Архитектура вычислительных систем.

Вариант 300

Задание 6, Функция 22

Выполнил:  
Лобанович Михаил Михайлович,  
группа БПИ-208

**Задача**

**Сущности:**

1. Самолеты (дальность полета – целое, грузоподъемность – целое)

2. Поезда (количество вагонов – целое)

3. Корабли (водоизмещение – целое; вид судна – перечислимый тип = (лайнер, буксир, танкер)

**Общие для всех альтернатив переменные:**

1. Скорость – целое;

2. Расстояние между пунктами отправления и назначения – действительное.

**Общая для всех альтернатив функция:**

Идеальное время прохождения пути (действительное число).

**Общая для все альтернатив структура:**

Транспорт.

### **Функционал**

1. Провести отладку и тестирование разработанной программы на заранее подготовленных тестовых наборах данных. При необходимости, программа должна правильно обрабатывать переполнение по данным. Тестовые наборы до 10 элементов должны вводиться из заранее подготовленных тестовых файлов с использованием сведений из рассматриваемой в задаче предметной области. Количество тестовых наборов данных (тестов, сделанных вручную), в которых четко представлены параметры предметной области (корректные названия, имена, фразы) – не менее пяти. Число уникальных элементов в тестовых наборах должно варьироваться от нуля до 10000. Тестовые данные с большим числом элементов должны порождаться программно с использованием генераторов случайных наборов данных. Данные, формируемые генератом случайных наборов должны поддерживать допустимые для обработки значения. Управление вводом данных задается из командной строки.
2. Использование модульной структуры и классов для представления программных объектов дает дополнительные бонусы к базовой оценке.
3. Описать архитектуру виртуальной Python машины с наложением на нее обобщенной схемы разработанной программы.
4. Зафиксировать для отчета основные характеристики программы, такие как: число интерфейсных модулей (заголовочных файлов) и модулей реализации (фалов с определением программных объектов), общий размер исходных текстов, полученный размер исполняемого кода (если он формируется), время выполнения программы для различных тестовых наборов данных.
5. Привести результаты сравнительного анализа полученных характеристик с теми, которые были получены для предыдущей программы. Сделать выводы о достоинствах и недостатках этого и предшествующего решения относительно друг друга.

**Команды для запуска**

**Случайная генерация**

-n \*число элементов( 0 < x < 10001)\* \*первый файл для вывода\* \*второй файл для вывода\*

**Чтение из файла**

-f \*имя файла с входными данными\* \*первый файл для вывода\* \*второй файл для вывода\*

--1--.

1. Интерфейсные модули:

Отсутствуют

2) Модули реализации

transport.py

plane.py

train.py

rnd.py

ship.py

container.py

build\_container.py

transport\_error.py

Всего: 8

3) Общий размер исходных текстов = 1 + 1 + 3 + 1 + 2 + 1 + 2 + 1 = 12КБ

4)Время выполнения программы для различных тестовых наборов данных: (тесты рассчитаны на значения в диапазоне от 1 до 20 вводимых элементов, начиная с 500 элементов программа тестировалась при помощи генерации случайных входных данных)

1 тест (1 элемент):

0.00096 sec

2 тест (3 элемента):

0.000996 sec

3 тест (5 элементов):

0.000997 sec

4 тест (10 элементов):

0.000999 sec

5 тест (15 элементов):

0.0009979 sec

6 тест (20 элементов):

0.000997 sec

7 тест (500 элементов):

0.0043 sec

8 тест(2500):

0.09737 sec

9 тест(5000 элементов):

0.1558 sec

10 тест(10000 элементов):

0.29712 sec

--2--.

Программа загружена в систему Github.

--3--.

Тесты находятся в папке tests

--4--.

1. Отображение на память методов классов (таблица для класса Container):

|  |  |  |
| --- | --- | --- |
| Память программы | Таблица имён | Память данных |
| 1. main.py | container | Container container |
|  | input\_stream | file file\_name |
|  | input\_string | file file\_name |
|  | Str\_array | List [...] |
|  | size | int <…> |
|  | input\_file | string “…” |
|  | output\_file1 | string “…” |
|  | output\_file2 | string “…” |
|  | output\_stream1 | file file\_name |
|  | output\_stream2 | file file\_name |
|  |  |  |
| 2)Container.print | store | List [..] |
|  |  |  |
| 3)Container.sort\_by\_perfect\_time | store | List [..] |
|  | get\_perfect\_time | List […] |
|  | avg\_time | double <…> |

**Таблица классов**

|  |
| --- |
| **Container** |
| def init(self) |
| def print(self) |
| def write(self, out\_stream) |
| def sort\_by\_perfect\_time(self) |
| field max\_len(int) |

|  |
| --- |
| **transport, plane, ship, train** |
| def init (self) |
| def read\_transport\_parameters(self, \*args) |
| def generate\_parameters(self) |
| def print(self) |
| def write(self, out\_stream) |
| get\_perfect\_time(self) (Только для transport) |
| field speed(int) |
| field distance(double) |

**Формат ввода:**

Для ввода одного транспорта следует ввести следующие данные:

Формат ввода меняется в зависимости от типа вводимого транспорта:

1. Самолет: X1 X2 X3 X4 X5,

Где X1 – 1, тип самолета

X2 – скорость самолета

X3 - расстояние между пунктами отправления и назначения

X4 – дальность полета

X5 - грузоподъемность

1. Поезд: X1 X2 X3 X4,

Где X1 – 2, тип поезда

X2 – скорость поезда

X3 - расстояние между пунктами отправления и назначения

X4 – количество вагонов

1. Корабль: X1 X2 X3 X4 X5,

Где X1 – 3, тип корабля

X2 – скорость корабля

X3 - расстояние между пунктами отправления и назначения

X4 – водоизмещение

X5 – тип корабля из перечисляемых

**Расположение входных и выходных данных**

Входные данные лежат в:

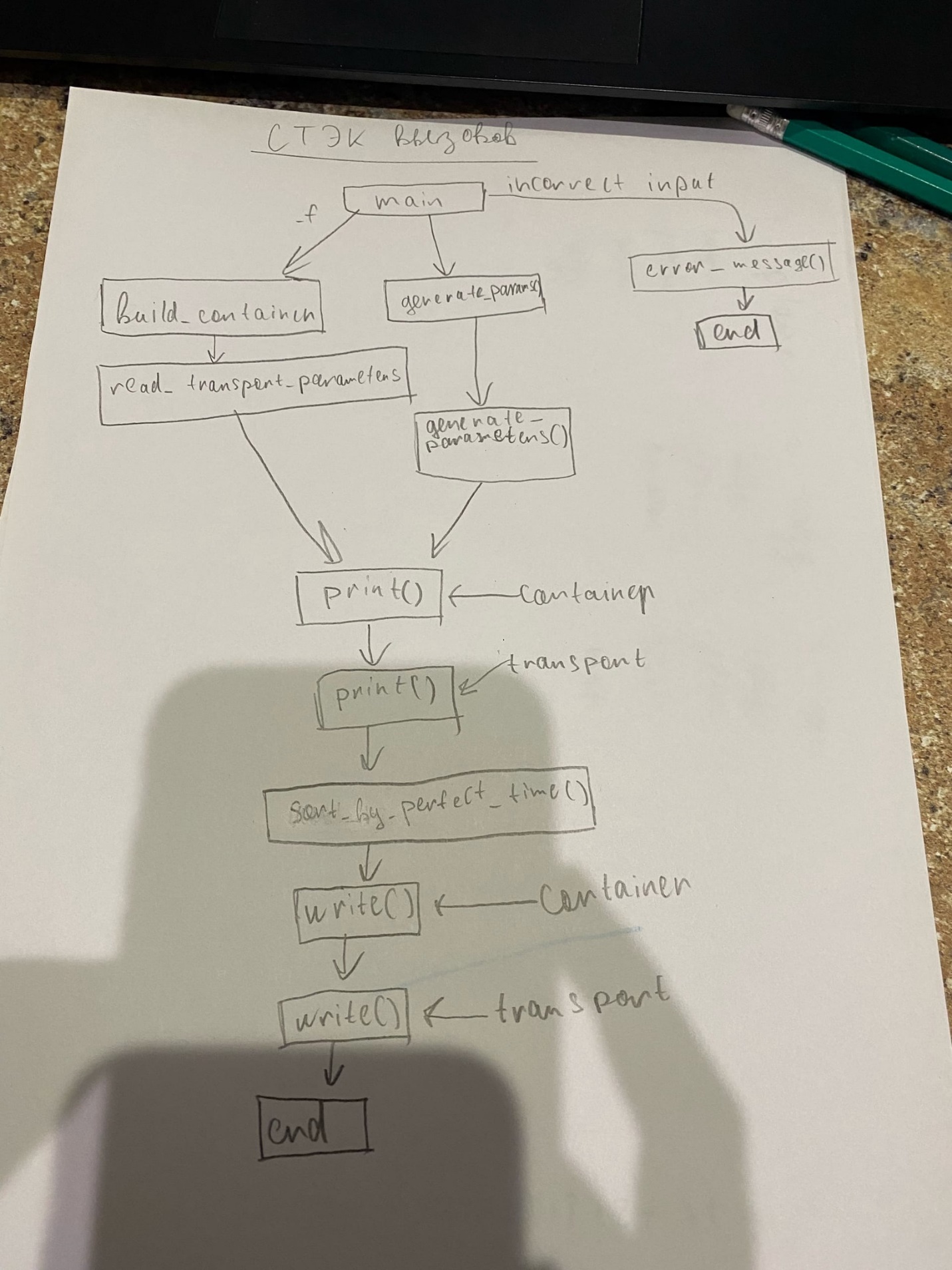
tests/input

Выходные лежат в:

tests/output

**Cхема**

1. HEAP (Куча) - int\*
2. Стек



**Сравнительная таблица**

**По памяти**

|  |  |
| --- | --- |
| Статическая типизация | Динамическая типизация |
| 24кб | 12кб |

**По времени**

|  |  |
| --- | --- |
| Статическая типизация | Динамическая типизация |
| 0.001 | 0.00096 |
| 0.001 | 0.000996 |
| 0.001 | 0.000997 |
| 0.001 | 0.000999 |
| 0.001 | 0.0009979 |
| 0.001 | 0.000997 |
| 0.003 | 0.0043 |
| 0.01099 | 0.09737 |
| 0.021 | 0.1558 |
| 0.04599 | 0.2971 |

Из таблицы можно понять, что при динамической типизации размер исполняемых файлов значительно меньше. Код получился более компактным. В то же время на тестах с большим количеством элементов время работы программы с динамической типизацией значительно увеличилось, из чего можно сделать вывод, что данный подход является более медленным для решения задачи.