Task: Implementation (Parking)

You need to create the backend for a new computerized management system for a parking garage. When customers enter the garage, the system detects their car size and automatically generates a ticket with a parking space number and directions to the space. When leaving the garage, the driver provides the ticket and credit card, and is charged accordingly.

The garage provides three levels of parking. Level one has six rows - rows one and two are for large vehicle parking, the next two rows are for handicapped-only parking, and the last two rows are for motorcycle-only parking. The remaining levels each have eight rows - the first four rows with compact size parking and the last four rows has large vehicle parking. Each row contains ten parking spaces of equal size.

The garage also accommodates buses, which take up five large parking spaces sequentially in the same row.

Each parking size is charged in 15 minute intervals as follows:

- motorcycle parking spaces: $2.50/hr. For motorcycles only.
- compact car parking spaces: $5/hr. For compact cars or motorcycles.
- large car parking spaces: $7.50/hr. For large or compact cars and motorcycles. Buses may also park here, if there are five sequential spaces available in the row (charge is $37.50/hr).
- handicapped parking spaces: $5/hr. For car driving handicapped placard holders only. If no handicapped spaces are available, a regular single space may be provided instead (larger preferred). No motorcycles or buses may park here, handicapped placards are only meaningful for cars.

Your task is to implement, in Python 2.7, the functions **park()**, **unpark()**, and **init()** in file **park_unpark.py**. park() and unpark() will be called from a legacy multi-threaded API that is triggered by the ticket machines when a car enters and leaves the garage entrance. init() is called on system startup. Declarations for these functions are as follows:

```
# defines the 15 minute minimum parking interval
MINIMUM_PARKING_INTERVAL_SECONDS = 15*60


class InvalidInputError(Exception):
    """
    Raised if the provided API input is invalid.
    """


def park(size, has_handicapped_placard):
```

```
    """
    Return the most appropriate available parking space for this vehicle. The most
    appropriate space is always the cheapest valid available space for the vehicle.
    Spaces closer to the entrance are preferred (lower levels, rows and spaces are
    closest to the entrance -  eg: level 1 row 4 space 5 is closer than level 2 row 3
    space 1), unless returning a closer space is not the cheapest or will result in
    less available bus parking.

    :param size: vehicle size. For now this is 1=motorcycle, 2=compact car, 3=large car,
         4=bus
    :type size: `int`
    :param has_handicapped_placard: if True, provide handicapped space (if available).
    :type has_handicapped_placard: `bool`
    :returns: parking location. tuple of (level, row, space), or None if no spaces available.
        Level, row and space numbers start at 1.
    :rtype: tuple(`int`,`int`,`int`)
    :raises InvalidInputError: if size invalid.
    """


def unpark(location):
    """
    Return the charge for parking at this location based on location type and time spent.
    Parking time is rounded up to the nearest MINIMUM_PARKING_INTERVAL_SECONDS.
    Amount charged to rounded up to the nearest penny.

    :param location: parking space the vehicle was parked at as tuple (level, row, space)
    :type location: tuple(`int`,`int`,`int`)
    :returns: The amount that the parker should be charged.
    :rtype: float
    :raises InvalidInputError: if location invalid or empty.
    """


def init():
    """
    Called on system initialization before any park/unpark function is called.
    """
```

Make reasonable assumptions (and document these) when requirements are not clear. Your code should reflect your best design and coding practices.

Your solution should be returned as a ZIP archive with the following contents:

- README: optional. Place any additional information that you would like us to know about your solution in this text file, if applicable.
- park_unpark.py: contains the completed park, unpark, init functions declared above.
- <all other files>: All other python source files or other files/directories in your solution.