1. (2 pt) Write a function `counter` that takes a string as an argument and outputs to the screen a count of the number of characters in each word in the string. A “word” is a sequence of characters not including a space. The function should output: 4 2 1 7 when called as follows: `counter("This is a string.");`

2. (3 pts) Given a linked list class defined as follows:

   ```cpp
class NumberList
   {
   private:
       struct Node
       {
           double value;
           struct Node *next;
       };
       ListNode *head; //points to the first node
   public:
       NumberList();
   }
   ```

   Write a member function `remove(int i)` to remove the element at position `i`. If `i` is equal to or greater than the length of the list (or less than 0) then the list should remain unchanged.

3. (1 pts) Write a recursive function `containsOnly` that takes three parameters: an array, its size, and a target value. The array contains integers, and the target value is an integer. The function should return true if every value in the list is equal to the target value, otherwise it should return false. Note: your function should return true if the size is 0.
4. **(4 pts)** Given the following *Block* class that represents a square image in a video game, and *BlockList* that stores a collection of *Blocks*:

```cpp
class Block {
private:
    string color;
    float x;
    float y;
public:
    Block() {color = "Black"; x = 0; y = 0;}
    void setColor(string newColor) {color = newColor;}
    void setX(float newX) {x = newX;}
    void setY(float newY) {y = newY;}
    string getColor() {return color;}
    float getX() {return x;}
    float getY() {return y;}
bool operator==(Block);    // define this
};

class BlockList {
private:
    Block list[100];  // partially filled array of blocks
    int count;        // the number of blocks currently in list
public:
    BlockList() {count = 0;}
    bool addBlock(Block b);  // add a Block to the list
    int findBlock(Block b);  // return index of b in the list
    BlockList operator+(BlockList);  // concatenate 2 lists
};
```

Implement the four undefined methods from the *Block* and *BlockList* classes:

- **Block::operator==(Block)**: two Blocks are equal if their three attributes are equal.
- **BlockList::addBlock(Block b)**: put b in next open slot in the array, return true. Return false if it failed (because the list is full).
- **BlockList::findBlock(Block b)**: use == that you defined in Block. Return -1 if b is not in the list.
- **BlockList::operator+(BlockList)**: return a new BlockList.