# CS19001: Programming and Data Structures Laboratory

**Assignment No. 3A (Iterations and Looping)**

**Date: 23-August-2019**

---

**Problem Statement:**

Madam Alice from Wonderland School wants to show the multiplication-table in her mathematics class. She wants to automate that by writing a C-program that will produce the multiplication-table for any number of entries ($< 10^6$) and upto any depth ($< 10^6$). Your task is to help Alice in writing this program which takes as input the number of multiplication-table entries and the multiplication depth (both as positive integers). The program outputs the multiplication-table in a structured format (refer to the sample cases below). However, there is one twist in the program. This program also takes an additional input to consider the base-value (decimal, binary, octal etc.) within $[2-10]$ and the multiplication-table data should be written considering this base. You have to implement such a multiplication-table in two ways:

- *Multiplicative Approach.* You need to multiply each entry in the table with each depth iteratively and provide the results after converting it under appropriate base-value from the decimal answer. For example, when you are producing the table for entry 4 – then for depth 6, you obtain $4 * 6 = 24$; for depth 7, you obtain $4 * 7 = 28$; and so on.
- *Additive Approach.* You are not allowed to multiply each entry with the depth, rather you can use repeated addition to get the next depth result from the present depth result for an entry. For example, when you are producing the table for entry 4 – then for depth 1, you start with 4 as the result; subsequently for depth 2, you obtain $4 + 4 = 8$; for depth 3, you obtain $8 + 4 = 12$; and so on.

Finally, you have to convert the each of these results under appropriate base-value format as indicated. For example, if you are specified with base 8 (octal number-system), then the result 28 in decimal will be written as 34 ($= 3 \times 8^1 + 4 \times 8^0$) in the multiplication-table.

---

**Example Inputs/Outputs:**

*Sample-1:*

```
Enter Number of Multiplication Table Entries: 9
Enter Multiplication Depth: 7
Enter Base Value (within range [2-10]): 10

++ Multiplicative Approach ++
[     1] ==>       1       2       3       4       5       6       7
[     2] ==>       2       4       6       8      10      12      14
[     3] ==>       3       6       9      12      15      18      21
[     4] ==>       4       8      12      16      20      24      28
[     5] ==>       5      10      15      20      25      30      35
[     6] ==>       6      12      18      24      30      36      42
[     7] ==>       7      14      21      28      35      42      49
[     8] ==>       8      16      24      32      40      48      56
[     9] ==>       9      18      27      36      45      54      63

++ Additive Approach ++
[     1] ==>       1       2       3       4       5       6       7
[     2] ==>       2       4       6       8      10      12      14
[     3] ==>       3       6       9      12      15      18      21
[     4] ==>       4       8      12      16      20      24      28
[     5] ==>       5      10      15      20      25      30      35
[     6] ==>       6      12      18      24      30      36      42
[     7] ==>       7      14      21      28      35      42      49
[     8] ==>       8      16      24      32      40      48      56
[     9] ==>       9      18      27      36      45      54      63
```

*Sample-2:*

```
Enter Number of Multiplication Table Entries: 10
Enter Multiplication Depth: 6
Enter Base Value (within range [2-10]): 2

++ Multiplicative Approach ++
[     1] ==>       1      10      11     100     101     110
[     2] ==>      10     100     110    1000    1010    1100
[     3] ==>      11     110    1001    1100    1111   10010
[     4] ==>     100    1000    1100   10000   10100   11000
[     5] ==>     101    1010    1111   10100   11001   11110
[     6] ==>     110    1100   10010   11000   11110  100100
[     7] ==>     111    1110   10101   11100  100011  101010
[     8] ==>    1000   10000   11000  100000  101000  110000
[     9] ==>    1001   10010   11011  100100  101101  110110
[    10] ==>    1010   10100   11110  101000  110010  111100

++ Additive Approach ++
[     1] ==>       1      10      11     100     101     110
[     2] ==>      10     100     110    1000    1010    1100
[     3] ==>      11     110    1001    1100    1111   10010
[     4] ==>     100    1000    1100   10000   10100   11000
[     5] ==>     101    1010    1111   10100   11001   11110
[     6] ==>     110    1100   10010   11000   11110  100100
[     7] ==>     111    1110   10101   11100  100011  101010
[     8] ==>    1000   10000   11000  100000  101000  110000
[     9] ==>    1001   10010   11011  100100  101101  110110
[    10] ==>    1010   10100   11110  101000  110010  111100
```

*Sample-3:*

```
Enter Number of Multiplication Table Entries: 8
Enter Multiplication Depth: 8
Enter Base Value (within range [2-10]): 8

++ Multiplicative Approach ++
[     1] ==>       1       2       3       4       5       6       7      10
[     2] ==>       2       4       6      10      12      14      16      20
[     3] ==>       3       6      11      14      17      22      25      30
[     4] ==>       4      10      14      20      24      30      34      40
[     5] ==>       5      12      17      24      31      36      43      50
[     6] ==>       6      14      22      30      36      44      52      60
[     7] ==>       7      16      25      34      43      52      61      70
[     8] ==>      10      20      30      40      50      60      70     100

++ Additive Approach ++
[     1] ==>       1       2       3       4       5       6       7      10
[     2] ==>       2       4       6      10      12      14      16      20
[     3] ==>       3       6      11      14      17      22      25      30
[     4] ==>       4      10      14      20      24      30      34      40
[     5] ==>       5      12      17      24      31      36      43      50
[     6] ==>       6      14      22      30      36      44      52      60
[     7] ==>       7      16      25      34      43      52      61      70
[     8] ==>      10      20      30      40      50      60      70     100
```

---

**Submit a single C source file. Do not use global/static variables.**

## CS19001: Programming and Data Structures Laboratory

**Assignment No. 3B (Iterations and Looping)**

**Date: 23-August-2019**

**Problem Statement:**

Madam Alice from Wonderland School wants to show in her mathematics class how the multiplication for two big numbers is computed using naïve school-book method. In this approach, Alice multiplies the multiplicand with the digits of the multiplier one at a time (starting from rightmost towards the left) and write the results obtained at every intermediate phases (shifting/aligning it to one position left of the previous line). Finally, she add all these intermediate stage results and produce the final answer. The two examples she gave in class are shown below.

```
     234    <== Multiplicand ==>         9876
x     56    <==  Multiplier  ==>    x     543
--------                           ----------
   1404                                29628
+ 1170                               39504
--------                           + 49380
  13104                            ----------
                                     5362668
```

Your task is to implement/automate this technique in terms of a C-program so that Alice need not have to do it by-hand everytime while showing it in her class. The program takes as input two non-negative integers (each of which is $< 10^9$) as the multiplicand and the multiplier. It outputs all the intermediate stages of results (as shown in the above examples) padding the extreme right fields with '0's (instead of blanks) and then provide the final answer by adding all these intermediate step-results.

**Example Inputs/Outputs:**

*Sample-1:*

```
Enter Multiplicand: 99999
Enter Multiplier: 888
++ School-book Multiplication Procedure ++
                           99999
                             888
---------------------------------
                          799992
                         7999920
                        79999200
---------------------------------
                        88799112
```

*Sample-2:*

```
Enter Multiplicand: 1111
Enter Multiplier: 203

++ School-book Multiplication Procedure ++
                            1111
                             203
---------------------------------
                            3333
                               0
                          222200
---------------------------------
                          225533
```

**Submit a single C source file. Do not use global/static variables.**

## CS19001: Programming and Data Structures Laboratory

**Assignment No. 3C (Iterations and Looping)**

**Date: 23-August-2019**

---

**Problem Statement:**

Madam Lorina from Wonderland School wants to show in her drawing class how a square block can be designed by drawing boundaries of the block with only numbers. The pattern she uses is to gradually decrement the number from outer boundary to the inner region boundary *by one*, and gradually reach the center point with number 1 in this process. For a given input $N$, the square block design is of $(2N-1)$ length. The outermost perimeter consists of only the number $N$, the immediate inner layer boundary consists of only the number $(N-1)$, the next inner layer boundary with $(N-2)$, ... so on, until the center is 1. The example she gave in class (for input 6) is shown below.

```
6  6  6  6  6  6  6  6  6  6  6
6  5  5  5  5  5  5  5  5  5  6
6  5  4  4  4  4  4  4  4  5  6
6  5  4  3  3  3  3  3  4  5  6
6  5  4  3  2  2  2  3  4  5  6
6  5  4  3  2  1  2  3  4  5  6
6  5  4  3  2  2  2  3  4  5  6
6  5  4  3  3  3  3  3  4  5  6
6  5  4  4  4  4  4  4  4  5  6
6  5  5  5  5  5  5  5  5  5  6
6  6  6  6  6  6  6  6  6  6  6
```

Your task is to implement/automate this technique in terms of a C-program so that Lorina need not have to do it by-hand everytime while showing it in her class. The program takes as input a positive integer (within range $[1-99]$) denoting the number to start marking outer boundary. The output will be the design as shown above.

---

**Example Inputs/Outputs:**

*Sample-1:*

```
Enter Positive Number for Outermost Boundary [1-99]: 5
The Drawing:
  5  5  5  5  5  5  5  5  5
  5  4  4  4  4  4  4  4  5
  5  4  3  3  3  3  3  4  5
  5  4  3  2  2  2  3  4  5
  5  4  3  2  1  2  3  4  5
  5  4  3  2  2  2  3  4  5
  5  4  3  3  3  3  3  4  5
  5  4  4  4  4  4  4  4  5
  5  5  5  5  5  5  5  5  5
```

*Sample-2:*

```
Enter Positive Number for Outermost Boundary [1-99]: 1
The Drawing:
  1
```

*Sample-3:*

```
Enter Positive Number for Outermost Boundary [1-99]: 2
The Drawing:
  2  2  2
  2  1  2
  2  2  2
```

---

**Submit a single C source file. Do not use global/static variables.**