Count Driven Loops

1. Consider the following program:

```
1.  #include <stdio.h>
2.
3.  int main( void ) {
4.       int limit = 4;
5.       int counter;
6.
7.       printf( "start: " );
8.       counter = 0;
9.       while (counter < limit) {
10.           printf( "%d ", counter);
11.           counter++;
12.      }
13.      printf( "end!\n" );
14.
15.      return 0;
16. }
```

The following is trace table demonstrating the output of this program.

Line #s	limit	counter < limit	output	counter
4,5	4			-
7			start:	
8				0
9,10,11		true (1)	0	1
9,10,11		true (1)	1	2
9,10,11		true (1)	2	3
9,10,11		true (1)	3	4
9		false(0)		
13			end!	

Output: start: 0 1 2 3 4 end!

How will this program change if:

a) Line 8 is removed?

counter never gets initialized and is some garbage value, the output is undeterminable

b) Line 11 is removed?

counter never gets updated in the loop and will cause an infinite loop

2. Consider the following program:

```
1.   #include <stdio.h>
2.
3.   int main( void ) {
4.        int limit = 4;
5.        int counter;
6.
7.        printf( "start: " );
8.        for (counter = 0; counter < limit; counter++) {
9.             printf( "%d ", counter);
10.       }
12.
13.       printf( "end!\n" );
14.
15.       return 0;
16.  }
```

a) Complete the trace table provided to compare this program to that in Question 1.

limit	counter < limit	output	counter
4		start:	-
			0
	T	0	1
	T	1	2
	T	2	3
	T	3	4
	F	end!	

Output: **start: 0 1 2 3 end!**

b) Below is a copy of the program above. Edit the program so that it counts down from limit to 0 **inclusive**. Cross out the code you do not want and add code you need.

```
1.   #include <stdio.h>
2.
3.   int main( void ) {
4.        int limit = 4;
5.        int counter;
6.
7.        printf( "start: " );
8.        for (counter = 0limit; counter < limit >=0; counter++ counter--) {
9.             printf( "%d ", counter);
10.       }
12.
13.       printf( "end!\n" );
14.
15.       return 0;
16.  }
```

3. Trace the following program to determine its behavior. Update the documentation, function name and variable names to reflect its behavior.

```
#include <stdio.h>

int sum_one_to_limit(int limit);
int main( void ) {
    int result;
    result = sum_one_to_limit(4);
    printf("result is: %d\n ", result);

    return 0;
}
/*
 * Purpose: sums the numbers from 1 to limit-1 inclusive
 * Parameters: int limit, a number > 0
 * Returns: int – the sum
 */
int sum_one_to_limit(int limit) {
    int sum = 0;
    int current_num;

    for (current_num = 1; current_num < limit; current_num++) {
        sum += current_num;
    }
    return sum;
}
```

4. Design a function that takes a positive integer n and prints the square values of all the integers from 0 to n-1.
   ie: square value of 4 is 4*4
   Your function should assume the argument passed is not negative.
   Include the prototype, test call to this function from main, documentation and the function definition.

5. Design a function that takes a positive integer n and computes and returns the sum of all the square values of the integers from 0 to n-1.
   Your function should assume the argument passed is not negative.
   Include the prototype, test call to this function from main, documentation and the function definition.

```
#include <stdio.h>

void print_squares (int limit);
int sum_squares(int limit);

int main( void ) {
    int result;

    print_squares(0); // should print nothing
    print_squares(4); // should print 0,1,4,9


    result = sum_squares(0);
    printf("should be 0, sum is: %d\n", result);

    result = sum_squares(4);
    printf("should be 14, sum is: %d\n", result);

    return 0;
}

/*
 * Purpose: prints the squares from 0 to limit-1 inclusive
 * Parameters: int limit, a number >=0
 */
void print_squares (int limit) {
    int current_num, sqr;

    if (limit > 0) {
        printf("0");
    }
    for (current_num = 1; current_num < limit; current_num++) {
        sqr = current_num * current_num;
        printf(",%d", sqr);
    }
    printf("\n");
}

/*
 * Purpose: sums the squares from 0 to limit-1 inclusive
 * Parameters: int limit – a number >=0
 * Returns: int – the sum
 */
int sum_squares (int limit) {
    int current_num, sqr;
    int sum = 0;

    for (current_num = 0; current_num < limit; current_num++) {
        sqr = current_num * current_num;
        sum += sqr;
    }

    return sum;
}
```

6. Write a function that takes an integer n and prints n copies of the character ***** on a line ending with a newline. Your function should assume the argument passed is not negative.
Include the prototype, a test call to this function from main, documentation and the function definition.

```
#include <stdio.h>

void print_n_stars(int n);

int main( void ) {
    print_n_stars(5);
    return 0;
}

/*
 * Purpose: prints n copies of * on one line
 * Parameters: int n – a number >=0
 */
void print_n_stars(int n) {
    int count = 0;

    for (count = 0; count < n; count++) {
        printf("*");
    }
    printf("\n");
}
```

7. You are now asked to write a function that takes a positive integer n and prints n copies of the character `$`. You realize this is VERY similar to the last function you designed in Question 6. Update the function you designed in Question 6 so that it is more flexible and you eliminate the redundancy that would occur if you had to write a function for every character type that could be printed.
Don't forget to update the prototype, test calls in main and the documentation!
Recall: a `char` is a type in C that holds a single character enclosed in single quotes (ie. `'*'` or `'$'` or `'X'`)
The format specifier for printing a `char` is `%c`

```
#include <stdio.h>

void print_n_chars(int n, char ch);

int main( void ) {
    print_n_chars(5, '$'); // should print $$$$$
    print_n_chars(3, '*'); // should print ***

    return 0;
}

/*
 * Purpose: prints n copies of ch on one line
 * Parameters: int n, a number >=0
 *       char ch
 */
void print_n_chars(int n, char ch) {
    int count = 0;

    for (count = 0; count < n; count++) {
        printf("%c", ch);
    }
    printf("\n");
}
```