Архитектура вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ПРАКТИЧЕСКОЙ РАБОТЕ НА ТЕМУ:

«Архитектура ВС с динамической типизацией»

Работу выполнил:

студент группы БПИ207
Пендищук Владислав

Москва
2021 г.

ОГЛАВЛЕНИЕ

ОПИСАНИЕ ЗАДАНИЯ

Задача состояла в разработке программного продукта с динамической типизацией на языке Python.

Требования к функционалу

В соответствии с полученным вариантом задания (258) функционал, требуемый к реализации в программе, содержал в себе следующие пункты:

1. Реализация обобщённого артефакта – пассажирского транспорта, и его параметров:
    a. Скорость – целое число;
    b. Расстояние между пунктами отправления и назначения – действительное число;
2. Реализация базовых альтернатив и уникальных параметров, задающих их отличительные признаки:
    a. Самолёт, отличительные признаки:
        i. Дальность полёта – целое число;
        ii. Грузоподъёмность – целое число.
    b. Поезд, отличительные признаки:
        i. Количество вагонов – целое число.
    c. Корабль, отличительные признаки:
        i. Водоизмещение – целое число;
        ii. Вид судна – перечислимый тип (лайнер, буксир, танкер).
3. Реализация общей для всех альтернатив функции – вычисления идеального времени прохождения пути (действительное число).
4. Реализация контейнера для объектов типа обобщённого артефакта с массивом фиксированной длины в своей основе.
5. Реализация функции удаления из контейнера тех элементов, 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.

Требования к запуску и вводу\выводу

К процессу запуска программы и ввода\вывода при работе с ней были представлены следующие требования:

1. Запуск программы осуществляется из командной строки, в которой указываются: имя запускаемой программы; имя файла с исходными данными; имя файла с выходными данными.
2. Для каждого программного объекта, загружаемого в контейнер, исходный файл с тестовым набором должен содержать: признак

альтернативы, а также список параметров, необходимых этой альтернативе. Этот список должен быть представлен в формате, удобном для обработки компьютером.

3. При больших данных во входном файле должны быть указаны только параметры для генератора случайных наборов данных, который и заполняет контейнер.
4. В выходной файл необходимо вывести введенные в контейнер данные. Помимо этого, необходимо вывести информацию об общем количестве объектов, содержащихся в контейнере. После этого в тот же файл необходимо вывести новые данные в соответствии с результатами, полученными в ходе работы программы. Информация для вывода должна быть представлена в форме, удобной для восприятия пользователем.

СТРУКТУРА ИЗУЧАЕМОЙ АРХИТЕКТУРЫ ВС

Объектом изучения в данной работе являлась динамически типизированная архитектура ВС. Разработка велась на языке Python с применением объектно-ориентированной парадигмы программирования. Отобразим данную архитектуру на обобщённой схеме разработанной программы на примере 4 функций и таблицы классов.

Схема №1 – main.py

На данной схеме в обобщённом виде на память отображена функция, содержащаяся в main.py, для случая успешного чтения из файла:

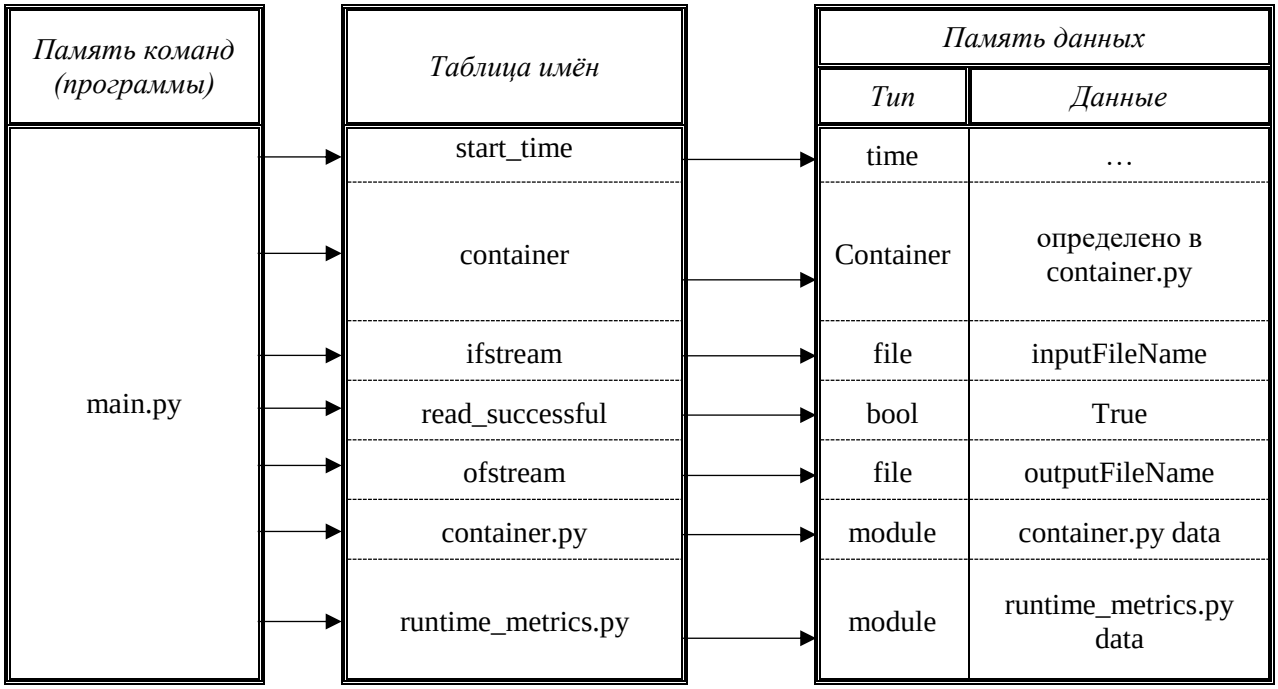
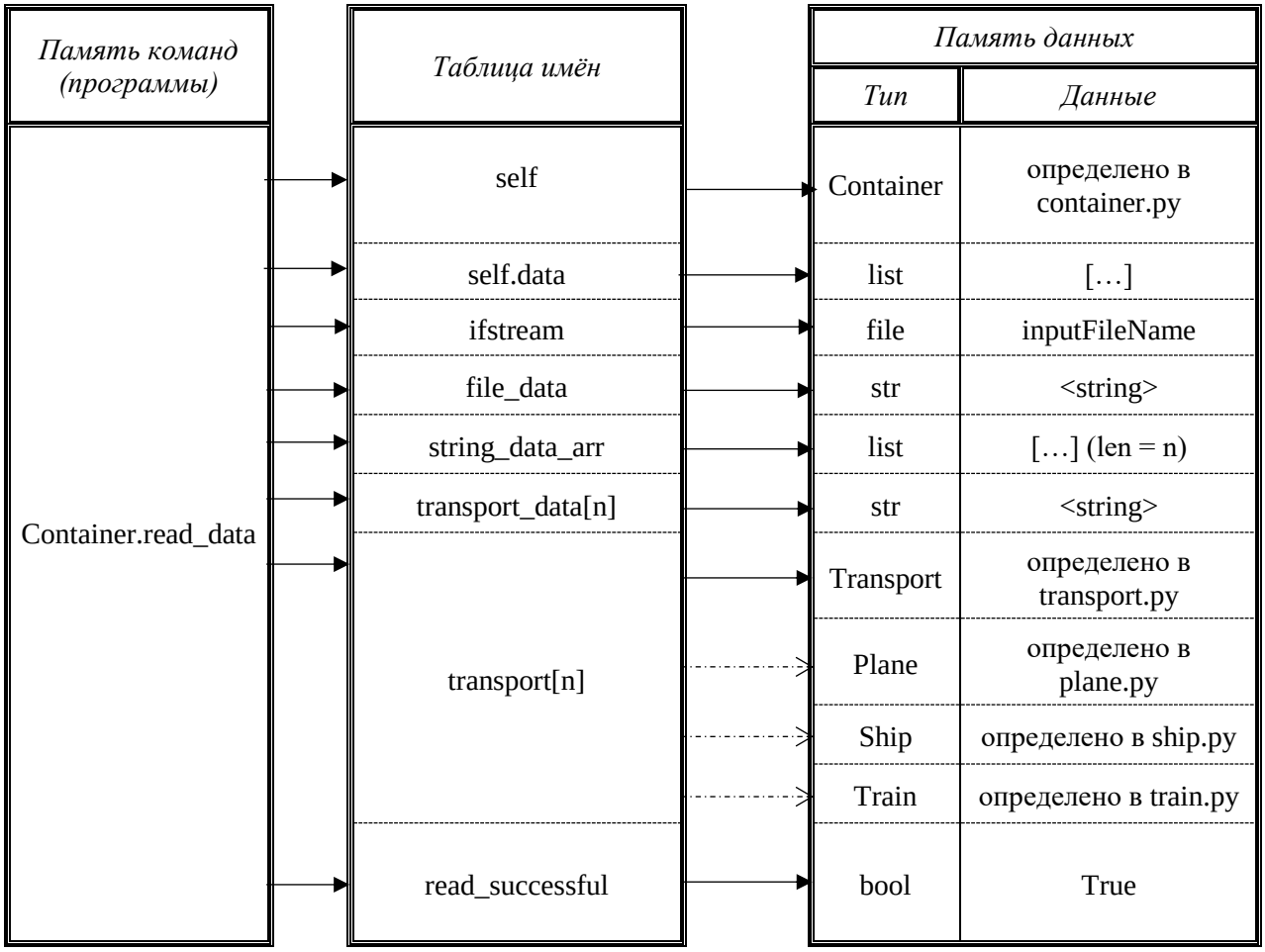


Схема №2 – Container.read_data

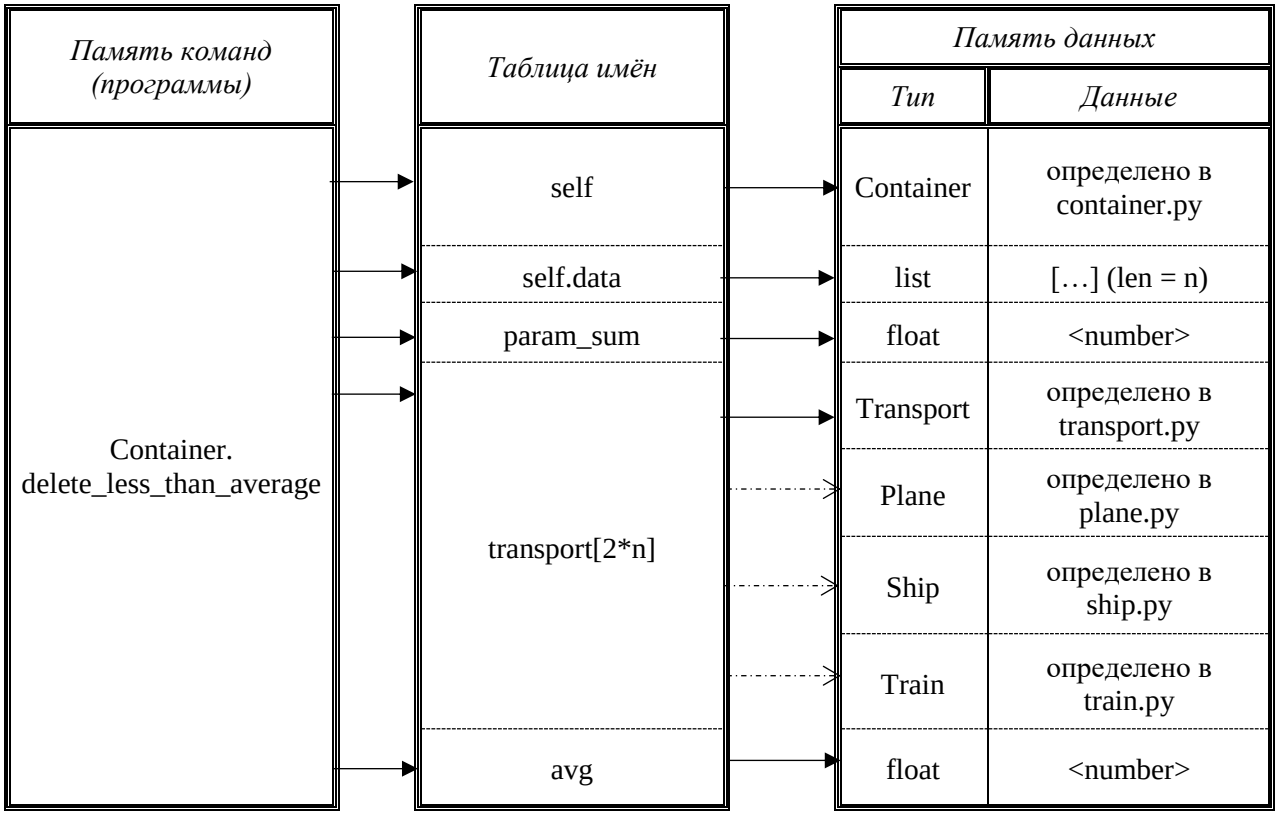
На данной схеме в обобщённом виде на память отображена функция Container.read_data для случая с успешным чтением данных из файла:



Примечание: символьным обозначением *[n]* обозначено, что переменные создаются в цикле с n повторениями.

Схема №3 – Container. delete_less_than_average

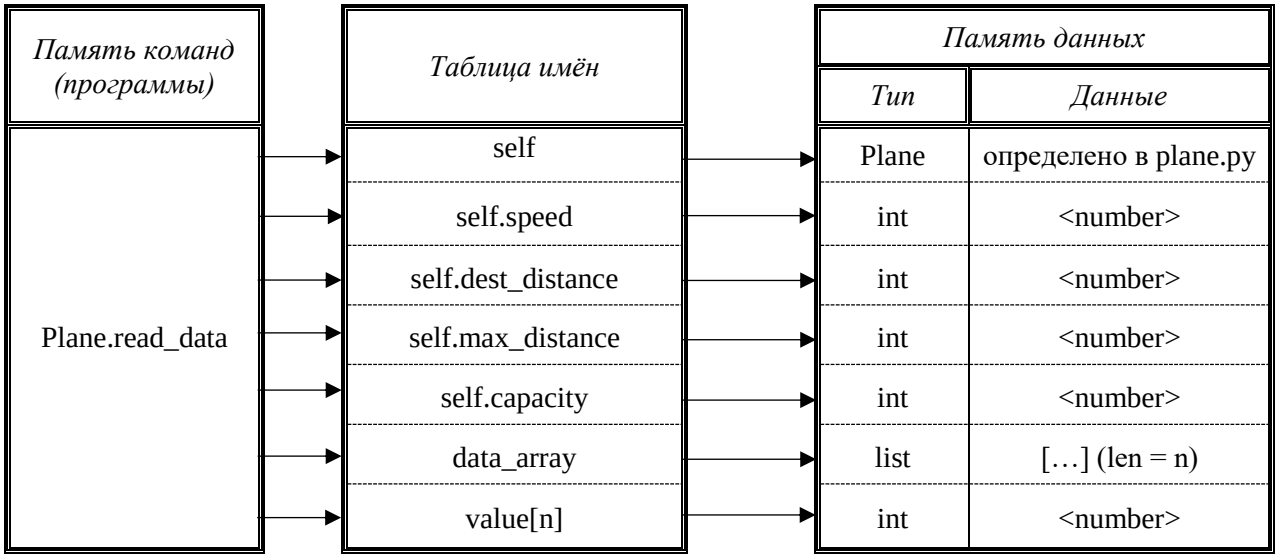
На данной схеме в обобщённом виде на память отображена функция Container.delete_less_than_average:



Примечание: символьным обозначением *[n]* обозначено, что переменные создаются в цикле с n повторениями. В данном случае переменная transport используется в двух циклах с n повторениями.

Схема №4 – Plane.read_data

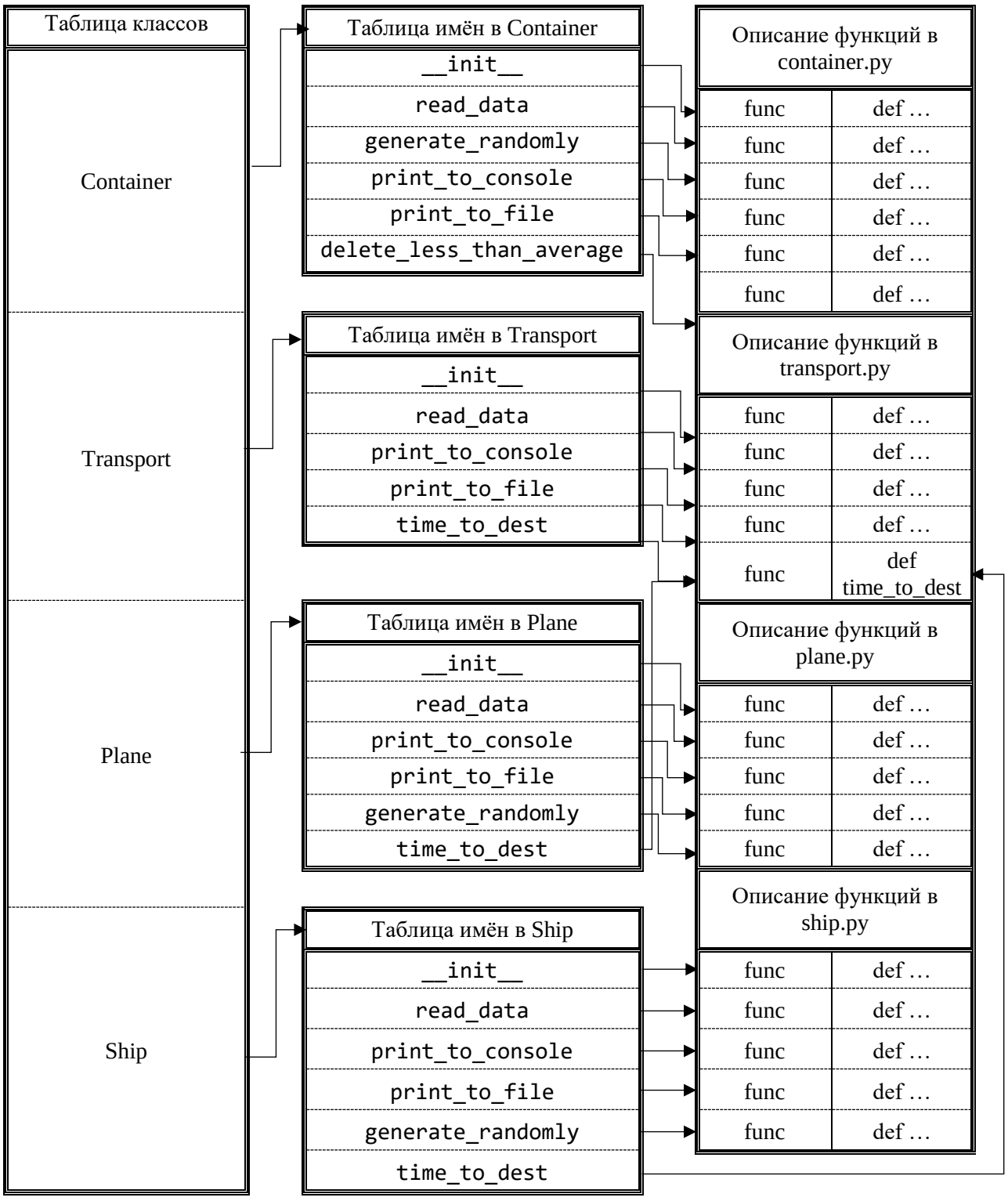
На данной схеме в обобщённом виде на память отображена функция Plane.read_data для случая с успешным чтением данных из файла:



Примечание: символьным обозначением *[n]* обозначено, что переменные создаются в цикле с n повторениями.

Схема №5 – Таблица классов

На данной схеме в обобщённом виде изображена таблица классов и отображение содержимого классов в ней (их функциональных членов) на память:



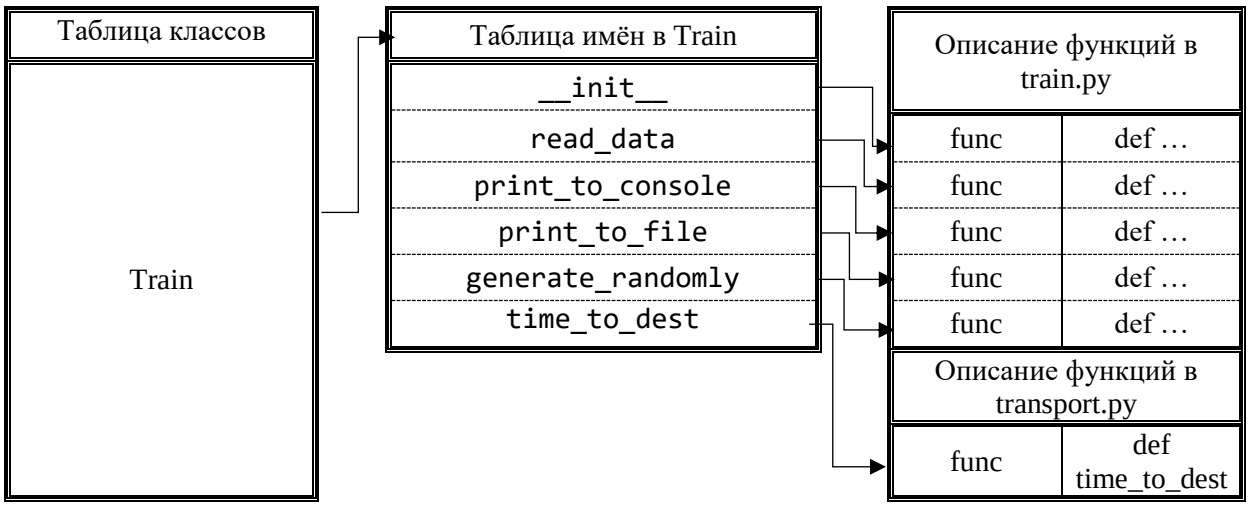


Таблица классов
Train
Таблица имён в Train
__init__
read_data
print_to_console
print_to_file
generate_randomly
time_to_dest
Описание функций в train.py
func
def …
func
def …
func
def …
func
def …
func
def …
Описание функций в transport.py
func
def time_to_dest

ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ПО

Исходный код программы содержится в 9 файлах. Все из них являются модулями реализации - файлами .py с определением программных объектов. Общий размер исходных текстов следующий:

	С форматированием и комментариями	*Без форматирования и комментариев*
Количество строк	557	325

Исполняемый файл в силу используемого языка программирования не собирался.

Результаты тестов с использованием тестовых файлов в соответствующих директориях следующие:

Тестовый файл	Тестовый кейс	Время работы программы
correct/test1.txt	12 корректных элементов	0.00099 секунды
correct/test2.txt	19 корректных элементов	0.00099 секунды
correct/test3.txt	13 корректных элементов	0.00099 секунды
error/test1.txt	Некорректное число аргументов в строке	0.00099 секунды
error/test2.txt	Некорректный идентификатор альтернативы	0.00099 секунды
error/test3.txt	Удалённые параметры	0.00099 секунды

Результаты тестов с использованием случайной генерации объектов контейнера следующие:

Количество элементов	*Время работы программы*
20 элементов	0.0009 секунды
1000 элементов	0.0369 секунды
10000 элементов	0.601 секунды

СРАВНИТЕЛЬНАЯ ХАРАКТЕРИСТИКА ПО

Ранее в ходе курса программа с идентичным функционалом была разработана на языке C с применением процедурного подхода и на языке C++ с применением объектно-ориентированного подхода для статически типизированной архитектуры ВС. От данной реализации предыдущие прежде всего отличались принципом типизации, так как в использованных ранее языках она была статической. В данной характеристике не будет рассматриваться отличие от реализации программы на C в парадигме ввиду того, что они не входят в рамки объекта изучения в данной работе. Таким образом, в рассматриваемой реализации можно выделить следующие различия:

1. Тип объектов, используемых в программе, определяется в течении времени исполнения, в отличие от статически типизированной архитектуры, где он задавался в коде программы при описании переменных. Данный подход позволяет уменьшить объём кода и изменять тип объектов в ходе выполнения программы, однако в значительной мере уменьшает её производительность из-за дополнительной нагрузки связанной с необходимостью интерпретации типа значений, присваиваемых переменным.
2. Некоторые типы объектов в статически типизированных языках хранятся в стеке, в то время как при динамической типизации все значения хранятся в куче до удаления garbage collector’ом, что дополнительно накладывает штраф на производительность.
3. Функции и операции в статически типизированных языках строго привязаны своей сигнатурой к типам аргументов и не могут принять аргументы иных типов. В динамически типизированной архитектуре функции могут принимать в качестве аргумента объект любого типа и проводить операции с ним за счёт принципа утиной типизации.

Сравнение характеристик текущей реализации с предыдущей также выявило ряд различий:

1. Объём исходных текстов значительно уменьшился, что связано со сменой языка программирования и, соответственно, используемым архитектурой способом задания однозначности.
2. Исполняемый файл в данном случае не был сгенерирован ввиду особенностей используемого языка.
3. Время работы программы значительно увеличилось во всех случаях. Различия во времени исполнения варьируется от 600 до 1200% (6-12 раз). Причины падения производительности описаны выше в сравнении механизмов типизации в использованных языках. Таким образом,

наиболее быстрой остаётся реализация с использованием процедурного подхода в статически типизированной архитектуре, в то время как динамически типизированная архитектура стала наиболее медленной в данном показателе.

Тестовый файл	*Тестовый кейс*	*Время работы программы*
correct/test1.txt	5 корректных элементов	0.00021 секунды
correct/test2.txt	19 корректных элементов	0.00024 секунды
correct/test3.txt	13 корректных элементов	0.00022 секунды
error/test1.txt	Некорректное число аргументов в строке	0.00013 секунды
error/test2.txt	Некорректный идентификатор альтернативы	0.00019 секунды
error/test3.txt	Удалённая строка параметров	0.00016 секунды

Количество элементов	*Время работы программы*
20 элементов	0.00023 секунды
1000 элементов	0.00298 секунды
10000 элементов	0.13457 секунды

Рис. 1: время выполнения программы для реализации на языке C (процедурный подход)

Тестовый файл	Тестовый кейс	Время работы программы
correct/test1.txt	12 корректных элементов	0.00025 секунды
correct/test2.txt	19 корректных элементов	0.00022 секунды
correct/test3.txt	13 корректных элементов	0.00022 секунды
error/test1.txt	Некорректное число аргументов в строке	0.00011 секунды
error/test2.txt	Некорректный идентификатор альтернативы	0.00008 секунды
error/test3.txt	Удалённая строка параметров	0.00014 секунды

Количество элементов	*Время работы программы*
20 элементов	0.0003 секунды
1000 элементов	0.086 секунды
10000 элементов	0.15291 секунды

Рис. 2: время выполнения программы для реализации на языке C++ (объектно-ориентированный подход)