* 1. Добрый день, в этом видео я расскажу вам о своем курсовом проекте. Целью курсового проекта было продемонстрировать умение работать с файлами и структурами данных на примере двусвязного списка. Подходящей для меня темой оказалась реализация концепции "Черепашья графика". Я написал программу, которая реализует концепцию черепашьей графики, то есть по исходному набору команд строит изображение. Программу я писал на языке программирования C++, потому что для меня это полезная практика, а так же потому, что первоочередной задачей я ставил расширяемость написанного мною кода без особых сложностей. Программа представляет собой утилиту командной строки.
  2. Теперь я расскажу немного о самой черепашьей графике. Итак, черепашья графика — это один из методов построения графических примитивов, основанный на метафоре рисующего устройства, а именно Черепахи. Это устройство способно перемещаться по экрану и поворачиваться в заданных направлениях, при этом оставляя за собой нарисованный след заданного цвета и ширины. Так же это понятие включает в себя способ задания команд такому устройству.
  3. Впервые черепашья графика была применена при реализации языка программирования Лого. Сейчас библиотека по работе с черепашьей графикой есть в языке python. Черепашья графика хорошо подходит для случаев, когда нужно рисовать несложные рисунки. Так же она очень удобно подходит для рисования рекурсивных узоров. Я не мог использовать готовые решения, которые представлены, например, в языке python, поэтому я реализовал собственный язык команд черепахи.

2.1) Сейчас я покажу как моя программа работает. Для этого я продемонстрирую одну итерацию работы программы… Итак, мы создаем исходный файл с командами для черепахи /открываю файл/. В нем находятся команды, которые и будут испольняться моей программой. Так же у нас есть исполняемый файл, который нужно запустить с параметром - имя исходного файла с командами /пишу параметры и запускаю/. Вот результат работы программы /открываю появившийся .bmp файл/. Так же вторым параметром для утилиты может служить имя результирующего файла /демонстрирую запуск утилиты с двумя параметрами/.

2.2) Так же я хочу показать результаты работы еще нескольких исходных файлов с командами /показываю программы и другие красивые рисунки/.

3) Процесс работы программы логически можно разделить на несколько этапов. Первый этап - это считывание содержимого исходного файла, его синтаксический разбор на команды и составление дерева команд. За считывание информации из файла отвечает объект класса Text\_File\_Reader. Он принимает в конструктор имя исходного файла и у него есть только один метод - read(), который возвращает строку, в которой находится содержимое файла. За синтаксический разбор отвечает объект класса Parser. Он принимает в конструктор строку и обработчик команд. У него так же есть только один метод, который называется handle(). Метод handle() класса Parser сначала разбирает исходную строку на список строк, а затем вызывает метод handle() объекта обработчика для каждой строки полученного списка. Класс обработчик содержит в себе список обработчиков каждой отдельной команды, которые идентифицируют поступающую команду и добавляют ее в дерево команд. Следующий этап - это исполнение команд, из которых было сформировано дерево. На этом этапе каждая команда посредством вызова метода execute() вызывает методы черепахи для совершения ею определенных действий, то есть, например, переместиться, повернуть в какую-либо сторону и так далее. Объект класса Turtle имеет соответствующие методы. Когда вызываются методы черепахи, отвечающие за ее перемещение, то эти методы, в свою очередь вызывают методы объекта класса Canvas, то есть объекта холста, у которого есть метод рисования линий drawLine(). Этот метод в свою очередь, вызывает метод drawLine() инкапсулированного объекта класса Drawer, который реализует соответствующую логику и алгоритмы рисования линий, основанные на алгоритме Брезенхэма. Заключительный этап, это вызов метода uploadToBmp(), который выгружает получившийся рисунок в .bmp файл. В программе мною самостоятельно реализованы структуры хранения данных "Двусвязный список" и "Строка" - это классы list и string. Хранение дерева команд реализовано по схеме паттерна Компоновщик, который реализован в классе Composite. Генерация объектов классов команд выполняется по схеме паттерна "Простая фабрика", который реализован в классе Command\_Factory.

4) Подводя итог я хочу сказать, что созданная программа может использоваться как отдельно, в виде самодостаточной утилиты, так и встраиваться в другие проекты, как на уровне передачи данных через файлы, так и на уровне встраивания классов в проекты C++. Например, пользуясь моим модулем можно удобно и быстро реализовать проект по построению графиков различных функций. В качестве примера могу привести этот несложный скрипт на языке javascript, который генерирует файл с командами Черепахи /показываю файл на javascript/.Так же мой проект поддается довольно серьезному расширению засчет добавления новых классов команд, новых классов обработчиков команд, и новых классов перьев. Например, я реализовал возможность рисовать только круглым пером, но при желании можно добавить несколько различных перьев, например квадратное перо, или перо с размытыми краями. На этом я завершаю свой рассказ. До новых встреч.