Куляхтина Алёна Алексеевна, группа БПИ204
Task 12 (Животные), function 10 (Straight Insertion)

Отчет о написании программы

Описание полученного задания

Обобщенный артефакт, используемый в задании	Базовые альтернативы (уникальные параметры, задающие отличительные признаки альтернатив)	Общие для всех альтернатив переменные	Общие для всех альтернатив функции
12. Животные	1. Рыбы (место проживания – перечислимый тип: река, море, озеро…) 2. Птицы (отношение к перелету: перелетные, остающиеся на зимовку – булевская величина) 3. Звери (хищники, травоядные, насекомоядные… – перечислимый тип)	1. Название – строка символов 2. Вес в граммах (целое)	Частное от деления суммы кодов названия животного на вес (действительное число)

10. Упорядочить элементы контейнера по убыванию используя сортировку с помощью прямого включения (Straight Insertion). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

Общие сведения

asm_clean_code – исходный код.

asm_code_with_object – исходный код с файлами типа object и выходными файлами после проведения тестов.

Ниже приведены характеристики проекта asm_clean_code, так как он является оптимальным для перемещения и полной проверки работы.

- Начало работы (из файла Makefile) :
  - (make clean для удаления)
  - make objects
  - make link
- Команды:
  - ./program_asm -f <input file> <output file> - работа с данными из файла
  - ./program_asm -r <number of animals> <output file>- работа со сгенерированными данными

Примеры:

./program_asm -f test/input3.txt output3.txt

./program_asm -r 15 output15.txt

- Обработка ошибок
  - При некорректном вводе параметров или проблем с чтением/записью в файл программа сообщает об этом и завершает работу.
  - При возникновении ошибок при чтении из файла программа пропускает некорректную строку и продолжает работу (сообщение об ошибке выводится в консоль).
- Файлы с примерами вводимых данных находятся в папке tests и в названии содержат слово “input”. Файлы input2.txt и input3.txt направленны на проверку корректности работы сортировки контейнера. Файлы inputErr1.txt и inputErr2.txt показывают корректность работы обработки ошибок при чтении данных из файла.

Спецификации разработки

- Операционная система: Ubuntu (64-bit)
- Архитектура: x86-64
- RAM: 15GB
- Средства сборки – GCC(v11.1.0), NASM(v2.15.05), CMake(v3.21.3)
- Примененные библиотеки на Си
  - time.h
  - stdlib.h
  - stdio.h

Структура проекта

- main.asm – точка входа;
- animal.asm – файл с описанием общего животного;
- container.asm – структура контейнера для чтения и обработки животных;
- репозиторий tests:
  - input1.txt, …, input5.txt – файлы с тестовыми входными данными;
  - inputErr1.txt, inputErr2.txt – файлы с некорректными данными.
- Makefile – набор инструкций для работы с программой

Характеристики проекта

- Количество программных объектов –
- Размер используемых файлов - ~29.3Кб
- Размер проекта - ~ 36.0Кб
- Время генерации животных (рассматривается среднее значение) :

• 10 элементов	• 0,000401 с
• 100 элементов	• 0,001835 с
• 1000 элементов	• 0,120518 с
• 10000 элементов	• 10,18309 с

Дополнительные характеристики:

- Модульная структура
- Обработка ошибок при работе с файлами
- Обработка ошибок при некорректных параметрах и данных в файлах
- Расширенные тестовые наборы
- Документация кода

Сравнение с другими работами

	Python	**C**	**NASM**
Размер проекта (без учета репозитория с тестами)	~ 10.6Мб	~40.0Кб	~160.0Кб
Время			
• 10 эл	0,01724 с	0,000573 с	0,000401 с
• 100 эл	0,05518 с	0,001862 с	0,001835 с
• 1000 эл	1,93840 с	0,098861 с	0,120518 с
• 10000 эл	165,109 с	0,160910 с	10,18309 с

*для каждого языка взято среднее значение по времени из 10 экспериментов

Код на NASM работает примерно с той же скоростью, что и C для небольших наборов данных, но очень отстаёт по времени при обработке и генерации более 10000 элементов. Несмотря на то, что по показателям времени и памяти самым не эффективным языком оказался Python, ассемблер все же более неудобен при написании и использовании. Для написания программы я была вынуждена писать отдельный shell-скрипт, чтобы после воспользоваться objconv для конвертации кода C в NASM. Это помогло перенести основные структуры программы, но после пришлось вручную удалять лишние компоненты, а также конвертировать определенные участки кода из шестнадцатеричного формата (Hex) в строки (ASCII). Такой способ потребовал оптимизации кода на C, который был написан для Задания 1. Помимо неудобств при конвертации, ассемблеру не хватает возможностей абстракции от базовых функций. Отсутствие этой возможности делает язык неудобным для применения в написании проектов. Данный язык подходит для оптимизаций существующих продуктов.