Review Questions:

1. What are the two different kinds of division that the / operator can do? Under what circumstances does it perform each?
2. What are the definitions of the “Boolean" values *true* and *false* in C?
3. Name three uses for the semicolon in C.
4. What would the equivalent code, using a while loop, be for the example

```
for(i = 0; i < 10; i = i + 1)
        printf("i is %d\n", i);
```

5. What is the numeric value of the expression 3 < 4 ?
6. Under what conditions will this code print ``water"?

```
if(T < 32)
        printf("ice\n");
else if(T < 212)
        printf("water\n");
else    printf("steam\n");
```

7. What would this code print?

```
int x = 3;
if(x)
        printf("yes\n");
else    printf("no\n");
```

8. What would this code print?

```
int i;

for(i = 0; i < 3; i = i + 1)
        printf("a\n");
        printf("b\n");
printf("c\n");
```

Tutorial Section

1. Type in and run this program, and compare its output to that of the original “Hello, world!" program (Tutorial 1).

   ```
   #include <stdio.h>

   int main()
   {
           printf("Hello, ");
           printf("world!\n");
           return 0;
   }
   ```

   You should notice that the output is identical to that of the original “Hello, world!" program. This shows that you can build up output using multiple calls to printf, if you like. You mark the end of a line of output (that is, you arrange that further output will begin on a new line) by printing the “newline" character, \n.

2. Type in and run this program:

   ```
   #include <stdio.h>
   int main()
   {
           int i;
           printf("statement 1\n");
           printf("statement 2\n");
           for(i = 0; i < 10; i = i + 1)
                   {
                   printf("statement 3\n");
                   printf("statement 4\n");
                   }
           printf("statement 5\n");

           return 0;
   }
   ```

   This program doesn't do anything useful; it's just supposed to show you how control flow works--how statements are executed one after the other, except when a construction such as the for loop alters the flow by arranging that certain statements get executed over and over. In this program, each simple statement is just a call to the printf function.

   Now delete the braces {} around statements 3 and 4, and re-run the program. How does the output change? (See also question 8 above.)

3. Type in and run this program:

```
#include <stdio.h>

int main()
{
        int i, j;

        printf("start of program\n");

        for(i = 0; i < 3; i = i + 1)
                {
                printf("i is %d\n", i);
                for(j = 0; j < 5; j = j + 1)
                        printf("i is %d, j is %d\n", i, j);
                printf("end of i = %d loop\n", i);
                }

        printf("end of program\n");

        return 0;
}
```

   This program doesn't do much useful, either; it's just supposed to show you how loops work, and how loops can be nested. The outer loop runs i through the values 0, 1, and 2, and for each of these three values of i(that is, during each of the three trips through the outer loop) the inner loop runs, stepping the variable j through 5 values, 0 to 4. Experiment with changing the limits (initially 3 and 5) on the two loops. Experiment with interchanging the two loops, that is, by having the outer loop manipulate the variable j and the inner loop manipulate the variable i. Finally, compare this program to the triangle-printing program of Assignment 1 (exercise 4) and its answer as handed out this week.

4. Type in and run this program:

```
#include <stdio.h>

int main()
{
        int day, i;

        for(day = 1; day <= 3; day = day + 1)
                {
                printf("On the %d day of Christmas, ", day);
                printf("my true love gave to me\n");
```

```
				for(i = day; i > 0; i = i - 1)
					{
					if(i == 1)
						{
						if(day == 1) printf("A ");
						else	printf("And a ");
						printf("partridge in a pear tree.\n");
						}
					else if(i == 2)
						printf("Two turtledoves,\n");
					else if(i == 3)
						printf("Three French hens,\n");
					}
				printf("\n");
				}

			return 0;
		}
```

The result (as you might guess) should be an approximation of the first three verses of a popular (if occasional tiresome) song.

a. Add a few more verses (calling birds, golden rings, geese a-laying, etc.).
b. Here is a scrap of code to print words for the days instead of digits:

```
	if(day == 1)
		printf("first");
	else if(day == 2)
		printf("second");
	else if(day == 3)
		printf("third");
```

Incorporate this code into the program.

c. Here is another way of writing the inner part of the program:

```
        printf("On the %d day of Christmas, ", day);
        printf("my true love gave to me\n");
        if(day >= 3)
                printf("Three French hens,\n");
        if(day >= 2)
                printf("Two turtledoves,\n");
        if(day >= 1)
                {
                if(day == 1) printf("A ");
                else    printf("And a ");
                printf("partridge in a pear tree.\n");
                }
```

Study this alternate method and figure out how it works. Notice that there are no else's between the if's.

Exercises:

1. Write a program to find out how many of the numbers from 1 to 10 are greater than 3. (The answer, of course, should be 7.) Your program should have a loop which steps a variable (probably named i) over the 10 numbers. Inside the loop, if i is greater than 3, add one to a second variable which keeps track of the count. At the end of the program, after the loop has finished, print out the count.

2. Write a program to compute the average of the ten numbers 1, 4, 9, ..., 81, 100, that is, the average of the squares of the numbers from 1 to 10. (instead of printing each square as it is computed, add it in a variable sum which keeps track of the sum of all the squares, and then at the end, divide the sum variable by the number of numbers summed.)

   If you keep track of the sum in a variable of type int, and divide by the integer 10, you'll get an integer-only approximation of the average (the answer should be 38). If you keep track of the sum in a variable of type float or double, on the other hand, you'll get the answer as a floating-point number, which you should print out using %f in the printf format string, *not* %d. (In a printf format string, %d prints only integers, and%f is one way to print floating-point numbers. In this case, the answer should be 38.5.)

3. Write a program to print the numbers between 1 and 10, along with an indication of whether each is even or odd, like this:

   ```
   1 is odd
   2 is even
   3 is odd
   ...
   ```

   (Hint: use the % operator.)

4. Write a program to print the first 7 positive integers and their factorials. (The factorial of 1 is 1, the factorial of 2 is 1 * 2 = 2, the factorial of 3 is 1 * 2 * 3 = 6, the factorial of 4 is 1 * 2 * 3 * 4 = 24, etc.)

5. Write a program to print the first 10 Fibonacci numbers. Each Fibonacci number is the sum of the two preceding ones. The sequence starts out 0, 1, 1, 2, 3, 5, 8, …

6. Make some improvements to the prime number printing program. Other than 2, no even numbers are prime, so have the program test only the *odd* numbers between 3 and 100. Rather than recomputing sqrt(i) each time through the inner loop, compute it just once for each new value of i (that is, compute it at the beginning of the body of the outer loop, just before the beginning of the inner loop) and assign its value to another variable.

Assignment #2 ANSWERS

Question 1. ***What are the two different kinds of division that the / operator can do? Under what circumstances does it perform each?***
The two kinds of division are integer division, which discards any remainder, and floating-point division, which yields a floating-point result (with a fractional part, if necessary). Floating-point division is performed if either operand (or both) is floating-point (that is, if either number being operated on is floating-point); integer division is performed if both operands are integers.

Question 2. ***What are the definitions of the ``Boolean'' values true and false in C?***
False is always represented by a zero value. Any nonzero value is considered true. The built-in operators <, <=, >, >=, ==, !=, &&, ||, and ! always generate 0 for false and 1 for true.

Question 3. ***Name three uses for the semicolon in C.***
Terminating declarations, terminating statements, and separating the three control expressions in a for loop.

Question 4. ***What would the equivalent code, using a while loop, be for the example***

```
for(i = 0; i < 10; i = i + 1)
        printf("i is %d\n", i);
```

Just move the first (initialization) expression to before the loop, and the third (increment) expression to the body of the loop:

```
i = 0;
while(i < 10)
        {
        printf("i is %d\n", i);
        i = i + 1;
        }
```

Question 5. ***What is the numeric value of the expression 3 < 4 ?***
1 (or “true''), because 3 is in fact less than 4.

Question 6. ***Under what conditions will this code print ``water''?***

```
if(T < 32)
        printf("ice\n");
else if(T < 212)
        printf("water\n");
else    printf("steam\n");
```

If T is greater than or equal to 32 *and* less than 212. (If you said ``greater than 32 and less than 212'', you weren't quite right, and this kind of distinction--paying attention to the difference between ``greater than'' and ``greater than or equal''--is often extremely important)
Question 7. ***What would this code print?***

```
int x = 3;
if(x)
        printf("yes\n");
else    printf("no\n");
```

It would print ``yes'', since x is nonzero.

Question 8. ***What would this code print?***

```
int i;

for(i = 0; i < 3; i = i + 1)
        printf("a\n");
        printf("b\n");
printf("c\n");
```

It would print

```
a
a
a
b
c
```

The indentation of the statement printf("b\n"); is (deliberately, for the sake of the question) misleading. It *looks* like it's part of the body of the for statement, but the body of the for statement is always a single statement, or a list of statements enclosed in braces {}. In this case, the body of the loop is the call printf("a\n");. The two statements printf("b\n"); and printf("c\n"); are normal statements following the loop.
The code would be *much* clearer if the printf("b\n"); line were indented to line up with the printf("c\n"); line. If the intent was that ``b'' be printed 3 times, along with ``a'', it would be necessary to put braces {}around the pair of lines printf("a\n"); and printf("b\n"); .

Exercise 1. ***Write a program to find out how many of the numbers from 1 to 10 are greater than 3.***

```
#include <stdio.h>

int main()
{
int i;
int count = 0;

for(i = 1; i <= 10; i = i + 1)
        {
        if(i > 3)
                count = count + 1;
        }

printf("%d numbers were greater than 3\n", count);
return 0;
}
```

Tutorial 4b. ***Print ``On the first day'' instead of ``On the 1 day'' in the days-of-Christmas program.***

Simply replace the lines

```
printf("On the %d day of Christmas, ", day);
printf("my true love gave to me\n");
```

with

```
printf("On the ");
if(day == 1) printf("first");
else if(day == 2) printf("second");
else if(day == 3) printf("third");
else if(day == 4) printf("fourth");
else if(day == 5) printf("fifth");
else if(day == 6) printf("sixth");
else    printf("%d", day);
printf(" day of Christmas, ");
printf("my true love gave to me\n");
```

The final

```
else printf("%d", day);
```

is an example of ``defensive programming.'' If the variable day ever ends up having a value outside the range 1-6 (perhaps because we later added more verses to the song, but forgot to update the list of words), the program will at least print something.

Exercise 2. ***Write a program to compute the average of the ten numbers 1, 4, 9, ..., 81, 100.***

```
#include <stdio.h>

int main()
{
int i;
float sum = 0;
int n = 0;

for(i = 1; i <= 10; i = i + 1)
        {
        sum = sum + i * i;
        n = n + 1;
        }

printf("the average is %f\n", sum / n);

return 0;
}
```

Exercise 3. ***Write a program to print the numbers between 1 and 10, along with an indication of whether each is even or odd.***

```
#include <stdio.h>

int main()
{
        int i;

        for(i = 1; i <= 10; i = i + 1)
                {
                if(i % 2 == 0)
                        printf("%d is even\n", i);
                else    printf("%d is odd\n", i);
                }

        return 0;
}
```

Exercise 4. ***Print out the 8 points of the compass, based on the x and y components of the direction of travel.***

```
	if(x > 0)
		{
		if(y > 0)
			printf("Northeast.\n");
		else if(y < 0)
			printf("Southeast.\n");
		else	printf("East.\n");
		}
	else if(x < 0)
		{
		if(y > 0)
			printf("Northwest.\n");
		else if(y < 0)
			printf("Southwest.\n");
		else	printf("West.\n");
		}
	else	{
		if(y > 0)
			printf("North.\n");
		else if(y < 0)
			printf("South.\n");
		else	printf("Nowhere.\n");
		}
```

(You will notice that this code also prints ``Nowhere'' in the ninth case, namely when x == 0 and y == 0.)

Exercise 5. ***Write a program to print the first 7 positive integers and their factorials.***

```
#include <stdio.h>

int main()
{
        int i;
        int factorial = 1;

        for(i = 1; i <= 7; i = i + 1)
                {
                factorial = factorial * i;
                printf("%d   %d\n", i, factorial);
                }

        return 0;
}
```

The answer to the extra credit question is that 8!, which is how mathematicians write ``eight factorial,'' is 40320, which is *not* guaranteed to fit into a variable of type int. To write a portable program to compute factorials higher than 7, we would have to use long int.

Exercise 6. ***Write a program to print the first 10 Fibonacci numbers.***

```
#include <stdio.h>
int main()
{
        int i;
        int fibonacci = 1;
        int prevfib = 0;
        int tmp;

        for(i = 1; i <= 10; i = i + 1)
                {
                printf("%d   %d\n", i, fibonacci);
                tmp = fibonacci;
                fibonacci = fibonacci + prevfib;
                prevfib = tmp;
                }
        return 0;
}
```

(There are many ways of writing this program. As long as the code you wrote printed 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, it's probably correct. If the code you wrote is clearer than my solution above, so much the better!)

Exercise 7. ***Make some improvements to the prime number printing program.***

Here is a version incorporating both of the suggested improvements:

```
#include <stdio.h>
#include <math.h>

int main()
{
int i, j;
double sqrti;

printf("%d\n", 2);

for(i = 3; i <= 100; i = i + 2)
        {
        sqrti = sqrt(i);
        for(j = 2; j < i; j = j + 1)
                {
                if(i % j == 0)
                        break;                  /* not prime */
                if(j > sqrti)
                        {                       /* prime */
                        printf("%d\n", i);
                        break;
                        }
                }
        }

return 0;
}
```