Section Handout #3

For the following linked-list questions, assume that the following definition exists:

```
typedef struct cellT {
    int value;
    struct cellT *link;
} cellT;
```

Problem 1: Mini-trace

Trace through the following code and draw the state of memory at the end:

```
cellT *p;

p = New(cellT *);
p->value = 1;
p->link = New(cellT *);
p = p->link;
p->value = 2;
p->link = New(cellT *);
p->link->link = p;
p->link->link->link->value = 3;
```

Problem 2: Recursive ReversePrint

Write a recursive function that prints a linked list in reverse order.

Problem 3: Append

Write a function that given two lists will append the second list onto the first. For example, given the first list `(1 4 6)` and the second list `(3 19 2)`, the function would destructively modify the first list to contain `(1 4 6 3 19 2)`. It is easiest to write this function recursively.

Problem 4: Stutter

Write a function `Stutter` that given a linked list will duplicate every cell in the list. If the incoming list contains the elements `(11 5 21 7 7)`, the function will destructively modify the list to contain `(11 11 5 5 21 21 7 7 7 7)`.

Problem 5: RemoveDuplicates (aka Unstutter)

Write a function `RemoveDuplicates` that given a linked list will remove and free the second of all **neighboring** duplicates found in the list. If the incoming list is `(5 5 22 37 89 89 15 15 22)` the function will destructively modify the list to contain `(5 22 37 89 15 22)`. Don't worry about handling duplicate sequences longer than 2 or duplicates that aren't right next to each other in the list.