Logical error

Problem (1):

The function/method countDigits return an integer representing the remainder when the given number is divided by the number of digits in it.
The function/method countDigits accepts an argument-num,an integer representing the given number.

The function/method countDigits compiles code successfully but fails to get the desirable result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

Test case 1:

Input: 782

Expected return value: 2

Test case 2:

Input: 21340

Expected return value: 0

```
int countDigits(int num)
{
	int count=0;
	while(num!=0)
	{
		num=num/10;
		count++;
	}
	return (num%count);
}
```

---
solution:

```
int countDigits(int num)
{
	int count=0,temp=num;
	while(temp!=0)
	{
		temp=temp/10;
		count++;
	}
	return (num%count);
}
```

Problem (2):

The function/ method arrayReverse modify the input list by reversing its element. The function/method arrayReverse accepts two arguments - len an integer representing the length of the list and arr, list of integers representing the input list respectively

For example, if the input list arr is {20 30 10 40 50}.
The function/method arrayReverse compiles code successfully but fails to get the desirable result for some test cases due to logical errors. Your task is to fix the code so that it passes all the test cases.

Testcase 1:

Input:
4,[4,2,8,6]

Expected return value:
6 8 2 4

Testcase 2:

Input:
3,[11,20,17]

Expected return value:
17 20 11

```
void arrayReverse(int len,int *arr)
{
	int i,temp,originalLen=len;
	for(i=0;i<=originalLen/2;i++)
	{
		temp=arr[len-1];
		arr[len-1]=arr[i];
		arr[i]=temp;
		len-=1;
	}
}
```

Problem (3):

The function/method printCharacterPattern accepts an integer num. It is supposed to print the first num(0<=num<=26) lines of the pattern as shown below:

For example,if num=4, the pattern is

a
ab
abc
abcd

The function/method printCharacterPattern compiles successfully but fails to get desirable output. Fix the logical errors.

```
void printCharacterPattern(int num)
{
	int i,j;
	char ch='a';
	char print;
	for(i=0;i<num;i++)
	{
		print=ch;
		for(j=0;j<=i;j++)
		{
			printf("%c",ch++);
		}
		printf("\n");
	}
}
```

Solution:

```
void printCharacterPattern(int num)
{
	int i,j;
	char ch='a';
	char print;
	for(i=0;i<num;i++)
	{
		print=ch;
		for(j=0;j<=i;j++)
		{
			printf("%c",print++);
		}
		printf("\n");
	}
}
```

Probelm (4):

The function/method removeElement prints space seperated integer that remains after removing the integer at the given index from the input list.

The function/method removeElement accepts three arguments,size an integer representing the sizeof input list, indexValue an integer representing given index and inputList, a list of integers representing the input list.

The function/method removeElement compiles code successfully but fails to get the desirable result for some test cases due to incorrect implementation of function/method removeElement. Your task is to fix the code so that it passes all the test cases.

Note: zer based indexing is follwed to access list elements

Test case 1:

Status: wrong
Expected:
1 2 3 5 6 7 8 9

Returned
1 2 3 4 4 6 6 8

Test case 1:

Status: correct
Expected:
11 23 12 34 54 32

Returned
11 23 12 34 54 32

```
void removeElement(int size,int indexValue,int *inputList)
{
	int i,j;
	if(indexValue<size)
	{

		for(i=indexValue;i<size;i++)
		{
			inputList[i]=inputList[i++];
		}

		for(i=0;i<size-1;i++)
		{
			printf("%d",inputList[i]);
		}
	}
	else
	{
		for(i=0;i<size;i++)
		{
			printf("%d",inputList[i]);
		}
```

---

solution:

```
void removeElement(int size,int indexValue,int *inputList)
{
	int i,j;
	if(indexValue<size)
	{

		for(i=indexValue;i<size;i++)
		{
			inputList[i]=inputList[i+1];//1 2 3 4 5 6 7 8 9
		}

		for(i=0;i<size-1;i++)
		{
			printf("%d",inputList[i]);
		}
	}
	else
	{
		for(i=0;i<size;i++)
		{
			printf("%d",inputList[i]);
		}
	}
```

}

Probelm (5):

The function calculateMatrixSum(int ** matrix, int m, int n) accepts a two dimensional array matrix of dimensions m, n as input and returns the sum of odd elements whose ith and jth index are same.

The function compiles successfully but fails to return the desired result for some test cases

PROGRAM

```
int calculateMatrixSum(int rows,int columns,int ** matrix))
{
	int i,j,sum=0;
	if((row>0)&&(column>0))
	{
		for(i=0;i<row;i++)
		{
			sum=0;
			for(j=0;j<column;j++)
			{
				if(i==j)
				{
					if(matrix[i][j]/2!=0)
						sum+=matrix[i][j];
				}
			}
		}
		return sum;
	}
	else
		return sum;
}
```

---

solution:

```
int calculateMatrixSum(int rows,int columns,int ** matrix))
{
	int i,j,sum=0;
	if((row>0)&&(column>0))
	{
		for(i=0;i<row;i++)
		{

			for(j=0;j<column;j++)
			{
				if(i==j)
				{
					if(matrix[i][j]%2!=0)
						sum+=matrix[i][j];
				}
			}
		}
		return sum;
	}
	else
		return sum;
```

```
}
```

Problem (6):

The function/method sortArray modify the input list by sorting its elements in descending order. It accepts two arguments-len,representing the length of the list and arr, a list of integers representing the input list respectively.

It compiles successfully but fails for some test case. Fix the code..

```
void sortArray(int len,int *arr)
{
	int i,max,location,j,temp;
	for(i=0;i<len;i++)
	{
		max=arr[i];
		location=i;
		for(j=i;j<len;j++)
		{
			if(max>arr[j])
			{
				max=arr[j];
				location=j;
			}
		}
		temp=arr[i];
		arr[i]=arr[location];
		arr[location]=temp;
	}
}
```
---
solution:

```
void sortArray(int len,int *arr)
{
	int i,max,location,j,temp;
	for(i=0;i<len;i++)
	{
		max=arr[i];
		location=i;
		for(j=i;j<len;j++)
		{
			if(max<arr[j])
			{
				max=arr[j];
				location=j;
			}
		}
		temp=arr[i];
		arr[i]=arr[location];
		arr[location]=temp;
	}
	for(i=0;i<len;i++)
	{
	   printf("%d ",arr[i]);
	}
}
```

Problem (7):

The function/method replaceValues is modifying the input list in such a way - if the length of the input list is odd, then all the elements of the input list are supposed to be replaced by 1s and in case it is even, the elments should be replaced by 0s.

For example, given the input list[0,1,2], the function will modify the input list like [1 1 1].It accepts two arguments size, an integer representing the size of input listand inputlist , a list of integers.

The function complies successfully but fails to return the desired results due to logical errors

Your task is to debug the program to pass all the test cases

```
void replaceValues(int size,int *inputList)
{
	int i,j;
	if(size%2==0)
	{
		i=0;
		while(i<size)
		{
			inputList[i]=0;
			i+=2;
		}
	}
	else
	{
		j=0;
		while(j<size)
		{
			inputList[j]=1;
			j+=2;
		}
	}
}
```

---

solution:

```
void replaceValues(int size,int *inputList)
{
	int i,j;
	if(size%2==0)
	{
		i=0;
		while(i<size)
		{
			inputList[i]=0;
			i+=1;
		}
	}
	else
	{
		j=0;
		while(j<size)
		{
```

```
                inputList[j]=1;
                j+=1;
            }
        }
}
```

Problem (8):

The function productMatrix accepts a two dimensional array matrix of dimensions m, n as input and returns the sum of odd elements whose ith and jth index are same.

The function compiles line but fails to return the desired result for some test cases

PROGRAM

```
int productMatrix(int rows,int columns,int **matrix)
{
    int result=0;
        for(i=0;i<row;i++)
        {

            for(j=0;j<column;j++)
            {
                if((i==j)||matrix[i][j]%2!=0)
                        result*=matrix[i][j];
            }
        }
        if(result<=1)   return 0;

        else
            return result;
}
```

---

solution:

```
int productMatrix(int rows,int columns,int **matrix)
{
    int result=0,flag=0;
        for(i=0;i<row;i++)
        {

            for(j=0;j<column;j++)
            {
                if((i==j) && matrix[i][j]%2!=0)
                {
                        result*=matrix[i][j];
                        flag=1;
                }
            }
        }
        if(flag==0)
            return 0;

        else
            return result;
}
```

Problem (9):

The function maxReplace is supposed to replace every element of the input array arr with the maximum element of arr.

The function complies unsuccessfullydue to compilation errors.

Your task is to debug the program to pass all the test cases

PROGRAM:

```
void maxReplace(int size,int &inputList)
{
	int i;
	if(size>0)
	{
		int max=inputList[0];
		for(i=0;i<size;i++)
		{
			if(max<inputList[i])
			{
				max=inputList[i];	}
		}
	}
	for(i=0;i<size,i++)
	{
		inputList[i]=max
		printf("%d",inputList[i]);
	}
}
```

---

solution:

```
void maxReplace(int size,int *inputList)
{
	int i,max;
	if(size>0)
	{
		 max=inputList[0];
		for(i=0;i<size;i++)
		{
			if(max<inputList[i])
			{
				max=inputList[i];	}
		}
	}
	for(i=0;i<size;i++)
	{
		inputList[i]=max;
		printf("%d",inputList[i]);
	}
}
```

Problem (10):

The function/method sumElement return an integer representing the sum of the elements in the input array that are greater than twice the input number K and present at the even index. It accepts three aguments-size of array, numK representing input number and inputarray list of integers.

It compiles successfully but fails to return the desirable result.

```
int sumElement(int size, int numK,int *inputArray)
{
	int i,sum=0;
	for(i=0;i<size;i++)
	{
		if(inputArray[i]>2*numK && i/2==0)
		{
			sum=inputArray[i];
		}
	}
	return sum;
}
```

---

solution:

```
int sumElement(int size, int numK,int *inputArray)
{
	int i,sum=0;
	for(i=0;i<size;i++)
	{
		if(inputArray[i]>2*numK && i%2==0)
		{
			sum+=inputArray[i];
		}
	}
	return sum;
}
```

Problem (11):

The function/method manchester: for each element in the input array arr, a counter is incremented if the bit arr[i] is same as arr[i-1]. Then the increment counter value is added to the output array to store the result.

If the bit arr[i] and arr[i-1] are different, then 0 is added to output array. for the first bit in the input array, assume its previous bit to be 0. For example if arr is {0,1,0,0,1,1,1,0}, then it should print 1 0 0 2 0 3 4 0.

It accepts two arguments-size and arr-list of integers. Each element represents a bit 0 or 1.

It compiles successfully but fails to print the desirable result. Fix the code.

```
void manchester(int size,int *arr)
{
	bool result;
	int *res=(int *)malloc(sizeof(int)*size);
	int count=0;
	for(int i=0;i<size;i++)
	{
		if(i==0)
			result=(arr[i]==0);
		else
			result=(arr[i]==arr[i-1]);
		res[i]=(result)?0:(++count);
	}
	for(int i=0;i<size;i++)
```

```
	{
		printf("%d",res[i]);
	}
}
```
---
solution:

```
void manchester(int size,int *arr)
{
	bool result;
	int res[size];
	int count=0;
	for(int i=0;i<size;i++)
	{
		if(i==0)
			result=(arr[i]==1);
		else
			result=!(arr[i]==arr[i-1]);
		res[i]=(result)?0:(++count);
	}
	for(int i=0;i<size;i++)
	{
		printf("%d ",res[i]);
	}
}
```

Problem (12):

The function/method printFibonacci accepts an integer num, representing a number. The function/method printFibonacci prints first num numbers of fibonacci series.

For example,given input 5, the function should print the string "01123"(without quotes).

It compiles successfully but fails to give the desirable result for some test cases. Debug the code.

```
void printFibonacci(int num)
{
	long num1=0;
	long num2=1;
	for(int i=1;i<num;++i)
	{
		printf("%ld",num1);
		long sum=num1+num2;
		num2=sum;
		num1=num2;
	}
}
```

Solution:

```
void printFibonacci(int num)
{
	long num1=0;
	long num2=1;
	for(int i=1;i<=num;++i)
	{
```

```
                printf("%ld ",num1);
                long sum=num1+num2;

                num1=num2;
                num2=sum;
        }
}
```

Problem (13):

The function/method selectionSortArray performs an in-place selection sort on the given input list which will be sorted in ascending order.

It accepts two arguments-len,an integer representing the length of input list and arr, alist of integers representing the input list respectively.

It compiles successfully but fails to get the desired result .

```
void selectionSortArray(int len,int *arr)
{
        int x=0,y=0;
        for(x=0;x<len;x++)
        {
                int index_of_min=x;
                for(y=x;y<len;y++)
                {
                        if(arr[index_of_min]>arr[x]){
                        index_of_min=y;
                        }
                }
                int temp=arr[x];
                arr[x]=arr[index_of_min];
                arr[index_of_min]=temp;
        }
}
```
---
solution:

```
void selectionSortArray(int len,int *arr)
{
        int x=0,y=0;
        for(x=0;x<len;x++)
        {
                int index_of_min=x;
                for(y=x;y<len;y++)
                {
                        if(arr[index_of_min]>arr[y]){
                        index_of_min=y;
                        }
                }
                int temp=arr[x];
                arr[x]=arr[index_of_min];
                arr[index_of_min]=temp;
        }
        for(x=0;x<len;x++)
        {
           printf("%d ",arr[x]);
```

```
        }
}
```

Probelm (14):

The function/method descendingSortArray performs an in-place sort on the given input list which will be sorted in descending order.

It accepts two argument-len of an array and array of elements.

It compiles succeszsfully but fails to get the desirable output.

```
void descendingSortArray(int len, int *arr)
{
        int small,pos, i,j,temp;
        for(i=0;i<=len-1;i++)
        {
                for(j=i;j<len;j++)
                {
                        temp=0;
                        if(arr[i]>arr[j])
                        {
                                temp=arr[i];
                                arr[i]=arr[j];
                                arr[j]=temp;
                        }
                }
        }
}
```
---
solution:

```
void descendingSortArray(int len, int *arr)
{
        int small, i,j,temp;
        for(i=0;i<=len-1;i++)
        {
                for(j=i+1;j<len;j++)
                {
                        temp=0;
                        if(arr[i]<arr[j])
                        {
                                temp=arr[i];
                                arr[i]=arr[j];
                                arr[j]=temp;
                        }
                }
        }
                for(i=0;i<=len-1;i++)
        {
           printf("%d ",arr[i]);
        }
}
```
---
problem (15):

The function/method patternPrint accepts an argument num,an integer.

The function/method patternPrint prints num lines in the following pattern.

For example, num=4, the pattern should be

1
11
111
1111

It compiles successfully but fails to print desirable result.

```
-----------------------------------------
void patternPrint(int num)
{
	int print=1,i,j;
	for(i=0;i<num;i++)
	{
		for(j=0;j<=i;j++);
		{
			printf("%d", print);
		}
		printf("\n");
	}
}
-----------------------------------------
```

solution:

```
void patternPrint(int num)
{
	int print=1,i,j;
	for(i=0;i<num;i++)
	{
		for(j=0;j<=i;j++)
		{
			printf("%d", print);
		}
		printf("\n");
	}
}
-----------------------------------------
```

Problem (16):

Lisa always forgets her birthday which is on the 5th of july. So,develop a function/method which will be helpful to remember her birthday.

It return an integer "1" if it is her birthday else return 0. It accepts two argument-month of her birthday, day of her birthday.

It compiles successfully but fails to return the desirable output.

```
-----------------------------------------
int checkBirthday(char *month,int day)
{
```

```
	if(strcmp(month,"July"))||(day==5))
		return 1;
	else
		return 0;
}
```
---
solution:

```
int checkBirthday(char *month,int day)
{
	if(strcmp(month,"July"))&&(day==5))
		return 1;
	else
		return 0;
}
```

Problem (17):

The function/method sameElementCount returns an integer representing the number of elements of the input list which are even numbers and equal to the elementto its right. For eg, if the input list is [4 [4 4 1 8 4 1 1 2 2] then the function should return the output 3 as it has three similar groups i.e., (4,4), (4,4),(2,2)

It compiles successfully but fails to return desirable output.

```
int sameElementCount(int size,int inputList)
{
	int i,count=0;
	for(i=0;i<size;i++)
	{
		if((inputList[i]%2==0)&&(inputList[i]==inputList[i++])
		{
			count++;
		}
	}
	return count;
}
```
---
solution:

```
int sameElementCount(int size,int *arr)
{
	int i,count=0;
	for(i=0;i<size-1;i++)
	{
		if((inputList[i]%2==0) && inputList[i]==inputList[i+1])
		{
			count++;
		}
	}
	return count;
}
```

Problem (18):
The function/method manchester: for each element in the input array arr, a counter is incremented if the bit arr[i] is same as arr[i-1]. Then the element of the list is 0, otherwise 1.

for the first bit in the input array, assume its previous bit to be 0. For example if arr is {0,1,0,0,1,1,1,0}, then it should print {0,1,1,0,1,0,0,1}

It accepts two arguments-size and arr-list of integers. Each element represents a bit 0 or 1.

It compiles successfully but fails to print the desirable result. Fix the code.

```
void manchester(int len,int *arr)
{

	int res[len];
	res[0]=arr[0];
	for(int i=1;i<len;i++)
	{
		res[i]=(arr[i]==arr[i-1]);
	}
	for(int i=0;i<len;i++)
	{
		printf("%d",res[i]);
	}
}
```

---

solution:

```
void manchester(int len,int *arr)
{

	int res[len];
	res[0]=arr[0];
	for(int i=1;i<len;i++)
	{
		res[i]=!(arr[i]==arr[i-1]);
	}
	for(int i=0;i<len;i++)
	{
		printf("%d",res[i]);
	}
}
```

---

problem (19):

The function/method patternPrint accepts an argument num,an integer.

The function/method patternPrint prints num lines in the following pattern.

For example, num=4, the pattern should be

```
11
1111
111111
11111111
```

It compiles successfully but fails to print desirable result.

```
void drawPrintPattern(int num)
{
	int i,j,print=1;
	for(i=1;i<=num;i++)
	{
		for(j=1;j<=2*i;j++);
		{
			printf("%d",print);
		}
		printf("\n");
	}
}
```
----------solution:

```
void drawPrintPattern(int num)
{
	int i,j,print=1;
	for(i=1;i<=num;i++)
	{
		for(j=1;j<=2*i;j++)
		{
			printf("%d",print);
		}
		printf("\n");
	}
}
```

Problem (20):

The function/method countOccurence return integer representing the count of occurence of given value in input list.

It accepts three arguments-len,value,arr list.

It compiles successfully but fails to return the desirable result.

```
int countOccurence(int len, int value,int *arr)
{
	int i=0,count=0;
	while(i<len)
	{
		if(arr[i]==value)
		{
			count+=1;
		}

	}
	return count;
}
```
----------solution:

```
int countOccurence(int len, int value,int *arr)
{
	int i=0,count=0;
	while(i<len)
	{
		if(arr[i]==value)
		{
```

```
                        count+=1;
                }
                i++;
        }
        return count;
}
```

Complete the code:

Problem (21):

You are given predefined structure Time containing hour,minute and second as members. A collection of functions/methods for performing some common operations on times is also available. You must use these functions/methods to calculate and return the difference.

It accepts two arguments time1 and time 2 representing two times and is supposed to return an integer represents the difference in the number of seconds.

You must complete the code so that it passes all the test cases.

Note:

The following structure is used to represent the time and is already implemented in the default code.

```
typedef struct
{
        int hour,
        int minute,
        int second;
}Time;
int Time_compareTo(const Time* tim1,const Time* time2)
{
        /* Return 1, if time1>time2
        Return -1 if time1<time2
        or, return 0, if time1==time2
        This can be called as

        *if time1 and time2 are two times then
        *Time_compareTo(time1,time2);*/
}

void Time_addSecond(Time * time)
{
        /* Add one second in the time;

}
```

```
int difference_in_times(Time *time1,Time * time2)
{

        //write code
```

```
}
```
---

solution:

```
#include<stdio.h>

typedef struct
{
        int hour;
        int minute;
        int second;
}Time;

int Time_compareTo(const Time*,const Time*);
int difference_in_times(Time*,Time*);
int t1s,t2s;
int main()
{
Time t1,t2;
t1.hour=11;t1.minute=43;t1.second=50;
t2.hour=9;t2.minute=27;t2.second=57;
t1s=t1.hour*60*60+t1.minute*60+t1.second;
t2s=t2.hour*60*60+t2.minute*60+t2.second;
printf("difference of times duration:%d",difference_in_times(&t1,&t2));
}

int Time_compareTo(const Time* time1,const Time* time2)
{
        if(t1s>t2s) return 1;
        else return -1;
        return 0;
}

int difference_in_times(Time* time1,Time* time2)
{
   if(Time_compareTo(time1,time2)==1) return t1s-t2s;
   else return t2s-t1s;
   return 0;
}
```

Problem (22):

The function findMaxElement(int *arr1,int len1,int *arr2,int len2) accepts twointeger arrays arr1,arr2 of length len1,len2 respectively.It is supposed to return the largest element in both the input arrays.

Another function sortArray(int *arr,int len) sorts the input array arr of length lenin ascending order and returns the sorted array.

Your task is to use sortArray(int *arr,int len) function and complete the codein findMaxElement(int *arr1,int len1,int *arr2,int len2) so that it passes all testcases"

```
int *sortArray(int len,int *arr)
{
```

```
	int i=0,j=0,temp=0;
	for(i=0;i<len;i++)
	{
		for(j=i+1;j<len;j++)
		{
			if(arr[i]>arr[j])
			{
				temp=arr[i];
				arr[i]=arr[j];
				arr[j]=temp;
			}
		}
	}
	return arr;
}

int findMaxElement(int len1,int *arr1,int len2,int *arr2)
{
	//write code here
}
```

Solution:

```
int *sortArray(int len,int *arr)
{
	int i=0,j=0,temp=0;
	for(i=0;i<len;i++)
	{
		for(j=i+1;j<len;j++)
		{
			if(arr[i]>arr[j])
			{
				temp=arr[i];
				arr[i]=arr[j];
				arr[j]=temp;
			}
		}
	}
	return arr;
}

int findMaxElement(int len1,int *arr1,int len2,int *arr2)
{
	int sort_array1[len1]=sortArray(int len1,int *arr1);
		int sort_array2[len2]=sortArray(int len2,int *arr2);
		if(sort_array1[len1-1]>sort_array2[len2-1])
		{
			printf("%d",sort_array1[len1-1]);
		}
		else
		{
			printf("%d",sort_array2[len2-1]);
		}

}
```

Problem (23):

The function allExponent(int base,int exponent)accepts two integers base and exponent as inputs.it is supposed to calculate and return of exponentation of base raised to power exponent for all input values.

The function positiveExponent(int base,int exponent) will return the exponentiation value if exponent value is positive. try to complete the code such that allExponent should return value if exponent value is negative

PROGRAM

```
 float allExponent( int base,int exponent )
{
float res=1;
if(exponent >=0)
{
Res = (float)positiveExponent(base,exponent)
}
Else
{
//write ur code here
}
return res;
}
```

---

solution:

```
 float allExponent( int base,int exponent )
{
float res=1;
if(exponent >=0)
{
res = (float)positiveExponent(base,exponent);
}
else
{
//write ur code here
res = 1/((float)positiveExponent(base,-(exponent)));
}
return res;
}
int positiveExponent(int base,int exponent)
{
   int sol=pow(base,exponent);
   return sol;
}
```

Problem (24):

You are given a predefined structure Point and related functions.
The function/method isTriangle which accepts three points P1,P2,P3 as inputs and checks whether the given three points form a triangle.
If they form a triangle the function return 1 else 0.

```
typedef struct point
{
```

```
int X;
int Y;
}Point;

double Point_calculateDistance(Point *point1,Point *point2)
{
	//returns the distance between pont 1, point2.  It can be called as Point_calculateDistance(P1,P2);
}
```

```
int isTriangle(Point *P1,Point *P2,Point *P3)
{
	//write code
}
```

---

solution:

```
int isTriangle(Point *P1,Point *P2,Point *P3)
{
	int a=P1.X*(P2.Y-P3.Y)+P2.X*(P3.Y-P1.Y)+P3.X*(P1.Y-P2.Y);

	if(a==0)
		return 0;
	else
		return 1;
}
```

Problem (25):

Charlie has a magic mirror. The mirror shows right rotated versions of a given word.
To generate different right-rotations of a word, write the word in a circle in clockwise order,
then start reading from any given character in clockwise order till you have covered all the characters.
For example: In the word "sample", if we start with 'p', we get the right rotated word as
"plesam". There are six such right rotations of "sample" including itself.

The inputs to the function isSameReflection consists of two strings, word1 and word2.
The function returns 1 if word1 and word2 are right rotations of the same word and -1 if they are not. Both word 1 and word2 will strictly contain characters between 'a'-'z' (lower case letters).

-->Useful commands:

strlen() is used to calculate the length of the string.
The statement -int len = strlen(str)
returns the lenght of the srting str

```
using namespace std;
int isSameReflection(char *word1,char *word2)
{
	//write code here

	int size1=strlen(word1);
	int size2=strlen(word2);
	char *temp;
	void *ptr;
```

```
	if(size1!=size2)
		return -1;
	temp[0]='';
	strcat(temp,word1);//concat empty string with word1(sample)
	strcat(temp,word1);//concat sample along with sample again(samplesample)
	//temp=samplesample  word2=zlesam

	ptr=strstr(temp,word2);//word2 is there in temp and return index of first char of word2 in temp, if word2 is not a substring it will return null.

	if(ptr!=null)
		return 1;
	else
		return -1;

}
```

----c++-----

```
int isSameReflection(String str1,String str2)
{
	if(str1.length()!=str2.length())
		return false;
	string temp=str1+str1;
	if(temp.find(str2)!=string::npos))
	{
		return 1;
	}
	else return -1;
}
```

Problem (26):

You are given a predefined structure Point and related functions.
The function/method isRightTriangle which accepts three points P1,P2,P3 as inputs and checks whether the given three points form a right angled triangle.
If they form a right triangle the function return 1 else 0.

```
typedef struct point
{
int X;
int Y;
}Point;

double Point_calculateDistance(Point *point1,Point *point2)
{
	//returns the distance between pont 1, point2.  It can be called as Point_calculateDistance(P1,P2);
}
```

---

solution:

```
int isRightTriangle(Point *P1,Point *P2,Point *P3)
{
	double d1,d2,d3;
	d1=Point_calculateDistance(P1,P2);
```

```
        d2=Point_calculateDistance(P2,P3);
        d3=Point_calculateDistance(P3,P1);

        if(pow(d1,2)+pow(d2,2)==pow(d3,2)||
          pow(d2,2)+pow(d3,2)==pow(d1,2)||
          pow(d3,2)+pow(d1,2)==pow(d2,2))
        {
                return 1;
        }
        else
                return 0;
}
```

Problem (27):

The function/method median accepts two arguments-size and inputList,an integer representing the length of a list and a list of integers respectively.

It is supposed to calculate and return an integer representing the median of elements in the input list. However the function/method median works only for odd-length lists because of incomplete code.

you must complete the code for even-length as well.

```
int quick_select(int*inputList,int start_index,int end_index,int median_order)
{
        //It calculate the median value
}
```

---

solution:

```
float median(int size,int *inputList)
{
        int start_index=0;
        int end_index=size-1;
        float res=-1;
        if(size%2!=0)
        {
                int med_1=size/2;
                int med_2=(size/2)+1;

                res=((float)quick_select(inputList,start_index,end_index,med_1));
        }
        else
        {
                //enter your code here


        res=((float)quick_select(inputList,start_index,end_index,med_1)+(float)quick_select(inputList,start_index,en
d_index,med_2))/2;
        }

        return res;
}
```

Compilation error:

Problem (28):

The function/method matrixSum returns an integer representing the sum of elements of the input matrix.
The function/method matrixSum accepts three arguments- rows, an integer representing the number of rows ,columns an integer representing number of columns, matrix is going to be the input matrix.
The function/method matrixSum has some compilation error. Rectify the error and get the desired output:

```
int matrixSum(int rows,int columns, int **matrix)
{
	int i,j,sum=0;
	for(i=0;i<rows;i++)
	{
		for(j=0;j<columns;j++)
		{
			sum+=matrix(i)(j);
		}
	}
	return sum;
}
```

---

solution:

```
int matrixSum(int rows,int columns, int **matrix)
{
	int i,j,sum=0;
	for(i=0;i<rows;i++)
	{
		for(j=0;j<columns;j++)
		{
			sum+=matrix[i][j];
		}
	}
	return sum;
}
```

Problem (29):
The function/method countElement return an integer representing the number of elements in the input list which are greater than twice the input number K.

The function/method accepts three arguments-size, an integer representing the size of input list, numK an integer representing the input number K and inputList, a list of integers representing the input list, respectively.

The function/method countElement compiles unsuccessfully due to systematical error. Fix the code so that it passes all test cases.

```
int countElement(int size,int num,int *inputList)
{
	int i,co-unt=0;
	for(i=0;i<size;i++)
	{
		if(inputList[i]>2numK)
```

```
                {
                        co-unt+=1;
                }
        }
        return co-unt;
}
```
---
solution:

```
int countElement(int size,int num,int *inputList)
{
        int i,count=0;
        for(i=0;i<size;i++)
        {
                if(inputList[i]>2*numK)
                {
                        count+=1;
                }
        }
        return count;
}
```

Problem (30):

The function/method deleteNonRepeat accpets two arguments-size, an integer representing the size of the list and inputList,representing the list of integers.

It removes the non-repeated integres from the inputList and prints the remaining repeated integers seperated by space from inputList.

If all the elements are unique then the function should not print anything.

For example, for the given inputList{2,3,2,2,5,6,6,7}, th expected output is 2 2 2 6 6 .

It compiles unsuccessfully due to syntactical error. Debug the code.

```
void deleteNonRepeat(int size,int *inputList)
{
        int count=0,i,j;
        int counter[size];
        for(i=0;i<size;i++)
        {
                counter[i]=1;
        }
        for(i=0;i<size;i++)
        {
                for(j=i+1;j<size;j++)
                {
                        if(inputList[i]==inputList[j])
                        {
                                counter[i]++;
                                counter[j]++;
                        }
                }
        }
        for(i=0;i<size;i++)
```

```
		if(counter[i]>1)
			count++;
	j=0;
	for(i=0;i<size;i++)
		if(counter[i]>1)
			printf("%d",inputList[]);
}
```
---

solution:

```
void deleteNonRepeat(int size,int *inputList)
{
	int count=0,i,j;
	int counter[size];
	for(i=0;i<size;i++)
	{
		counter[i]=1;
	}
	for(i=0;i<size;i++)
	{
		for(j=i+1;j<size;j++)
		{
			if(inputList[i]==inputList[j])
			{
				counter[i]++;
				counter[j]++;
			}
		}
	}
	for(i=0;i<size;i++)
		if(counter[i]>1)
			count++;
	j=0;
	for(i=0;i<size;i++)
		if(counter[i]>1)
			printf("%d",inputList[i]);
}
```

Problem (31):

The function/method multiplyNumber returns an integer representing the multiplicative product of the maximum two of three numbers. It accepts three integers- numA,numB,and NumC,representing the input numbers.

It compiles unsuccessfully due to syntactical error. Debug the code.

```
int multiplyNumber(int numA,int numB,int numC)
{
	int result,min,max,mid;
	max=(numA>numB)?numA>numC)?numA:numc):((numB>numC)?numB:numC);
	min=(numA<numB)?((numA<numC)?numA:numc):((numB<numC)?numB:numC);
	mid=(numA+numB+numC)-(min+max);
	result=(max*int mid);
	return result;
}
```
---

solution:

```
int multiplyNumber(int numA,int numB,int numC)
{
	int result,min,max,mid;
	max=(numA>numB)?((numA>numC)?numA:numc):((numB>numC)?numB:numC);
	min=(numA<numB)?((numA<numC)?numA:numc):((numB<numC)?numB:numC);
	mid=(numA+numB+numC)-(min+max);
	result=(max* mid);
	return result;
}
```
