Объектно-ориентированное программирование

Лабораторные работы (очная форма обучения)

- Вместо лабораторных работ (кроме второй) можно разрабатывать собственный проект по договорённости с преподавателем. Индивидуальный проект должен предполагать самостоятельный процесс объектно-ориентированного проектирования.
- Рекомендуется использовать систему контроля версий.
- Рекомендуемый стандарт кодирования: https://tproger.ru/translations/stanford-cpp-style-guide
- Лабораторную работу можно сдавать по частям.

Семестр I (2018-2019)

Задание 1. Работа с Git.

Продемонстрировать[1] работу с системой управления версиями git на примере одного из проектов[2] (программ) созданных на лабораторных занятиях.

Сценарий использования git для примера:

1. Создать репозитарий.
2. Добавить файлы к отслеживанию.
3. Просмотреть состояние репозитория
4. Сделать коммит (зафиксировать изменения).
5. Исправить сообщение предыдущего коммита.
6. Сделать коммит. Просмотреть разницу между коммитами.
7. Создать новую ветку.
8. Переключится на неё.
9. Внести изменения. Посмотреть разницу (diff) между ветками.
10. Зафиксировать изменения.
11. Посмотреть что содержится в файлах на ветке master.
12. Объединить ветки.
13. Клонировать удалённый репозитарий

1 Вместо демонстрации всех действий, можно показать отчёт (текстовый документ) о проделанных действиях и выполнить отдельные действия во время защиты работы. Отчёт должен фиксировать все этапы выполнения работы (например в виде скриншотов) и результаты выполняемых действий.

2 Git может воспринимать файлы исходных текстов созданные в Visual Studio как бинарные, поэтому нельзя будет корректно просмотреть разницу между версиями файлов.

14. Отправить изменения в удалённый репозитарий, забрать изменения из удалённого репозитария.
15. *Дополнительно (если не выполнено, максимальная оценка на экзамене - 3):*
    1. Создать worktree для ветки master.
    2. Моделировать конфликт. Решить конфликт.
    3. Использовать теги
16. Дополнительно: изучить интеграцию git в вашу любимую среду программирования.

Рекомендуется использовать систему управления версиями git при работе над другими заданиями.

Вопросы.

1. Что такое система управления версиями? Для чего она используется?
2. Что такое репозитарий (локальный и удалённый)?
3. Какие файлы следует добавлять к отслеживанию, а какие нет?
4. В каких случаях создавать ветку?
5. Что такое конфликт? Как исправить?

Ссылки

- Слайды про git – github.com/VetrovSV/Programming/blob/master/git_lec.pdf
- GitBook – uleming.github.io/gitbook/index.html
- worktree – inoyakaigor.ru/blog/85

Задание 2. Простой класс

Классы на выбор:

- Геометрическая фигура. Задание сторон, координат на плоскости. Вычисление площади и периметра.
- Комплексное число. Операторы сложения, вычитания, умножения (на комплексное и действительное число). Вычисление аргумента и модуля.
- Кватернион. Аналогично комплексному числу.
- Вектор. Задаётся своими компонентами. Вычисление длины, углов между осями; операторы сложения и вычитания, умножения на число.
- Время. Сложение, вычитание. Добавление минут, секунд, часов и т.п. Перевод времени в секунды, часы, минуты. Преобразование в строку.
- Дата. Реализовать то же самое, что и для времени.
- Обыкновенная дробь. Операции сложения, вычитания, умножения, деления, сравнения.
- Другой класс по согласованию с преподавателем.

1. Описать АДТ.
2. Представить класс в виде UML диаграммы.
3. Описать класс на C++. Реализовать методы для доступа и изменения данных, контроль постоянства, конструктор с параметрами. Операторы и генерирование исключительных ситуаций если необходимо
4. Продемонстрировать работу с классом, с основными методами, операторами.
5. Создать массив из объектов. Программа не обязательна должна взаимодействовать с пользователем, главная цель — показать пример использования класса.
6. Записать состояние объектов в файл, загрузить из файла.
7. Для класса привести документацию для класса описав его назначение, принципы использования, смысл методов и их параметров. Если необходимо привести пример использования класса в документации.
    1. *Дополнительно: описать документацию в markdown?*

Для проверки корректности всех методов класса можно использовать юнит-тест. см. «Задание 8. Юнит-тест» из второй части курса.

Вопросы

1. Что такое АДТ?
2. Что такое предусловия? Для чего нужны? Что такое постусловия?
3. Что такое класс? Что такое объект?
4. Что такое абстрагирование?
5. Что такое инкапсуляция? Что такое метод? Что такое поле класса? Что такое конструктор?
6. Что такое принцип сокрытия? Что такое «чёрный ящик»?
7. Что такое оператор?
8. Как вызвать метод конкретного объекта находящегося в массиве?
9. Чем отличаются обращения к методам в C++ с использованием объекта, ссылки на объект и указателя на объект?
10. Что такое равенство объектов? Когда объекты идентичны?
11. Чем отличается присваивание объектов от присваивания указателей на объекты?
12. Как обратится к методу или полю объекта находящегося в массиве?
13. Что такое поведение? Что такое состояние?

Ссылки

- Слайды лекции: github.com/VetrovSV/OOP/blob/master/OOP_1.1.pdf
- Слайды с лекции (АДТ и UML): github.com/VetrovSV/OOP/blob/master/OOP_1.0.pdf
- Пример класса: github.com/VetrovSV/OOP/tree/master/examples/simple_class
- Draw.io – создание диаграмм

Задание 3. Класс «матрица»

(не обязательное, -1 к максимальной оценке, если не сдана)
(upd 2019: обязательный минимум выделен курсивом)

Создать класс представляющий матрицу. В качестве основы использовать класс vector.

Реализовать в классе:

- *доступ к отдельным элементам матрицы*
- *заполнение матрицы одним значением*
- *заполнение матрицы случайными числами*
- *сложение и вычитание матриц (использовать оператор)*
- *умножение на число*
- умножение матрицы на матрицу
- транспонирование
- создание диагональной матрицы.
- Дополнительно:
  - вычисление определителя
  - вычисление обратной матрицы
  - доступ к строкам матрицы
  - Операторы *=, -=, +=?
  - применения функции к элементам матрицы

- *Наглядно продемонстрировать работу всех методов.* Недопустимые или невозможные операции над матрицами обрабатывать с помощью механизма генерации исключений.

- Рекомендуется использование системы контроля версий при разработке.

- *Для класса привести документацию описав его назначение, принципы использования, смысл методов и их параметров.* Если необходимо привести пример использования класса в документации.

- Дополнительно: описать документацию в markdown.

Пример: QGenericMatrix - http://doc.qt.io/qt-5/qgenericmatrix.html

Возможна замены темы задания по согласованию с преподавателем.

Вопросы

1. Какие бывают виды конструкторов?
2. Объясните правило большой пятёрки. Когда его стоит применять?
3. Как должен быть реализован оператор присваивания?
4. Что такое оператор?
5. Какие бывают виды операторов?
6. Как определяется бинарный оператор внутри класса?
7. Как определяется бинарный оператор вне класса (дружественная функция)?
8. Когда оператор следует определять как метод, а когда как дружественную функцию?
9. Чем отличается класс vector от класса list?

Задание 4. Наследование

1. Создать UML диаграмму из 3-х (или более) классов имеющих отношение типа наследование.
2. Реализовать классы на C++. Продемонстрировать работу с ними.

Пример классов для задания:

- Геометрическая фигура на плоскости, квадрат, круг, прямоугольник.
- Шахматная фигура, пешка, ладья, …

Для классов привести документацию описав их назначение, принципы использования, смысл методов и их параметров. При необходимости приведите пример использования классов в документации.

Вопросы

1. Что такое наследование? Как оно изображается на UML?
2. Когда стоит использовать наследование?
3. Какие классы называется базовыми и производными?
4. Сколько предков может иметь класс?
5. Что такое множественное наследование? Чем оно опасно? Когда его можно использовать?
6. Что такое перегрузка метода? Что такое переопределение метода?
7. Как работает преобразование типов связанных наследованием?
8. Что такое интерфейс (ООП)?
9. Как влияют области видимости внутри класса на наследуемые методы и поля?

Ссылки

- Слайды лекции (Наследование): https://github.com/VetrovSV/OOP/blob/master/OOP_1.2.pdf

Задание 5. Диаграмма классов

Составить UML диаграмму классов.

- Не менее 4 классов.
- Каждый взаимодействует хотя бы с одним другим. Взаимодействие не должно быть только последовательным.
- Объекты не должны быть однотипными.
- Должно быть как минимум по одному отношению: ассоциация, агрегация (композиция), наследование.
- Указать мощность связей.
- Диаграмму оформить в электронном и твёрдом формате. Сохранение диаграммы исключительно в формат растровых изображений не допускается.
- Возможна презентация схемы на доске или проекторе, обсуждение в группе.

Диаграмма классов должна быть целостной: полностью описывать предметную область, не иметь лишних сущностей, свойств или отношений.

Вопросы

1. Какие отношения возможны между классами?
2. Какие отношения возможны между объектами?
3. Что такое мощность отношения?
4. Как в C++ может быть реализовано отношение *ассоциация (связь)?*
5. Чем отличается агрегация от композиции?
6. Как в C++ может быть реализовано отношения агрегации и композиции?
7. Когда следует использовать наследование, а когда агрегацию (композицию)?

Ссылки

- Слайды с лекции (UML): https://github.com/VetrovSV/OOP/blob/master/OOP_1.0.pdf

Задание 6. Реализовать лабораторную 5 на C++

(не обязательное, +1 к оценке на экзамене)

Реализовать предыдущую лабораторную работу. Как минимум один метод должен быть переопределён. Каждый класс должен располагаться в отдельном модуле.

Для классов привести документацию описав их назначение, принципы использования, смысл методов и их параметров. При необходимости приведите пример использования классов в документации.

Представить состояние и изменение объектов наглядно в программе с GUI.

Рекомендации: вести разработку «сверху вниз», использовать систему контроля версий.

Вопросы

1. Какие отношения возможны между классами?
2. Какие отношения возможны между объектами?
3. Что такое мощность отношения?
4. Как в C++ может быть отображено отношение *ассоциация (связь)?*
5. Какие возможности среды программирования использовались? Автоматическая генерация методов, конструктора? Рефакторинг? Автоматическое изменение сигнатуры методов?

Ссылки

- Слайды лекции (Отношения между классами): https://github.com/VetrovSV/OOP/blob/master/OOP_1.1.pdf

Задание 7. Калькулятор

Создать калькулятор с графическим интерфейсом пользователя. Калькулятор должен корректно обрабатывать любые входные данные. Сделать обработку исключительных ситуаций. Помимо арифметических операций и возведения в любою степень калькулятор должен вычислять функции: sin, cos, tan, ln, exp.

Дополнительно:

- хранить историю вычислений
- взаимодействие с интерфейсом без мыши
- разбирать математические выражения, например: `2 x 4 / (2 x 2)`; можно использовать сторонние библиотеки

Возможно изменение задания работы после согласования с преподавателем.

Вопросы

1. Что такое исключительная ситуация?
2. Как работает механизм обработки исключительных ситуаций?
3. Что такое бизнес-логика?
4. В какой части программы должна быть реализована бизнес-логика?
5. Изобразите диаграмму классов для приложения.
6. Какие возможности среды программирования использовались? Автоматическая генерация методов, конструктора? Рефакторинг? Автоматическое изменение сигнатуры методов? Система управления версиями?