Описание задания:

Номер варианта: 298
Номер задачи: 4
Номер функции обработки данных в контейнере: 22

Обобщенный артефакт, используемый в задании	**Базовые альтернативы**	**Общие переменные**	**Общая функция**
4. Объемная (трехмерная) геометрическая фигура.	**1. Шар (целочисленный радиус) 2. Параллелепипед (три целочисленных ребра) 3. Правильный тетраэдр (длина ребра – целое)**	**Плотность материала фигуры (действитель ное число)**	**Вычисление площади поверхности (действительн ое число)**

Функция:

22.Переместить в конец контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, больше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы сдвинуть к началу без изменения их порядка.

Ниже приведена таблица типов:

Таблица типов			
Название	Размер, Байт	Название	Размер, Байт
base type: int	4	Class Ball	32
base type: bool	1	radius: int	4
base type: double	8	density: double	8
base type: char**	8	Class Barallelepiped	32
base type: FILE	216	edge_a: int	4
Class container	320004	edge_b: int	4
len: int	4	edge_c: int	4
cont: **shape	320000	density: double	8
Class Random	8	Class Betrahedron	32
		edge: int	4
ball: Ball	32	density: double	8
parallelepiped: Parallelepiped	32		
tetrahedron: Tetrahedron	32		

Ниже приведено описании работы функции main в рамках архитектуры:

Heap
argv[0] = "/bin/project"
argv[1] = "-f"
argv[2] = "iofiles/test1.txt"
argv[3] = "iofiles/out.txt"
argv[4] = "iofiles/out2.txt"

Stack
ShowErrorMessage
Init
InContainer
InRndContainer
OutContainer
RearrangeContainer
OutContainer
Clear

Память программы	
исходный файл main.cpp	размер
average: double	8
time: unsigned int	4
argc: int	4
argv: char**	8
container: container	320004
p_file: FILE	216
is_file_input: bool	1
size: int	4

Глобальная память
-

Описание работы функции RearrangeContainer в рамках архитектуры:

Stack
RearrangeContainer
Square?
memcpy

Heap
c: container

Глобальная память
-

Память программы	
void RearrangeContainer(…)	
	Память, Байт
c: container	320004
average: double	8
size: int	4
cpy_array: shape*	320004
i: int	4
j: int	4

Примечание: вопросительный знак означает, что вызов функции зависит от условий.

Описание работы функции AverageMean в рамках архитектуры:

Stack
AverageMean
Square

Heap
c: container

Глобальная память
-

Память программы	
double AverageMean(…)	
	Память, Байт
c: container	320004
sum: double	8
i: int	4

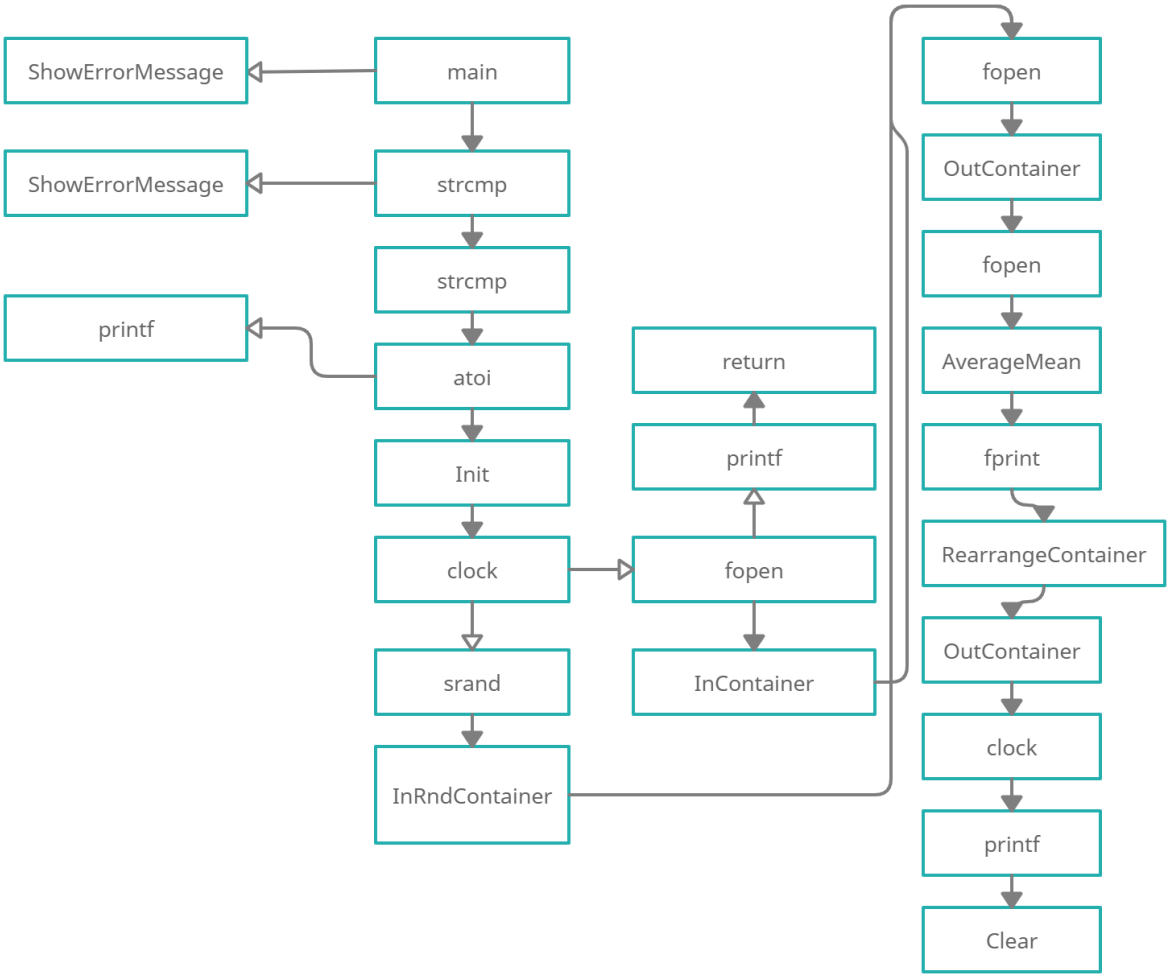
Описание работы функции OutBall в рамках архитектуры:

Stack
OutBall
Square
Heap
-
Глобальная память
-

Память программы	
void OutBall(…)	
	Память, Байт
ball: ball	32
file: FILE	8

Блок схема возможного стека, в результате работы функции main (с глубиной в 1 шаг):

Примечание: Пустая стрелочка обозначает то, что этот шаг не обязательно будет выполнен.



Основные характеристики программы:

Число header файлов: 6

Число source файлов: 6

Размер скомпилированное исполняемого файла: 52 Kb (дз1) vs 55 Kb (дз 2)

Размер файлов с кодом:

main.cpp - 2,96 Kb
shape.cpp - 1,64 Kb
ball.cpp - 1,16 Kb
tetrahedron.cpp - 1,15 Kb
parallelepiped.cpp - 1,56 Kb
container.cpp - 3,53 Kb

tetrahedron.h - 1,07 Kb
ball.h - 0,948 Kb
parallelepiped.h - 1,2 Kb
container.h - 1,48 Kb
shape.h - 1,3 Kb
rnd.h - 1,11 Kb

Итого общий размер: 17,6 Kb (дз 1) vs 19,1 Kb (дз 2)

Суммарное количество строк кода: 560 (дз 1) vs 573 (дз 2)

Время выполнения для включенных в состав программы тестов:

тест1 - 7,11 ms (дз 1) vs 1,23 ms (дз 2)	(10 фигур на вход)
тест2 - 13,54 ms (дз 1) vs 1,33 ms (дз 2)	(59 фигур на вход)
тест3 - 33,42 ms (дз 1) vs 2,96 ms (дз 2)	(517 фигур на вход)
тест4 - 13,36 ms (дз 1) vs 1,07 ms (дз 2)	(24 фигуры на вход)
тест5 - 302,61 ms (дз 1) vs 36,63 ms (дз2)	(9867 фигур на вход)
тест 6 - 124,71 ms (дз 1) vs 9,54 ms (дз 2)	(2671 фигур на вход)

рандомная генерация 5000 фигур - 139,79ms (дз 1) vs 32,68ms (дз 2)

Сравнение:

В отличие от процедурного подхода, в ООП увеличился объем кода, размер файлов, размер исполняемого файла. Теперь компилятор вынужден выделять ряд памяти под сам класс, скрытые ссылки на виртуальные таблицы, выравнивание байтов в памяти. Код стал проще и понятнее за счет возможностей полиморфизма и наследования. На практике же ООП подход оказывается предпочтительнее, тк проще конструировать логику программы + время выполнения сократилось почти в 10 раз в каждом из тестов.