# Exercise #1

## Score 25/25

In this challenge, you will learn to implement the basic functionalities of pointers in C. A [pointer](https://www.hackerrank.com/external_redirect?to=http://en.wikipedia.org/wiki/Pointer_%28computer_programming%29) in C is a way to share a memory address among different contexts (primarily functions). They are primarily used whenever a function needs to modify the content of a variable, of which it doesn't have ownership.  
  
In order to access the memory address of a variable, , we need to prepend it with  sign. E.g., &val returns the memory address of .  
  
This memory address is assigned to a pointer and can be shared among various functions. E.g.  will assign the memory address of  to pointer . To access the content of the memory to which the pointer points, prepend it with a \*. For example, \*p will return the value reflected by  and any modification to it will be reflected at the source ()

void increment(int \*v) {

(\*v)++;

}

int main() {

int a;

scanf("%d", &a);

increment(&a);

printf("%d", a);

return 0;

}

You have to complete the function void update(int \*a,int \*b), which reads two integers as argument, and sets  with the difference (a-b) of them, and  with the absolute difference of them (b-a) if the result of a-b is even or the product of them if a-b is odd.

* *a'=a-b*
* if a' is odd: *b'=|b-a|*
* if a' is even:  *b'=a\*b*

**Explanation**

* a = 0, b = 4
* a'=0-4=-4 //a' is even
* b'=0\*4=0

**For example:**

| **Test** | **Result** |
| --- | --- |
| int a, b;  int \* pa = &a, \* pb = &b;  \*pa = 0; \*pb = 4;  update(&a,&b);  printf("a = %d, b = %d\n",a,b);  \*pa = 1; \*pb = 2;  update(&a,&b);  printf("a = %d, b = %d\n",a,b);  \*pa = 10; \*pb = 4;  update(&a,&b);  printf("a = %d, b = %d\n",a,b);  \*pa = 13; \*pb = 11;  update(&a,&b);  printf("a = %d, b = %d\n",a,b); | a = -4, b = 0  a = -1, b = 1  a = 6, b = 40  a = 2, b = 143 |

# Exercise #2

## Score 0/25

In mathematics, the Fibonacci numbers are the numbers in the following integer sequence, called the Fibonacci sequence, and characterized by the fact that every number after the first two is the sum of the two preceding ones:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377...

By definition, the first two numbers in the Fibonacci sequence are either 1 and 1, or 0 and 1, depending on the chosen starting point of the sequence, and each subsequent number is the sum of the previous two.

This series can be broken down as the following series:

* Fib0 = 0
* Fib1 = 1
* Fibn = Fibn-1 + Fibn-2 , n > 1

**Tribonacci sequence:**

The tribonacci sequence is a generalization of the [Fibonacci sequence](https://brilliant.org/wiki/fibonacci-series/) where each term is the sum of the **three preceding**terms.  
This series can be broken down as the following series:

* Trib0= 0
* Trib1= 1
* Trib2= 1
* Tribn= Tribn-1 + Tribn-2 + Tribn-3, n>2

The sequence begins

You will create an array smaller than n using malloc/calloc to store the values of the series on the heap. You will use realloc to resize the array as needed. **You will recompute the Tribonacci series with every iteration (even if you already have the values).**

**Input Format**

Input will contain one integer, n, as the amount of series to compute and print.

**Output Format**

You will print ***n***tribonacci series up to **i** where **{i|0<=i<=n}** in new lines separeted by a single space. Each new line will print the tribonacci series for **i+1**.

**For example:**

| **Test** | **Result** |
| --- | --- |
| tribonacci(3);  tribonacci(0);  tribonacci(2);  tribonacci(20); | 0  0 1  0 1 1  0 1 1 2  0  0  0 1  0 1 1  0  0 1  0 1 1  0 1 1 2  0 1 1 2 4  0 1 1 2 4 7  0 1 1 2 4 7 13  0 1 1 2 4 7 13 24  0 1 1 2 4 7 13 24 44  0 1 1 2 4 7 13 24 44 81  0 1 1 2 4 7 13 24 44 81 149  0 1 1 2 4 7 13 24 44 81 149 274  0 1 1 2 4 7 13 24 44 81 149 274 504  0 1 1 2 4 7 13 24 44 81 149 274 504 927  0 1 1 2 4 7 13 24 44 81 149 274 504 927 1705  0 1 1 2 4 7 13 24 44 81 149 274 504 927 1705 3136  0 1 1 2 4 7 13 24 44 81 149 274 504 927 1705 3136 5768  0 1 1 2 4 7 13 24 44 81 149 274 504 927 1705 3136 5768 10609  0 1 1 2 4 7 13 24 44 81 149 274 504 927 1705 3136 5768 10609 19513  0 1 1 2 4 7 13 24 44 81 149 274 504 927 1705 3136 5768 10609 19513 35890  0 1 1 2 4 7 13 24 44 81 149 274 504 927 1705 3136 5768 10609 19513 35890 66012 |

# Exercise #3

## Score 0/25

In this problem, you will implement a decoder that produces strings from ciphertexts by:

    1. Converting the string into an nxn matrix in column-major order

    2. Generating the new string by reading the matrix in row major order

For example, the following matrix would be constructed for the input string "WECGEWHYAAIORTNU":

* W E A R
* E W A T
* C H  I N
* G Y O U

Taking characters in row-major order from this matrix, the decrypted text would read:

* WEAREWATCHINGYOU

**Input Format**

Input starts with a line consisting of a single number T. T test cases follow. Each test case consists of one line. This line contains the cipher text. The cipher text contains either UPPERCASE letters or blank spaces.

**Constraints**

Total number of character in the text will not be more 10,000.

**Output Format**

For each test case, the output contains a single line containing the decrypted text. If the number of characters in the input text is not square of any number, then give the output ‘INVALID’

**For example:**

| **Test** | **Result** |
| --- | --- |
| char cmd[MAX\_CHARS];  strcpy(cmd, "WE OUE OUT LNGSAO H RWDN!");  decode(cmd); // Expected: WE ARE LOW ON DOUGHNUTS !  memset(cmd,0,MAX\_CHARS);  strcpy(cmd, "S EFEM FNOCEDROE");  decode(cmd); // Expected: SEND MORE COFFEE  memset(cmd,0,MAX\_CHARS);  strcpy(cmd, "CSERULES");  decode(cmd); // Expected: INVALID  memset(cmd,0,MAX\_CHARS); | WE ARE LOW ON DOUGHNUTS !  SEND MORE COFFEE  INVALID |

# Exercise #4

## Score 25/25

Complete the following script to sort a list of words/numbers.

Use the pointer "word" and the double pointer "word\_list" to store the words given by the STD Input.

NOTE: Complete the code following the TODO hints

**For example:**

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| Expected:  AAA  BBB  CCC  DDD | 4  CCC  BBB  DDD  AAA | AAA  BBB  CCC  DDD |