана накладывается ряд ограничений, которые необходимо учитывать при реализации класса. При этом программист не застрахован от внесения новых ошибок в ходе исправления старых, а также при дальнейшем развитии класса. Многократное **ручное** тестирование всех методов класса после вносимых в него изменений может отнимать много времени и часто избегается программистами. Это, в свою очередь, может привести к тому, что случайно внесенные ошибки не будут своевременно замечены и устранены. Поэтому часто применяется автоматическое тестирование программ, основанное на разработке вспомогательного тестирующего кода, выполняющего проверку соответствия поведения классов и функций их спецификациям.

> Достаточно эффективной техникой программирования является **разработка тестов для класса до его реализации** или параллельно с нею, управляя при этом его разработкой. Данный подход к разработке кода называется «**Test-driven development**» (разработка через тестирование, **TDD**) и способствует разработке надежного и устойчивого к сбоям кода.
>
> Кроме того, если класс разрабатывается с учетом его тестирования отдельно от других классов программы, то он получается легче в использовании и дальнейшем сопровождении, т.к. содержит меньше зависимостей от внешнего кода - чем больше зависимостей в классе от внешних условий, тем сложнее написать для него тесты.
>
> Разработка же тестов для уже написанного класса обладает меньшей эффективностью по целому ряду причин.

Во-первых, такие тесты составляются менее тщательно, т.к., по мнению программиста, «цель уже достигнута» и такая работа кажется обузой. При TDD, напротив, составление тестов является естественной составляющей процесса разработки.

Во-вторых, когда класс уже разработан, разработка тестов для него затрудняется его зависимостями от других классов, которыми класс успел обрасти в процессе разработки. Устранение таких зависимостей, необходимое для нормального тестирования класса, зачастую связано с необходимостью изменения архитектуры класса (**рефакторинга**), в процессе которого существует вероятность сломать уже «работающий» по мнению программиста код при отсутствии средств автоматического контроля его работы. Разработка через тестирование, напротив, поощряет внесение изменений в код, поскольку качественно написанные тесты значительно повышают вероятность обнаружения ошибок в процессе рефакторинга.

В конце концов, программист может вообще прийти к мысли, что разработка тестов для его кода – не барское занятие, что код он уже 100 раз перепроверил, и ошибок в нем нет. Тем не менее, даже если ошибок сегодня в коде действительно нет (что случается очень редко), существует риск внести ошибку в дальнейшем, например, при наращивании функциональности класса. При TDD ошибка при модификации кода может быть обнаружена с большей вероятностью.

Для проверки работоспособности методов класса **еще до их полноценной реализации** напишем несколько тестов в основном файле программы:

```
#include "stdafx.h"
#include "Glass.h"
#include <assert.h>

using namespace std;

void TestGlassCreation()
{
    // проверяем свойства созданного стакана
    CGlass validGlass(10);

    assert(validGlass.GetCapacity() == 10);
    assert(validGlass.GetWaterAmount() == 0);

    // пытаемся создать скакан отрицательной вместимости
    CGlass invalidGlass(-10);

    assert(invalidGlass.GetCapacity() == 0);
    assert(invalidGlass.GetWaterAmount() == 0);

    cout << "Glass creation test is ok\n";
}

void TestAddingWater()
{
    // создаем стакан вместимостью 100 мл.
    CGlass someGlass(100);

    // добавляем в него воды по 1 мл, убеждаясь в невозможности переполнения
    for (int i = 1; i < 200; ++i)
    {
        const int addedWater = someGlass.AddWater(1);
        if (i <= someGlass.GetCapacity())
        {
            assert(addedWater == 1);
            assert(someGlass.GetWaterAmount() == i);
        }
        else
```

```
        {
            assert(addedWater == 0);
            assert(someGlass.GetWaterAmount() == someGlass.GetCapacity());
        }
    }

    CGlass anotherGlass(100);
    anotherGlass.AddWater(10);
    assert(anotherGlass.GetWaterAmount() == 10);

    // пытаемся добавить отрицательное количество воды
    const int addedWater = anotherGlass.AddWater(-10);
    assert(addedWater == 0);
    assert(anotherGlass.GetWaterAmount() == 10);

    cout << "Adding water test is ok\n";
}

void TestPouringWater()
{
    // создаем стакан
    CGlass glass(100);

    // наполняем его доверху
    glass.AddWater(glass.GetCapacity());
    assert(glass.GetWaterAmount() == glass.GetCapacity());

    // выливаем воду по 1 мл, убеждаясь в невозможности вылить больше, чем есть воды
    for (int i = glass.GetWaterAmount() - 1; i >= -100; --i)
    {
        const int pouredWater = glass.PourWater(1);

        if (i >= 0)
        {
            assert(pouredWater == 1);
            assert(glass.GetWaterAmount() == i);
        }
        else
        {
            assert(pouredWater == 0);
            assert(glass.GetWaterAmount() == 0);
        }
    }

    // создаем другой стакан
    CGlass anotherGlass(100);
    anotherGlass.AddWater(10);
    assert(anotherGlass.GetWaterAmount() == 10);

    // пытаемся добавить отрицательное количество воды
    const int pouredWater = anotherGlass.PourWater(-10);
    assert(pouredWater == 0);
    assert(anotherGlass.GetWaterAmount() == 10);

    cout << "Pouring water test is ok\n";
}

void TestIntegerOverflow()
{
    CGlass glass(100);
    glass.AddWater(10);
    assert(glass.GetWaterAmount() == 10);
    // пытаемся вызвать целочисленное переполнение
    const int addedWater = glass.AddWater(INT_MAX);
    assert(addedWater == glass.GetCapacity() - 10);
    assert(glass.GetWaterAmount() == glass.GetCapacity());
```

```
    CGlass anotherGlass(100);
    // пытаемся вызвать целочисленное переполнение
    const int pouredWater = anotherGlass.PourWater(INT_MAX);
    assert(pouredWater == 0);
    assert(anotherGlass.GetWaterAmount() == 0);

    cout << "Integer overflow test is ok\n";
}

int main(int argc, char* argv[])
{
    TestGlassCreation();
    TestAddingWater();
    TestPouringWater();
    TestIntegerOverflow();

    return 0;
}
```

Для того чтобы тесты проходили компиляцию, необходимо в теле методов класса и конструкторах поместить заглушки.

Теперь, когда с интерфейсной частью класса мы определились, пришло время определиться с его реализацией.

Для хранения информации о вместимости стакана и количестве находящейся в нем воды нам понадобятся соответствующие целочисленные переменные. Для сокрытия подробностей реализации класса разместим члены-данные класса в закрытой области объявления класса:

```
class CGlass
{
public:
    // создаем стакан заданной вместимости
    CGlass(int capacity);

    ~CGlass();

    // возвращаем вместимость стакана
    int GetCapacity()const;

    // возвращаем количество воды в стакане
    int GetWaterAmount()const;

    // добавляем воду в стакан, возвращая фактически добавленное количество
    int AddWater(int amount);

    // выливаем воду из стакана, возвращаем фактически вылитое количество
    int PourWater(int amount);
private:
    int     m_capacity;       // вместимость
    int     m_waterAmount;    // количество воды
};
```

Для отличия имен переменных-членов класса от локальных переменных и параметров его методов мы используем префикс **m_** для их именования.

В файле Glass.cpp напишем реализацию методов и конструктора. Их разработка во многом облегчилась благодаря предварительно написанным тестам.

```
#include "StdAfx.h"
#include "Glass.h"

// создаем пустой стакан заданной неотрицательной вместимости
```

```
CGlass::CGlass(int capacity)
:m_capacity((capacity >= 0) ? capacity : 0)
,m_waterAmount(0)
{
}

CGlass::~CGlass()
{
}

// возвращаем вместимость стакана
int CGlass::GetCapacity()const
{
    return m_capacity;
}

// возвращаем количество воды в стакане
int CGlass::GetWaterAmount()const
{
    return m_waterAmount;
}

// добавляем воду в стакан, возвращая фактически добавленное количество
int CGlass::AddWater(int amount)
{
    if (amount < 0)
    {
        amount = 0;
    }
    // проверка на целочисленное переполнение при сложении и на переполнение стакана
    else if (
        (m_waterAmount + amount < m_waterAmount) ||
        (m_waterAmount + amount > m_capacity))
    {
        amount = m_capacity - m_waterAmount;
    }

    m_waterAmount += amount;
    return amount;
}

// выливаем воду из стакана, возвращаем фактически вылитое количество
int CGlass::PourWater(int amount)
{
    // нельзя вылить больше, чем имеется в наличии, а также отрицательное количество
    amount =
        (amount < 0) ? 0 :
        (m_waterAmount >= amount) ? amount : m_waterAmount;

    m_waterAmount -= amount;
    return amount;
}
```

Резюме

Вы получили основное представление об объектно-ориентированном подходе к моделированию сущностей предметной области, о том, как создавать собственные классы на языке C++, а также вести разработку с использованием подхода «Test driven development».

Практические задания

На оценку «**удовлетворительно**» необходимо выполнить хотя бы часть **обязательных заданий** и набрать **не менее 40 баллов**.

На оценку «**хорошо**» необходимо выполнить все обязательные и часть дополнительных заданий и набрать **не менее 250 баллов**.

На оценку «**отлично**» необходимо выполнить все обязательные и часть дополнительных заданий и набрать **не менее 500 баллов**.

Внимание, дополнительные задания принимаются только после успешной защиты обязательных заданий.

Обязательные задания

Задание 1

Выполните один из предложенных вариантов задания.

Вариант 1 – 50 баллов

Спроектируйте **с использованием TDD** класс CTVSet , моделирующий телевизор, который может находиться либо в выключенном, либо включенном состоянии. Находясь во включенном состоянии, телевизор способен отображать один из 99 каналов (от 1 до 99).

Над телевизором можно выполнять следующие действия:

- Выключить. В выключенном состоянии нельзя переключать каналы.
- Включить. При своем включении телевизор включается на том канале, на котором он был ранее выключен. При самом первом включении телевизор включается на первом канале.
- Выбрать заданный канал (от 1 до 99) или остаться на том же самом канале, если номер канала за пределами данного диапазона.

Телевизор обладает следующими свойствами:

- Включен или выключен
- Номер текущего канала. В выключенном состоянии номер текущего канала должен быть равен нулю.

Разработайте программу, использующую разработанный Вами класс CTVSet, которая обрабатывает команды пользователя, вводимые им со стандартного потока ввода:

- **TurnOn**. Включает телевизор, если он был выключен
- **TurnOff**. Выключает телевизор, если он был включент
- **SelectChannel** <*номер канала*>. Выбирает указанный номер канала.
- **Info**. Выводит текущее состояние телевизора (выключен или включен, номер канала).

Вариант 2 – 140 баллов

Разработать **с использованием TDD** класс CCar, моделирующий автомобиль в следующей предметной области.

Двигатель автомобиля может находиться как во включенном состоянии, так и в выключенном.

В автомобиле может быть включена одна из следующих передач:

- Задний ход (-1)

- Нейтральная передача (0)
- Первая передача (1)
- Вторая передача (2)
- Третья передача (3)
- Четвертая передача (4)
- Пятая передача (5)

Каждая передача автомобиля имеет свой диапазон скоростей:

Передача	**Диапазон скоростей**
Задний ход	0 – 20
Нейтраль	Без ограничений
Первая	0 – 30
Вторая	20 – 50
Третья	30 – 60
Четвертая	40 – 90
Пятая	50 – 150

На каждой передаче можно развить скорость в пределах отведенного данной передаче диапазона. **Исключение – нейтральная передача**, на которой скорость можно изменить **только в сторону нуля**.

При включенном двигателе переключиться можно с любой передачи на любую при условии, что текущая скорость автомобиля находится в диапазоне скоростей новой передачи, и направление движения автомобиля допускает включение данной передачи. Например:

- на задний ход можно переключиться только с нейтральной или первой передачи **на нулевой скорости**;
- с заднего хода можно переключиться на первую передачу **только на нулевой скорости;**
- переключившись на заднем ходу на нейтральную передачу на ненулевой скорости, переключиться на первую передачу **можно только после остановки**

Двигатель данного автомобиля может быть **выключен** только при **нулевой** скорости на **нейтральной** передаче. При выключенном двигателе переключиться можно только на нейтральную передачу. Как следствие, автомобиль **после включения двигателя** находится на **нейтральной** передаче **в состоянии покоя**.

Автомобиль обладает следующими свойствами:

- Состояние двигателя (включен или выключен)
- Направление движения (вперед, назад или стоим на месте)
- Текущая скорость движения (целое число от 0 до максимальной скорости)
- Текущая выбранная передача (-1..5)

Автомобиль может выполнять следующие действия:

- Включить двигатель (если он выключен). Возвращает true, если двигатель включился и false – если двигатель был уже включен.
  bool **TurnOnEngine**()
- Выключить двигатель (если он включен и текущая передача – нейтральная, а автомобиль стоит). Возвращает true, если двигатель был успешно выключен, и false, если двигатель не может быть в данный момент выключен, либо он был выключен ранее).
  bool **TurnOffEngine**()

- Выбрать указанную передачу (-1..5). В случае успешного переключения передачи (в том числе и на саму себя) возвращает true.
  bool **SetGear**(int gear)
- Задать указанную скорость. Возвращает true, если скорость удалось изменить и false, если изменить скорость движения на указанную невозможно (например, на нейтральной передаче нельзя разогнаться).
  bool **SetSpeed**(int speed)

На основе данного класса разработать приложение, позволяющее пользователю поуправлять виртуальным автомобилем при помощи команд, вводимых со стандартного потока ввода:

- **Info**. Выводит состояние двигателя автомобиля, направление движения, скорость и передачу
- **EngineOn**. Включает двигатель
- **EngineOff**. Выключает двигатель
- **SetGear *<передача>***. Включает заданную передачу. В случае ошибки сообщает о причине невозможности переключения передачи
- **SetSpeed *<скорость>***. Устанавливает указанную скорость движения. В случае невозможности изменения скорости сообщает о причине невозможности изменить скорость на указанную.

Задание 2 – 50 баллов

Разработайте **с использованием TDD** класс CRectangle, моделирующий сущность «Прямоугольник».

Прямоугольник обладает следующими свойствами[1] (не путать с членами-данными[2]):

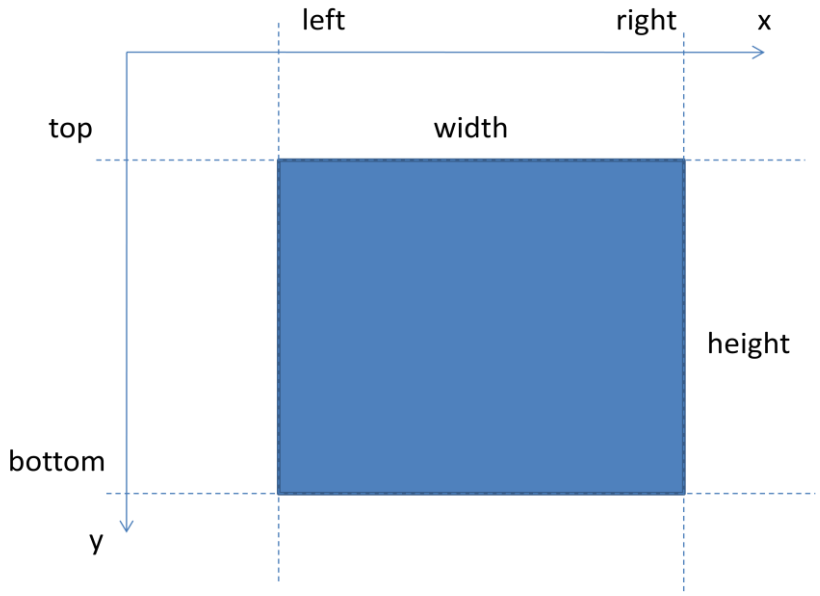
- Ширина (width), доступна как для чтения, так и для записи
- Высота (height), доступна как для чтения, так и для записи
- Координата X левого края (Left), доступна как для чтения, так и для записи
- Координата Y верхнего края (Top), доступна как для чтения, так и для записи
- Координата X правого края (Right), доступна как для чтения, так и для записи
- Координата Y нижней стороны (Bottom), доступна как для чтения, так и для записи
- Площадь (Area), доступна только для чтения
- Периметр (Perimeter), доступен только для чтения

Координаты и размеры прямоугольника **задаются целыми числами**.

Ширина и высота

Размеры (ширина и высота) прямоугольника не могут быть отрицательными (конструкторе и set-методах следует заменять отрицательные размеры на 0).

[1] Т.к. в C++ отсутствует поддержка свойств, следует вместо них использовать соответствующие Get* и Set* методы.
[2] Используйте тот набор переменных-членов класса, который необходим и достаточен для реализации упомянутых свойств



Конструктор данного класса принимает координаты левого верхнего угла прямоугольника, а также его ширину и высоту.

Над прямоугольником можно выполнить следующие действия:

- Перенести вдоль заданного вектора на dx и dy, не изменяя размеров прямоугольника
  void Move(int dx,int dy)
- Отмасштабировать прямоугольник с использованием масштабных коэффициентов sx и sy
  void Scale(int sx, int sy)
  При масштабировании координаты левого верхнего угла прямоугольника остаются без изменения, а изменяется только его размер.
  Если sx или sy является отрицательным числом, то масштабирования не происходит
- Найти пересечение данного прямоугольника с другим прямоугольником:
  bool Intersect(CRectangle const& other)
  Данный метод возвращает true, если прямоугольники пересекаются, и изменяет характеристики текущего прямоугольника. Если прямоугольники не пересекаются, то данный метод возвращает false и сбрасывает ширину и высоту прямоугольника в 0 (координаты верхнего левого угла остаются без изменений).

Разработать на основе данного класса программу, выполняющую считывание двух текстовых файлов, имена которых передаются программе через параметры командной строки, содержащих произвольное количество команд (по одной команде в каждой строке), управляющих размерами и трансформациями двух прямоугольников.

- **Rectangle *<left> <top> <width> <height>***. Инициализирует текущий прямоугольник указанными координатами и размерами.
- **Move *<dx> <dy>***. Переносит текущий прямоугольник вдоль заданного вектора, не изменяя его размеров
- **Scale *<sx> <sy>***. Масштабирует (увеличивает в указанное количество раз ширину и высоту) текущий прямоугольник с использованием масштабных коэффициентов относительно его верхнего левого угла

До первого появления команды Rectangle в файле текущий прямоугольник **имеет нулевые размеры и нулевые координаты верхнего левого угла**.

После выполнения команд, задаваемых в текстовых файлах программа должна вывести в стандартный поток вывода координаты и размеры прямоугольников после применения указанных преобразований, их периметр и площадь, а также результат их пересечения в следующем формате:

```
Rectangle 1:
	Left top: (<left1>; <top1>)
	Size: <width1>*<height1>
	Right bottom: (<right1>; <bottom1>)
	Area: <area1>
	Perimeter: <perimeter1>
Rectangle 2:
	Left top: (<left2>; <top2>)
	Size: <width2>*<height2>
	Right bottom: (<right2>; <bottom2>)
	Area: <area2>
	Perimeter: <perimeter2>
Intersection rectangle:
	Left top: (<left>; <top>)
	Size: <width>*<height>
	Right bottom: (<right>; <bottom>)
	Area: <area>
	Perimeter: <perimeter>
```

Бонус в 50 баллов за визуализацию исходных прямоугольников и результата их пересечения

Разработайте класс CCanvas, моделирующий прямоугольное растровое полотно для рисования в текстовом режиме, а также метод FillRectangle, выполняющий закрашивание области, соответствующей прямоугольнику на данном полотне.

Каркас класса CCanvas представлен ниже:

```
/*
Класс, моделирующий сущность Canvas (полотно, холст для рисования, картинка, канва),
хранящую прямоугольный массив пикселей. Для каждого пикселя изображения можно задать
свой код символа, что позволяет выводить простейшие картинки в текстовом режиме,
вроде таких:

   +----+
  /    /|
 +----+ |
 |    | +
 |    |/
 +----+

*/
class CCanvas
{
public:
   // Создает канву для рисования размером width*height
   // После своего создания содержимое канвы заполнено пробельными символами
   // Допускается создание канвы нулевых размеров
   CCanvas(unsigned width, unsigned height);

   // Возвращает ширину канвы
   unsigned GetWidth()const;

   // Возвращает высоту канвы
   unsigned GetHeight()const;

   // Очищает канву (заполняет содержимое символами с указанным кодом)
   // Если код символа находится в диапазоне от 0 до ' ', команда игнорируется
   void Clear(char code = ' ');

   // Задает код символа code для пикселя в координатах (x, y)
   // Координаты верхнего левого угла канвы принимаются равными 0, 0.
   // Если координаты выходят за пределы канвы, либо код символа
   // находится в диапазоне от 0 до (' ' - 1), содержимое канвы не должно меняться
```

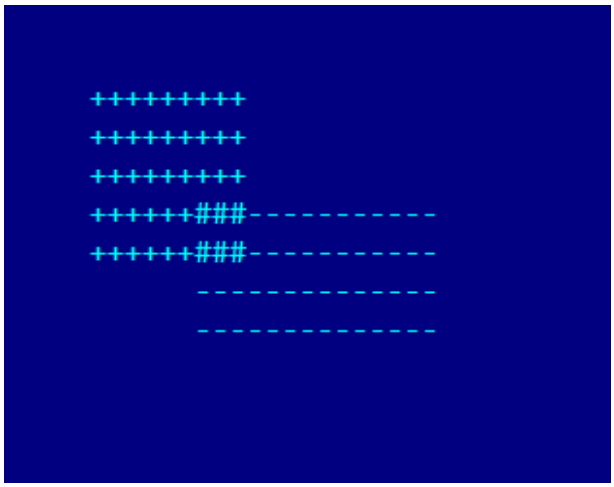
```
   void SetPixel(int x, int y, char code);

   // Возвращает код символа пикселя в координатах (x, y)
   // Координаты верхнего левого угла канвы принимаются равными 0, 0.
   // Если координаты пикселя выходят за пределы канвы, должен возвращаться
   // код символа "пробел"
   char GetPixel(int x, int y)const;

   // Выводит содержимое в поток вывода, производный от std::ostream
   // (например, std::cout, экземпляр ofstream, или ostringstream)
   // В конце каждой строки должен выводиться символ \n
   void Write(std::ostream & ostream)const;
private:
   // Закрытые данные и методы класса
};
```

Объявление функции FillRectangle представлено ниже

```
// Заполняет пиксели, соответствующие прямоугольнику rect в канве,
// символом с кодом code
void FillRectangle(CRectangle const& rect, char code, CCanvas & canvas);
```

Первый прямоугольник должен быть отображен при помощи символов “+”, второй – при помощи символа “-“, а результат их пересечения – при помощи символа “#”. На следующем рисунке показан ожидаемый результат работы программы при следующих входных данных:

- Прямоугольник 1. Left: 5, Top: 2, Width: 9, Height: 5
- Прямоугольник 2. Left: 11, Top: 5, Width: 14, Height: 4



Программа должна вывести визуальный результат пересечения прямоугольников в файл, имя которого передано в качестве 3-го аргумента командной строки, а при отсутствии 3-го аргумента – в стандартный поток вывода (сразу после текстовой информации, указанной в обязательной части задания). В обоих случаях принят размер полотна для рисования равным 60 * 20.

Дополнительные задания

Задание 3 – 100 баллов

Разработайте **с использованием TDD** класс CFile (выполненный в виде «умной оберткой» над функциями стандартной библиотеки языка C для работы с файлами, вроде fopen/fread/fwrite/fclose и т.п.), предоставляющий следующие методы для работы с файлами:

- **Open** - выполняет открытие файла (если файл был открыт, то он сначала закрывается) с указанным именем в указанном режиме (аналогично режимам, принимаемым функцией fopen). Возвращает true, если файл был открыт и false, если файл открыт не был
- **Close** - закрывает файл (если он был открыт)
- **IsOpened** – возвращает true, если файл открыт и false если нет.
- **IsEndOfFile** – возвращает true, если встречен символ конца файла
- **GetChar** – считывает символ из файла и возвращает его код. Возвращает значение EOF, если был встречен символ конца файла. Возвращает значение FILE_ERROR (задайте в виде перечислимой константы данное значение), если произошла ошибка (например, файл не был открыт, либо произошла ошибка чтения из файла)
- **PutChar** – записывает символ в файл. Возвращает значение true, если символ был записан успешно и false, если произошла ошибка (например, файл не был открыт, либо произошла ошибка записи)
- **GetLength** – возвращает длину файла в байтах. Возвращает значение -1, если произошла ошибка (например, файл не был открыт)
- **Seek** – позволяет выполнить позиционирование в файле (аналогично функции fseek). Возвращает значение false, если произошла ошибка (например, файл не был открыт, либо что-то еще)
- **GetPosition** – возвращает текущую позицию в файле. Возвращает значение -1, если произошла ошибка (например, файл не был открыт)
- **Write** – позволяет записать в файл данные (аналогично функции fwrite). Возвращает количество фактически записанных элементов. Возвращает значение -1, если произошла ошибка (например, файл не был открыт)
- **Read** – аналогично методу Write, но выполняет блочное считывание данных из файла (аналогично функции fread).

Указания:

Объявление класса и реализация его методов должны располагаться в отдельных файлах.

Переменные класса должны находиться в приватной области.

Для каждого из разработанных методов **класса обоснуйте использование или неиспользование** квалификатора **const**, а также способ передачи параметров (по значению, по ссылке, по константной ссылке, по указателю, по указателю на константные данные).

Класс должен иметь конструктор по умолчанию, инициализирующий значения переменных класса значениями по умолчанию (по умолчанию файл не открыт).

Деструктор класса должен вызывать метод Close. Это позволит автоматизировать процесс закрытия файла при разрушении экземпляра класса и упростит использование класса, фактически избавив от необходимости явного вызова метода Close в подавляющем большинстве случаев.

На основе разработанного класса разработайте приложение, выполняющее считывание входного текстового файла в память (для выделения массива данных нужного размера используйте класс **vector**), реверсирование массива символов, считанных из файла, в памяти (**без создания копии vector-а**) и запись

результата в выходной файл. Имена входного и выходного файлов передаются в виде параметров командной строки.

Задание 4 – 400 баллов

Разработайте с использованием TDD программу **AddressBook**, представляющую из себя адресную книгу, хранящую информацию о почтовых адресах, телефонах и адресах электронной почты абонентов. Программа должна позволять осуществлять поиск информации об абоненте по фрагменту почтового адреса, телефонному номеру, имени абонента или адресу электронной почты. Должна иметься возможность редактирования информации об абонентах (добавление, правка, удаление).

Данные об абонентах должны храниться в базе данных, представляющей из себя один или несколько файлов на диске. По окончании работы с программой в случае внесения изменений в базу данных программа должна предложить сохранить изменения в базу данных. Программа должна производить работу с файлом базы данных самостоятельно. Для этого спроектируйте подходящий формат хранения данных адресной книги в файле.

Если разработанный формат хранения данных накладывает ограничения на используемые в полях адресной книги символы, необходимо осуществлять соответствующие проверки при вводе данных пользователем. Т.е. если символ «точка с запятой» используется для разделения соседних полей, его использование внутри полей может в ряде случаев вызывать проблемы при дальнейшем считывании файла. Допустимым, хотя и не самым лучшим, вариантом решения данной проблемы будет запрет на использование символа «;» в составе имени или адреса.

Каждый абонент характеризуется следующей информацией:

- Имя
    - Имя состоит из одного или нескольких слов и не может быть пустым.
    - Допускается существование различных абонентов с одинаковым именем
- Почтовый адрес
    - Город
    - Улица
    - Номер дома (допускаются цифры и буквы)
    - Номер квартиры
    - **Примечание**: один и тот же адрес могут иметь несколько разных абонентов (пример – семья). Любое поле из адреса может быть пропущено, как, впрочем, и сам адрес
- Номера телефонов
    - Один абонент может иметь произвольное количество телефонных номеров (0 и больше)
    - Один и тот же номер телефона может быть у разных абонентов (пример – домашний номер телефона)
- Адреса электронной почты
    - Один абонент может иметь произвольное количество адресов электронной почты (ноль и больше)
    - Разные абоненты не могут иметь один и тот же адрес электронной почты

Спроектируйте классы, соответствующие сущностям предметной области. Следующие сущности являются обязательными (Вы можете ввести дополнительные сущности, соответствующие Вашему решению задачи).

- Абонент
- Адресная книга
    - Обязательные операции
        - Получить коллекцию абонентов

        - Загрузить коллекцию абонентов из файла
        - Сохранить коллекцию абонентов в файл
- Коллекция абонентов
    - Обязательные операции:
        - Выполнить поиск абонентов по указанным параметрам (известные данные адреса, имени, телефона, адреса эл. почты». Результатом данной операции будет коллекция абонентов (или их индексов в коллекции), удовлетворяющих условиям поиска
        - Удалить информацию об абоненте с указанным индексом
        - Перезаписать информацию об абоненте с указанным индексом
- Почтовый адрес
    - Обязательные операции:
        - Сравнить адрес с указанным адресом. Адреса считаются равными, если все их соответствующие непустые поля совпадают (сравнение происходит без учета регистра символов). Любое незаполненное поле почтового адреса, принимается равным соответствующему полю другого адреса, пустому или нет.
            - Результат сравнения адреса «Москва, Шаболовка, 37, 54» с адресом «, Шаболовка, 37,» будет положительным
            - Результат сравнения адреса «Москва, Шаболовка, 37, 54» с адресом «Йошкар-Ола, Шаболовка, ,» будет отрицательным
- Имя
    - Обязательные операции:
        - Сравнить имя с указанной строкой. Сравнение происходит без учета регистра символов и лишних пробелов и сравнивает данные, указанные в ФИО, с данными, переданными в строке. Например, результат сравнения ФИО «Иванов Сергей Петрович» со следующими строками будет положительным[3]:
            - *Сергей иванов*
            - *Иванов Сергей*
            - *Сергей Петрович*
            - *Иванов ПетровиЧ*
            - *Иванов*
            - *Петрович*
            - *Сергей Петрович Иванов*
            - *Иванов Сергей Петрович*
        - Результат сравнения имени «Иванов Сергей Петрович» со следующими строками будет отрицательным[4]:
            - *Сергеев Иван Петрович*
            - *Иванов Иван Петрович*
            - *Иван*
            - *Семён Семёныч Горбунков*

Для хранения номера телефона и адреса электронной почты можно воспользоваться строковыми типами.

Программа должна предоставлять пользователю меню возможных в данный момент действий. Выбор пункта меню пользователь осуществляет путем ввода его порядкового номера или иной строки. Например:

---

[3] Все слова, входящие в состав проверяемой строки, совпадают (без учета регистра символов) со словами, входящими в состав имени

[4] Хотя бы некоторые слова проверяемой строки не совпадают (без учета регистра символов) со словами, входящими в состав имени

```
Выберите действие:
1. Поиск абонента
2. Добавить абонента
3. Отредактировать/удалить абонента
Q. Выйти
```

Выбор одного пункта меню может предложить пользователю новое подменю. Например, для редактирования абонента нужно будет ввести известные данные абонента, в результате чего программа должна предложить меню из списка абонентов, удовлетворяющих критериям поиска, чтобы пользователь выбрал из них абонента (например, по порядковому номеру), которого он хотел бы отредактировать, либо ввести R для возврата в предыдущее меню. После выбора абонента пользователю было бы предложено меню, позволяющее удалить данного абонента, изменить ФИО, отредактировать адрес, добавить номер телефона, удалить номер телефона, отредактировать один из номеров телефона, добавить, удалить или отредактировать один из адресов электронной почты.

Весьма полезным может оказаться оформить код, выполняющий взаимодействие с пользователем, в виде одного или нескольких классов, имеющих доступ к модели данных (или ее части).

Ссылки

1. Разработка через тестирование