Арунова Анастасия БПИ208

Пояснительная записка

Описание полученного задания

Вариант: 282

Начальное условие задач: 2.

Обобщенный артефакт, используемый в задании	Базовые альтернативы (уникальные параметры, задающие отличительные признаки альтернатив)	Общие для всех альтернатив переменные	Общие для всех альтернатив функции
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)

Функция обработки данных в контейнере: 21.

Удалить из контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив меньше, чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы передвинуть к началу контейнера с сохранением порядка.

Требуемые метрики, определяющие характеристики программы, для различных тестовых прогонов.

Программа содержит 0 интерфейсных модулей (заголовочных файлов) и 9 модулей реализации (файлов с определением программных объектов):

- `triangle.py`
- `rectangle.py`
- `shape.py`
- `container.py`
- `main.py`
- `circle.py`
- `random_shape_generator.py`
- `build_container.py`
- `colors.py`

Общий размер исходных текстов: 16,1 КБ.

Полученный размер исполняемого кода: – (не создаётся).

Время выполнения программы для различных тестовых наборов данных:

Количество фигур	Время работы программы при вводе данных из файла	Время работы программы при случайной генерации данных
1 фигура	0,2 ms	0,2 ms
3 фигуры	0,4 ms	0,2 ms
8 фигур	0,4 ms	0,5 ms
12 фигур	0,6 ms	0,7 ms
14 фигур	0,6 ms	0,9 ms
100 фигур	2 ms	2 ms
1000 фигур	16 ms	20 ms
10000 фигур	1,7 sec	2 sec

Программа была протестирована на 12 тестах. Все тесты расположены в папке проекта «…\CSA_HW3\tests» и имеют название «test<i>.txt», где i – числа от 0 до 11. Причём некорректный ввод проверяют файлы «test1.txt» (некорректно заданные фигуры), «test7.txt» (пустой контейнер), «test8.txt» (переполнение контейнера). При некорректном цвете фигуры, её координатах программа генерирует их случайным образом. Тесты «test9.txt», «test10.txt», «test11.txt» написаны для получения времени работы программы на 10000, 100 и 1000 фигурах соответственно.

Результаты прохождения i-го теста записываются в два файла: «output<i>.txt» и «output_deleted<i>.txt». В первом из них содержится информация о контейнере, во втором информация о контейнере после применении функции, данной в задании. Файлов «output7.txt» и «output_deleted7.txt», «output8.txt» и «output_deleted8.txt» нет, так как на соответствующих им тестах программа в виду некорректных данных заканчивает работу. Также проект содержит два файла: «outputR.txt» и «output_deletedR.txt», которые были получены в ходе тестирования программы на случайно сгенерированном контейнере. Все файлы с результатами работы находятся в папке проекта «…\CSA_HW3\outputs».

Формат аргументов командной строки и тестов

Для тестирования программы на тестах необходимо ввести следующую команду (i – число от 1 до 11):

```
-f tests/test<i>.txt outputs/output<i>.txt outputs/output_deleted<i>.txt
```

Для тестирования программы на случайно сгенерированных тестах необходимо ввести следующую команду:

```
-n <number> outputs/outputR.txt outputs/output_deletedR.txt
```

Также в обоих случаях допустимо указание других файлов вывода.

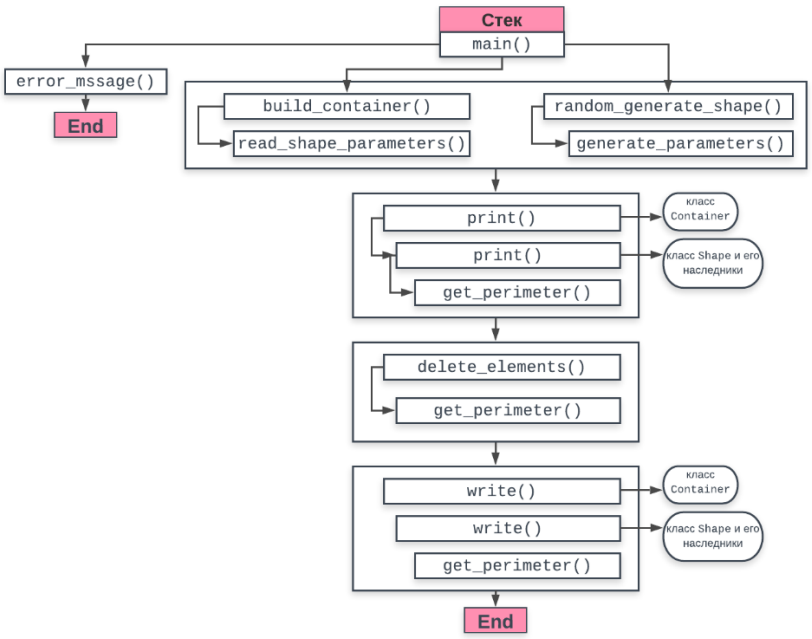
Фигуры в тестах написаны в следующем формате:

```
<figure_number> <color_number> <coordinates>
```

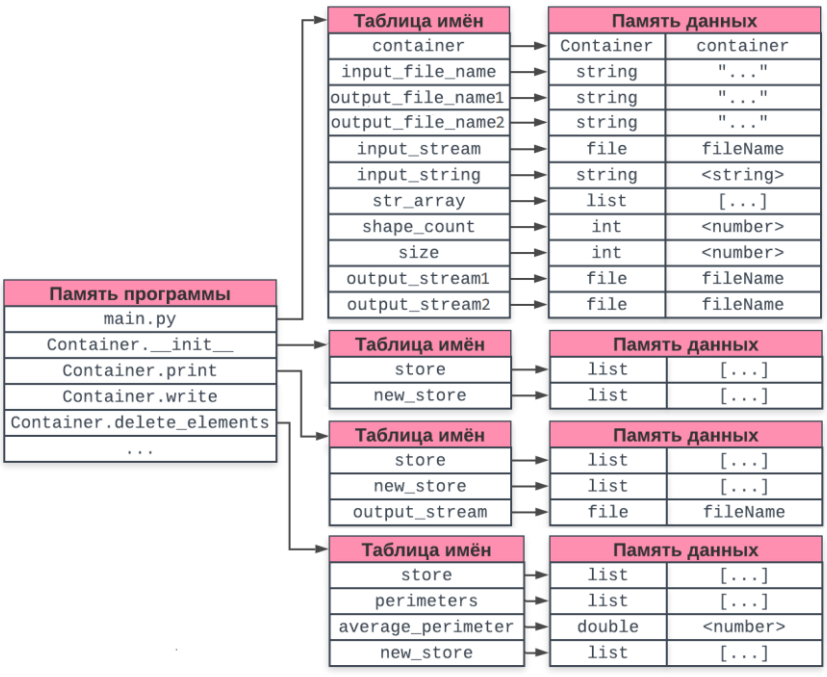
Номер фигуры – цифра от 1 до 3. Цифра 1 соответствует кругу, 2 – прямоугольнику, 3 – треугольнику. Номер цвета – цифра от 0 до 6, где каждая цифра соответствует цвету радуги по порядку. Координаты – числа, количество которых для каждой фигуры своё. У круга 3 числа в следующем порядке: x-координата цента, y-координата центра и радиус. У прямоугольника 4 числа в порядке x-координата левого верхнего угла, y-координата левого верхнего угла, x-координата правого нижнего угла, y-координата правого нижнего угла. У треугольника 6 чисел, аналогично по порядку три (x, y) координаты вершин.

Структурная схема изучаемой архитектуры ВС с размещенной на ней разработанной программы

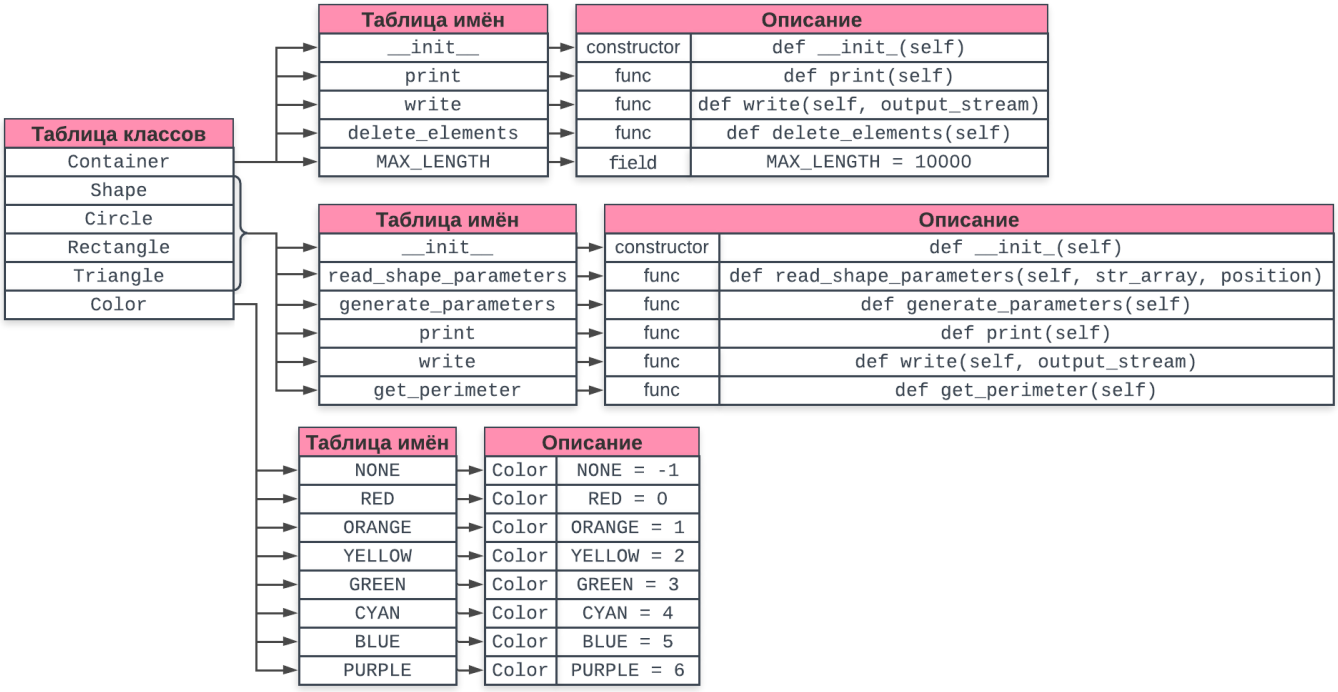
Стек вызовов методов:



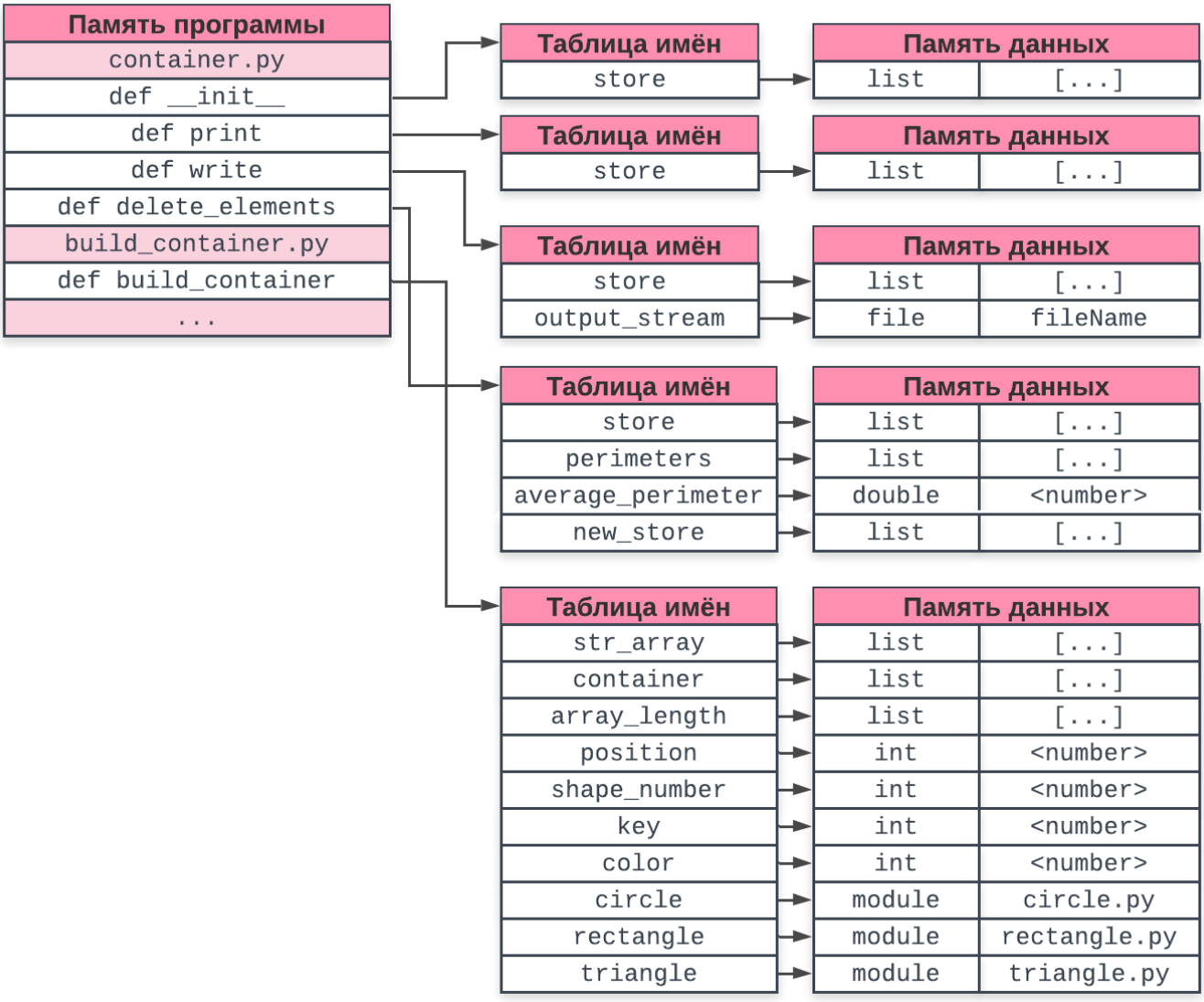
Отображение на память методов классов (несколько методов класса `Container`):



Отображение содержимого классов (всех классов программы):



Отображение на память содержимого модуля `container.py` и `build_container.py`.



Разница в работе данной и предыдущих программ

Программа с динамической типизацией и ООП работает значительно медленнее на больших тестах, чем программа со статической типизацией. Это связано с тем, что требуется время на определение интерпретатором всех типов во время работы программы, они не проверяются компилятором ещё до запуска, как это было в предыдущих двух домашних работах. Несмотря на то, что динамическая типизация значительно замедляет выполнение программы, её разработка с ООП позволила получить время меньшее, чем было зафиксировано в первой домашней работе с процедурным подходом, на маленьких тестах (< 100 фигур) и почти одинаковое на больших тестах (от 100 до 10000 фигур). Однако, размер исполняемого кода при ООП подходе получается в 1,5 раза больше.

Исходные тексты по сравнению с предыдущими домашними работами занимают меньше памяти, что может быть связано с тем, что динамическая типизация делает код лаконичным и простым.