**Задание 14**. Автомобильный транспорт

Сущности:

1. Грузовик (грузоподъемность кг – целое)

2. Автобус (пассажировместимость – короткое целое)

3. Легковой автомобиль (максимальная скорость – короткое целое)

Общие характеристики:

1. Емкость топливного бака в литрах (целое)

2. Расход топлива на 100 км в литрах (действительное)

Функция:

Максимальное расстояние, которое может пройти автомобиль в км (действительное число)

**Задание 22**. Переместить в конец контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, больше, чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы сдвинуть к началу без изменения их порядка.

число интерфейсных модулей: 1

число модулей реализации: 6

общий размер исходных текстов: 7,79Кб

**./program -n 10 tests/out\_randomn10.txt tests/out\_randomn10\_sorted.txt**

**Время работы = 0m0.038s –** 10 транспортных средств (т.с.)

**./program -n 30 tests/ out\_randomn30.txt tests/out\_randomn30\_sorted.txt**

**Время работы = 0m0.020s –** 30 т.с.

**./program -n 50 tests/ out\_randomn50.txt tests/out\_randomn50\_sorted.txt**

**Время работы = 0m0.016s –** 50 т.с.

**./program -n 500 tests/ out\_randomn500.txt tests/out\_randomn500\_sorted.txt**

**Время работы = 0m0.022s –** 500 т.с.

**./program -n 1000 tests/ out\_randomn1000.txt tests/out\_randomn1000\_sorted.txt**

**Время работы = 0m0.16s –** 1000 т.с.

**Test01.txt: Время работы = 0m0.0008s**

**Test02.txt: Время работы = 0m0.0012s**

**Test03.txt: Время работы = 0m0.0015s**

**Test04.txt: Время работы = 0m0.0010s**

**Test05.txt: Время работы = 0m0.0011s**

**Запуск программы:**

Исходные данные для тестирования содержатся в каталоге tests.

**Пример тестирования:**

python main.py 1000 .\tests\out\_randomn1000.txt

**Описание ВС**

|  |  |  |  |
| --- | --- | --- | --- |
| **Память программы** | **Таблица имен** | **Память имен** | |
| main.py | strArray | list | 20+4\*длина |
| cont | list | 20+4\*длина |
| container | module | container.py |
| figNum | int | Станд. 12 байт, но зависит от системы |
| container.py | module |  |
| inputFileName | str | 21+длина |
| outputFileName | str | 21+длина |
| container.py |  |  |  |
| def ReadStrArray | cont | list | 20+4\*длина |
| strArray | list | 20+4\*длина |
| i | int | Станд. 12 байт, но зависит от системы |
| figNum | int | Станд. 12 байт, но зависит от системы |
| inputStr | str | 21+длина |
| key | int | Станд. 12 байт, но зависит от системы |

**Сравнительный анализ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Показатель** | **С** | **С++** | **Python** | **Данные для сравнения** |
| Время работы | -> | На небольшом количестве данных значений времени исполнения программы особо не отличаются, однако на большом объеме наблюдается значительное уменьшение времени | На большом кол-ве данных видно сильное преимущество по времени работы программы | На 10 значениях: 0,052s/0,089s/0.038s  На 30 значениях  0,046s/0,110s/0.020s  На 1000 значениях  4,119s/1,102s/0.016s |
| Размер исходных текстов | 70,0 Кб | 46,7 Кб | 7,79 Кб | - |
| Агрегирование | Процедурное | ООП | Динамическое |  |

Преимущества и недостатки ОО перед процедурным агрегированием:

- ОО агрегирование предоставляет более устойчивую ассоциацию объекта, поддержанную непосредственным размещением процедур внутри классов,

- наличие на один аргумент меньше в методах класса против дополнительной передачи объекта в процедурном подходе, что сказывается на производительности,

- использование классов улучшает восприятие кода, так как при реализации структуры методы обработки и дальнейшей работы реализуются отдельно от структуры, что уменьшает читабельность кода,

- однако использование структур позволяет инкапсулировать методы обработки и скрыть ненужный для пользователя код;

Преимущества и недостатки динамического агрегирования:

- Благодаря определению типов во время исполнения программы сильно облегчается метапрограммирование

- Благодаря гибкости кода в рантайме и интроспекции получается на порядок проще и быстрее писать универсальные алгоритмы

- с точки зрения ошибок типизации (а собственно их и ставят в недостаток динамически типизированным языкам) разница минимальна

- по сравнению с другими, в динамической типизации нужно держать в голове намного больше сущностей для эффективного и правильного использования

- скорость при использовании динамической типизации, в общем случае, сильно больше. Однако в рамках данной задачи такого не наблюдается