# Practice: 1-6 (Solve using all three loops)

# Assignment: 7-11 (Solve using a for and a while loop). Demonstration is due at the beginning of your next-next class. You may code both loops in the same project.

### Simple Loop

1. Write a program to print the numbers 2 to 10 in increments of two on the same line. The output of your program should be 2 4 6 8 10
2. Write a Program to compute the sum of eight numbers. You will prompt the user for each number. Your prompt should also indicate which number is currently being accepted e.g. your prompt can be **"Please enter value #5: "**
3. Write a Program to compute the average of ten numbers. You will prompt the user for each number.
4. Write a C# program that displays a conversion table from gallons to liters. The program should display gallons from 10 to 20 in one-gallon increments and the corresponding liter equivalents. Use the relationship that 1 gallon contains 3.785 liters. Your table must look professional!
5. Write a C# program that converts feet to meters. The program should display feet from 3 to 30 in three-foot increments and the corresponding meter equivalents. Use the relationship that 1 meter is equivalent to 3.28 feet. Your table must look professional!
6. Write a program to produce a table of the numbers 0 through 20 in increments of 2, with their squares and cubes. Your table must look professional!
7. Write a program to produce a table of numbers from 10 to 1, with their squares and cubes. Again, your table must look professional!
8. Write a program that displays a conversion table from Fahrenheit to Celsius. The program must request the starting Fahrenheit value, the ending Fahrenheit value, and the increment. Thus, instead of the condition checking for a fixed count, the condition checks for the ending Fahrenheit value. Your table must look professional!
9. Write and run a C# program that calculates and displays the amount of money available in a bank account that initially has $1,000 deposited in it and that earns 8 percent interest a year. Your program should display the amount available at the end of each year for a period of ten years. Use the relationship that the money available at the end of each year equals the amount of money in the account at the start of the year plus .08 times the amount available at the start of the year.  
     
     
   Your output should be formatted as shown below

Year Balance  
 1 $1,080.00

2 $1,166.40

3 $1,259.71

4 $1,360.49

5 $1,469.33

6 $1,586.87

7 $1,713.82

8 $1,850.93

9 $1,999.00  
10 $2,158.92

1. Write a C program that continuously requests a grade.   
   If the grade is less than 0 or greater than 100, your program should print an appropriate message informing the user that an invalid grade has been entered,   
   else the grade should be added to a total.

Try to use the **break** and **continue** keywords.

If the grade is 999, the program should exit the loop and display the sum, number of valid grades and the average of the valid grades entered. (You must NOT display an invalid message)  
P.S. The value 999 is use to terminate the cycle and should not be used in any of the calculation. You must not display an “Invalid grade”

1. Print the decimal, octal, and hexadecimal values of all characters between the start and stop characters entered by a user. For example, if the user enters an a and a z, the program should print all the characters between a and z and their respective numerical values. Make sure that the second character entered by the user occurs later in the alphabet than the first character. If it does not, write a loop that repeatedly asks the user for a valid second character until one is entered.  
   Your output should be formatted as shown below

You need separate loops to read the start and stop character and to display the table.

Letter Decimal Octal Hex  
a 97 141 61  
b 98 142 62  
c 99 143 63  
d 100 144 64  
e 101 145 65  
to obtain an int from a char use an explicit cast  
to obtain an octal from an int use **Convert.ToString(«number\_value», 8)**  
to obtain a hexadecimal from an int use **Convert.ToString(«number\_value», 16)**

1. Write a program to compute the (x, y) pairs for the equation **y = 2x2 –x - 6** for x in the range 1 to 5 in 0.5 increments.  
    x 2x2 -x -6 y  
   1.0 2.0 -1.0 -6 -5.0  
   1.5 4.5 -1.5 -6 -3.0  
   2.0 8.0 -2.0 -6 0.0  
   2.5 12.5 -2.5 -6 4.0
2. Write a program to reverse the digits of a positive integer number. For example, if the number 8735 is entered, the number displayed should be 5378. (Hint: Use a do statement and continuously strip off and display the units digit of the number. If the variable num initially contains the number entered, the units digit is obtained as (num % 10). After a units digit is displayed, dividing the number by 10 sets up the number for the next iteration. Thus, (8735 % 10) is5and (8735 / 10) is 873. The do statement should continue as long as the remaining number is not zero).
3. Write a program to compute the (x, y) pairs for the equation **y = x4 –5x2 - 4** for x in the range -2.5 to 2.5 in 0.5 increments.  
    x x4 -5x2 4.0 y  
   -2.5 39.0625 -31.25 4.0 11.8125  
   -2.0 16.0000 -20.00 4.0 0.0  
   -1.5 5.0625 -11.25 4.0 -2.1875  
   -1.0 1.0000 -5.00 4.0 0.0

### Nested Loops

1. Write a C# program that calculates and displays the yearly amount available if $1,000 is invested in a bank account for 10 years. Your program should display the amounts available for interest rates from 6 percent to 12 percent inclusively, at 1 percent increments. Use a nested loop, with the outer loop having a fixed count of 7 and the inner loop a fixed count of 10. The first iteration of the outer loop should use an interest rate of 6 percent and display the amount of money available at the end of the first 10 years. In each subsequent pass through the outer loop, the interest rate should be increased by 1 percent. Use the relationship that the money available at the end of each year equals the amount of money in the account at the start of the year plus the interest rate times the amount available at the start of the year.  
   **The output of this program should be in the form of a table.**
   1. Modify the program written for Question 1 to initially prompt the user for the amount of money initially deposited in the account.
   2. Modify the program written for Question 1 to initially prompt the user for both the amount of money initially deposited and the number of years that should be displayed.
   3. Modify the program written for Question 1 to initially prompt for the amount of money initially deposited, the interest rate to be used, and the number of years to be displayed.

### More on loops – the continue and break keywords

The do-while is a post-test loop, this means that the body of the loop is processed and then the condition is checked. This implies that the body is guaranteed to be processed at least once.

The other loop is the pre-test loop, this means that the condition is checked before the body is processed. This implies that the body might not be processed at all!

#### The continue keyword

When the continue keyword is seen, the rest of the statements in the loop body are ignored and processing carry on from the start of the loop.

#### The break keyword

When the break keyword is seen, the rest of the statements in the loop body are ignored and the loop is terminated

int a = 1;  
do  
{  
 Console.Write(a++);  
 if(a % 3 == 0)  
 break;  
 Console.WriteLine()  
*}while(a < 8);*

*//outputs  
123*

int a = 1;  
do  
{  
 Console.Write(a++);  
 if(a % 3 == 0)  
 continue;  
 Console.WriteLine()  
}while(a < 8);

//outputs  
123  
456  
78

Nested Loops

Write programs to output the following patterns. You must not hard-code the size of the figure. You should be able to output different sizes by changing the value of one or two variables

\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*

\*  
 \*  
 \*  
 \*  
 \*  
 \*  
 \*

\*  
 \*  
 \*  
 \*  
 \*  
 \*  
\*

\* \*  
 \* \*  
 \* \*  
 \* \*  
 \*  
 \* \*  
 \* \*  
 \* \*  
\* \*

\*\*\*\*\*\*\*  
\* \*\*\*\*\*\*  
\*\* \*\*\*\*\*  
\*\*\* \*\*\*\*  
\*\*\*\* \*\*\*  
\*\*\*\*\* \*\*  
\*\*\*\*\*\* \*  
\*\*\*\*\*\*\*

\*\*\*\*\*\*  
\*\*\*\*\* \*  
\*\*\*\* \*\*  
\*\*\* \*\*\*  
\*\* \*\*\*\*  
\* \*\*\*\*\*  
 \*\*\*\*\*\*

\*\*\*\*\*\*\*\*  
\* \*  
\* \*  
\* \*  
\* \*  
\* \*  
\* \*  
\*\*\*\*\*\*\*\*

\* \* \* \* \*   
 \* \* \* \* \*   
 \* \* \* \* \*   
 \* \* \* \*   
\* \* \* \* \*   
 \* \* \* \* \*   
 \* \* \* \* \*

\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*

\*  
 \*\*\*  
 \*\*\*\*\*  
 \*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* \*  
 \* \*  
 \* \*  
 \* \*  
 \* \*  
 \*

\*  
 \*\*\*  
 \*\*\*\*\*  
 \*\*\*\*\*\*\*  
\*\*\*\*\*\*\*\*\*  
 \*\*\*\*\*\*\*  
 \*\*\*\*\*  
 \*\*\*  
 \*