Question 1. [10 MARKS]

Write the body of the function below so that it satisfies its docstring. Assume that module `stack.py` defines a class `Stack` that provides the usual methods: `is_empty()`, `push(item)`, `pop()`.

For full marks, your code must *not* depend on any details of the implementation of class `Stack`. In other words, the only thing you can do with a `Stack` object is to call some of its methods.

```
from stack import Stack

def size(stk):
    """(Stack) -> int
    Return the number of items on Stack stk, *without* modifying stk.
    (It's OK if the contents of stk are modified during the execution of this
    function, as long as everything is restored before the function returns.)
    """
    # Hint: You can use more than one stack.
```

```
# this question is alike to the one using stack implement
# Queue, we need two ~~stack~~ stack or a stack and a queue,
#or a stack and a list.


# An accumulator
  temp = Stack().
  count = 0
# empty stk, put everything into temp.
  while not stk.is_empty():
          temp.push( stk.pop())
          count += 1    # count the num of element.
# put everything back.
  while not temp.is_empty():
          stk.push ( temp.pop())

  return count
```

Question 1. [10 marks]

```
class B:
    def __init__(self: 'B', name: str) -> None:
        self._name = name

    def __str__(self: 'B') -> str:
        return ("I'm a B named " + self._name)

    def mB(self: 'B', n: int) -> None:
        print("mB", self._name, str(n))

class C(B):
    def __init__(self: 'C', name: str) -> None:
        B.__init__(self, name)
        self._name_len = len(name)

    def __str__(self: 'C') -> str:
        return ("I'm a C named " +
                self._name + "-" +
                str(self._name_len))

    def mC(self: 'C', n: int) -> None:
        print("mC", end = " ")
        B.mB(self, n+1)

class D(C):
    def __init__(self: 'D', name: str) -> None:
        C.__init__(self, "Super " + name)
        self._junk = self._name_len + 3

    def mC(self: 'D', n: int) -> None:
        print(str(self._junk), end = " ")
        C.mC(self, n+1)

if __name__ == "__main__":
    b = B("Bob")
    c = C("Carole")
    d = D("Dan")

    print("1:", b)
    print("2:", c)
    print("3:"d)     # No comma here.
    print("4:")
    b.mB(12)
    print("5:")
    c.mB(13)
    print("6:")
    d.mB(14)
    print("7:")
    b.mC(15)         # B has no method MC
    print("8:")
    c.mC(16)
    print("9:")
    d.mC(17)
```

Note.
newlines
Here

Write the output of the code in the box below. If a line of code would cause Python to crash, write CRASH, and then continue tracing as though that line had been commented out of the main block.

1: I'm a B named Bob
2: I'm a C named. Carole-6.
3: Crash.
4:
mB Bob 12
5:
mB Carole 13
6:
mB Super Dan 14.
7:
Crash.
8:
mC mB carole 17.
9:
12 mC mB Super Dan 19

Question 3. [25 MARKS]

Write a series of classes, complete with documentation that satisfiy the following specification.

- A `building` has an `address` (an arbitrary string), and a number of rooms, provided at the time of construction.
- A `room` has a `name` (an arbitrary string) and a `square_footage` (float), provided at the time of construction.
- When printed, a `building` prints the sum of the square footages of all of its rooms
- A `house` is a type of `building` with at most 10 rooms, and prints `"Welcome to our house"`, plus the details of all of its rooms (name and square footage, separated by commas)
- If a `house` is created with too many rooms, a `BuildingCodeViolationError` should be raised
- A `business` may have any number of rooms, but no room may be named `Bedroom`, or have a square footage of less than 100, or else a `InvalidBusinessError` should be raised.
- It is possible to rename any room in any building (by specifying an old and a new name), but only a business can change the square footage of their rooms (by specifying the room name and the new square footage)
- Any invalid/improper input to any parameter (aside from those already mentioned) should raise a `BuildingCreationException`

Write a main code body (that should only execute when this file is run directly, not when it is imported), to perform the following:

- prompt the user for a type of building, address and number of rooms
  - *reminder:* ***`input("prompt")`*** *prompts the user for input and returns their response*
- prompt the user for room names and square footages until all rooms have been named
- if the user's inputs cause `BuildingCreationException` print `"oops..."`
- if the user's inputs cause any other type of error print `"you can't do that"`
- your main body code should not test the input, all decisions about what is/isn't valid input must be made by the classes you created in the first part of the question

```
'''这道题的内容较多，负荷量可以相当于一个小assignment.
   如下是我自己做的这道题的答案供小伙伴们参考，design的方式
   并不唯一，只要可以达到目标就是对的， 我写了大量的comments，
   里面有我在分析这道题时候的思路，如果想不出来可以参考一下.
                                        - Gary
'''


class Building:
    # 题中第一条给出了init中的parameter address 和 num_of_rooms
    def __init__(self, address, num_of_rooms):
        self.address = address
        self.num_of_rooms = num_of_rooms
        # 根据题中最后一句话， 在 class里面 check Exception.
        if num_of_rooms <= 0:
            raise BuildingCreationException()
        # 注意： room list在此题中为隐含条件
        self.rooms = []

    # 题中第三条给出了__str__的structure
    def __str__(self):
        footage = 0
        for room in self.rooms:
            # 注意 这里 square_footage的名字要和下面room class中一致
            footage += room.square_footage
        return str(footage)  # 这里要注意， __str__返回的应该是个string

    # 题中倒数第二条写明可以rename room, 并且写明是要根据name
    def rename(self, old_name, new_name):
        # find the room with old_name
        for room in self.rooms:
            if room.name == old_name:
                room.name = new_name
                break  # break to make it only rename once (optional)
    # 根据 题中第六条中， 声明了business 在添加room的时候不可以名字叫bedroom,
    # 我们可以推测出需要有一个method来添加room, 并且在business中需要做更改
    def add_room(self, name, square_footage):
        if len(self.rooms) == self.num_of_rooms:
            raise BuildingCreationError()  # 根据最后一条
        else:
            self.rooms.append(Room(name, square_footage))


class Room:
    # 题中第二条给出了init中的parameter， name 和 square_footage
    def __init__(self, name, square_footage):
        self.name = name
        self.square_footage = square_footage

    # 题中第四条后半句有隐含提出要print每一个room, 所以我们要implement __str__
    def __str__(self):
        # 模板来啦!
        return "{}: {}".format(self.name, self.square_footage)

class House(Building):  # 即使是 common sense， house也是building的subclass啦!
    # 这道题类似我们课上那道题，__init__中有了room的数量限制，是锦上添花型.
    def __init__(self, name, num_of_rooms):
        # check num of rooms
        if num_of_rooms > 10:
            raise BuildingCodeViolationError() # create corresponding class below
        Building.__init__(self, name, num_of_rooms)

    # __str__ 根据题中第四条， __str__ 为完全override.
    def __str__(self):
        # 由于不知道room的数量， 我们使用accumulator来construct string
        result = ""
        for room in self.rooms:
            result = result + str(room) + ","
        return "Welcome to our house " + result[:-1]  # delete the last ","
```

```python
class Business(Building):
    # no change on __init__
    # changes on add_room
    def add_room(self, name, square_footage):
        if name == 'Bedroom' or square_footage < 100:
            raise InvalidBusinessError()
        Building.add_room(self, name, square_footage) # 锦上添花型

    # 根据第七条, Business可以 change square footage.
    def change_footage(self, name, footage):
        for room in self.rooms:
            if room.name == name:
                room.square_footage = footage
                break

# Exceptions
class BuildingCodeViolationError(Exception):
    pass

class InvalidBusinessError(Exception):
    pass

class BuildingCreationException(Exception):
    pass


if __name__ == '__main__':  # 当这个file被 run directly的时候

    try: # everything in try block to catch Exceptions
         # create the building with prompts
        b_type = input('\nplease give the type of building:')
        name = input('\nplease give the name of building:')
        num_of_rooms = int(input('\nplease give the number of rooms of building:'))
        if b_type == "business":
            building = Business(name, num_of_rooms)
        elif b_type == "house":
            building = House(name, num_of_rooms)
        elif b_type == "building":
            building = Building(name, num_of_rooms)
        else:
            raise BuildingCreationException()

        # Repeatly add room to it.
        i = 0
        while i < num_of_rooms:
            building.add_room(input('\nplease give the name of room:'),
                              float(input('\nplease give the square footage of room:')))
            i += 1
    except BuildingCreationException:
        print("oops...")
    except Exception:
        print("you can't do that")
```