Отчёт АВС по 3-м заданиям.

Подготовил Веснин Константин БПИ203

Описание задания:

2. Плоская геометрическая фигура, размещаемые в координатной сетке.	1. Круг (целочисленные координата центра окружности, радиус) 2. Прямоугольник (целочисленные координаты левого верхнего и правого нижнего углов) 3. Треугольник (целочисленные координаты трех углов)	Цвет фигуры (перечислимый тип) = {красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый}	Вычисление периметра фигуры (действительное число)

7. Упорядочить элементы контейнера по возрастанию используя сортировку методом деления пополам (Binary Insertion). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

Как запускать?

Первые две программы: <Файл> <Команда> <Аргумент1> <Аргумент2>

Третья программа: <Файл><Аргумент1><Аргумент2>, потому что не успел реализовать на ней автогенерацию, работает только с файлами.

Где файл:

- Для первых двух заданий найти файл .exe в папке debug и использовать его.
- Для третьего задания это main.py

Команда:

- “-fromFile” – позволяет загрузить данные из файла. В случае этой команды Аргумент1 – это путь к файлу (то есть полное название).
- “-autoGen” – позволяет сгенерировать данные автоматически. В случае этой команды Аргумент1 – это количество фигур, которые необходимо создать.

Аргумент2 – это путь к файлу (то есть полное название), в который необходимо вывести результат.

По результату работу, программы в консоли сообщат время, которое программа работала.

Описание структуры

1. Первое задание выполнено в процедурном подходе. Разделение по файлам носит по большей части условный характер, потому что программа скорее представляет из себя длинную цепочку команд. Изначально мне было удобнее написать её целиком в одном файле. Вот сам процесс:

   Загрузить контейнер (из файла или авто). Сохраняется в структуры Shape, которое хранит в себя подструктуры Rectangle, Circle и Triangle. Массив таких Shape хранится в структуре Container.

Вывести контейнер в файл. Для этого проведена целая цепочка, идущая по логике от контейнера к Circle, Rect и Tr, так как вывести надо каждый элемент.
Бинарно отсортировать контейнер по периметру. И вывести снова.

Писать такой код довольно сложно, потому что нет обобщения из ООП, то есть нельзя назвать квадрат и круг фигурами, приходится обходиться через условные операторы и всё выполнять раздельно.

2. Второе задание выполнено в ООП. Теперь Shape – это базовый виртуальный класс для Circle, Rect и Tr. Такие методы, как getPerimetr пишутся не раздельно. Во всём остальном работа программы очень похожа на процедурный подход.

3. Третье задание написано на Python с динамической типизацией. Внешне работа похожа на предыдущие задания, отличий больше в том, как этот код работает.

Протестируем программы по времени на разных наборах данных (в мс):

	Процедур.	ООП.	Дин. Тип.
In1.txt	~2.35	~2.35	~18
In2.txt	Incorrect	Incorrect	Incorrect
In3.txt	~4	~4	~19
In4.txt	Incorrect	Incorrect	Incorrect
In5.txt	~38	~44	~46

Из результатов мы видим, что динамическая типизация очень значительно уступает ООП и процедурным подходам при небольших наборах данных, однако когда данных становится больше, то разница в работе становится несущественной.