**ОПИСАНИЕ ЗАДАНИЯ (задание 2, дополнительная функция 20):**

Разработать программный продукт с использованием *объектно-ориентированного подхода и статической типизацией*. Программа должна содержать следующие структуры:

| **Обобщенный артефакт, используемый в задании** | **Базовые альтернативы** |
|---|---|
| Плоская геометрическая фигура, размещаемые в координатной сетке | 1. Круг (целочисленные координата центра окружности, радиус)<br>2. Прямоугольник (целочисленные координаты левого верхнего и правого нижнего углов)<br>3. Треугольник (целочисленные координаты трех углов) |

Для всех альтернатив общей переменной является **цвет** (перечислимый тип). Он может принимать значения: красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый. Общей функция всех альтернатив выступает вычисление периметра фигуры (действительное число). В качестве дополнительной функции контейнера необходимо удалять из него те фигуры, периметр которых больше, чем среднее арифметическое периметров всех фигур контейнера (остальные фигуры передвигать к началу контейнера с сохранением порядка).

Также нужно: разработать тестовые входные данные и провести тестирование и отладку программы на этих данных (при необходимости, программа должна правильно обрабатывать переполнение по данным); описать структуру используемой ВС с наложением на нее обобщенной схемы разработанной программы; зафиксировать количество заголовочных файлов, программных файлов, общий размер исходных текстов, полученный размер исполняемого кода (если он формируется), время выполнения программы для различных тестовых наборов данных.

(*) В качестве доп. функционала реализован способ генерации тестовых файлов.

**СТРУКТУРНАЯ СХЕМА АРХИТЕКТУРЫ ВС С ПРОГРАММОЙ:**

Программа разработана в 64 битной системе (дистрибутив Ubuntu на ярде Linux). Разработка велась на языке C++, с использований библиотек C (как указано в требованиях к заданию). Рассмотрим 4 функции, на примере которых отобразим графически структуру ВС с наложением на нее обобщенной схемы разработанной программы:

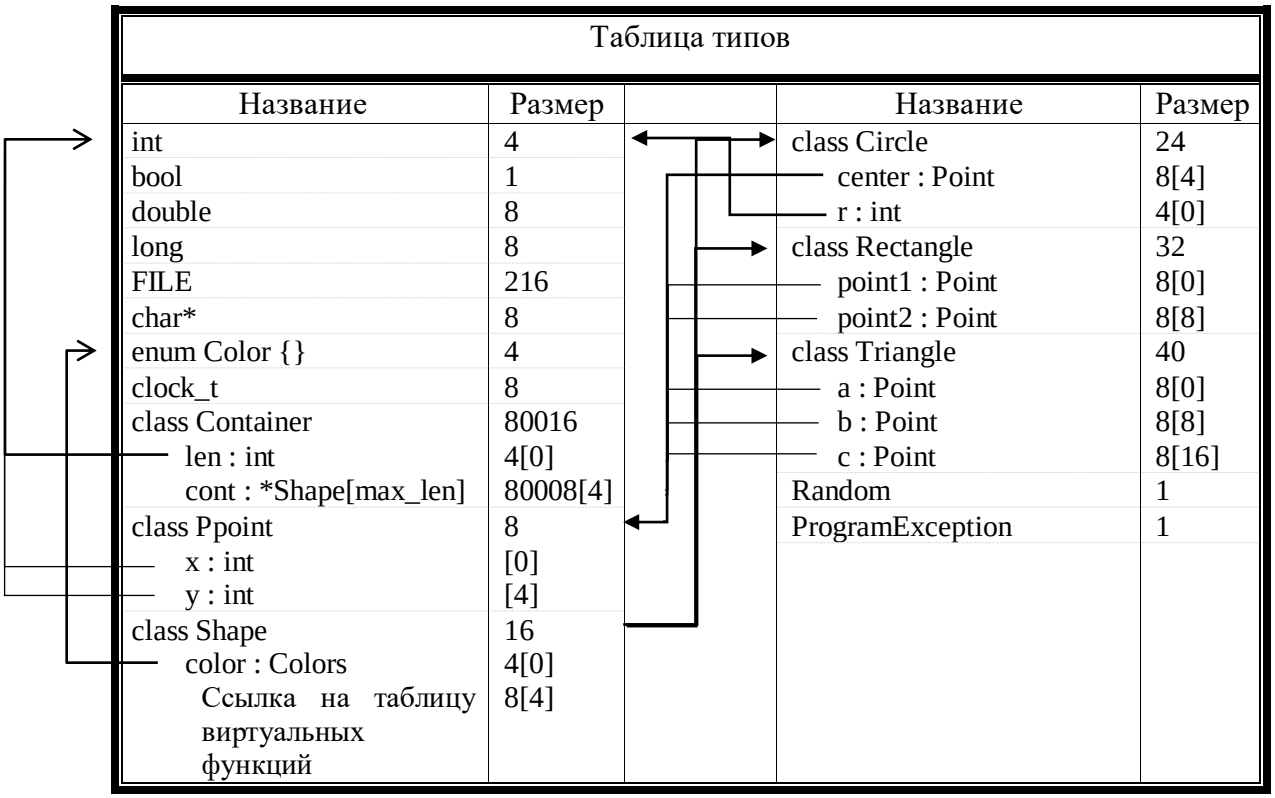
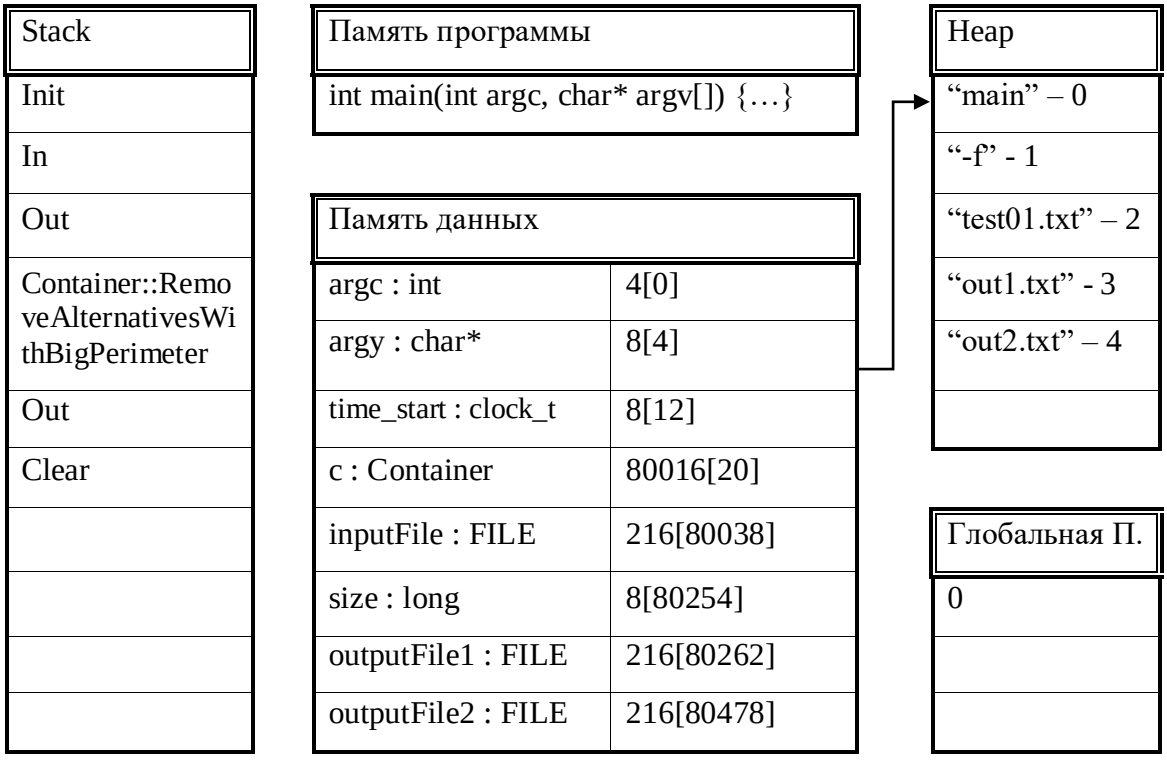


Таблица типов

| Название | Размер | Название | Размер |
|---|---|---|---|
| int | 4 | class Circle | 24 |
| bool | 1 | center : Point | 8[4] |
| double | 8 | r : int | 4[0] |
| long | 8 | class Rectangle | 32 |
| FILE | 216 | point1 : Point | 8[0] |
| char* | 8 | point2 : Point | 8[8] |
| enum Color {} | 4 | class Triangle | 40 |
| clock_t | 8 | a : Point | 8[0] |
| class Container | 80016 | b : Point | 8[8] |
| len : int | 4[0] | c : Point | 8[16] |
| cont : *Shape[max_len] | 80008[4] | Random | 1 |
| class Ppoint | 8 | ProgramException | 1 |
| x : int | [0] | | |
| y : int | [4] | | |
| class Shape | 16 | | |
| color : Colors | 4[0] | | |
| Ссылка на таблицу виртуальных функций | 8[4] | | |

Можно заметить, что размер классов больше объектов, которые они содержат. К сожалению, нам не рассказывали на лекциях почему так происходит. Поиски в интернете натолкнули меня на мысль, что так происходит из-за выравнивания компилятором объектов в памяти и особенностью выбранного компилятора (g++ 10).

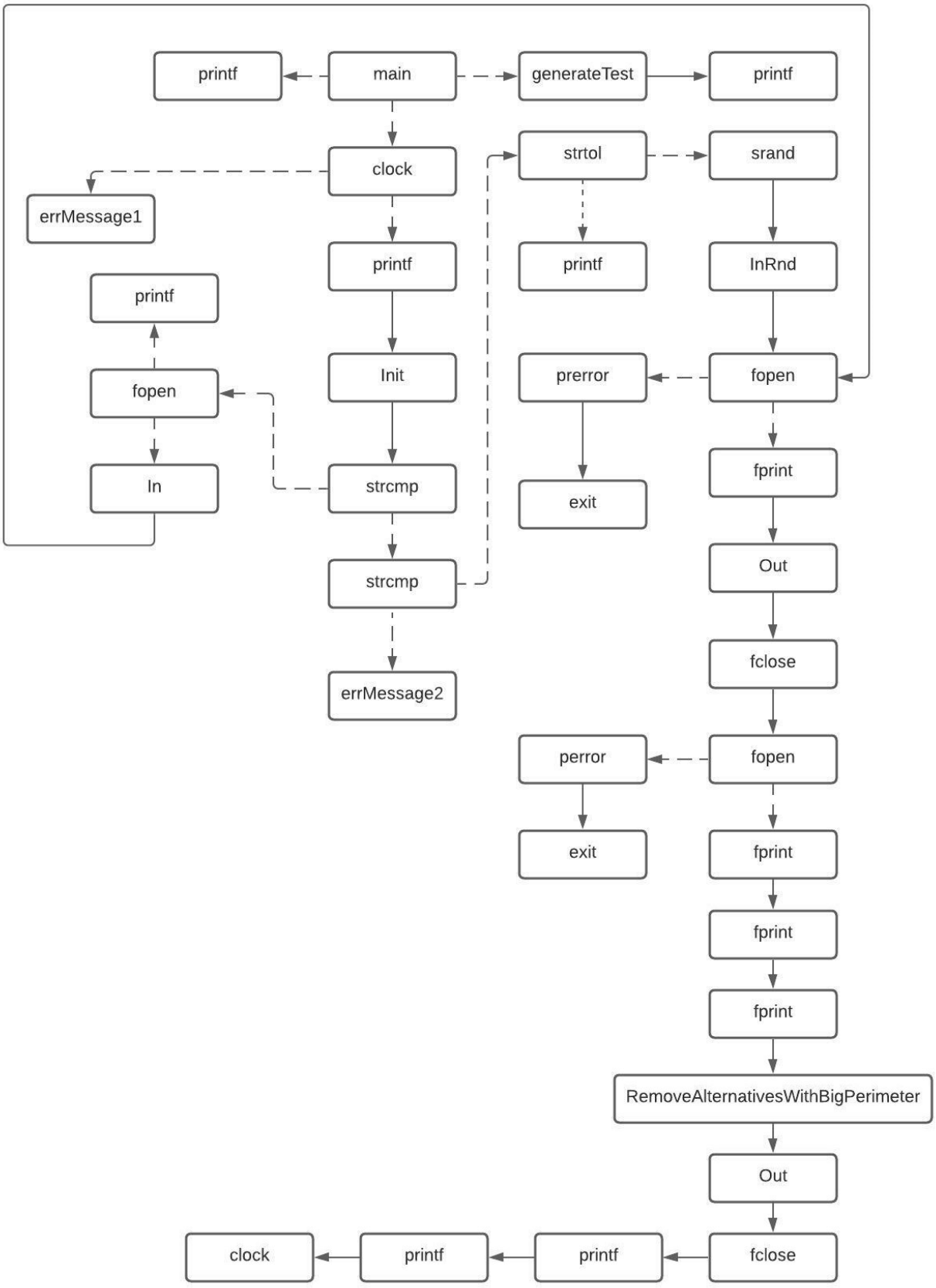
Описание работы функции main в рамках архитектуры:




Примечание:

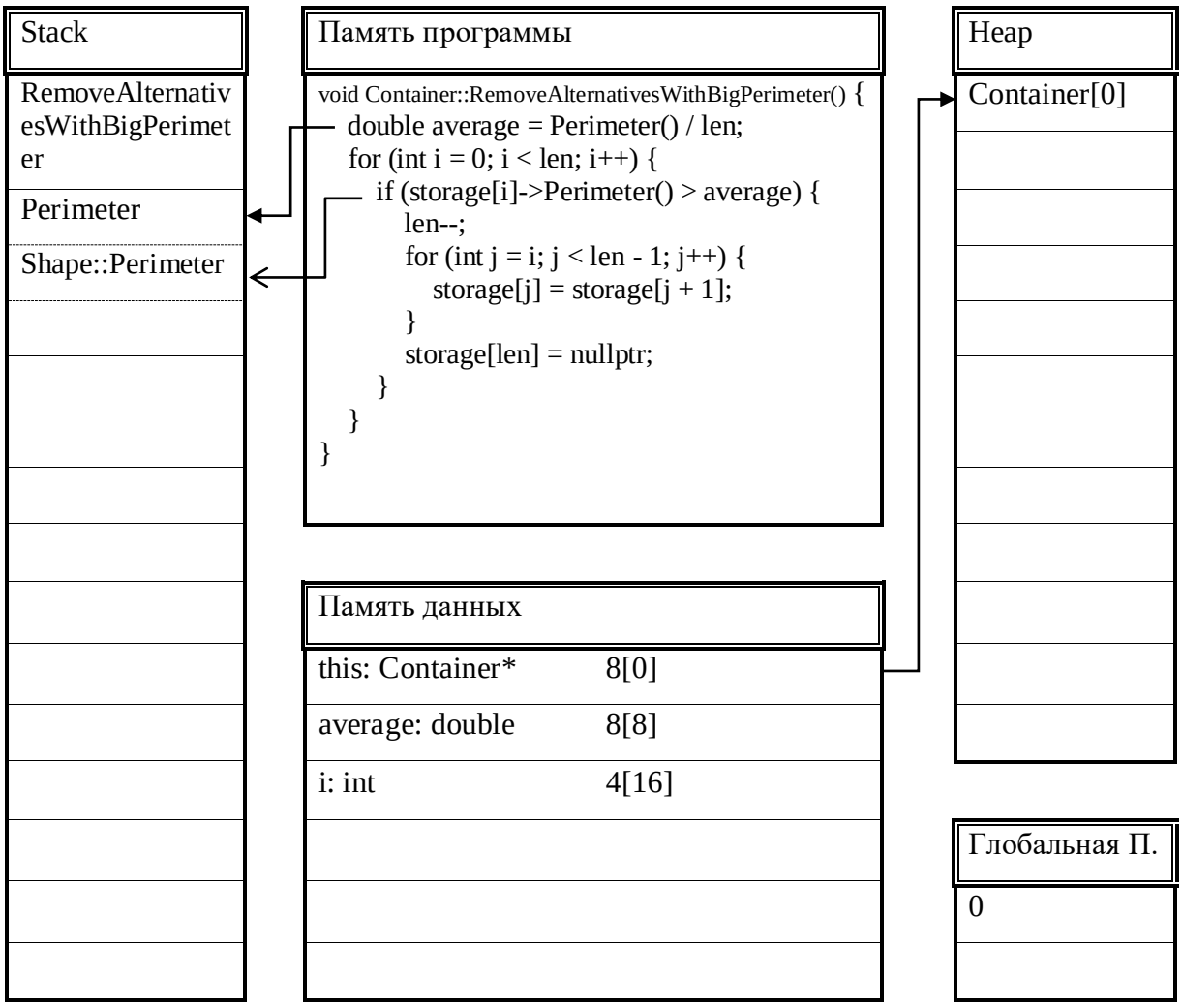
1. В таблице стек представлен для частного случая (загрузка данных из файла) и содержит только созданные в процессе написания программы функции). Все вариации возможного стека вызов(с глубиной в 1 шаг) изображены на следующей странице в виде блок схемы.
2. Под {…} подразумевается код функции main (он написан в исходном файле main.c).

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




*Пунктирными стрелками обозначена вариативность стека.

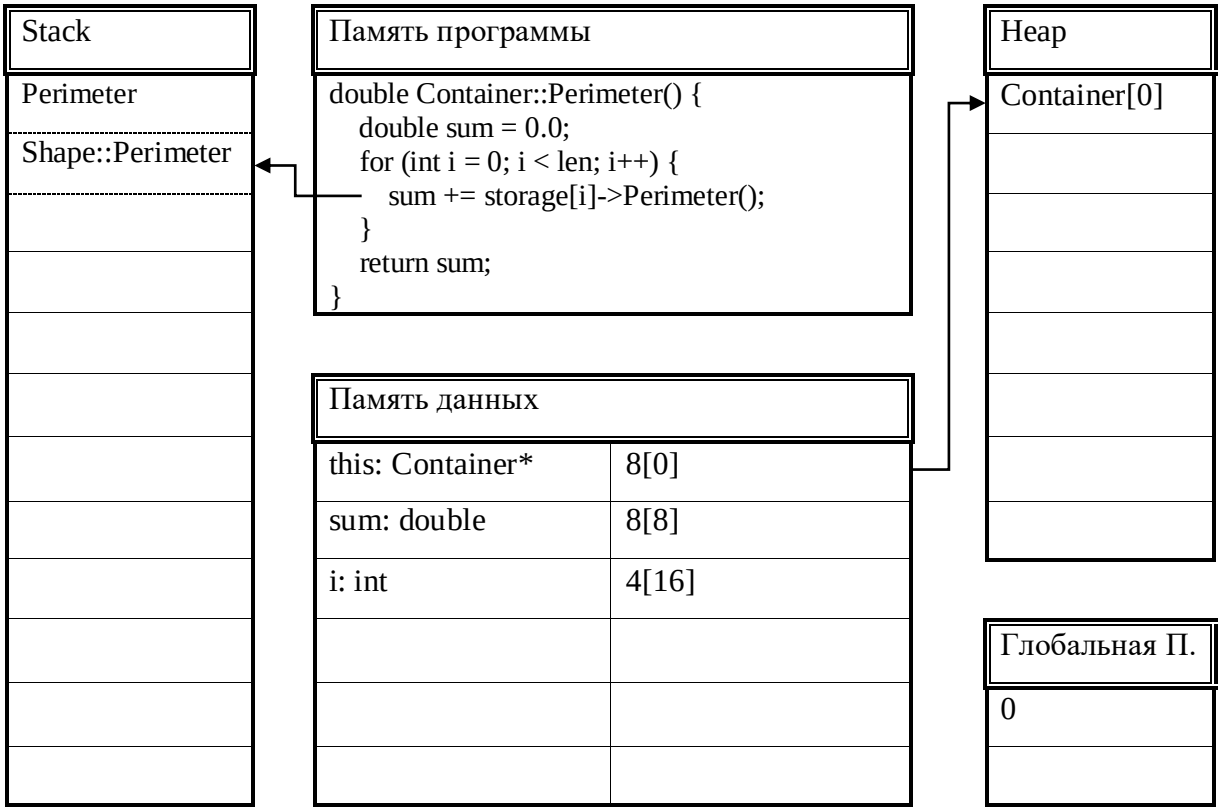
Описание работы для функции RemoveAlternativesWithBigPerimeter в рамках архитектуры:



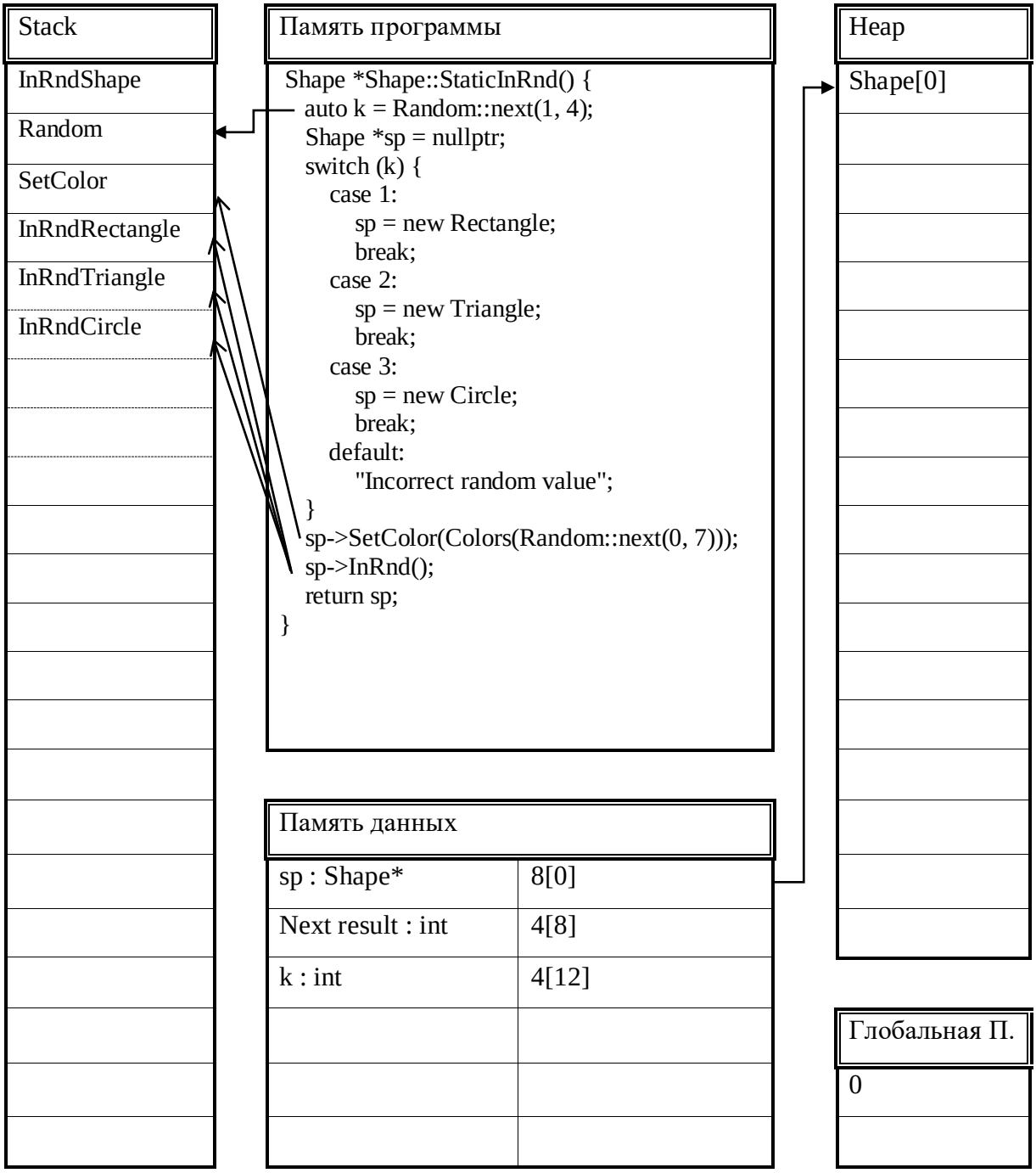

Примечание:

В данной (и последующих) таблицах стек изображен полностью (вариативная часть отображается выделением функции пунктирными линиями).

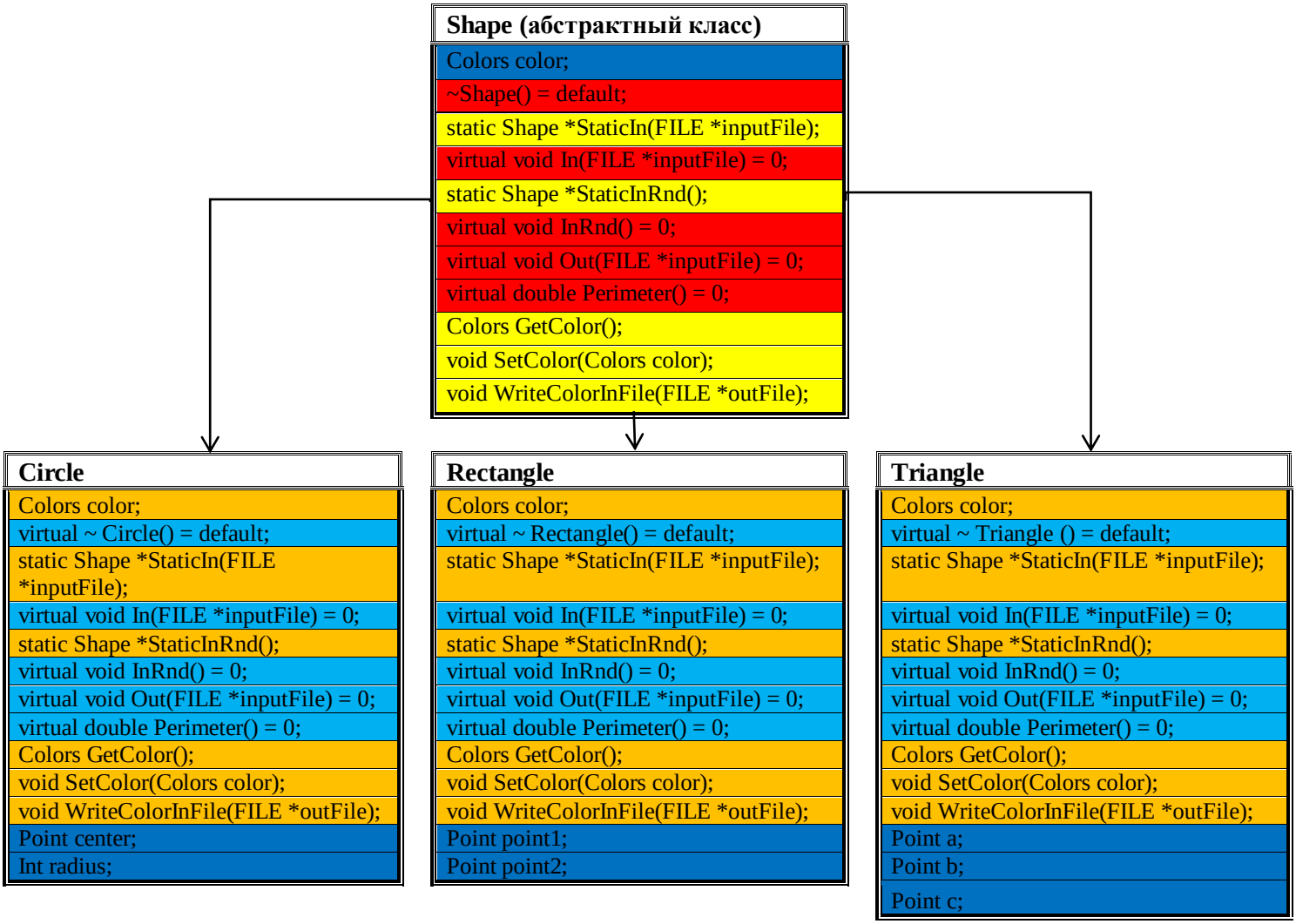
Описание работы для функции Perimeter в рамках архитектуры:



Описание работы для функции InRndShape в рамках архитектуры:



В качестве дополнительной работы я решил изобразить иерархию наследования в данной программе с названиями всех функций и полей, которые содержит каждый класс:




Примечание:

- Желтый – функция, определенная и реализованная в данном классе
- Красный – виртуальная функция
- Синий – поле
- Оранжевый – методы и поля родителя.
- Голубой – реализованный метод родителя.

**ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ПРОГРАММЫ:**

- Число заголовочных файлов – 9
- Число модулей реализации – 9
- Общий размер исходных текстов – 1110 строк кода вместе с комментариями (40.2 кб).
- Размер исполняемого файла – 127.2 кб.
- Время выполнения программы для различных тестовых прогонов:

| Номер теста | Время выполнения в секундах |
|---|---|
| Тест1 (3 корректных фигуры) | 0.000272 |
| Тест2 (пустой файл) | 0.000251 |
| Тест3 (некорректный ввод) | 0.000211 |
| Тест4 (некорректный цвет) | 0.000116 |
| Тест5 (некорректные аргументы) | 0.000234 |
| Тест6 (10000 элементов) | 0.066709 |

**СРАВНЕНИЕ С ПРЕДЫДУЩИМИ ВЕРСИЯМИ ПРОГРАММЫ:**

Предыдущее задание было посвящено разработки аналогичного консольного приложения, но с использованием процедурной парадигмы программирования. Объектно-ориентированный подход, дополнение необязательного функционала, расширение объёма документации и количества исполняемых/интерфейсных файлов, привел к ряду отличий данной версии программы, а именно:

- Объём памяти, занимаемый исполнимым файлом, увеличился в более чем в 3 раза. Абсолютно также объём исходных текстов значительно возрос – почти в 3 раза. Объём написанного кода при этом увеличился более чем в 2 раза.
- Программа стала работать быстрее (примерно на треть). Приведем таблицу с временными результатами тестовых наборов из предыдущего задания:

| Номер теста | Время выполнения в секундах |
|---|---|
| Тест1 (3 корректных фигуры) | 0.000349 |
| Тест2 (пустой файл) | 0.000385 |
| Тест3 (некорректный ввод) | 0.000358 |
| Тест4 (некорректный цвет) | 0.000407 |
| Тест5 (некорректные аргументы) | 0.000367 |

- Функционал программы увеличился – теперь есть возможность генерировать тестовые файлы автоматически.
- Исправлены ряд багов, связанных с обработкой тестов на экстремальных значениях.
- Разработанные типы данных стали занимать больше места, чем аналогичные в процедурном подходе. Приведем несколько примеров:

| Было | Стало |
|---|---|
| container – 80008 байт | Container – 800016 байт |
| shape – 32 байта | Shape – 16 байт |
| circle – 12 байт | Circle – 24 байт |
| triangle – 16 байт | Triangle – 32 байт |
| rectangle – 24 байт | Rectangle – 40 байт |

Это связано с тем, что компилятор вынужден выделять ряд памяти под сам класс, скрытые ссылки на виртуальные таблицы, выравнивание байтов в памяти.
К тому же альтернативы стали занимать гораздо больше места, в то время, как артефакт – меньше. Объясняется это тем, что в ООП подходе каждая альтернатива обязана хранить в себе все функциональные члены артефакта, в отличии от процедурного подхода (там же артефакт обязан хранить в себе как саму альтернативу, так и ее отличительный признак).