Отчет о домашнем задании по

«Архитектуре вычислительных систем» #4

Студент: Кунин Илья, БПИ203

1. Вариант задания

Variant number = 96 Number of task = 12 Number of function = 7

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 незашифрованной строки на число символов в этой строке на вес (действительное число)

7. Упорядочить элементы контейнера по возрастанию используя сортировку методом деления пополам (Binary Insertion). В качестве ключей для сортировки и других действий используются результаты функции, общей для всех альтернатив.

2. Примеры команд запуска (инструкций)

Команды необходимо запускать из папки с исполняемым файлом (task)

./task -n 1500 output.txt

./task -f input.txt output.txt

3. Сортировка

В программе производится Binary Insertion Sort по возрастанию параметра Value.

4. Как должен быть устроен файл INPUT.TXT (Формат данных)

Каждая строчка состоит из 4 слов и идет в формате A B C D, где

- A - число от 0 до 2 *(0 - fish, 1 - bird, 2 – beast)*
- B - строка без пробелов *(имя)*
- C - целое число *(вес в граммах)*
- D - число от 0 до 1 (*специальная характеристика)*

Пример строки файла: 0 FishName 100 1

5. Содержание архива

Файл отчет.pdf - этот файл

Подкаталог app – исходный текст программы, исполняемый файл, скрипт компиляции
Подкаталог tests – папка с программой для создания тестов, скриптом запуска тестов и четырьмя подкаталогами – тестовыми входными наборами (рандомными и ручными) и результатами работы на этих наборах

6. Основные характеристики программы

Тестовый файл	Количество элементов	Время C	Время C++	Время Python	Время NASM
input01.txt	3	0.0001	0.0001	0.0014	0.0001
input02.txt	20	0.0004	0.0003	0.0079	0.0002
input03.txt	20	0.0003	0.0003	0.0079	0.0003
input04.txt	40	0.0004	0.0005	0.0114	0.0004
input05.txt	100	0.0009	0.0010	0.0277	0.0009
input06.txt	500	0.0041	0.0053	0.1259	0.0027
input07.txt	500	0.0043	0.0052	0.1150	0.0029
input08.txt	500	0.0044	0.0052	0.1140	0.0023
input09.txt	10 000	0.1087	0.1422	3.6659	0.0824
input10.txt	10 000	0.1135	0.1533	3.6074	0.0824
random	10	0.0002	0.0001	0.0034	0.0001
random	100	0.0005	0.0008	0.0386	0.0004
random	1 000	0.0061	0.0113	0.2544	0.0053
random	5 000	0.0382	0.0624	1.7340	0.0325
random	10 000	0.0876	0.1455	3.8990	0.0817
Коэффициент	времени	1,00	1,35	31,71	0,85

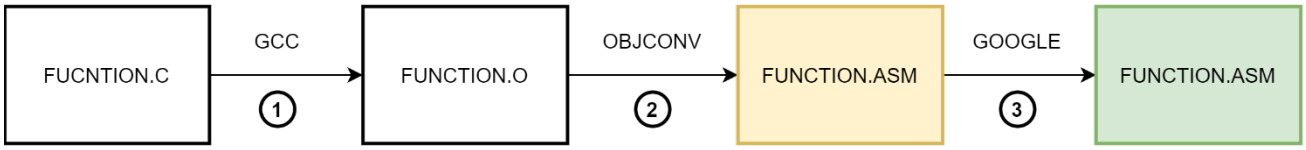
Характеристика	C	C++	Python	NASM
Число внешних интерфейсных модулей	4	3	3	17
Число внутренних интерфейс модулей	6	6	-	-
Число внутренних модулей реализаци	4	7	3	1
Общий размер исходных текстов программы (байт)	17 604	11 100	5 727	22 623
Общий размер исходных текстов программы на диске (байт)	28 672	24 576	8 192	24 576
Размер исполняемого файла (байт)	23 224	40 680	-	19 400
Размер исполняемого файла на диске (байт)	24 576	40 960	-	20 480

7. Дополнительные пояснения

Какие использовались инструменты для написания / компиляции программы?

GCC (Ubuntu 9.3.0-17ubuntu1 20.04)
OBJCONV (Object file converter 2.52 for x86 and x86-64 platforms)
NASM (NASM version 2.14.02)

Как происходило написание программы?



Написание происходило на языке NASM. Когда я упирался, в какой-то момент, который у меня не получается реализовать, я поступал следующим образом.

1. Писал код функции, которую хочу реализовать, на языке C. После чего, с помощью GCC, компилировал ее в function.o
2. Function.o в свою очередь конвертировал в файл function.asm с помощью OBJCONV. По идеи, OBJCONV принимает параметром -fnasm и понимает, что нужно выдавать код корректным под компилятор NASM. Но на практике даже небольшие программы, при попытке компиляции такого файла под NASM, начинают выдавать множество предупреждений (warning) и ошибок (error).
3. После поисков в интернете и исправления ошибок в файле function.asm, функция успешно компилируется под NASM.

После всех этих этапов берется файл task.asm и пишется своя функция по образу и подобию. Параллельно разбирается устройство кода function.asm.

Как происходила компиляция программы?

Получение исполняемого файла из исходного текст программы на языке NASM происходило в два этапа.

1. Сначала файл task.asm компилировался в файл task.o с помощью инструмента NASM.
2. После чего task.o «прилинковывался» к исполняемому файлу task с помощью gcc.

Данные команды можно найти в каталоге app, в скрипте compile.sh.