Тимохина Софья Константиновна, БПИ207

Пояснительная записка к домашнему заданию 3.

Архитектура ВС с динамической типизацией

Описание задания

Вариант 267, задание 1, функция 20.
Программа должна содержать следующие структуры:

Обобщенный артефакт, используемый в задании	Базовые альтернативы (уникальные параметры, задающие отличительные признаки альтернатив)
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)

Общая для всех альтернатив переменная - цвет фигуры, перечислимый тип = {красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый}.
Общая для всех альтернатив функция: вычисление площади фигуры (действительное число).
Дополнительная функция: удаление из контейнера тех элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, то есть функция вычисления площади, больше, чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы передвинуть к началу контейнера с сохранением порядка.

Также нужно:

1. Провести отладку и тестирование разработанной программы на заранее подготовленных тестовых наборах данных. Количество тестовых наборов данных – не менее пяти. Число уникальных элементов в тестовых наборах должно варьироваться от нуля до 10000. При необходимости программа должна правильно обрабатывать переполнение по данным. Тестовые наборы до 20 элементов должны вводиться из заранее подготовленных тестовых файлов. Тестовые данные с большим числом элементов должны порождаться программно с использованием генераторов случайных наборов данных. Данные

формируемые генератом случайных наборов должны поддерживать допустимые значения. Управление вводом данных задается из командной строки.

2. Описать структуру используемой ВС с наложением на нее обобщённой схемы разработанной программы.
3. 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.

Структурная схема архитектуры ВС

Разработка велась на языке Python версии 3.9

Таблица классов	Таблица имен	Память данных	
Container	__shapes	list	[Shape]
	__init__	func	def
	in_from_file	func	def
	in_random	func	def
	out	func	def
	average_area	func	def
	remove_items_with_ area_bigger_than_average	func	def
(Enum class) Color	cases (red, orange, yellow, green, light_blue, dark_blue, purple)	int	<number>
	get_color_from_int	func	def
Point	x	int	<number>
	y	int	<number>
	to_string	func	def
Shape	color	Enum	class
	__init__	func	def
	to_string	func	def
	area	func	def
Circle: Shape	__radius	int	<number>
	__center	Point	class
	color	Enum	class
	__init__	func	def
	to_string	func	def
	area	func	def
	get_random_circle	func	def
Triangle: Shape	__p1	Point	class
	__p2	Point	class
	__p3	Point	class
	color	Enum	class
	__init__	func	def
	to_string	func	def
	area	func	def
	get_random_triangle	func	def

Rectangle	__p1	Point	class
	__p2	Point	class
	color	Enum	class
	__init__	func	def
	to_string	func	def
	area	func	def
	get_random_rectangle	func	def

Отображение на память содержимого модулей

Память программы	Таблица имен	Память данных	
main.py			
def main	arvg	list	[str]
	start_time	datetime	datetime
	input_file	FileIO	class
	container	Container	class
	container.py	module	container.py
	output_file_1	FileIO	class
	output_file_2	FileIO	class
container.py			
def __init__	self	reference	Shape
	shapes	list	[Shape]
def average_area	self	reference	Shape
	__shapes	list	[Shape]
	area_sum	float	<number>
def remove_items_with_area_bigger_than_average	self	reference	Container
	__shapes	list	[Shape]
	current_len	int	<number>
def in_random	cls	reference	Container
	count	int	<number>
def out	self	reference	Container
	file	TextIO	class
def in_from_file	cls	reference	Container
	lines	list	[str]
	shapes	list	[Shape]
	shape_number	int	<number>
	file	TextIO	class
shape.py			
def __init__	self	reference	Shape
	color	Color	class
def to_string	self	reference	Shape

Основные характеристики программы

Число заголовочных файлов: 0
Число модулей реализации: 10
Общий размер кода: 351 строка (19 КБ)
Размер исполняемого файла: 0
Время выполнения программы для различных тестов:

Номер теста	Время (с)
1	0.000742
2	0.002105
3	0.001881
4	0.001971
5	0.00273

Время выполнения программы для автогенерируемых тестов:

Количество элементов	Время (с)
50	0.002847
100	0.00499
2000	0.103762
5000	0.137004
10000	0.288199

Сравнительный анализ

В ходе предыдущих домашних заданий были проработаны архитектуры статически типизированного универсального языка программирования, ориентированные на процедурный подход и ООП. В данном домашнем задании была разработана архитектура ВС с динамической типизацией.

В статической типизации типы для переменных и функций определяются на этапе компиляции, а в динамической типизации все типы определяются во время выполнения программы. Так, например, в случае обращения к переменной типа null (которой не присвоено значение или же ссылка равна nullptr) при использовании статически типизированного языка C++ могут возникнуть проблемы или предупреждения уже на этапе компиляции, при использовании динамически типизированного языка Python ошибка возникнет сразу после обращения к переменной, и программа приостановится.

Нетрудно заметить, что при использовании статической типизации программа отрабатывала в разы быстрее. Так, например, для генерации 10.000 элементов при использовании программы, написанной на C++ требовалось около 37 миллисекунд, то при использовании программы, написанной на Python, потребовалось 288 миллисекунд - то есть практически в 8 раз больше.

Также сократился общий размер кода: с 560 (ООП) и 667 (процедурный подход) до 351 строки - практически в два раза.

Как мне показалось, программы, написанные на Python, гораздо проще читать, отчасти ввиду синтаксиса, отчасти ввиду отсутствия заголовочных файлов. Помимо этого, программы на Python писать чуть легче и приятнее, потому что не нужно работать с указателями и очищать память. Из минусов отметила только долгую работу программы