# Общее описание

* Лабораторные работы - это поэтапная работа над одним проектом. **В связи с этим, перед выполнением работ, рекомендуется предварительно ознакомиться с текстом всех работ.**
* Лабораторные работы выполняются и защищаются индивидуально.
* За лаб. работу может быть выставлено неполное количество баллов в случае наличия ошибок, невыполнения требований или замечаний.

# Требования к коду

1. Обязательное соблюдение стиля кода: отсутствие транслитераций в названиях переменных, функций, классов и.т.д; все идентификаторы должны быть “говорящими, то есть по названию должен быть понятен смысл идентификатора; вложенность блоков должна быть обозначена табуляцией; каждому .cpp файлу (кроме файла с ф-цией main) должен соответствовать .h файл; в каждом .h файле объявлен только 1 класс.  
   *Гайд по стилю для примера* [*https://google.github.io/styleguide/cppguide.html*](https://google.github.io/styleguide/cppguide.html)
2. Весь проект должен быть написан в ООП стиле. Все функции кроме main должны быть методами классов. Количество static методов должно быть минимально, а их использование четко обосновано. В функции main должна быть только инициализация необходимых объектов и запуск логики.
3. Весь код должен быть написан на чистом C++ с использованием стандартной библиотеки. Сторонние библиотеки можно использовать только для написания GUI или CLI. *Исключение, библиотеки по работе с файлами, например JSON, но не библиотеки, которые реализуют больше логики чем работа с файлами (например, серализация/десериализация)*
4. Все классы должны реализованы так, чтобы гарантировалось сохранение инварианта.
5. Запрещается дублирование кода, в том числе смысловое.
6. Не должно быть “божественных” классов.

# Требования к отчету

1. Отчет должен быть оформлен согласно шаблону с ЛЭТИ
2. Для каждой лаб. работы должны быть UML-диаграммы классов. Диаграммы должны отображать только то, что было сделано в рамках лаб. работы.
3. Отчет должен содержать описание архитектурных решений. Недостаточно просто перечислить набор классов и методов, которые были разработаны. Необходимо указывать, для чего классы сделаны, почему они имеют такую реализацию, и как и почему классы связаны между собой.
4. В отчете должна быть отражена проверка классов и то, что программа работает.

# 

# Сроки сдачи лаб. работ

1. Сдача лаб. работы до дедлайна позволяет получить максимальный балл (10).
2. Если работа сдана до дедлайна на 8 и менее баллов, то **при защите следующей лаб. работы** можно показать исправления и добрать до 9 баллов за лаб. работу. (для последней лаб. работы исправления показываются отдельно). *Если будет свободное время, то можно будет отдельно показать исправления, но только 1 раз.*
3. Сдача лаб. работы в первый раз в течение недели после дедлайна оценивается максимум в 5 балла.
4. Сдача лаб. работы в первый раз через неделю после дедлайна оценивается максимум в 1 балл.
5. За одно занятие лаб. работу можно показать только **один** раз.

Таблица с дедлайнами и баллами

**Даты могут быть скорректированы в течение семестра**

\*необязательные лаб. работы

| **№ ЛР** | **10 баллов** | **5 баллов** | **1 балл** |
| --- | --- | --- | --- |
| 1 | ~~06.10~~ 15.10 | ~~13.10~~ 22.10 | после ~~13.10~~ 22.10 |
| 2 | ~~05.11~~ 12.11 | ~~12.11~~ 19.11 | после ~~12.11~~ 19.11 |
| 3 | 03.12 | 10.12 | после 10.12 |
| \*4 | 24.12 | 29.12 | после 29.12 |

# Оценка за семестр

* Оценку за семестр можно получить на основании набранных баллов за работы, если сданы все 4 лаб. работы.
* Можно воспользоваться возможностью поменять оценку сдав экзамен. В таком случае баллы “сгорают”.
* Для допуска к экзамену (получению оценки) необходимо сдать первые 3 работ. В таком случае можно не выполнять лаб. работу 4, и экзамен сдается в обязательном порядке.
* Если сданы 4 работы, но не набрано достаточное кол-во баллов, то экзамен сдается в обязательном порядке

**Соотношение оценки и кол-ва баллов (включительно)**

| **Оценка** | **Баллы** |
| --- | --- |
| Отлично | 35 |
| Хорошо | 27 |
| Удовлетворительно | 20 |

# Лабораторные работы

В рамках лабораторных работ необходимо реализовать простую игру “морской бой с модификациями”, систему управления, сохранения и загрузки.

**Запись на лаб. работы**

Очередь формируется по дате заполнения формы. Очередь отражается в соответствующих таблицах. Записывать нужно за день до занятия до 12:00 (например, если занятие в понедельник, то записываться надо до 12:00 воскресенья).

**Записываться только в случае если работа сделана и готова к защите!**

Если хотите прийти и показать только исправления, то в форме нужно отметить соответствующую галочку. При формировании очереди приоритет у тех, кто сдает лаб. работу.

Если будет записано меньше 5 человек в очередь, то занятие проводиться не будет.

Ссылки на запись:

3341 и 3342: <https://forms.gle/BRidvDbyNDRMyk7UA>

3343 и 3344: <https://forms.gle/xP4i6ChRgJJThAQN6>

3381 и 3382: <https://forms.gle/YgkHCHrwujYYYBcw7>

3383 и 3384: <https://forms.gle/A5RnKfJnJdMbQ3Pq8>

3385 и 3388: <https://forms.gle/kgGFtGNK4cfkP53JA>

## Лабораторная работа №1 - Создание классов

1. Создать класс корабля, который будет размещаться на игровом поле. Корабль может иметь длину от 1 до 4, а также может быть расположен вертикально или горизонтально. Каждый сегмент корабля может иметь три различных состояния: целый, поврежден, уничтожен. Изначально у корабля все сегменты целые. При нанесении 1 урона по сегменту, он становится поврежденным, а при нанесении 2 урона по сегменту, уничтоженным. Также добавить методы для взаимодействия с кораблем.
2. Создать класс менеджера кораблей, хранящий информацию о кораблях. Данный класс в конструкторе принимает количество кораблей и их размеры, которые нужно расставить на поле.
3. Создать класс игрового поля, которое в конструкторе принимает размеры. У поля должен быть метод, принимающий корабль, координаты, на которые нужно поставить, и его ориентацию на поле. Корабли на поле не могут соприкасаться или пересекаться. Для игрового поля добавить методы для указания того, какая клетка атакуется. При попадании в сегмент корабля изменения должны отображаться в менеджере кораблей.

Каждая клетка игрового поля имеет три статуса:

* 1. неизвестно (изначально вражеское поле полностью неизвестно),
  2. пустая (если на клетке ничего нет)
  3. корабль (если в клетке находится один из сегментов корабля).

Для класса игрового поля также необходимо реализовать конструкторы копирования и перемещения, а также соответствующие им операторы присваивания.

**Примечания:**

* Не забывайте для полей и методов определять модификаторы доступа
* Для обозначения переменной, которая принимает небольшое ограниченное количество значений, используйте enum
* Не используйте глобальные переменные
* При реализации копирования нужно выполнять глубокое копирование
* При реализации перемещения, не должно быть лишнего копирования
* При выделении памяти делайте проверку на переданные значения
* У поля не должно быть методов возвращающих указатель на поле в явном виде, так как это небезопасно

## Лабораторная работа №2 - Полиморфизм

1. Создать класс-интерфейс способности, которую игрок может применять. Через наследование создать 3 разные способности:
   1. Двойной урон - следующая атак при попадании по кораблю нанесет сразу 2 урона (уничтожит сегмент).
   2. Сканер - позволяет проверить участок поля 2х2 клетки и узнать, есть ли там сегмент корабля. Клетки не меняют свой статус.
   3. Обстрел - наносит 1 урон случайному сегменту случайного корабля. Клетки не меняют свой статус.
2. Создать класс менеджер-способностей. Который хранит очередь способностей, изначально игроку доступно по 1 способности в случайном порядке. Реализовать метод применения способности.
3. Реализовать функционал получения одной случайной способности при уничтожении вражеского корабля.
4. Реализуйте набор классов-исключений и их обработку для следующих ситуаций (можно добавить собственные):
   1. Попытка применить способность, когда их нет
   2. Размещение корабля вплотную или на пересечении с другим кораблем
   3. Атака за границы поля

**Примечания:**

* Интерфейс события должен быть унифицирован, чтобы их можно было единообразно использовать через интерфейс
* Не должно быть явных проверок на тип данных

## Лабораторная работа №3 - Связывание классов

1. Создать класс игры, который реализует следующий игровой цикл:
   1. Начало игры
   2. Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
   3. В случае проигрыша пользователь начинает новую игру
   4. В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

1. Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

**Примечание:**

* Класс игры может знать о игровых сущностях, но не наоборот
* Игровые сущности не должны сами порождать объекты состояния
* Для управления самое игрой можно использовать обертки над командами
* При работе с файлом используйте идиому RAII.

## \*Лабораторная работа №4 - Шаблонные классы

1. Создать шаблонный класс управления игрой. Данный класс должен содержать ссылку на игру. В качестве параметра шаблона должен указываться класс, который определяет способ ввода команда, и переводящий введенную информацию в команду. Класс управления игрой, должен получать команду для выполнения, и вызывать соответствующий метод класса игры.
2. Создать шаблонный класс отображения игры. Данный класс реагирует на изменения в игре, и производит отрисовку игры. То, как происходит отрисовка игры определяется классом переданном в качестве параметра шаблона.
3. Реализовать класс считывающий ввод пользователя из терминала и преобразующий ввод в команду. Соответствие команды введенному символу должно задаваться из файла. Если невозможно считать из файла, то управление задается по умолчанию.
4. Реализовать класс, отвечающий за отрисовку поля.

**Примечание:**

* Класс отслеживания и класс отрисовки рекомендуется делать отдельными сущностями. Таким образом, класс отслеживания инициализирует отрисовку, и при необходимости можно заменить отрисовку (например, на GUI) без изменения самого отслеживания
* После считывания клавиши, считанный символ должен сразу обрабатываться, и далее работа должна проводить с сущностью, которая представляет команду.
* Для представления команды можно разработать системы классов или использовать перечисление enum.
* Хорошей практикой является создание “прослойки” между считыванием/обработкой команды и классом игры, которая сопоставляет команду и вызываемым методом игры. Существуют альтернативные решения без явной “прослойки”
* При считывания управления необходимо делать проверку, что на все команды назначена клавиша, что на одну клавишу не назначено две команды, что на одну команду не назначено две клавиши.