For recursion:

1) Write your function with the loop form you are comfortable with (aka while/for).
2) Test the function to make sure it works
3) Take the inner-most loop in the function and convert that to its own recursive function.
4) Test it and make sure that works.
5) Replace the innermost loop with the recursive function
6) Test it and make sure it works
7) Take the next inner-most loop (which now appears to be the inner-most loop) and convert that to its own recursive function
8) Test it and make sure it works
9) Repeat recursively until you hit the stopping condition of no more loops (Did I just hear a collective groan?).

So, for example, if you have the following function:

```
int loopfunc1(int len1, int len2) {
	int tot = 0;
	for (int i = 0; i < len1; i++) {
		for (int j = i; j < len2; j++) {
			cout << "*\t";  //\t prints a tab so everything is evenly spaced
			tot+=1;
		}
		cout << endl;
	}
	return tot;
}
```

Take the inner loop:

```
		for (int j = i; j < len2; j++) {
			cout << "*\t";  //\t prints a tab so everything is evenly spaced
			tot +=1;
		}
```

What variables are used in this loop?

j, i, len2, and tot – so those have to be passed into the function as parameters.

i.e.,

```
innerloop(int j, int i, int len2, int tot) {
```

What is the stopping condition? (When will this loop stop?)

When j >= len2, so that will be the stopping condition:

```
innerloop(int j, int i, int len2, int tot) {
if (j >= len2) {
	return
}
```

What is calculated and should be returned from this loop? (if anything)?

In this case, we're increasing tot every time a star is printed, and that tot is used by the outer loop, so that should be returned when the stopping condition is hit (which means it should also be the function's return type!)

i.e.,

```
int innerloop(int j, int i, int len2, int tot) {
	if (j >= len2) {
		return tot
	}
```

Now, what changes in the loop each time so that the stopping condition is eventually reached?

j increases each time.

So in the recursive function call, make j increase:

```
int innerloop(int j, int i, int len2, int tot) {
	if (j >= len2) {
		return tot;
	}
	else {
		return (innerloop(j+1, i, len2, tot));
	}
}
```

And finally, plop the code in before the recursive call:

```
int innerloop(int j, int i, int len2, int tot) {
	if (j >= len2) {
		return tot;
	}
	else {
		cout << "*\t";  //\t prints a tab so everything is evenly spaced
		tot +=1;
		return (innerloop(j+1, i, len2, tot));
	}
}
```

Now replace the inner loop of the original function with a call to this function, initializing all values to be what they were going into this inner loop:

```
int loopfunc2(int len1, int len2) {
	int tot = 0;
	for (int i = 0; i < len1; i++) {
		tot = innerloop(i,i,len2,tot);
		cout << endl;
	}
	return tot;
}
```

We're almost there: Now repeat the process with the next loop:

For this, the variables used by and in this loop are i, len1, len2, and tot.

```
secondloop(int i, int len1, int len2, int tot) {
```

The stopping condition is when i is greater than or equal to len1:

```
secondloop(int i, int len1, int len2, int tot) {
	if (i >= len1){
		return
	}
```

The thing to be returned is the tot, which is an int, which is also the return function of the function:

```
int secondloop(int i, int len1, int len2, int tot) {
	if (i >= len1) {
		return tot;
	}
```

Each time the loop repeats, i increases, so that is what must happen in the recursive call:

```
int secondloop(int i, int len1, int len2, int tot) {
	if (i >= len1) {
		return tot;
	}
	else {
```

```
                return(secondloop(i+1,len1,len2,tot));
        }
}
```

And, finally, what happens in the loop? Plop that in the else condition before the recursive call:

```
int secondloop(int i, int len1, int len2, int tot) {
        if (i >= len1) {
                return tot;
        }
        else {
                tot = innerloop(i,i,len2,tot);
                cout << endl;
                return(secondloop(i+1,len1,len2,tot));
        }
}
```

Now just plop a function call to this second loop in the original function:

```
int loopfunc3(int len1, int len2) {
        int tot = 0;
        tot = secondloop(0,len1,len2,tot);
        return tot;
}
```

And we're good to go!

If you want to get fancy, you can just call secondloop with the initialized values and skip the loopfunc3, but it works either way and doesn't make a lot of difference in terms of efficiency.