# CS F211
# Data Structures and Algorithms
# Assignment - 1

Allowed Language: **C**

January 14, 2024

## General Tips

- Try to use functions as much as possible in your code. Functions increase reusability and the pass-by-value feature provides a significant help sometimes. Modularizing your code also helps you to debug efficiently.
- Use `scanf` to read characters/strings from STDIN. Avoid using `getchar`, `getc` or `gets`. Try to read up about character suppression in `scanf` as it will be very helpful in some of the problems.
- Use `printf` instead of `putc`, `putchar` or `puts` to print character/string output on STDOUT.
- Indent your code appropriately and use proper variable names. These increase readability and writability of the code. Also, Use comments wherever necessary.
- Use a proper IDEs like Sublime Text or VSCode as they help to run and test your code on multiple test-cases easily.

# A: Big Product

Now that in Assignment 0 you have done Addition pretty well, let's dive into some Multiplication. For this question, you will have to write a program to multiply 2 non-negative numbers, *Num1* and *Num2*. The numbers would be given in the input as strings and you are **not allowed** to use the **string.h** library to store the numbers. Compute their product and output it without any leading zeroes.

## Input

The first line contains a single non-negative integer *Num1* represented as a string. The second line contains a single non-negative integer *Num2* represented as a string. ($0 \leq Num2 \leq Num1 \leq 10^{1000}$).

## Output

Print a single string containing the product of the two long numbers provided to you. Note that do not add any leading zeroes.

---

input
72939269875772
45469387038502

output
331650389228722676760520973544

---

input
976492836403856038692837293025278
382752856829357029402739029393

output
373755422806977866255486846185241133203470127833335990333996254

---

input
2
1

output
2

---

# B: Karatsuba Multiplication

Karatsuba Algorithm is a straightforward modification of the recursive algorithm discussed in class and is described below (*Figure*1). In this problem, you are given 2 non-negative numbers as string inputs and you need to compute their product without any leading zeroes. The constraints are changed as necessary. All the necessary functions and computations should be clearly present in your code.
**Note:** You can assume that the length of the string $N$ is positive and a power of 2.

## Input

The first line contains a single non-negative integer *Num1* represented as a string. The second line contains a single non-negative integer *Num2* represented as a string. ($0 \leq Num2 \leq Num1 \leq 10^{100000}$).

## Output

Print a single string containing the product of the two long numbers provided to you. Note that do not add any leading zeroes.

---

```
input
38293821
10293098

output
394162052347458
```

---

```
input
5678
1234

output
7006652
```

---

```
input
2
1

output
2
```

---

`Karatsuba`

**Input:** two $n$-digit positive integers $x$ and $y$.
**Output:** the product $x \cdot y$.
**Assumption:** $n$ is a power of 2.

---

**if** $n = 1$ **then** `// base case`
    compute $x \cdot y$ in one step and return the result
**else** `// recursive case`
    $a, b :=$ first and second halves of $x$
    $c, d :=$ first and second halves of $y$
    compute $p := a + b$ and $q := c + d$ using grade-school addition
    recursively compute $ac := a \cdot c$, $bd := b \cdot d$, and $pq := p \cdot q$
    compute $adbc := pq - ac - bd$ using grade-school addition
    compute $10^n \cdot ac + 10^{n/2} \cdot adbc + bd$ using grade-school addition and return the result

Figure 1: Karatsuba Algorithm

# C: OOPs - I Did It Again

You get an NC in OOPs, and are required to give the make-up Compre. There you see one question. 100 Marks. 0 or 100. This question asks you to multiply two very large numbers using a *gmp.h* library which sir didn't teach in class. No Arrays, No Strings, No Recursion, No DP, Nothing! Will you be able to pass?

## Input

Let N = $10^{100000}$
The first line contains a single integer *Num1*.
The second line contains a single integer *Num2*.
($-10^N \leq$ *Num1*, *Num2* $\leq 10^N$).

## Output

Print a single integer containing the product of the two long numbers provided to you.

---

input
72939269875772
45469387038502

output
331650389228722676052097354 4

---

input
976492836403856038692837293025278
-382752856829357029402739029393

output
-373755422806977866255486846185241133203470127833335990333996254

---

# Bonus: Optimizing Karatsuba

You have successfully implemented the Karatsuba Algorithm for integer multiplication. Now, let's delve into optimizing and benchmarking your solution.

## 1. Benchmarking with GMP:

- Use the `time` command in Linux to measure the execution time of your Karatsuba implementation and the GMP library for varying input sizes (e.g., $2^{10}$, $2^{12}$, $2^{14}$, $2^{16}$).
- Record the results and observe the performance differences.

## 2. Profiling and Optimization:

- Employ a profiler (e.g., `gprof` or `perf`) to identify bottlenecks in your Karatsuba implementation.
- Optimize your code based on profiler insights to enhance performance.

## 3. GCC Optimization Flags:

- Experiment with optimization flags in GCC (e.g., `-O1`, `-O2`, `-O3`) to boost your Karatsuba implementation's efficiency.
- Assess the impact of different optimization levels on execution time.

## 4. Visualization:

- Generate plots or graphs showcasing the performance comparison between your Karatsuba and GMP for the tested input sizes.
- Clearly present any optimizations made to your code.
- Share insights gained from benchmarking and optimizing. Were there cases where your implementation outperformed GMP or vice versa?

# DD: Deep Dreams

After finishing your OOPs make-up Compre, you go into a deep sleep. You start dreaming about a DD quiz 4 which was never supposed to be there. You try all means to wake up but you get stuck in your dream. Giving up all hopes, you decide to solve the quiz. The quiz consists of 1 question, which requires you to find the number of 1s in the binary representation of the given number. But since you are a can't solve a DD question even in your dreams, you decide to write a C Program for it using the *gmp.h* library you learnt about in the OOPs make-up.

## Input

The first line contains a single integer $N$ ($0 \leq N \leq 10^{10000}$).

## Output

Print the number of 1s present in the binary representation of $N$.

---

input
0

output
0

---

input
1

output
1

---

input
8292923293847923

output
29

---

# E: TheHackerCat

Once upon a time, there was a guy named Vidyateja. He loved making websites. One day, he made a malicious website and sent it on your Whatsapp group using a non-suspicious handouts link. You open it you find all your devices locked. However, Vidyateja is very generous, so he told you that there is a *passcode* which is a non-negative integer which when multiplied with $N$ under mod $M$ gives 1. You also need to compute the *passcode* using only the *gmp.h* library which will allow you to break free from Vidyateja's Attack (and scroll reels of course).
**Note:** You can assume that *passcode* always exists.

## Input

The first line contains a single integer $N$ ($1 \leq N \leq 10^{10000}$).
The second line contains a single integer $M$ ($1 \leq M \leq 10^{10000}$).

## Output

Print a single integer *passcode*.

---

input
23281928469
283

output
174

---

input
271
2940

output
1291

---