Задача:

В далекой-далекой галактике… почтовая служба Новая-Галактика должна доставить грузы по назначению. В распоряжении почтовой службы есть только один грузовой космический корабль. Корабль имеет ограниченное пространство и грузоподъемность. Во время путешествия в космосе, корабль и экипаж расходуют ресурсы (еда, вода, топливо). Пополнить корабль ресурсами можно только на базе почтовой службы.

Нужно найти такой маршрут, при котором корабль проделает наименьший путь и доставит все грузы по назначению.

Входные данные:

Программа должна быть реализована на C++. Класс, который производит расчет пути, должен наследоваться от IGalaxyPathFinder, который находиться в IGalaxyPathFinder.h.

Написанная функция должна принимать на вход JSON-файл со следующей информацией:

- Информация о корабле: габариты кузова (половина по каждому измерению), максимальный подъемный вес (груз + ресурсы), максимальный вес ресурсов, расход ресурсов на расстояние
- Информация о точках назначения: координаты в пространстве
- Информация о каждом грузе: габариты (половина по каждому измерению), вес, точка назначения

Форма груза и грузового отсека - прямоугольный параллелепипед. Груз и грузовой отсек не могут быть повернуты.

Единицы измерения: вес в тоннах, расстояния в астрономических единицах (ae), размер груза и грузового отсека в метрах, расход топлива в т/ae.

Примеры входного JSON в файлах: inputDataX.json

Типы данных и единицы измерения:

```
{
   "ship":{
      "maxCarryingCapacity":{
         "half_x":0,                // int (м)
         "half_y":0,                // int (м)
         "half_z":0                 // int (м)
      },
      "maxResourcesWeight":0.0,     // float (т)
      "maxCarryingWeight":0.0,      // float (т)
      "resourcesConsumption":0.0    // float (т/ae)
   },
   "targetPoints":[
      {
         "y":0.0,                   // float (ae)
         "x":0.0,                   // float (ae)
         "z":0.0,                   // float (ae)
```

```
         "pointId":0              // int
      }
   ],
   "boxes":[
      {
         "half_x":0,              // int (м)
         "half_y":0,              // int (м)
         "half_z":0,              // int (м)
         "weight": 0.0,           // float (т)
         "targetPointId":0,       // int
         "boxId":0                // int
      }
   ]
}
```

База имеет pointId == 0. Корабль изначально находится в этой точке. Заправляться и загружать груз можно только на базе. Сумма веса груза и ресурсов не должна превышать максимальный подъемный вес корабля.

Выходные данные:

Написанная функция должна генерировать JSON-файл с записями перемещения корабля из одной точки в другую.

Выходной JSON содержит массив шагов, каждый шаг содержит информацию:

- Точка назначения
- сколько ресурсов загружено (если вылет с базы); ноль, если текущий шаг был сделан не с базы.
- какой груз доставляет и как груз располагается в кузове. Расположение каждой коробки относительно центра грузового отсека корабля.

При возвращении на базу, оставшиеся ресурсы обнуляются (теряются). В конце, когда весь груз доставлен, корабль всегда должен возвращаться на базу.

Пример выходного JSON в файле: outData.json

```
{
  "steps":[
    {
      "shippedBoxes":[
        {
          "boxId":0,              // int
          "x":0,                  // int (м)
          "y":0,                  // int (м)
          "z":0                   // int (м)
        }
      ],
      "shippedResources":0.0,     // float (т)
```

```
      "destinationPointId":0       // int
    }
  ]
}
```

Проверка результатов:

Результат будет оцениваться по нескольким критериям:

1. Решение задачи - минимальное суммарное расстояние поездки.
2. Скорость выполнения расчетов.
3. Расход памяти - количество выделений/освобождений, максимальное значение выделенной памяти.

Каждое решение будет запускаться на большом количестве разных входных данных, в том числе краевых/некорректных значениях (например, один из грузов больше отсека корабля). Программа должна правильно обрабатывать соответствующие входные данные, а именно, после доставки всех грузов и возвращения на базу, последним фиктивным шагом - “перевозка” всех грузов, которые невозможно доставить, с базы на базу (destinationPointId == 0) без упаковки (все коробки с координатами (0,0,0)).

Для проверки работы программы будет использоваться только cpp-файл с классом наследником IGalaxyPathFinder, который будет добавлен в общий проект со всеми решениями. Смотри ExampleSolution.cpp.

Программа будет компилироваться с помощью VisualStudio 2019 в схеме Release, с оптимизацией “/O2”

С любыми вопросами обращайтесь к контактным лицам.
Контактные лица: Владимир (v.ivanov@dragonlk.com), Евгения (e.beirak@dragonlk.com), Иван (i.kozmyk@dragonlk.com), Ольга (o.strukova@dragonlk.com)