Пояснительная записка

Описание полученного задания (вариант 12, функция 11):
Создать консольное приложение, обрабатывающее данные в контейнере.

- Обобщенный артефакт, используемый в задании - животные.
- Базовые альтернативы и уникальные параметры - рыбы (место проживания – перечислимый тип: река, море, озеро…), птицы (отношение к перелету: перелетные, остающиеся на зимовку – булевская величина), звери (хищники, травоядные, насекомоядные… – перечислимый тип).
- Общие для всех альтернатив переменные - название (строка символов), вес в граммах (целое).
- Функция для обработки данных в контейнере - упорядочить элементы контейнера по убыванию используя сортировку с помощью прямого выбора (Straight Selection). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.
- Функция, общая для всех альтернатив - частное от деления суммы кодов незашифрованной строки на вес (действительное число).

Структура проекта:
project/
    cmake-build-debug/
    tests/
    main.cpp
    container.h
    container.cpp
    animal.h
    animal.cpp
    beast.h
    beast.cpp
    bird.h
    bird.cpp
    fish.h
    fish.cpp
    rnd.h

Работа программы:
Пользователь может ввести данные в контейнер двумя способами:

- Задать входной файл с тестовыми данными с помощью команды
  -f *input path* *output path* , где *input path* и *output path* - пути входного и выходного файлов соответственно.
- Задать параметры для генерации тестовых данных:
  -n *size* *files path* , где *size* - количество элементов в контейнере, *files path* - путь, куда будут сохранены сгенерированный файл с входными данными

(***_generated.txt) и с обработанными выходными данными (***_generated.output.txt).

Результаты тестирования программы сохранены в папке project/tests. Тесты 1-5 с использованием входных данных из файла, тесты 6-10 с использованием сгенерированных входных данных.

Примеры использованных команд:

```
./project -f ../tests/test1 ../tests/test1_output
./project -n 100 ../tests/test6
```

Основные характеристики программы:

- Число интерфейсных модулей = 6
- Число модулей реализации = 6
- Общий размер исходных текстов ≈ 19.7 КБ
- Общий размер исполняемых файлов ≈ 29.0 КБ
- Время работы программы для различных тестовых наборов данных:

Тип входных данных	Кол-во элементов в контейнере	Время работы программы со структурами (мс)	Время работы программы с классами (мс)
Входные данные из файла	10	0.000385	0.000442
	20	0.000418	0.000521
Входные данные генерируются	100	0.000929	0.000989
	1000	0.025607	0.033843
	5000	0.723708	0.763432
	10000	3.162394	3.538942

Как мы можем заметить, время работы программы с классами больше, чем со структурами. Предположительно это связано с тем, что при проходе по массиву, объектом которого являются экземпляры класса, увеличивается кол-во ссылок. А при прохождении по массиву, состоящему из структур, при обращении к объекту создается его неизменяемая копия, смотрящая на ту же область памяти.

Также работу замедляет наличие виртуальных методов. Каждый класс, содержащий хотя бы один виртуальный метод, хранит адрес таблицы виртуальных функций. В ходе работы при вызове виртуальной функции сначала берется указатель на объект, затем происходит поиск реализации виртуальной функции в таблице и только потом ее вызов. Процесс поиска конкретной функции по указателю на объект называется поздним связыванием и выполняется во время работы программы.

Инструментальные средства:

- Виртуальная машина Oracle VM VirtualBox
  - Класс ОС: Linux
  - Версия ОС: Ubuntu (64-bit)
  - Кол-во процессоров виртуальной машины: 2
  - Оперативная память: 4096 МБ
- Языки программирования: C/C++
- Библиотеки: stdio.h, stdlib.h, time.h, string.h
- Интегрированная среда разработки: CLion
- Средства сборки проектов: CMake

Структура архитектурного решение: