Архитектура вычислительных систем

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ПРАКТИЧЕСКОЙ РАБОТЕ НА ТЕМУ:

«Статически типизированная архитектура ВС,
ориентированная на объектно-ориентированный подход»

*Работу выполнил:*

студент группы БПИ207
Пендищук Владислав

Москва
2021 г.

# ОГЛАВЛЕНИЕ

# ОПИСАНИЕ ЗАДАНИЯ

Задача состояла в разработке программного продукта с использованием объектно-ориентированного подхода и статической типизацией на языке C++.

### Требования к функционалу

В соответствии с полученным вариантом задания (258) функционал, требуемый к реализации в программе, содержал в себе следующие пункты:

1. Реализация обобщённого артефакта – пассажирского транспорта, и его параметров:
    a. Скорость – целое число;
    b. Расстояние между пунктами отправления и назначения – действительное число;
2. Реализация базовых альтернатив и уникальных параметров, задающих их отличительные признаки:
    a. Самолёт, отличительные признаки:
        i. Дальность полёта – целое число;
        ii. Грузоподъёмность – целое число.
    b. Поезд, отличительные признаки:
        i. Количество вагонов – целое число.
    c. Корабль, отличительные признаки:
        i. Водоизмещение – целое число;
        ii. Вид судна – перечислимый тип (лайнер, буксир, танкер).
3. Реализация общей для всех альтернатив функции – вычисления идеального времени прохождения пути (действительное число).
4. Реализация контейнера для объектов типа обобщённого артефакта с массивом фиксированной длины в своей основе.
5. Реализация функции удаления из контейнера тех элементов, для которых значение, полученное с использованием функции, общей для всех альтернатив, меньше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции.

### Требования к запуску и вводу\выводу

К процессу запуска программы и ввода\вывода при работе с ней были представлены следующие требования:

1. Запуск программы осуществляется из командной строки, в которой указываются: имя запускаемой программы; имя файла с исходными данными; имя файла с выходными данными.
2. Для каждого программного объекта, загружаемого в контейнер, исходный файл с тестовым набором должен содержать: признак

альтернативы, а также список параметров, необходимых этой альтернативе. Этот список должен быть представлен в формате, удобном для обработки компьютером.

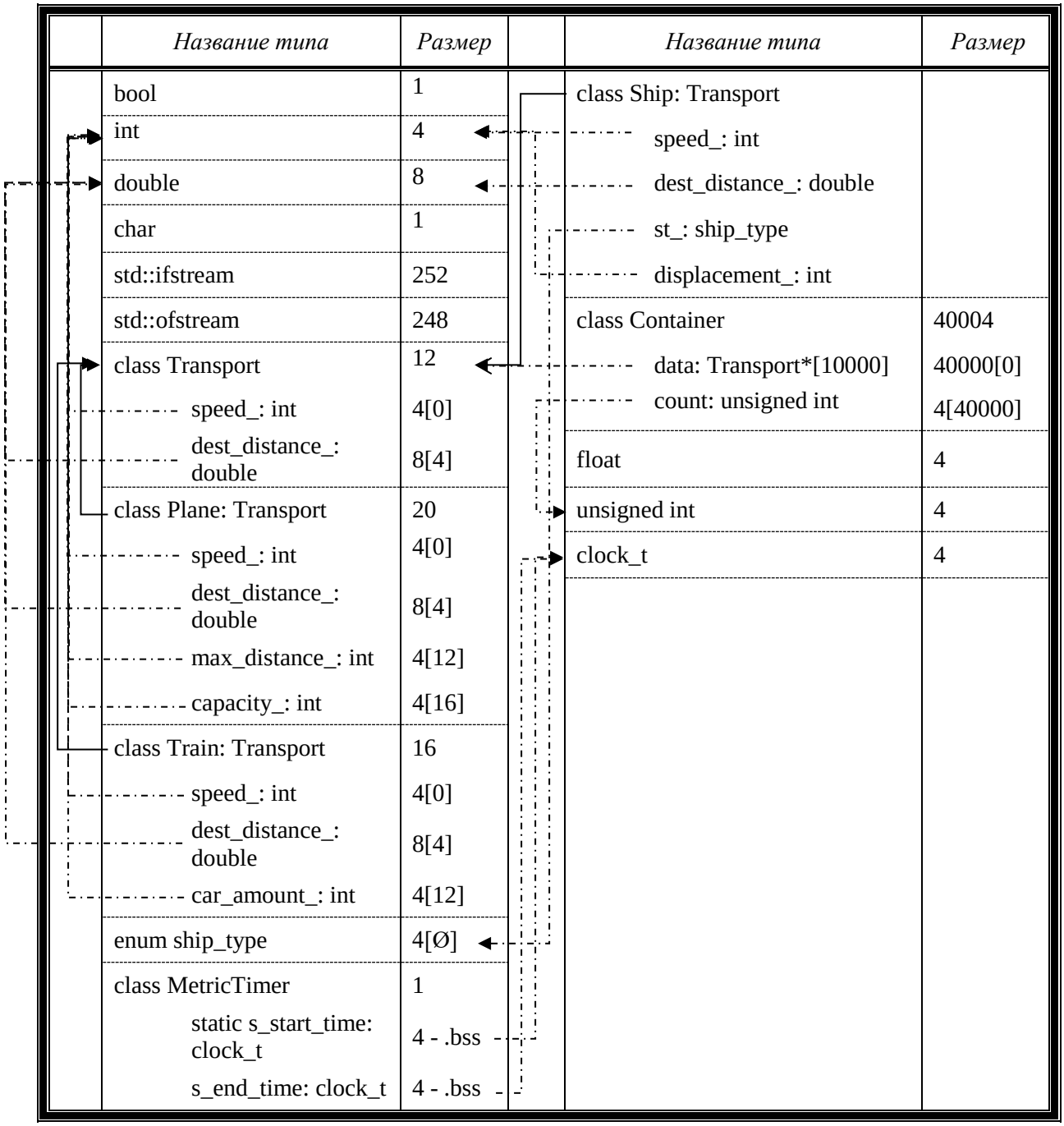
3. При больших данных во входном файле должны быть указаны только параметры для генератора случайных наборов данных, который и заполняет контейнер.
4. В выходной файл необходимо вывести введенные в контейнер данные. Помимо этого, необходимо вывести информацию об общем количестве объектов, содержащихся в контейнере. После этого в тот же файл необходимо вывести новые данные в соответствии с результатами, полученными в ходе работы программы. Информация для вывода должна быть представлена в форме, удобной для восприятия пользователем.

# СТРУКТУРА ИЗУЧАЕМОЙ АРХИТЕКТУРЫ ВС

Объектом изучения в данной работе являлась статически типизированная архитектура ВС, ориентированная на объектно-ориентированный подход. Разработка велась на языке С с соответствующими типами. Отобразим данную архитектуру на обобщённой схеме разработанной программы на примере 4 функций и иерархии классов для архитектуры x86-64.

### Таблица типов

Таблица типов, используемых в программе, необходима в силу статической типизации рассматриваемой архитектуры.



## Схема №1 – main

На данной схеме в обобщённом виде на статически типизированную архитектуру ВС отображена функция main:

| *Heap* |
|---|
| “proc-c-1” |
| “-f” |
| “in.txt” |
| “out.txt” |
| &c |
| |
| … |

| *Память данных*[1] | |
|---|---|
| argc: int | 4[0] |
| argv: char** | 8[4] |
| start: clock_t | 4[12] |
| c: Container | 40004[16] |
| ifstream: std::ifstream | 252[40020] |
| ofstream: std::ofstream | 248[40272] |
| end: clock_t | 4[40520] |
| seconds: float | 4[40524] |

| *Глобальная память* |
|---|
| s_start_time, s_end_time |

| *Память команд (программы)* |
|---|
| `int main(int argc, char* argv[]) {…}` |

| *Stack 1 (ввод из файла)* |
|---|
| clock |
| fprintf |
| Initialize |
| strcmp |
| open |
| In |
| StaticTransportIn[n] |
| Append[n] |
| Out |
| (Transport)::Out[n] |
| DeleteLessThanAverage |
| TimeToDest[n] |
| RemoveAt[n] |
| Out |
| (Transport)::Out[n] |

| *Stack 2 (ошибка)* |
|---|
| clock |
| fprintf |
| … |
| ArgNumError / InputModeError |
| PrintRuntimeDuration |
| exit |

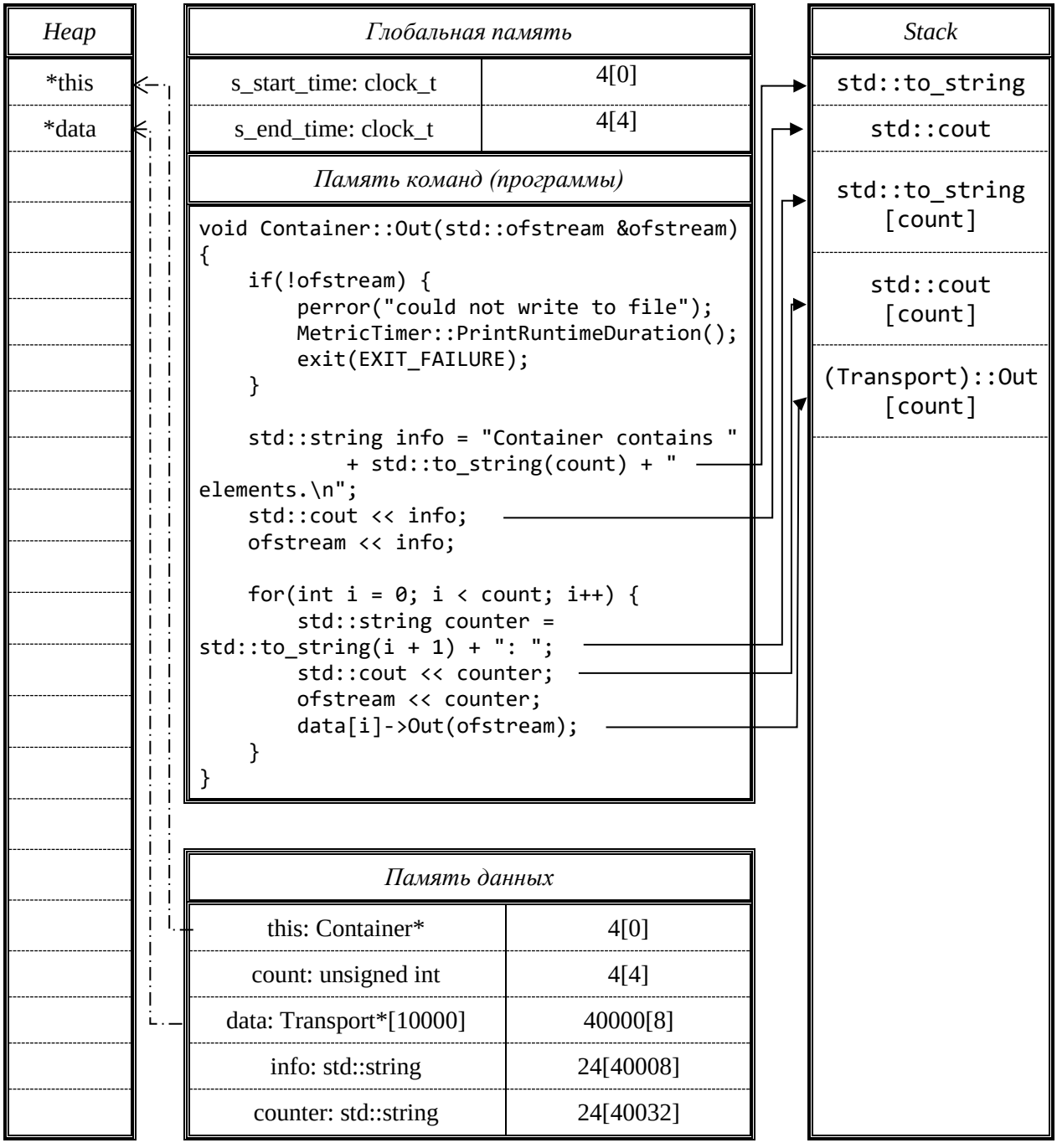
| *Stack 3 (ген. контейнера)* |
|---|
| clock |
| fprintf |
| Initialize |
| strcmp |
| InRand |
| StaticInRnd[n] |
| Append[n] |
| Out |
| (Transport)::Out[n] |
| DeleteLessThanAverage |
| TimeToDest[n] |
| RemoveAt[n] |
| Out |
| (Transport)::Out[n] |

| *Stack 4 (ген. тестов)* |
|---|
| clock |
| fprintf |
| Initialize |
| strcmp |
| printf |
| GenerateTestFiles |
| open[n] |
| GenerateTest[n] |
| std::cout[n |
| printf |

[1] Память данных представлена для случая с успешным вводом из файла

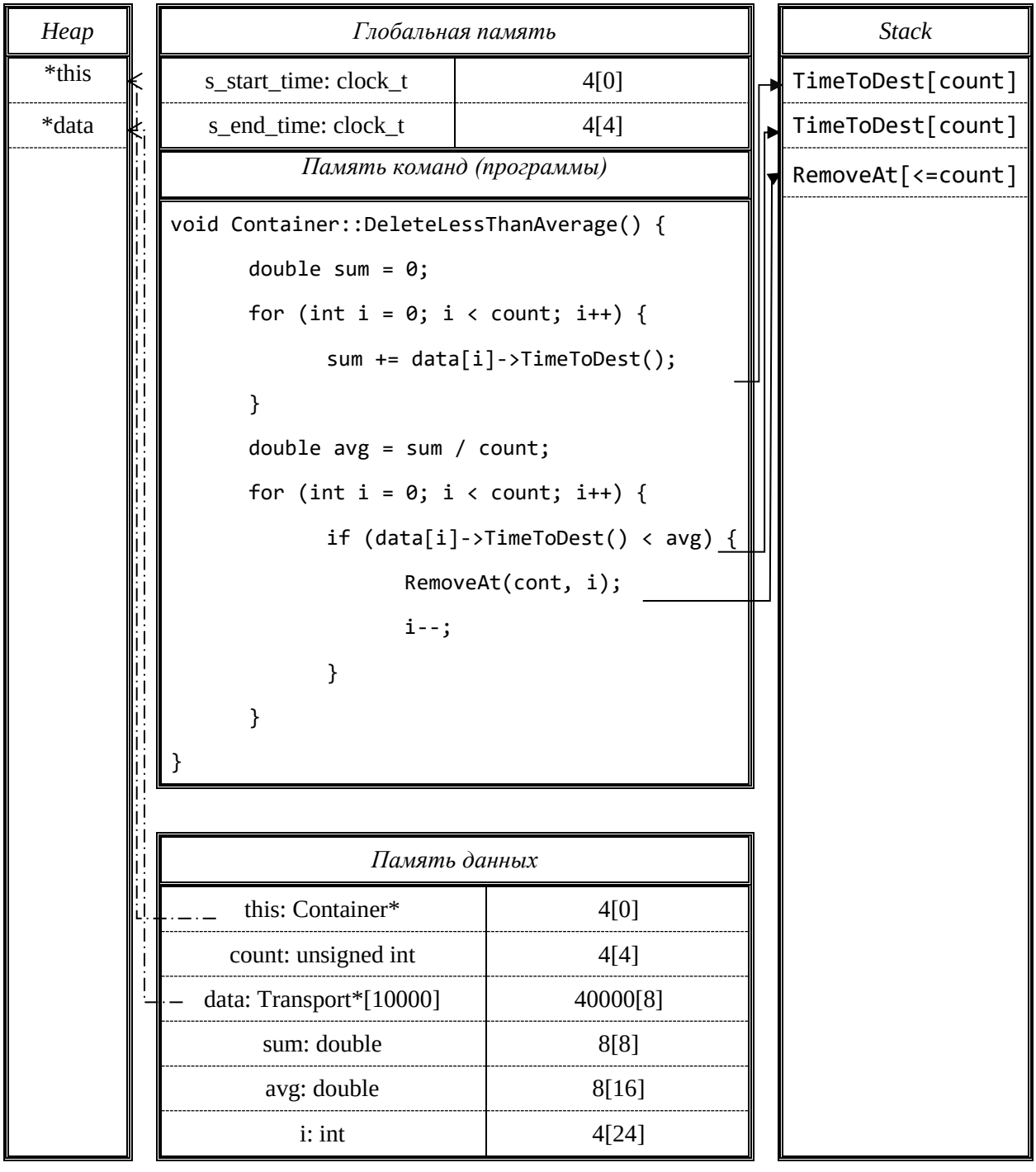
## Схема №2 – Container::Out

На данной схеме в обобщённом виде на статически типизированную архитектуру ВС отображена функция Container::Out для случая с заданным потоком вывода:



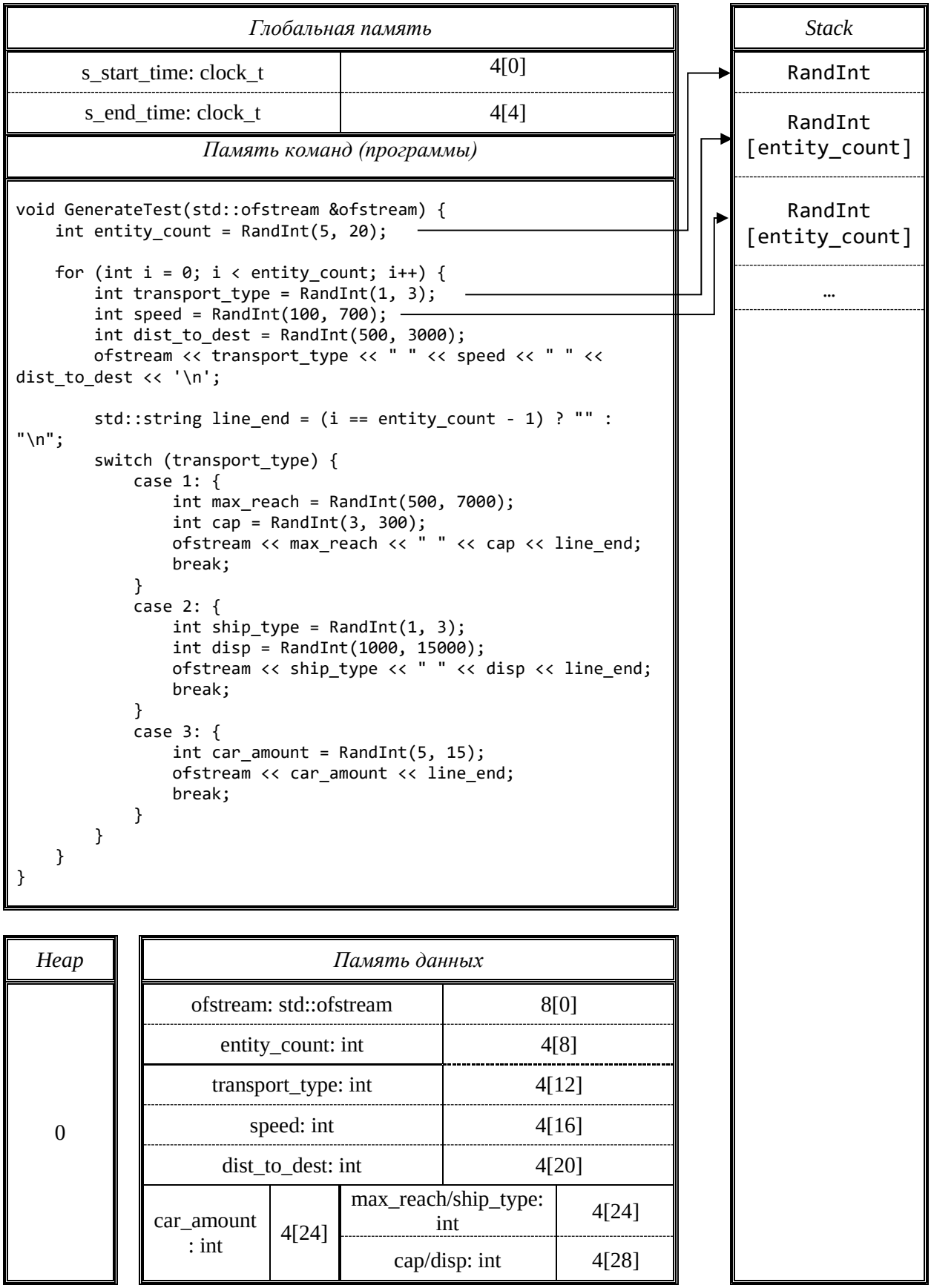
## Схема №3 – Container::DeleteLessThanAverage

На данной схеме в обобщённом виде на статически типизированную архитектуру ВС отображена функция Container::DeleteLessThanAverage:



## Схема №4 – GenerateTest

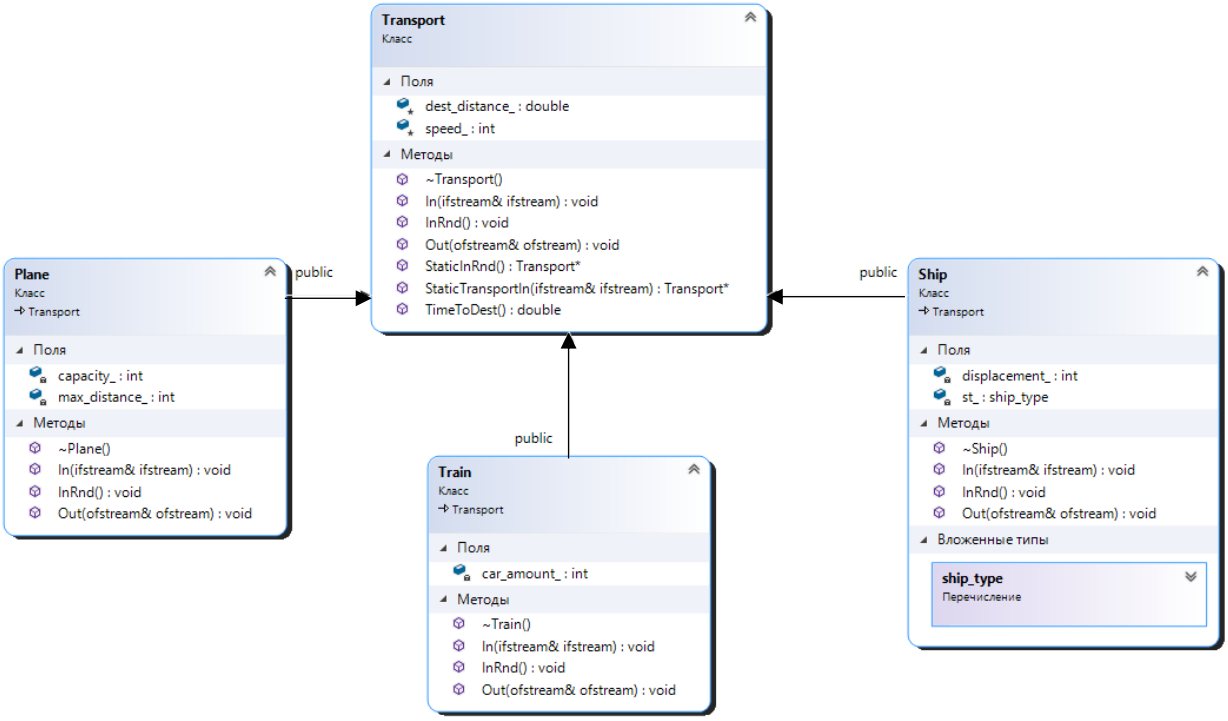
На данной схеме в обобщённом виде на статически типизированную архитектуру ВС отображена функция GenerateTest:



В данных схемах через пометку «(Transport)::» обозначена принадлежность метода к соответствующей типу объекта альтернативе, наследуемой от класса Transport.

**Схема №5 – Иерархия наследования класса Transport**

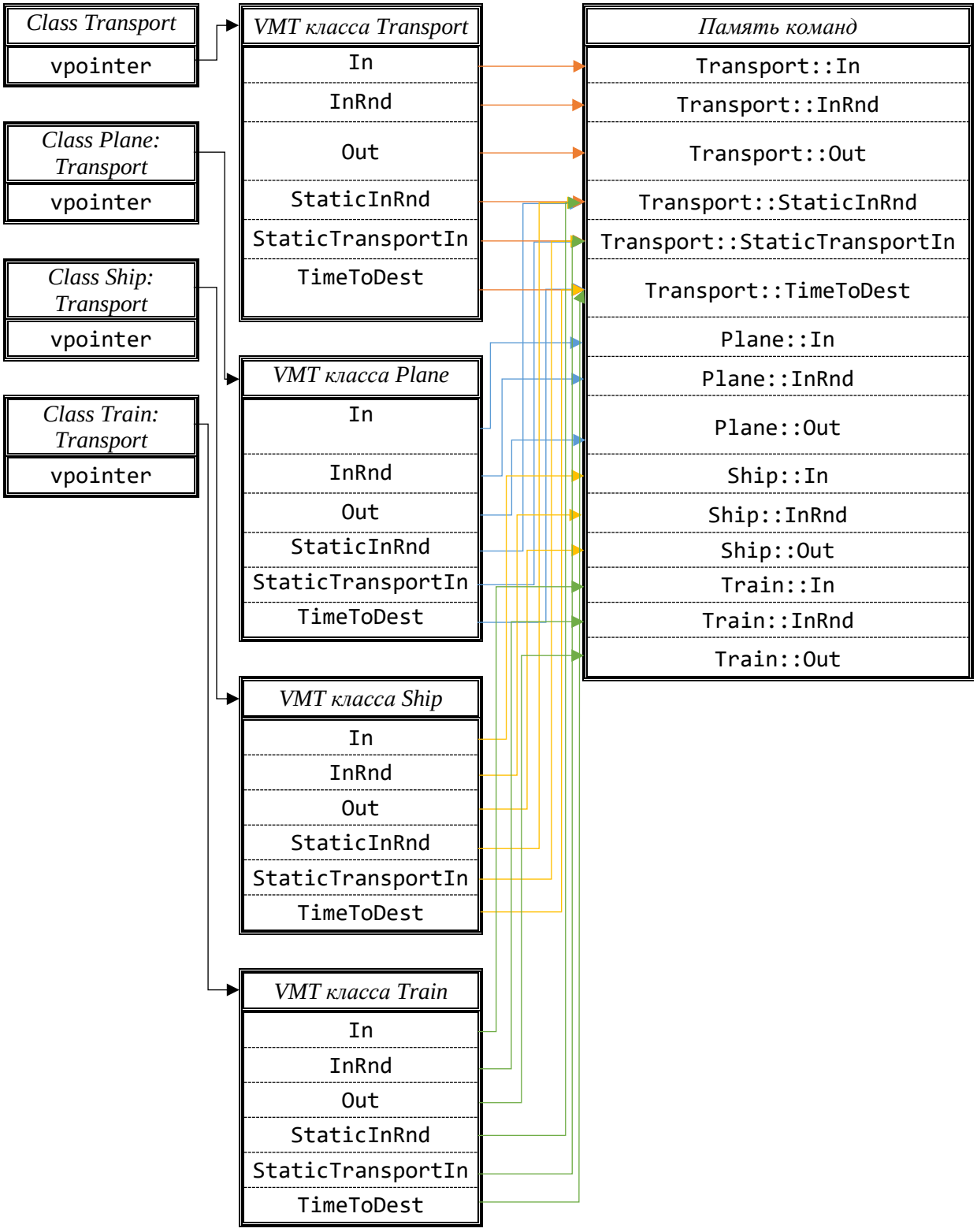
На данной схеме в обобщённом виде изображена иерархия наследования класса Transport для реализации программы при помощи объектно-ориентированного подхода:




Классы Plane, Train и Ship, наследуясь от класса Transport, наследуют от него также protected (закрытые для всех классов, кроме их содержащего и его наследников) поля dest_distance_ и speed_ и содержат собственные private (закрытые для всех классов, кроме их содержащего) поля, например, capacity_ и max_distance_ для класса Plane. Это является примером реализации принципа инкапсуляции объектно-ориентированного подхода.

## Схема №6 – Таблицы виртуальных методов класса Transport и его наследников

На данной схеме в обобщённом виде изображены таблицы виртуальных методов (VMT) класса Transport и его наследников – основной механизм для реализации динамического соответствия и принципа полиморфизма объектно-ориентированного подхода в статически типизированной архитектуре ВС:



# ОСНОВНЫЕ ХАРАКТЕРИСТИКИ ПО

Исходный код программы содержится в 17 файлах. 8 из них являются интерфейсными модулями (заголовочными файлами C++), 9 – модулями реализации (файлами .cpp с определением программных объектов). Общий размер исходных текстов следующий:

| | *С форматированием и комментариями* | *Без форматирования и комментариев* |
|---|---|---|
| *С интерфейсными модулями* | 802 | 436 |
| *Без интерфейсных модулей* | 543 | 339 |

Размер исполняемого файла, полученного после компиляции кода на ОС Linux, равен 303816 байт (296,7 килобайт).

Результаты тестов с использованием тестовых файлов в соответствующих директориях следующие:

| Тестовый файл | Тестовый кейс | Время работы программы |
|---|---|---|
| correct/test1.txt | 12 корректных элементов | 0.00025 секунды |
| correct/test2.txt | 19 корректных элементов | 0.00022 секунды |
| correct/test3.txt | 13 корректных элементов | 0.00022 секунды |
| error/test1.txt | Некорректное число аргументов в строке | 0.00011 секунды |
| error/test2.txt | Некорректный идентификатор альтернативы | 0.00008 секунды |
| error/test3.txt | Удалённая строка параметров | 0.00014 секунды |

Результаты тестов с использованием случайной генерации объектов контейнера следующие:

| *Количество элементов* | *Время работы программы* |
|---|---|
| 20 элементов | 0.0003 секунды |
| 1000 элементов | 0.086 секунды |
| 10000 элементов | 0.15291 секунды |

# СРАВНИТЕЛЬНАЯ ХАРАКТЕРИСТИКА ПО

Ранее в ходе курса программа с идентичным функционалом была разработана на языке C с применением процедурного подхода для статически типизированной архитектуры ВС. От данной программы ввиду выбранной парадигмы предыдущий продукт отличался реализацией типов данных, функционала базового артефакта и альтернатив, а также их связью между собой в исходном коде. Таким образом, можно выделить основные отличия в реализациях, связанных с выбором парадигм программирования:

1. Для организации функционала альтернатив в процедурном подходе применялись обработчики конкретной параметрической специализации, вызываемые для соответствующего типа передачей ссылки на его инстанцию. В объектно-ориентированном подходе, альтернативный функционал реализован при помощи реализации обобщённых в базовом классе (типе) процедур в его альтернативах. В языке C++ для этого используются виртуальные методы и динамическое соответствие.
2. Для связи функций с типом данных, представителем которого является субъект операции, выполняемой функцией, в процедурном подходе применялись указатели на инстанцию этого типа. Функции в таком подходе были ответственны за трансформацию переданной им информации, содержащейся в открытом виде в типе данных, и не были привязаны к самому типу. В объектно-ориентированном подходе функции, выполняющие операции над определенным типом данных, связаны с ним в один класс. Данные класса инкапсулированы и не могут быть получены функциями извне.
3. В процедурном подходе отсутствует концепция полиморфизма, тип данных связан с его расширением при помощи ссылки на его инстанцию. В объектно-ориентированном программировании базовый артефакт связан с альтернативой через механизм наследования и полиморфизма, в связи с чем инстанция альтернативы в то же время является инстанцией базового класса и имеет прямой доступ к его полям и методам.

Сравнение характеристик текущей реализации с предыдущей также выявило ряд различий:

1. Объём исходных текстов несколько уменьшился, что в первую очередь связано со сменой языка программирования.
2. Размер исполняемого файла увеличился в 6,5 раз, что также связано с сменой парадигмы программирования. В частности, ввиду реализации полиморфизма через динамическое соответствие, классы занимают дополнительную память для хранения таблиц виртуальных методов.

Альтернативы же занимают больше памяти ввиду хранения в себе полей и ряда не виртуальных методов обобщённого артефакта.

3. Время работы программы незначительно уменьшилось для случая чтения из файла и в значительной мере (до 20 раз) увеличилось для случая случайной генерации содержания контейнера. Для справки ниже приведена таблица результатов тестов для предыдущей реализации функционала программы:

| *Тестовый файл* | *Тестовый кейс* | *Время работы программы* |
|---|---|---|
| correct/test1.txt | 5 корректных элементов | 0.00021 секунды |
| correct/test2.txt | 19 корректных элементов | 0.00024 секунды |
| correct/test3.txt | 13 корректных элементов | 0.00022 секунды |
| error/test1.txt | Некорректное число аргументов в строке | 0.00013 секунды |
| error/test2.txt | Некорректный идентификатор альтернативы | 0.00019 секунды |
| error/test3.txt | Удалённая строка параметров | 0.00016 секунды |

| *Количество элементов* | *Время работы программы* |
|---|---|
| 20 элементов | 0.00023 секунды |
| 1000 элементов | 0.00298 секунды |
| 10000 элементов | 0.13457 секунды |