Задание №1

Задача:

Разработать приложение для конвертации между единицами измерения расстояния с поддержкой метрической и имперской систем мер. Соотношения для конвертации вы можете взять из таблицы. По умолчанию, приложение должно распознавать метры (*m*), сантиметры (*cm*), дюймы (*in*) и футы (*ft*), и поддерживать конвертацию между любыми единицами.

Также необходимо реализовать возможность расширять список поддерживаемых единиц путем задания правил конвертации посредством JSON файла. Формат JSON файла - на ваше усмотрение. В качестве примера, расширьте ваше приложение добавив в файл значения для миллиметров (*mm*), ярдов (*yd*) и километров (*km*).

Входящие параметры:

Объект в JSON формате, содержащий расстояние заданное для конвертации (*distance*) со значением (*value*) и шкалой (*unit*), а также обозначение единицы для шкалы, в которую должна быть произведена конвертация (*convert_to*). Например:

```
{“distance”: {“unit”: “m”, “value”: 0.5}, “convert_to”: “ft”}
```

Выходные данные:

Объект в JSON формате, содержащий полученное значение расстояния, округленное до сотых, а также обозначение соответствующей единицы измерения, например:

```
{“unit”: “ft”, “value”: 1.64}
```

Задание №2

Задача:

Разработать простое приложение для сортировки и отбора данных по заранее заданным правилам. Приложение должно уметь работать со списками JSON объектов произвольной структуры, отбирать объекты, содержащие ключи с определенными значениями, а также сортировать объекты по значениям, используя естественный порядок сортировки.

Например, если для данных вида:

```
{“data”: [{“name”: “John”, “email”: “john2@mail.com”},
          {“name”: “John”, “email”: “john1@mail.com”},
          {“name”: “Jane”, “email”: “jane@mail.com”}]}
```

задать условие:

```
{“condition”: {“include”: [{“name”: “John”}], “sort_by”: [“email”]}}
```

содержащее два правила - *include* и *sort_by* (где правило *include* принимает набор пар ключ:значение для проверки записей на соответствие, а правило *sort_by* принимает набор ключей для сортировки), результатом будет объект, содержащий только записи с именем *John*, отсортированные по ключу *email*:

```
{“result”: [{“name”: “John”, “email”: “john1@mail.com”},
            {“name”: “John”, “email”: “john2@mail.com”}]}
```

Планируя подход к дизайну кода приложения, необходимо предусмотреть возможность расширять функционал через добавление в код новых “модулей” с правилами. Важно, чтобы все модули имели между собой идентичную структуру, были изолированы друг от друга и остального кода приложения, и взаимодействовали с основным кодом, используя один и тот же подход. В качестве примера, вы можете добавить модуль с дополнительным правилом *exclude*, которое будет отбрасывать записи, содержащие ключи с определенным значением.

Входящие параметры:

JSON объект со списком данных (*data*), и условием для обработки (*condition*):

```
{“data”: [{“user”: “mike@mail.com”, “rating”: 20, “disabled”: false},
          {“user”: “greg@mail.com”, “rating”: 14, “disabled”: false},
          {“user”: “john@mail.com”, “rating”: 25, “disabled”: true}],
 “condition”: {“exclude”: [{“disabled”: true}], “sort_by”: [“rating”]}}
```

Выходные данные:

JSON объект с данными полученными после применения условия обработки (*result*):

```
{“result”: [{“user”: “greg@mail.com”, “rating”: 14, “disabled”: false},
            {“user”: “mike@mail.com”, “rating”: 20, “disabled”: false}]}
```

Задание №3

Задача:

Необходимо реализовать программируемый опросник, в котором порядок и список вопросов зависят от переданной конфигурации в JSON формате. Опросник должен поддерживать только вопросы с вариантами ответов, например:

```
{"What is your marital status?": ["Single", "Married"]}
{"Are you planning on getting married next year?": ["Yes", "No"]}
{"How long have you been married?": ["Less than a year", "More than a year"]}
{"Have you celebrated your one year anniversary?": ["Yes", "No"]}
```

Вопросы в опроснике должны динамически определяться на основании ответов пользователя - следующий вопрос должен зависеть от ответа на предыдущий. Вам необходимо продумать, как будет работать эта логика, и разработать формат JSON конфигурации (он будет отличаться от примера выше), которая позволит задавать правила, связывающие вопросы с конкретными ответами.

Для тестирования работы опросника требуется создать скрипт, работающий с кодом логики опросника и проходящий по всем возможным путям опросов. Для решения задачи не требуется реализовывать пользовательский интерфейс, достаточно написать скрипт и логику опросника, последовательно предоставляющую вопрос и принимающую ответ.

Входящие параметры:

JSON конфигурация, в выбранном вами формате, с вопросами и допустимыми ответами, задающая связь между ответами пользователя и последующими вопросами.

Выходные данные:

JSON объект, являющийся результатом работы скрипта тестирования, с информацией о количестве всех возможных путей опросов (*paths.number*), и всеми возможными последовательностями вопросов с ответами (*paths.list*):

```
{paths: {number: 3, list: [
    [{"What is your marital status?": "Single"},
     {"Are you planning on getting married next year?": "Yes/No"}],
    [{"What is your marital status?": "Married"},
     {"How long have you been married?": "Less than a year"}],
    [{"What is your marital status?": "Married"},
     {"How long have you been married?": "More than a year"},
     {"Have you celebrated your one year anniversary?": "Yes/No"}],
]}}
```

Задание №4 * (опциональное)

Задача:

Необходимо разработать алгоритм, позволяющий определить порядок задействования топливных капсул ионных двигателей спутника для совершения заранее заданного ряда маневров. Капсулы имеют 5 разновидностей, и разработаны для получения прироста скорости в **2, 4, 6, 8 или 10** м/с. Каждый маневр требует получить прирост скорости **от 1 до 12** м/с. Для совершения одного маневра спутник может одновременно использовать два двигателя:

- первый - позволяет получить прирост скорости **равный значению** используемой капсулы. Например, для капсулы 4 м/с, прирост будет ровно 4 м/с.
- второй - позволяет получить прирост скорости **равный половине значения** капсулы. Например, для капсулы 4 м/с, прирост будет ровно 2 м/с.

Для совершения одного маневра запускать каждый двигатель можно максимум **один раз**. Также, для одного маневра, допускается суммарный прирост скорости **меньше** требуемого (например если запаса капсул недостаточно), но **превышение заданного приращения скорости запрещено**. Капсулы **невозможно использовать повторно**.

Алгоритм должен определять такой порядок использования капсул, при котором сумма приращений скорости по всем маневрам, и при соблюдении всех условий, **будет максимальной**, задавая таким образом наиболее точную траекторию. Количество маневров, допустимое приращение скорости на каждом из них, а также доступный набор капсул может быть произвольным.

Мы рекомендуем решать эту задачу используя **генетический алгоритм** (возможно, с определенными модификациями).

Входящие параметры:

JSON объект, содержащий массив произвольной длины с целыми положительными приращениями скорости, которые требуется достичь на каждом из маневров (*corrections*); и массив произвольной длины содержащий список доступных топливных капсул (*cells*), например:

```
{“corrections”:  [1, 12, 7, 1], “cells”: [8, 4, 6, 2, 2] }
```

Выходные данные:

JSON объект, содержащий последовательность использования капсул для первого двигателя (*main_thruster*); последовательность использования капсул для второго двигателя (*sec_thruster*); и итоговое полученное приращение скорости (*delta_velocity*). Например:

```
{“main_thruster”: [0, 8, 6, 0], “sec_thruster”: [2, 4, 2, 0], “delta_velocity”: 18}
```

Примечания к выполнению заданий

Задачу под номером 4 решать не обязательно, но решение определенно будет плюсом - генетический алгоритм поиска, часто используемый для решения оптимизационных задач, работает аналогично механизмам эволюции и естественного отбора в природе, и мы надеемся, что данную задачу вам будет решать интересно.

Также, во время написания программ, обратите внимание на следующее:

- код тестовых приложений необходимо разбить по логическим блокам, так чтобы он был компактным, легко читаемым, и не содержал повторений
- приложения должны корректно реагировать на широкий спектр возможных входных значений, и обрабатывать исключительные ситуации
- все задачи должны быть решены наиболее оптимальным образом, с наименьшим использованием ресурсов памяти и процессора

Выполненное задание (исходный код) присылайте на email hr@sysgears.com, в качестве темы письма укажите: “Выполненные задания. [Имя Фамилия]”.

Дополнительно к письму необходимо прикрепить резюме.