Programming Practice: Numeral Arithmetic

1. Write a C program to add two binary numbers. The input data contains an even number of 0-1 strings with maximum 64 bits and no leading zeros. The last two numerals are "0 0". Repeat the addition operation until both input binary numbers are 0's. In each iteration, the program will read two 0-1 strings, n1 and n2, and add these two binary numbers with the result sum. In the output, print n1, n2, and sum aligning to the right with a "+" sign before n2 and a separated line below n2. Print "n1 + n2 = sum" in decimal. If the result exceeds 64 bits, print an overflow message. Solution: binary_addition.c.
   Sample input (binary_addition.txt):

```
111111 1
1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
1000010100101010101010101011
1010101010101011110101011
10000101001011101010101010101011111
10101011111010101011110101011
1010101010101010101010101010101010101010101010101010101010101010101010101010
1010101010101010101010101010101010101010101010101010101010101010101010101010
1111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111
0 0
```

   Sample output:

```
D:\>binary_addition < binary_addition.txt
   111111
+       1
---------
  1000000
63 + 1 = 64

    1111111111111111111111111111111111111111111111111111111111111111
+   1111111111111111111111111111111111111111111111111111111111111111
------------------------------------------------------------------
  10111111111111111111111111111111111111111111111111111111111111110
9007199254740991 + 18014398509481983 = 27021597764222974

  1000010100101010101010101011
+    1010101010101011110101011
--------------------------------
  1001101010000000001001010110
139635371 + 22370219 = 162005590

  10000101001011101010101010101011111
+     10101011111010101011110101011
-------------------------------------
  10001111111011010101010110100001010
4468856159 + 360535979 = 4829392138

  1010101010101010101010101010101010101010101010101010101010101010101010101010
+   1010101010101010101010101010101010101010101010101010101010101010101010101010
--------------------------------------------------------------------------------
  1111111111111111111111111111111111111111111111111111111111111111111111111111
12297829382473034410 + 6148914691236517205 = 18446744073709551615

   1111111111111111111111111111111111111111111111111111111111111111111111111111
+  1111111111111111111111111111111111111111111111111111111111111111111111111111
--------------------------------------------------------------------------------
  11111111111111111111111111111111111111111111111111111111111111111111111111110
18446744073709551615 + 18446744073709551615 = 18446744073709551614    **** Overflow!!!
```

2. Write a C program to add two octal numbers. The input data contains an even number of octal digit strings with maximum 64 bits and no leading zeros. The last two numerals are "0 0". Repeat the addition operation until both input octal numbers are 0's. In each iteration, the program will read two octal digit strings, n1 and n2, and add these two octal numbers with the result sum. In the output, print n1, n2, and sum aligning to the right with a "+" sign before n2 and a separated line below n2. Print "n1 + n2 = sum" in decimal. If the result exceeds 64 bits, print an overflow message. Solution: octal_addition.c.
   Sample input (octal_addition.txt):
   1 7777
   73064722 263004307726
   34704602726 26326027220
   27360774330202 2003044264521
   1252525252525252525252 525252525252525252525
   1777777777777777777777 1777777777777777777777
   0 0
   Sample output:

```
命令提示字元
D:\>octal_addition < octal_addition.txt
      1
+  7777
-------
  10000
1 + 4095 = 4096

      73064722
+ 263004307726
--------------
  263077374650
15493586 + 24026124246 = 24041617832

  34704602726
+ 26326027220
-------------
  63232632146
3876783574 + 3008900752 = 6885684326

  27360774330202
+  2003044264521
----------------
  31364040614723
1612893499522 + 137851136337 = 1750744635859

  1252525252525252525252
+  525252525252525252525
------------------------
  1777777777777777777777
1229782938247303441O + 6148914691236517205 = 18446744073709551615

  1777777777777777777777
+ 1777777777777777777777
------------------------
  3777777777777777777776
18446744073709551615 + 18446744073709551615 = 18446744073709551614    ****Overflow!!!


D:\>
微軟注音 半 :
```

3. Write a C program to multiply two binary numbers. The input data contains an even number of 0-1 strings with maximum 32 bits and no leading zeros. The last two numerals are "0 0". Repeat the multiplication operation until both input binary numbers are 0's. In each iteration, the program will read two 0-1 strings, n1 and n2, and multiply these two binary numbers with the result product of maximum 64 bits. Note that the length of product could be the total length of n1 and n2. Hence, no overflow will occur. In the

output, print n1, n2, and product aligning to the right with a "*" sign at the left-most character in the output line of n2 and a separated line below n2. Print "n1 * n2 = product" in decimal. Solution: binary_multiplication.c.

Sample input (binary_multiplication.txt):

```
111111 1
1011010
101010101
100001010101011
101010101110101011
1000011010101010101111
10101011110101011
10101010101010101010101010101010
10101010101010101010101010101010101
11111111111111111111111111111111
11111111111111111111111111111111
0 0
```

Sample output:

```
D:\>binary_multiplication < binary_multiplication.txt
  111111
*      1
--------
  111111
63 * 1 = 63

          1011010
*       101010101
-----------------
  111011111100010
90 * 341 = 30690

                   100001010101011
*               101010101110101011
-----------------------------------
  10110010000010101100000100111001
17067 * 175019 = 2987049273

                  1000011010101010101111
*                    10101011110101011
---------------------------------------
  10110100110010001111101110111100101
551599 * 87979 = 48529128421

                                 10101010101010101010101010101010
*                                 10101010101010101010101010101010101
-------------------------------------------------------------------
  11100011100011100011100011100001110001110001110001110001110010
2863311530 * 1431655765 = 4099276458915470450

                                   11111111111111111111111111111111
*                                  11111111111111111111111111111111
-----------------------------------------------------------------------------
  11111111111111111111111111111110000000000000000000000000000000001
4294967295 * 4294967295 = 18446744065119617025
```