C-4 Loops

The *while* Loop

```
#include <stdio.h>
#include <stdlib.h>
int main (void)
{
   int x, sum = 0, count = 0;
   float average;
   printf("Enter a number (zero to end):  ");
   scanf("%d", &x);

   while (x != 0)                    /* blue shows the loop */
   {
      sum += x;
      count += 1;
      printf("\nEnter a number (zero to end):  ");
      scanf("%d", &x);
   }
   average = (float) sum / (float) count;
   printf("\nThe average of these %d numbers is %f.\n\n",
         count, average);
   return EXIT_SUCCESS;
}
```

General Form of the while loop:

```
while (condition)
{
   statements;
}
```

If the loop has only one statement, the braces can be omitted.

Most loops have more than one statement.

On to the *do while* loop

```
int main (void)
{
    int x, sum = 0, count = 0;
    float average;

    do
    {
        printf("Enter a number (zero to end:  ");
        scanf("%d", &x);
        sum += x;
        count += 1;
    } while (x != 0);

    average = (float) sum / ((float) count -1);
    printf ("The average of %d numbers is %f.",
                count-1, average);
    return EXIT_SUCCESS;
}
```

General Form of the do-while loop:

```
do
{
    statements;
} while (condition);
```

One of the few structures we use that ends with a semicolon.

In the do-while loop, the test happens at the end. So it is guaranteed to do the “statements” section at least once.

On to the *for* loop…

The FOR loop

Degrees to Radians

0	0.000000
10	0.174533
20	0.349066
30	0.523599
40	0.698132
50	0.872665
60	1.047198
70	1.221731
80	1.396264
90	1.570796
100	1.745329
110	1.919862
120	2.094395
130	2.268928
140	2.443461
150	2.617994
160	2.792527
170	2.967060
180	3.141593
190	3.316126
200	3.490659
210	3.665192
220	3.839725
230	4.014258
240	4.188791
250	4.363324
260	4.537857
270	4.712389
280	4.886922
290	5.061455
300	5.235988
310	5.410521
320	5.585054
330	5.759587
340	5.934120
350	6.108653
360	6.283186

NOTE: This has been cut and pasted to fit on one slide.

```
/* Print a degree-to-radians table using a FOR loop structure */
#include <stdio.h>
#include <stdlib.h>
#define PI 3.1415
int main (void)
{
   int degrees;
   double radians;
   printf("\nDegrees to Radians \n");

   for (degrees = 0; degrees <= 360; degrees += 10)
   {
      radians = degrees * PI / 180;
      printf("%6i %9.6f \n", degrees, radians);
   }
   return EXIT_SUCCESS;
}
```

General Form of the for loop:

```
for (exp 1;  exp 2; exp 3)
{
   statements;
}
```

where:

exp 1 is used to initialize the loop-control variable

exp 2 specifies the condition that should be TRUE to continue the loop repetition

exp 3 specifies the modification to the loop-control variable

Picky details on the *for* loop

The minimum for a FOR loop is:

for (; ;) It must have the two semicolons and the ( )

if missing:

exp1 – no initialization performed

exp2 – then test is ALWAYS true

exp3 – no for-loop automatic incrementing or decrementing

```
while
(condition)
{
    statements;
}
```

```
do
{
    statements;
} while (condition);
```

```
for (exp 1;  exp 2; exp 3)
{
    statements;
}
```

break – used to exit any loop or structure immediately

continue – used to skip remaining statements in current pass of the loop or structure

For simple short for loops, one can just write down the valid loop counters, and count them up.

```
for (x = 0; x <=18; x+=2)
```

The valid loop counters would be:

0 2 4 6 8 10 12 14 16 18

1 2 3 4 5 6 7 8 9 10 MyCountingLine.

The list consists of 10 numbers, so the loop would execute 10 times.

Computing the number of times a for loop will execute:

(PS: *floor* is a function that rounds down.)

$$\textbf{floor}\left(\frac{\textbf{final} - \textbf{initial}}{\textbf{increment}}\right) + \textbf{1}$$

Example:

for (k = 5; k <= 83; k +=4)

$$\boxed{\text{floor}\left(\frac{83 - 5}{4}\right) + 1} = \boxed{\text{floor}\left(\frac{78}{4}\right) + 1} = \boxed{19 + 1 = 20}$$

Loops & printf & columns.
Using integers.

Lining up numbers under column headers:

CODE:

```
int a = 125, b = 789;

printf("First Number    Second Number \n");
printf("----------------     ------------------- \n");
printf ("%4i%4i\n\n", a, b);
```

OUTPUT:

```
First Number    Second Number
----------------     --------------------
 125 789
```

PROBLEM:

The numbers do not line up correctly under the column headers.

There are several solutions. Here is a first try:

CODE:

```
int a = 125, b = 789;

printf("First Number    Second Number \n");
printf("----------------     ------------------- \n");
printf ("%8i    %8i\n\n", a, b);
```

OUTPUT:

```
First Number    Second Number
----------------     -------------------
     125            789
```

PROBLEM:

The 125 is almost centered (if that is desired).
The 789 is still in the wrong place.

There are several solutions. Here is another try:

CODE:

```
int a = 125, b = 789;

printf("First Number    Second Number \n");
printf("----------------    -------------------- \n");
printf ("%8i          %8i\n\n", a, b);
```

OUTPUT:

```
First Number    Second Number
----------------    --------------------
     125                789
```

PROBLEM:

The 125 is almost centered (if that is desired).
Added 5 more spaces between.
Now the 789 is about right place.

Loops & printf & columns. Using variables with decimal points.

Lining up numbers under column headers:

CODE:

```
double a = 125.6, b = 7.89, c = 45.678, d=567.1234;

printf("1st Column   2nd Column \n");
printf("----------   ---------- \n");
printf("%f   %f \n", a, b);
printf("%f   %f \n", c, d);
```

OUTPUT 2:

```
1st Column    2nd Column
-----------   -----------
125.6000      7.8900
 45.6780    567.1234
```

PROBLEM:

Getting closer to a correct solution, but the spacing is still off.

CODE:

double a = 125.6, b = 7.89, c = 45.678, d=567.1234;

The most digits before the decimal point = 3
The decimal point will take one space. = 1
The most digits after the decimal point = 4
Solution = **%8.4f**

```
printf("%8.4f   %8.4f \n", a, b);
printf("%8.4f   %8.4f \n", c, d);
```

Output 3:

```
1st Column    2nd Column
-----------   -----------
125.6000      7.8900
 45.6780    567.1234
```

The decimal points line up but more space is required to get the numbers to line up with the headers.

CODE:

```
double a = 125.6, b = 7.89, c = 45.678, d=567.1234;
```

Both columns ought to shift to the right by 2 spaces.
Solution = **%10.4f**

```
printf("%10.4f   %10.4f \n", a, b);
printf("%10.4f   %10.4f \n", c, d);
```

Output 4:

```
1st Column    2nd Column
----------    ----------
  125.6000        7.8900
   45.6780      567.1234
```

The decimal points line up.
The numbers line up with the headers.
SUCCESS!

CODE:

double a = 125.6, b = 7.89, c = 45.678, d=567.1234;

I allowed 3 spaces between the two header lines.
I could have left those three spaces **out**,
and added the 3 to the second set of conversion specifiers.

```
printf("%10.4f%13.4f \n", a, b);
printf("%10.4f%13.4f \n", c, d);
```

Output 5:

```
1st Column   2nd Column
----------   -----------
  125.6000        7.8900
   45.6780      567.1234
```

C-4 Loops

THE END