Правительство Российской Федерации

Национальный исследовательский университет

«Высшая школа экономики»

Департамент программной инженерии

Факультет компьютерных наук

Плоские геометрические фигуры, размещаемые в координатной сетке.

(Вариант 2, вариант 8)

Отчёт

Дисциплина: «Архитектура вычислительных систем»

Исполнитель:

студент группы БПИ203

Насыхова Анастасия Артемовна

Москва 2021

Описание полученного задания

Вариант 2, варинат 8.

***Текст задания:*** Необходимо реализовать консольное приложение для обработки элементов контейнера с использованием динамической проверки типов во время выполнения программы.

В контейнере могут находиться плоские геометрические фигуры, размещаемые в координатной сетке. Базовыми альтернативами являются такие фигуры:

1. Треугольник (парметры: целочисленные координаты всех трех углов);
2. Прямоугольник (параметры: целочисленные координаты левого верхнего и правого нижнего углов);
3. Круг (параметры: целочисленные координаты центра и радиус).

Для каждой фигуры необходимо реализовать следующий функционал:

1. Ввод параметров фигуры из файла;
2. Случайный ввод параметров фигуры;
3. Вывод параметров в файл;
4. Вычисление периметра.

Для контейнера необходимо реализовать следующий функционал:

1. Инициализация контейнера;
2. Очистка контейнера;
3. Ввод содержимого контейнера из указанного потока;
4. Слуйчайный ввод содержимого контейнера;
5. Вывод содержимого контейнера в указанный поток;
6. Сортировка (QuickSort) содержимого контейнера (элементы сортируются по периметру).

Используемые метрики:

Число модулей реализации (файлов с определением программных объектов): 7.

Общий размер исходных текстов: 368 строк.

Полученный размер исполняемого кода: 12,2 Kb.

Время выполнения программы для различных тестовых наборов данных:

1. Test 1: 0,0015 c
2. Test 2: 0,00062 c
3. Test 3: 0,0011 c
4. Test 4: exception (обработанный)
5. Test 5: exception (обработанный)
6. Test 6: exception (обработанный)
7. Test 7: exception (обработанный)
8. Test 8: 0,0011 с
9. Test 9: 0,74 c
10. Test 10: exception (обработанный)

Сравнительный анализ полученных характеристик

1. Число модулей реализации уменьшилось.
2. Общий размер исходных текстов уменьшился в 2 раза (было 807).
3. Размер исполняемого файла уменьшился.
4. Время работы программы сильно увеличилось.

Структурная схема изучаемой архитектуры вычислительной системы

Таблица типов

|  |  |
| --- | --- |
| Name | Size |
| int | 4 |
| float | 4 |
| enum | 4 |

Память данных

|  |  |
| --- | --- |
| Name | Size |
| Class Shape | 0 |
| Class Color |  |
| Class Triangle  int x1  int x2  int x3  int y1  int y2  int y3  Color color | 28  4[0]  4[4]  4[8]  4[12]  4[16]  4[20]  4[24] |
| Class Rectangle  int x1  int x2  int y1  int y2  Color color | 20  4[0]  4[4]  4[8]  4[12]  4[16] |
| Class Circle  int x  int y  int r  Color color | 16  4[0]  4[4]  4[8]  4[12] |
| Class Container  enum max\_len  int len  Shape cont | 8008  4[0]  4[4]  8 \* 1000[8] |

Память команд

|  |  |
| --- | --- |
| main()  int argc  List[str] argv  container c  float start  float finish  float elapsed\_time | 8032  4[0]  8[4]  8008[12]  4[8020]  4[8024]  4[8028] |
| Triangle | |
| in\_file(x1, x2, y1, y2, x3, y3,color)  Triangle self  int color | 12  8[0]  4[8] |
| in\_rnd()  Triangle self  float a  float b  float c | 20  8[0]  4[8]  4[12]  4[16] |
| out()  Triangle self | 8  8[0] |
| perimeter()  Triangle self  double a  double b  double c | 32  8[0]  8[8]  8[16]  8[24] |

Исходный текст программы, разработанной в ходе выполнения задания

**1) main.py**

import sys  
import time  
  
from container import Container  
  
  
# Error message.  
def err\_message1():  
 print("incorrect command line!\n"  
 " Waited:\n"  
 " command -f infile outfile01 outfile02\n"  
 " Or:\n"  
 " command -n number outfile01 outfile02\n")  
  
  
# Error message.  
def err\_message2():  
 print("incorrect qualifier value!\n"  
 " Waited:\n"  
 " command -f infile outfile01 outfile02\n"  
 " Or:\n"  
 " command -n number outfile01 outfile02\n")  
  
  
# ------------------------------------------------------------------------------  
if \_\_name\_\_ == "\_\_main\_\_":  
 argv = sys.argv  
 argc = len(argv)  
  
 if argc != 5:  
 err\_message1()  
 exit(1)  
  
 start = time.process\_time()  
  
 print("Start\n")  
  
 container = Container()  
  
 if argv[1] == "-f":  
 container.in\_file(argv[2])  
 elif argv[1] == "-n":  
 if (int(argv[2]) > 10000) or (int(argv[2]) < 1):  
 print(f'incorrect number of figures! {argv[2]}.\n' \  
 f'Set 0 < number <= 10000\n')  
 exit(1)  
 container.in\_rnd(int(argv[2]))  
 else:  
 err\_message2()  
 exit(1)  
  
 container.out(argv[3])  
 container.sort()  
 container.out(argv[4])  
  
 container.clear()  
  
 print("Stop\n")  
  
 finish = time.process\_time()  
 elapsed\_time = finish - start  
 print('Time elapsed', str(elapsed\_time) + "s")

**2) circle.py**

# ------------------------------------------------------------------------------  
# Circle.py - содержит функции обработки круга  
# ------------------------------------------------------------------------------  
  
from rnd import random  
from shape import Shape  
import math  
  
  
# ------------------------------------------------------------------------------  
# Круг.  
class Circle(Shape):  
 # Ввод круга из файла.  
 def \_\_init\_\_(self, x = 0, y = 0, r = 0, color = 0):  
 super().\_\_init\_\_(color)  
 self.x = x  
 self.y = y  
 self.r = r  
  
 # Случайный ввод круга.  
 def in\_rnd(self):  
 self.x = random()  
 self.y = random()  
 self.r = random()  
 self.color = super().get\_color(random() % 7 + 1)  
  
 # Вывод круга в форматируемый поток.  
 def out(self):  
 return f'It is Circle: x = {self.x}, y = {self.y}, r = {self.r}, ' \  
 f'Color = {super().get\_str()}. Perimeter = {self.perimeter()}\n'  
  
 # Вычисление периметра круга.  
 def perimeter(self):  
 return 2.0 \* math.pi \* self.r

**3) rectangle.py**

# ------------------------------------------------------------------------------  
# Rectangle.py - содержит функции обработки прямоугольника.  
# ------------------------------------------------------------------------------  
  
from shape import Shape  
from rnd import random  
  
  
# ------------------------------------------------------------------------------  
# Прямоугольник.  
class Rectangle(Shape):  
 # Ввод прямоугольника из файла.  
 def \_\_init\_\_(self, x1 = 0, x2 = 0, y1 = 0, y2 = 0, color = 0):  
 super().\_\_init\_\_(color)  
 self.x1 = x1  
 self.y1 = y1  
 self.x2 = x2  
 self.y2 = y2  
  
 # Случайный ввод прямоугольника.  
 def in\_rnd(self):  
 self.x1 = random()  
 self.y1 = random()  
 while True:  
 self.x2 = random()  
 self.y2 = random()  
 if (self.x1 != self.x2) and (self.y1 != self.y2):  
 break  
  
 self.color = super().get\_color(random() % 7 + 1)  
  
 # Вывод прямоугольника в форматируемый поток.  
 def out(self):  
 return f'It is Rectangle: x1 = {self.x1}, y1 = {self.y1}, x2 = {self.x2}, y2 = {self.y2}, ' \  
 f'Color = {super().get\_str()}. Perimeter = {self.perimeter()} \n'  
  
 # Вычисление периметра прямоугольника.  
 def perimeter(self):  
 return 2.0 \* abs(self.x2 - self.x1) + 2.0 \* abs(self.y2 - self.y1)

**4) triangle.py**

# ------------------------------------------------------------------------------  
# Triangle.py - содержит функции обработки треугольника  
# ------------------------------------------------------------------------------  
  
import math  
from shape import Shape  
from rnd import random  
  
  
# ------------------------------------------------------------------------------  
# Треугольник.  
class Triangle(Shape):  
 # Ввод треугольника из файла  
 def \_\_init\_\_(self, x1 = 0, x2 = 0, y1 = 0, y2 = 0, x3 = 0, y3 = 0,  
 color = 0):  
 super().\_\_init\_\_(color)  
 self.x1 = x1  
 self.y1 = y1  
 self.x2 = x2  
 self.y2 = y2  
 self.x3 = x3  
 self.y3 = y3  
  
 # Случайный ввод треугольника  
 def in\_rnd(self):  
 self.x1 = random()  
 self.y1 = random()  
  
 while True:  
 self.x2 = random()  
 self.y2 = random()  
 if (self.x1 != self.x2) and (self.y1 != self.y2):  
 break  
  
 a = math.sqrt((self.x2 - self.x1) \* (self.x2 - self.x1) + (self.y2 - self.y1) \* (self.y2 - self.y1))  
  
 while True:  
 self.x3 = random()  
 self.y3 = random()  
  
 b = math.sqrt((self.x3 - self.x1) \* (self.x3 - self.x1) +  
 (self.y3 - self.y1) \* (self.y3 - self.y1))  
  
 c = math.sqrt((self.x3 - self.x2) \* (self.x3 - self.x2) +  
 (self.y3 - self.y2) \* (self.y3 - self.y2))  
  
 if (c < a + b) and (a < c + b) and (b < c + a):  
 break  
  
 self.color = super().get\_color(random() % 7 + 1)  
  
 # Вывод треугольника в форматируемый поток  
 def out(self):  
 return f'It is Triangle: x1 = {self.x1}, y1 = {self.y1}, x2 = {self.x2}, y2 = {self.y2}, ' \  
 f'x3 = {self.x3} , y3 = {self.y3}, Color = {super().get\_str()}. Perimeter = {self.perimeter()}\n'  
  
 # Вычисление периметра треугольника.  
 def perimeter(self):  
 a = math.sqrt((self.x2 - self.x1) \* (self.x2 - self.x1) +  
 (self.y2 - self.y1) \* (self.y2 - self.y1))  
  
 b = math.sqrt((self.x3 - self.x1) \* (self.x3 - self.x1) +  
 (self.y3 - self.y1) \* (self.y3 - self.y1))  
  
 c = math.sqrt((self.x3 - self.x2) \* (self.x3 - self.x2) +  
 (self.y3 - self.y2) \* (self.y3 - self.y2))  
  
 return a + b + c

**5) container.py**

# ------------------------------------------------------------------------------  
# Container.py - содержит функции обработки контейнера.  
# ------------------------------------------------------------------------------  
  
from rectangle import Rectangle  
from circle import Circle  
from triangle import Triangle  
from rnd import random  
  
  
# ------------------------------------------------------------------------------  
# Простейший контейнер на основе одномерного массива  
class Container:  
 # Инициализация контейнера.  
 def \_\_init\_\_(self):  
 self.cont = []  
  
 # Очистка контейнера от элементов(освобождение памяти).  
 def clear(self):  
 self.cont = []  
  
 # Ввод содержимого контейнера из указанного потока.  
 def in\_file(self, path):  
 with open(path, 'r') as file:  
 for shape in file:  
 try:  
 line = shape.strip().split()  
 if line[0] == "1":  
 self.cont.append(Rectangle(int(line[1]), int(line[2]), int(line[3]),  
 int(line[4]), int(line[5])))  
 elif line[0] == "2":  
 self.cont.append(Triangle(int(line[1]), int(line[2]), int(line[3]), int(line[4]),  
 int(line[5]), int(line[6]), int(line[7])))  
 elif line[0] == "3":  
 self.cont.append(Circle(int(line[1]), int(line[2]), int(line[3]), int(line[4])))  
 except Exception:  
 raise ValueError("Invalid argument")  
  
 # Случайный ввод содержимого контейнера.  
 def in\_rnd(self, size: int):  
 while len(self.cont) < size:  
 shape = random() % 3  
 if shape == 0:  
 self.cont.append(Rectangle())  
 elif shape == 1:  
 self.cont.append(Triangle())  
 else:  
 self.cont.append(Circle())  
 self.cont[-1].in\_rnd()  
  
 # Вывод содержимого контейнера в указанный поток.  
 def out(self, path):  
 with open(path, 'w') as file:  
 file.write("Filled container:\n")  
 file.write(f'Container contains {len(self.cont)} elements.\n')  
 i = 1  
 for shape in self.cont:  
 file.write(f'{i}: {shape.out()}')  
 i += 1  
  
 # Сортировка контейнера по возрастанию QuickSort.  
 def sort(self):  
 self.quicksort(self.cont, 0, len(self.cont))  
  
 def quicksort(self, cont, start, end):  
 if end - start > 1:  
 p = self.partition(cont, start, end)  
 self.quicksort(self.cont, start, p)  
 self.quicksort(self.cont, p + 1, end)  
  
 def partition(self, cont, start, end):  
 cont = self.cont  
 pivot = cont[start].perimeter()  
 i = start + 1  
 j = end - 1  
  
 while True:  
 while (i <= j) and (cont[i].perimeter() <= pivot):  
 i = i + 1  
 while (i <= j) and (cont[j].perimeter() >= pivot):  
 j = j - 1  
  
 if i <= j:  
 cont[i], cont[j] = cont[j], cont[i]  
 else:  
 cont[start], cont[j] = cont[j], cont[start]  
 return j

**6) shape.py**

# ------------------------------------------------------------------------------  
# Shape.py - содержит описание фигуры (обобщающего типа данных)  
# ------------------------------------------------------------------------------  
  
from enum import Enum  
  
  
class Color(Enum):  
 INVALID\_COLOR = 0  
 RED = 1  
 ORANGE = 2  
 YELLOW = 3  
 GREEN = 4  
 BLUE = 5  
 DARKBLUE = 6  
 PURPLE = 7  
  
  
# ------------------------------------------------------------------------------  
# Класс, обобщающий все имеющиеся фигуры.  
class Shape:  
 # Ввод обобщенной фигуры из файла.  
 def \_\_init\_\_(self, color):  
 self.color = self.get\_color(color)  
  
 @staticmethod  
 def get\_color(color: int):  
 if color == 1:  
 return Color.RED  
 elif color == 2:  
 return Color.ORANGE  
 elif color == 3:  
 return Color.YELLOW  
 elif color == 4:  
 return Color.GREEN  
 elif color == 5:  
 return Color.BLUE  
 elif color == 6:  
 return Color.DARKBLUE  
 elif color == 7:  
 return Color.PURPLE  
 else:  
 return Color.INVALID\_COLOR  
  
 def get\_str(self):  
 if self.color == Color.RED:  
 return "Red"  
 elif self.color == Color.ORANGE:  
 return "Orange"  
 elif self.color == Color.YELLOW:  
 return "Yellow"  
 elif self.color == Color.GREEN:  
 return "Green"  
 elif self.color == Color.BLUE:  
 return "Blue"  
 elif self.color == Color.DARKBLUE:  
 return "Darkblue"  
 elif self.color == Color.PURPLE:  
 return "Purple"  
 else:  
 return "Invalid color!"

**7) rnd.py**

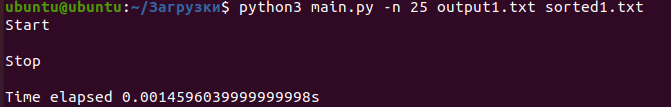
# ------------------------------------------------------------------------------  
# rnd.py - содержит генератор случайных чисел в диапазоне от 0 до 30  
# ------------------------------------------------------------------------------  
  
from random import randint  
  
  
def random():  
 return randint(0, 30)

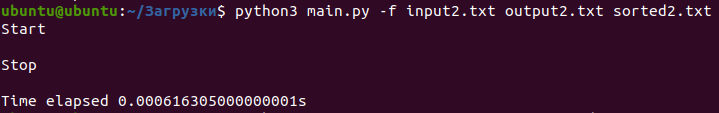
Тестовые наборы данных

1. python3 main.py -n 25 output1.txt sorted1.txt
2. python3 main.py -f input2.txt output2.txt sorted2.txt (5 элементов)
3. python3 main.py -n 15 output3.txt sorted3.txt
4. python3 main.py -n -1000 output4.txt sorted4.txt
5. python3 main.py
6. python3 main.py -n 0 output6.txt sorted6.txt
7. python3 main.py -n 10001 output7.txt sorted7.txt
8. python3 main.py -f input8.txt output8.txt sorted8.txt (20 элементов)
9. python3 main.py -n 10000 output9.txt sorted9.txt
10. python3 main.py er r o r

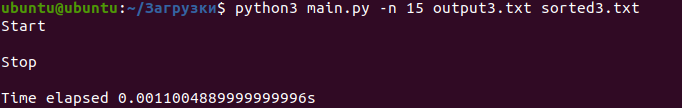
Результаты тестовых прогонов

Тест 1:

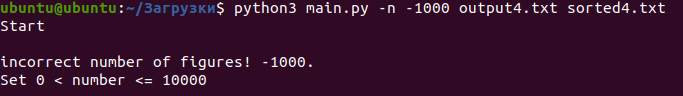
Тест 2:



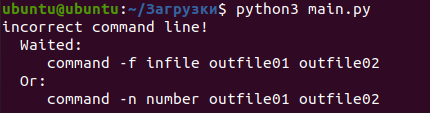
Тест 3:



Тест 4:



Тест 5:

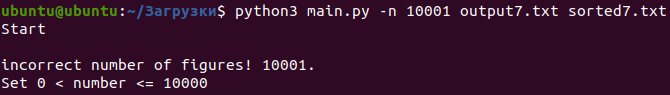


Тест 6:

Изображение выглядит как текст

Автоматически созданное описание

Тест 7:



Тест 8:

Изображение выглядит как текст

Автоматически созданное описание

Тест 9:

Изображение выглядит как текст

Автоматически созданное описание

Тест 10:

Изображение выглядит как текст

Автоматически созданное описание