Сухова Анна БПИ207

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

1. Описание задания

В рамках задания необходимо:

1. Использовать:

• вычислительную систему с архитектурой x86-64;

• язык программирования Python;

2. Разработать программу в виде консольного приложения с использованием динамически типизированного универсального языка программирования (объектно-ориентированный подход).

Запуск программы должен осуществляться из командной строки, в которой указываются: имя запускаемой программы; имя файла с исходными данными; имя файла с выходными данными.

Примеры:

command -f infile.txt outfile1.txt outfile2.txt (для ввода из файла)

command -n number_of_elements outfile1.txt outfile2.txt (для случайной генерации)

3. Реализовать в программе следующий функционал (**вариант 13**):

Обобщённый артефакт	Базовые альтернативы (и их отличительные параметры)	Общая для всех альтернатив переменная	Общая для всех альтернатив функция
Растение	1. Деревья (возраст – длинное целое) 2. Кустарники (месяц цветения – перечислимый тип) 3. Цветы (домашние, садовые, дикие –	Название – строка символов (макс. длина = 20 символов)	Частое от деления числа гласных букв в названии на общую длину названия

	перечислимый тип)		

4. Поместить данные объекты в контейнер и в соответствии с вариантом задания (**вариант 19**) 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 той же функции.

5. Провести отладку и тестирование разработанной программы на заранее подготовленных тестовых наборах данных (не менее 5). Тестовые данные с большим числом элементов должны порождаться программой с использованием генераторов случайных наборов данных. Управление вводом данных задается из командной строки.

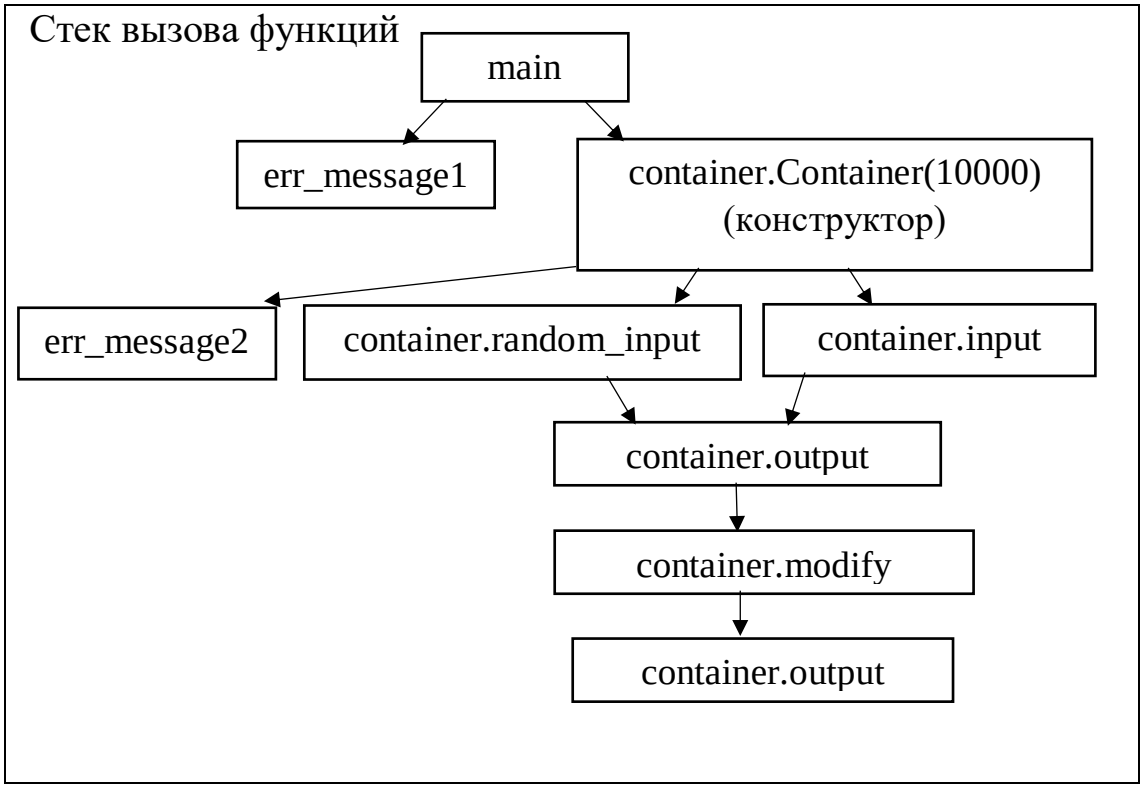
6. Создать отчёт по выполненному заданию, описав структуру используемой ВС с наложением на нее обобщённой схемы разработанной программы и зафиксировав основные характеристики программы.

7. Привести результаты сравнительного анализа полученных характеристик с теми, которые были получены для предыдущих программ. Сделать выводы о достоинствах и недостатках этого и предшествующих решений относительно друг друга.

2. Схема архитектуры ВС с размещённой на неё разработанной программой.

1. Схема памяти функции main.

Память программы
if __name__ == '__main__': …



Переменные функции	
argv	list
container	Container
size	int
input_file	_io.TextIOWrapper
output_file1	_io.TextIOWrapper
output_file2	_io.TextIOWrapper

Глобальная память

2. Схема памяти функции по подсчёту среднего значения.

Память программы

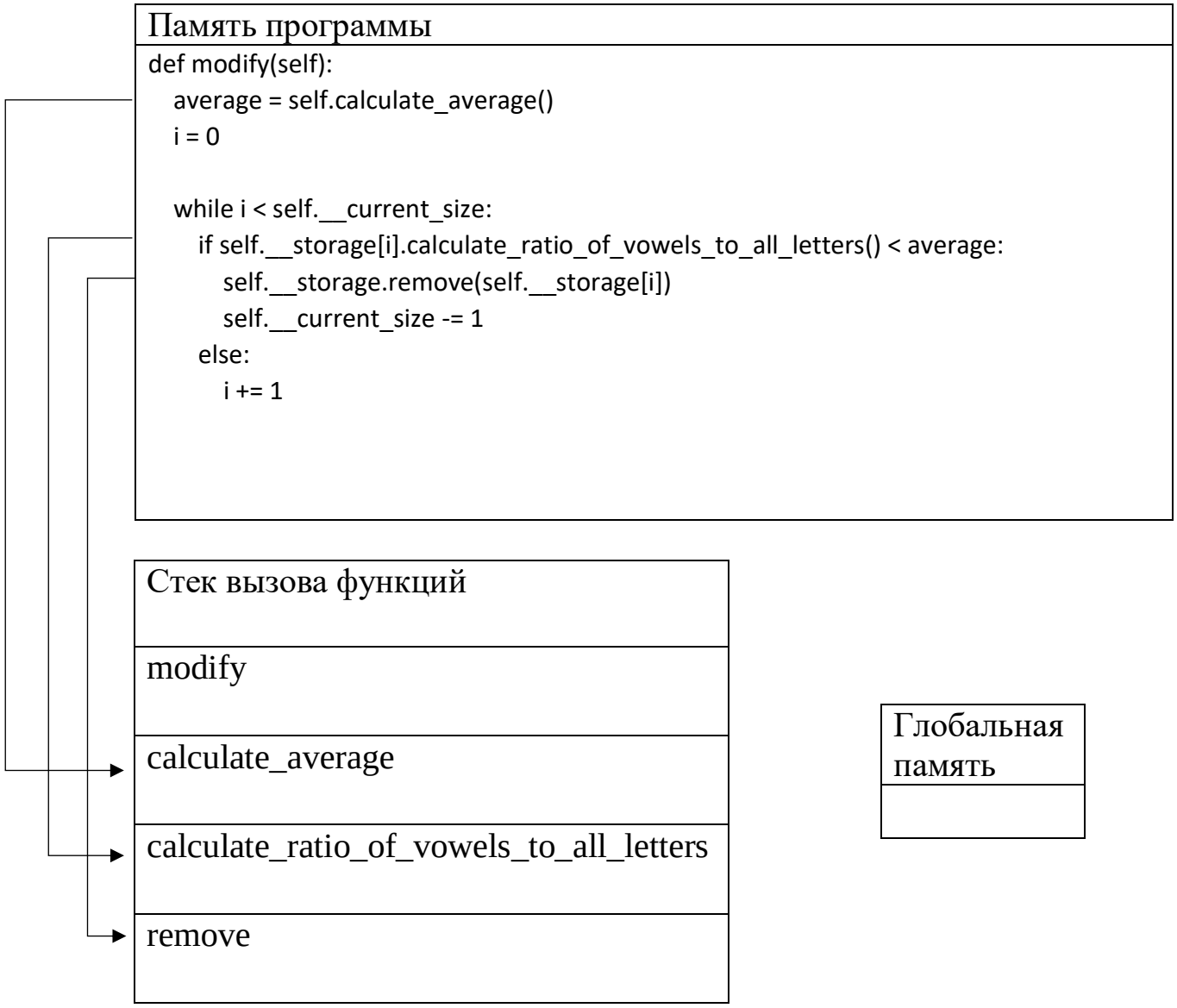
def calculate_average(self) -> Decimal:
  sum_of_results = Decimal("0")
 	
  for i in range(self.__current_size):
  sum_of_results += self.__storage[i].calculate_ratio_of_vowels_to_all_letters()
 	
  if self.__current_size == 0:
  return Decimal("0")
  return Decimal(sum_of_results) / Decimal(self.__current_size)

Стек вызова функций
calculate_average
calculate_ratio_of_vowels_to_all_letters

Глобальная память

Переменные функции	
sum_of_results	Decimal
self	Container
self.__current_size	int
self.__storage	list

3. Схема памяти функции по модификации контейнера в соответствии с вариантом 19.



Переменные программы (размер в байтах)	
average	Decimal
i	int
self	Container
self.__current_size	int
self.__storage	list

Таблица типов:

Тип	
int	<number>
double	<number>
Decimal	<class>
string	“…”
Container	<class>
list	[…, …]
Переменная	**Тип**
class Container	
__current_length	int
__storage	list
class Plant	Abs
__name	string
class Tree	class
__age	int
class Flower	class
class FlowerKey	Enum
__flower_type	FlowerKey
class Bush	class
class BushKey	Enum
__flowering_month	BushKey
argv	list
container	Container
size	int
input_file	_io.TextIOWrapper
output_file1	_io.TextIOWrapper
output_file2	_io.TextIOWrapper

Описание класса Container:

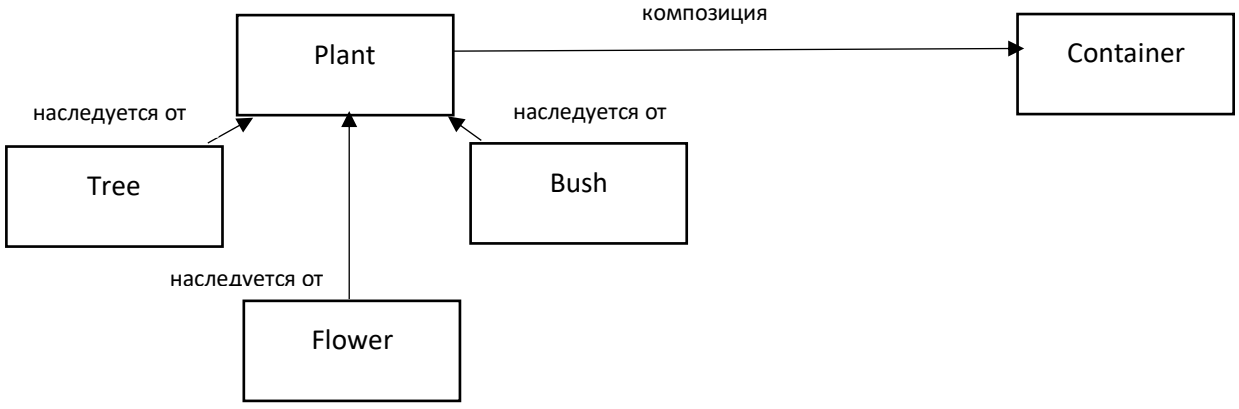
Методы	
__init__	Экземплярный метод (конструктор)
input	Экземплярный метод
random_input	Экземплярный метод

output	Экземплярный метод
modify	Экземплярный метод
calculate_average	Экземплярный метод
static_input	Статический метод
static_random_input	Статический метод
__del__	Экземплярный метод (деструктор)

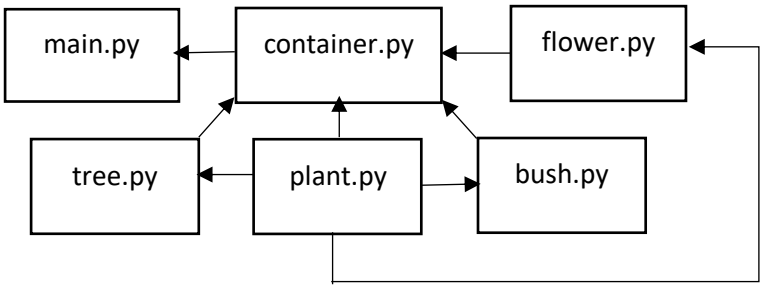
Описание класса Plant:

Методы	
__init__	Экземплярный метод (конструктор)
input	Абстрактный метод
random_input	Абстрактный метод
output	Абстрактный метод
generate_random_name	Статический метод
calculate_ratio_of_vowels_to_all_letters	Экземплярный метод
__del__	Экземплярный метод (деструктор)

Диаграмма классов:



Файловая схема:



3. Основные характеристики программы:

Число заголовочных файлов (библиотек): 6 (sys, typing.io, enum, random, decimal, abc)
Число модулей: 6
Общий размер исходных текстов программы: 12,7 Кбайт
Полученный размер исполняемого кода: 7144 Кбайт (.exe) (Python – интерпретируемый язык, не требует обязательного создания .exe файла)

Время выполнения программы для тестовых наборов данных:

№ теста	Время в задании №3 (в секундах)	Время в задании №2 (объектно-ориентированный подход на C++)	Время в задании №1 (процедурный подход на C++)
1	0.0026	0.000982	0.001703
2	0.0023	0.000843	0.00148
3	0.0033	0.000945	0.002179
4	0.0025	0.00132	0.01543
5	0.5483	0.1245	0.16013
6	0.1926	0.038996	0.081633
7	0.0329	0.00443	0.014050
8	0.0026	0.000752	0.001601
9	0.5763	0.11506	0.332784

4. Сравнительный анализ с предыдущими программами (из заданий №1, №2)

При сравнении программы на Python (объектно-ориентированный подход) и программ на C++ (объектно-ориентированный и процедурный подходы), решающих одну задачу, можно выделить следующие различия:

1. По таблице времени выполнения программы для тестовых наборов данных видно, что на одинаковых тестах программа на Python

выполняется медленнее программ на C++, и различия в скорости возрастают с увеличением количества обрабатываемых элементов контейнера.
2. Для описания данных сущностей (растение, дерево, цветок, кустарник, контейнер) при использовании ООП используются классы, между тем как при процедурном подходе используются структуры.
3. Так как Python динамически типизированный язык, типы переменных в программе не указываются явно (в отличие от C++), что может усложнить понимание кода.
4. В Python реализована автоматическая сборка мусора, что позволяет не создавать деструкторы в классах.
5. Благодаря наследованию и полиморфизму код с объектно-ориентированным подходом получается более читабельным (видно, что цветок, кустарник, дерево наследуются от класса растения => являются растениями и получают функционал родителя, но с разной его реализацией).
6. Размер исполняемого кода при объектно-ориентированном подходе на Python (7144 Кбайт) значительно больше наибольшего .exe файла программы на C++ (112 Кбайт).
7. Схемы подключения файлов различаются (файловая схема, пункт 2), так как в программе на Python нет заголовочных файлов (.h).
8. Количества модулей одинаковы у всех трёх программ.

Таким образом, программа на Python с применением объектно-ориентированного подхода, не эффективна по времени и памяти (если создавать .exe файл) в сравнении с программами на C++, а значит, наименее всего подходит для решения данной задачи.