1. Kim finds the digits 3, 4 and 7 to be lucky digits. A number is called Nearly Lucky if the count of lucky digits in a number is also a lucky digit.

You are given **N** numbers, tell how many of them are Nearly Lucky?

Input	Output	Output Description
1 40047	1	40047 has 3 lucky digits, which is a lucky digit.
2 40447 313	1	40447 is nearly lucky because it has 4 lucky digits, and the count 4 itself is a lucky digit. 313 is not nearly lucky because it has 2 lucky digits, and the count 2 is not a lucky digit.
2 41347 343	2	41347 has 4 lucky digits and the count 4 itself is a lucky digit. Hence 41347 is nearly lucky. 343 has 3 lucky digits and the count 3 is also a lucky digit. Hence 343 is also nearly lucky.

```
class Solution {

     static int nearlyLucky(List<int> Arr) {

        // Write your code here
        int sumcount=0;
        for(int i=0;i<Arr.Count;i++){
            int count=0;
           string temp= Arr[i].ToString();
            for(int j=0;j<temp.Length;j++){
                if(temp[j]=='3'||temp[j]=='4'||temp[j]=='7'){
                     count++;
                }
            }
            if(count==3||count==4||count==7){
                sumcount++;
            }
```

```
        }
        return sumcount;
    }

    static void Main(string[] args) {
        int N;
        N = Convert.ToInt32(Console.ReadLine());
        List<int> Arr = new List<int>();
        for(int j = 0; j < N; ++j){
            int ArrItem = Convert.ToInt32(Console.ReadLine());
            Arr.Add(ArrItem);
        }

        int result;
        result = nearlyLucky(Arr);
        Console.Write(result);
    }

}
```

---

2. You are given an array of integers, ***arr,*** of size **array_length**.

A **"couple"** is any two elements in a row. A **"triplet"** is any three elements in a row. The elements of a **couple** and a **triplet** must be continuous and non-circular i.e. they must **NOT** wrap around the end of the array.

For example, in the array **arr** = [1, 2, 3, 4], there are **three couples** and **two triplets** as shown below:

- **Couples**: (1, 2), (2, 3), and (3, 4)
- **Triplets**: (1, 2, 3) and (2, 3, 4)

Your task is to score ***arr*** as per the following rules:

1. For every **couple** in ***arr*** whose **sum** is **even**, increase the **score** by **5**.
2. For every **triplet** in ***arr*** whose **sum** is **odd AND** whose **product** is **even**, increase the **score** by **10**.
3. If neither of the above conditions are met, then the **score** is unchanged

Output the final value of **score** for ***arr.***

Notes:

- **Couples** and **triplets** may overlap with each other
- The **score** is initially 0
- The array may contain positive and negative numbers
- 0 is an **even** number.

Input	Output	Output Description
1 94	0	Since array_length is 1, there are no couplets or triplets. Therefore, the score is 0
5 -26 71 6 -3 -69	15	Only couple having even sum is (-3,-69) so we add 5 to score. And triple having even product and odd sum is (-26,71,6) so we add further 10 to the score. Final score=5+10=15
5 19 0 -95 -93 -37	10	Couple having even sum are (-95,-93) and (-93,-37) so we add 10 to score. And there is no triple having even product and odd sum. Final score=5+5=10

```
class Solution {
    static int score(int array_length, List<int> arr) {
        // Write your code here
        int score=0;
        for(int i=0;i<array_length-2+1;i++){
            if((arr[i]+arr[i+1])%2==0){
                score+=5;
            }
```

```
        }
        for(int j=0;j<array_length-3+1;j++){
            int sum=arr[j]+arr[j+1]+arr[j+2];
            int pro=arr[j]*arr[j+1]*arr[j+2];
            if((sum%2!=0)&&(pro%2==0)){
                score +=10;
            }
        }
    return score;
    }

    static void Main(string[] args) {
        int array_length = Int32.Parse(Console.ReadLine().Trim());

        List<int> arr = new List<int>();
        for(int j=0; j<array_length; j++) {
            arr.Add(Int32.Parse(Console.ReadLine().Trim()));
        }
        int result = score(array_length, arr);

        Console.WriteLine(result);
    }
}
```

---

4. You are given an integer **N** and integer array **A** as the input, where **N** denotes the length of **A**. Assume the sum of all the elements of **A** to be **S**. Write a program to return the product of **N** and **S**.

Input	Output	Output Description
2 23 45	136	Sum of the elements is 23+45=68. Product of the sum with the length is 2*68=136.
4 1 2 3 4	40	Sum of the elements is 1+2+3+4=10. Product of the sum with the length is 4*10=40.
3 264 334 309	2721	Sum of the elements is 264+334+309=907. Product of the sum with the length is 3*907=2721.

```
class Solution {
     static int check(List<int> A) {
        // Write your code here
        int sum=0;
        int pro=0;
        for(int i=0;i<A.Count;i++){
            sum=sum+A[i];
        }
        pro=sum*A.Count;
        return pro;
    }
    static void Main(string[] args) {
        int N;
        N = Convert.ToInt32(Console.ReadLine());
        List<int> A = new List<int>();
        for(int j = 0; j < N; ++j){
            int AItem = Convert.ToInt32(Console.ReadLine());
            A.Add(AItem);
        }
        int result;
        result = check(A);
```

```
        Console.Write(result);
    }

}
```

---

4. You are given an array of integers, ***arr***, of size **array_length**.

A "**couple**" is **any** two adjacent elements which are non-circular. However, elements must NOT wrap around the end of the array. It is given that the **two** elements of **each couple** are the length, **L**, and width **W**, of a rectangle. It is given that **couples** may overlap with each other.

A **couple** may be categorized as follows:
1) A couple is known as a "**Type A Couple**" if the **area of the rectangle** > **perimeter of the rectangle**
2) A couple is known as a "**Type B Couple**" if the **area of the rectangle** < **perimeter of the rectangle**
3) A couple is known as a "**Type C Couple**" if the **area of the rectangle** = **perimeter of the rectangle**

Find the **absolute difference** between the **total number** of **type A couples** and the **total number** of **type B couples** in **arr**.

Notes:

- **arr** is guaranteed to **only** contain positive integers.
- **Area** = **L * W** ; **Perimeter = 2L + 2W.**

Input	Output	Output Description
5 1 2 3 4 5	2	Couples = (1,2), (2,3), (3,4), (4,5) Type A couples = (4,5) Type B couples = (1,2), (2,3), (3,4) Absolute difference = | 1 - 3 | = 2
6 1 2 3 4 5 6	1	Couples = (1,2), (2,3), (3,4), (4,5), (5,6) Type A couples = (4,5), (5,6) Type B couples = (1,2), (2,3), (3,4) Absolute difference = | 2 - 3 | = 1
7 1 2 3 4 5 6 7	0	Couples = (1,2), (2,3), (3,4), (4,5), (5,6), (6,7) Type A couples = (4,5), (5,6), (6,7) Type B couples = (1,2), (2,3), (3,4) Absolute difference = | 3 - 3 | = 0

```
class Solution {
    static int absoluteDifferenceTypeATypeB(int array_length, List<int> arr) {
        // Write your code here
        int a=0,b=0;
        for(int i=0;i<array_length-1;i++){
            int l=arr[i];
            int w=arr[i+1];
            int area=l*w;
            int perimeter=2*l+2*w;
            if(area>perimeter){
```

```
                a++;
            }
            else if(area<perimeter){
                b++;
            }
        }
        int diff=Math.Abs(a-b);
        return diff;
    }

    static void Main(string[] args) {
        int array_length = Int32.Parse(Console.ReadLine().Trim());

        List<int> arr = new List<int>();
        for(int j=0; j<array_length; j++) {
            arr.Add(Int32.Parse(Console.ReadLine().Trim()));
        }
        int result = absoluteDifferenceTypeATypeB(array_length, arr);

        Console.WriteLine(result);
    }
}
```

---

5. You are given an integer, **N,** and an array of integers, **arr,** of size **array_length**.

Find the **total number** of occurrences of **N** in **arr.**

Input	Output	Output Description
4 5 1 2 3 4 4	2	The number 4 appears twice in arr, hence the output is 2.
0 5 1 2 3 4 5	0	The number 0 does not appear in arr, hence the output is 0.
4 5 1 2 3 4 -4	1	The number 4 appears once in arr, hence the output is 2.

```
class Solution {
    static int findFreq(int N, int array_length, List<int> arr) {
        // Write your code here
        int count=0;
        for(int i=0;i<array_length;i++){
            if(arr[i]==N)
            {
                count++;
            }
        }
```

```
        return count;
    }
    static void Main(string[] args) {
        int N = Int32.Parse(Console.ReadLine().Trim());

        int array_length = Int32.Parse(Console.ReadLine().Trim());

        List<int> arr = new List<int>();
        for(int j=0; j<array_length; j++) {
            arr.Add(Int32.Parse(Console.ReadLine().Trim()));
        }
        int result = findFreq(N, array_length, arr);

        Console.WriteLine(result);
    }
}
```

---

6. You are given an array of integers, ***arr***, of size **array_length**. Find the **total number** of **perfect squares** in **arr**.

A **perfect square** is **any** integer that can be written as **a*a** (i.e. **a^2**). This means that the numbers 0, 1, 4, 9, are **perfect squares**.

Notes:

- The array may contain positive and negative numbers. However, negative numbers cannot be perfect squares

9 -9 11 12 19 14 -10 14 -9 13	0	There are no perfect squares in this array. Hence, output = 0.
10 -15 -16 4 25 20 -20 12 22 -12 -11	2	There are 2 perfect squares in this array: 25 and 4.
10 14 -11 -25 24 -15	1	There is 1 perfect square in this array: 25.

```
class Solution {
    static int numPerfectSquares(int array_length, List<int> arr) {
        // Write your code here
        int count=0;
        for(int i=0;i<array_length;i++){
```

```
            int pro=(int)Math.Sqrt(arr[i]);
            if(pro*pro==arr[i]){
                count++;
            }
        }
        return count;

    }

    static void Main(string[] args) {
        int array_length = Int32.Parse(Console.ReadLine().Trim());

        List<int> arr = new List<int>();
        for(int j=0; j<array_length; j++) {
            arr.Add(Int32.Parse(Console.ReadLine().Trim()));
        }
        int result = numPerfectSquares(array_length, arr);

        Console.WriteLine(result);
    }
}
```

---

7. You are given a string, ***str.***

You need to find the **score** of ***str*** as per the following rules:
1) For each palindrome of **length 4** in *str*, add **5** to the score
2) For each palindrome of **length 5** in *str*, add **10** to the score

It is given that **palindromes** can overlap within ***str***. However, the characters of the palindrome must be continuous and must NOT be circular (i.e. they must NOT wrap around the end of the string.

Output the **final value** of **score.**

Notes:

• A **palindrome** is a word, phrase, or sequence that reads the same backwards as forwards. Examples of **palindromes** are words like "madam", "racecar", "abCba" and "aBccBa", and "ABBA"
• Palindromes are **case-sensitive** in this problem (e.g. "Madam" and "ABCcba" **are NOT** palindromes)
• The **score** is initially 0
• There are **no** whitespaces in **str.**

```
class Solution {

     static int scoreString(String str) {
        // Write your code here
    int sum = 0;
for (int i = 0; i < str.Length; i++)
{
    if (i + 5 <= str.Length)
    {
        string temp = str.Substring(i, 5);
        char[] temp1 = temp.Reverse().ToArray();

        if (temp.SequenceEqual(temp1))
            sum += 10;
    }
    if (i + 4 <= str.Length)
    {
        string temp = str.Substring(i, 4);
        char[] temp1 = temp.Reverse().ToArray();

        if (temp.SequenceEqual(temp1))
            sum += 5;
    }

}
return sum;
    }

    static void Main(string[] args) {
        string str;
        str = Console.ReadLine();
        int result;
        result = scoreString(str);
        Console.Write(result);
    }

}
```

8. You are given an integer, **N**, and an integer array, **arr**, of size **array_length**.

A "**couple**" is **any two** adjacent elements in **arr** which are non-circular. This means that elements of a **couple** must NOT wrap around the end of the array.

Find the **total number** of **couples** in **arr** whose **sum** is perfectly divisible by **N**.

Notes:

- **arr** may contain positive and negative integers.
- **N** may be positive or negative.

Input	Output	Output Description
3 5 1 2 3 4 5	2	Couples = (1,2), (2,3), (3,4), (4,5) Sum of couples = 3, 5, 7, 9 Only 3 and 9 are divisible by 3, hence the output is 2.
2 5 1 2 3 4 5	0	Couples = (1,2), (2,3), (3,4), (4,5) Sum of couples = 3, 5, 7, 9 None of these numbers is divisible by 2, hence the output is 0
3 5 -1 -2 -3 -4 -5	2	Couples = (-1,-2), (-2,-3), (-3,-4), (-4,-5) Sum of couples = -3, -5, -7, -9 Only -3 and -9 are divisible by 3, hence the output is 2

```
class Solution {

     static int numDivisible(int N, List<int> arr) {
        // Write your code here
        int count=0;
        for(int i=0;i<arr.Count-2+1;i++){
            int temp=arr[i]+arr[i+1];
            if(temp%N==0){
                count++;
            }
        }
        return count;
    }

    static void Main(string[] args) {
        int N;
        N = Convert.ToInt32(Console.ReadLine());
        int array_length;
        array_length = Convert.ToInt32(Console.ReadLine());
        List<int> arr = new List<int>();
        for(int j = 0; j < array_length; ++j){
            int arrItem = Convert.ToInt32(Console.ReadLine());
            arr.Add(arrItem);
        }

        int result;
        result = numDivisible(N,arr);
        Console.Write(result);
    }

}
```

---

9. You are given an array of integers, **arr**, of size **array_length**. You need to find the **pyramid sum** of **arr**.

A **pyramid sum** can be described as follows.

It is known that there are (**array_length** - 1) adjacent pairs for **arr**. Hence, if we repeatedly calculate the **sum** of **adjacent pairs** in an array and store this in a new array, we will eventually have an array of length 1. The value stored in the array of length 1 is known as the **pyramid sum**.

Let **arr** = [1, 2, 3], the steps required to find the **pyramid sum** of **arr** are as follows:

1. 1+2, 2+3] ==> [3, 5]
2. [3+5] ==> [8]. Thus, **pyramid sum** = 8

Find the **pyramid sum** of **arr**.

Notes:

- All **pyramid sums** in this problem are between - (2^31 - 1) and + (2^31 - 1)

Input	Output	Output Description
1 500	500	The array contains just one element, so the pyramid sum is that number itself
5 1 2 3 4 5	48	A=[1,2,3,4,5] After one operation A=[3,5,7,9] After second operation A=[8,12,16] After third operation A=[20,28] After 4th operation A=[48]
3 -238 89 273	213	A=[-238,89,273]] After one operation A=[-149,362] After second operation A=[213]

```
class Solution {
      static int FindPyramidSum(List<int> arr)
    {
        while (arr.Count > 1)
        {
            List<int> temp = new List<int>(arr.Count - 1);
            for (int i = 0; i < arr.Count - 1; i++)
            {
                temp.Add(arr[i] + arr[i + 1]);
            }
            arr = temp;
        }
        return arr[0];
    }

    static int pyramidSum(int array_length, List<int> arr)
    {
        int pyramidSum = FindPyramidSum(arr);
        return pyramidSum;
    }

    static void Main(string[] args) {
        int array_length = Int32.Parse(Console.ReadLine().Trim());

        List<int> arr = new List<int>();
        for(int j=0; j<array_length; j++) {
            arr.Add(Int32.Parse(Console.ReadLine().Trim()));
        }
        int result = pyramidSum(array_length, arr);

        Console.WriteLine(result);
    }
}
```

---

10. You are given a positive integer, **num**. You have to reverse **num** and store it in **A.**

Output the value of **A.**

Note:

- Your answer must not contain leading zeroes. For example if num is 200 then you should return 2 and not 002.

Input	Output	Output Description
896	698	The reverse of 896 is 698.
573	375	The reverse of 573 is 375.
589	985	The reverse is 589 is 985.

```
class Solution {

     static int reverseNum(int num) {
        // Write your code here
        int rev=0;
        while(num>0){
            int rem=num%10;
            rev=rev*10+rem;
            num=num/10;
        }
        return rev;
    }

    static void Main(string[] args) {
        int num;
        num = Convert.ToInt32(Console.ReadLine());
        int result;
        result = reverseNum(num);
        Console.Write(result);
    }

}
```