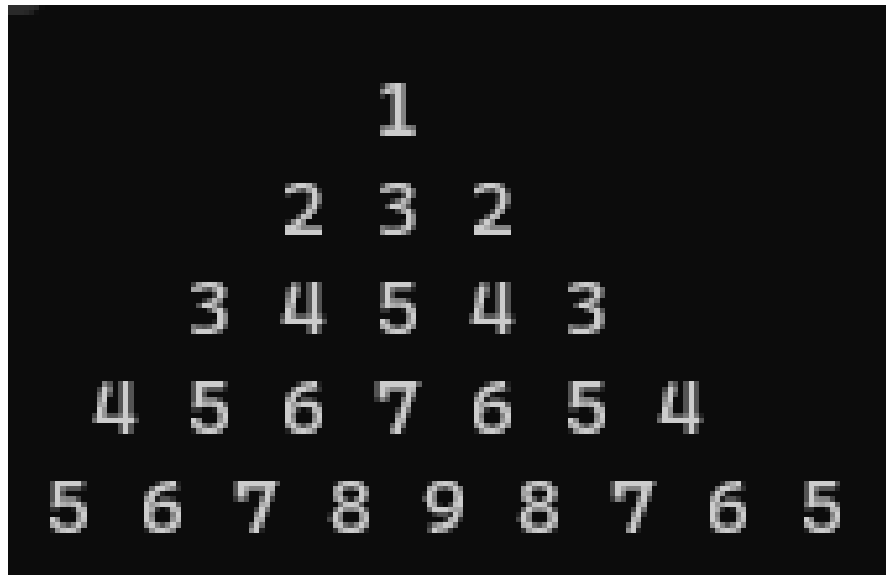


Loops

Pattern1 :



Solution:

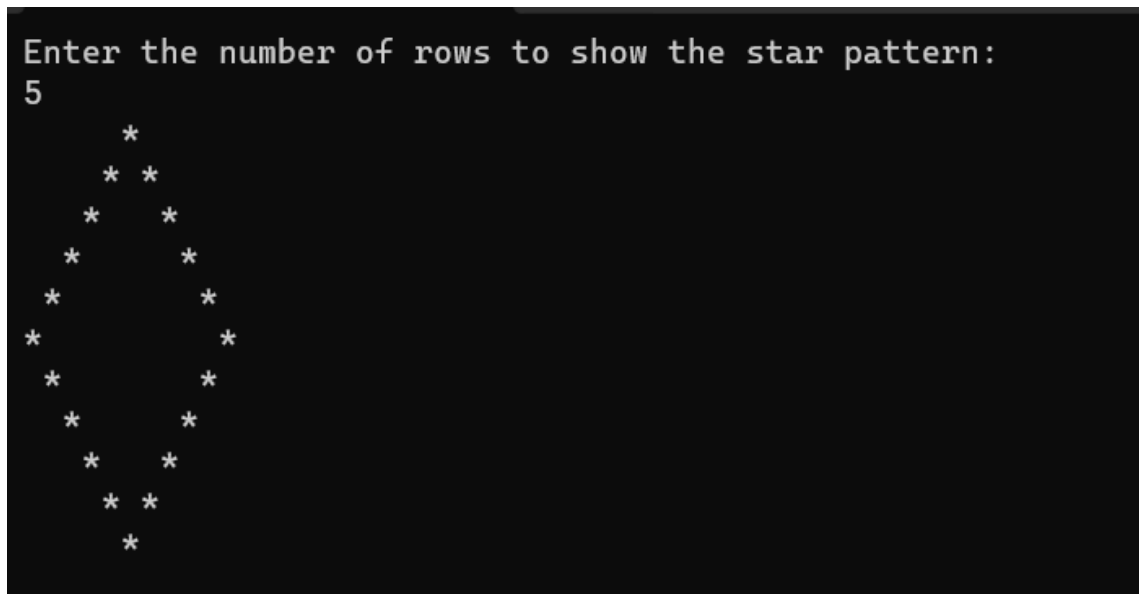
```
// ***** Pattern 1 *****
int i = 0, j = 0, range = 5, k = 0, count = 0, count1 = 0;
for (i = 1; i <= range; ++i)
{
    for (j = 1; j <= range - i; ++j)
    {
        cout << " ";
        ++count;
    }
    for (j = 1; j <= (range - 1) - i; ++j)
    {
        cout << " ";
    }
    while (k != 2 * i - 1)
```

```

    {
        if (count <= range - 1)
        {
            cout << i + k << " ";
            ++count;
        }
        else
        {
            ++count1;
            cout << i + k - 2 * count1 << " ";
        }
        ++k;
    }
    count1 = count = k = 0;
    cout << "\n";
}

```

Pattern 2



```

cout << "Enter the number of rows to show the star pattern: ";
int n, x, y, s = 1, k;
cin >> n;
for(x = 0; x <= n; x++)
{
    for(y = n; y > x; y--)

```

```

{
cout << " ";
}
cout << "**";
if (x > 0)
{
for(k = 1; k <= s; k++)
{
cout << " ";
}
s += 2;
cout << "**";
}
cout << "\n";
}
s -= 4;
for(x = 0; x <= n -1; x++)
{
for(y = 0; y <= x; y++)
{
cout << " ";
}
cout << "**";
for(k = 1; k <= s; k++)
{
cout << " ";
}
s -= 2;
if(x != n -1)
{
cout << "**";
}
//ending line after each row
cout << "\n";
}

```

Pattern 3

Output: ?

```

int main()
{
    int m = 7, n = m / 2 + 1;

```

```

char ch = '*';
for (int i = 0; i < n - 1; i++)
{
    for (int j = 0; j < i; j++)
    {
        cout << " ";
    }
    for (int j = 0; j < m; j++)
    {
        if (j == m - 1 || j == 0)
            cout << ch;
        else
            cout << " ";
    }
    m -= 2;
    cout << endl;
}

return 0;
}

```

Pattern 4

Output: ?

```

int main()
{
    int n;
    cout << "Enter a number: " << endl;
    cin >> n;
    int x = 2 * n - 1;
    int inner_space = 1;
    int k = n - 1;
    for (int i = 1; i <= n; i++)
    {
        if (i == 1) {

            for (int i = 1; i <= x; i++)
            {
                cout << "**";
            }
            cout << endl;

```

```

    }
    if (i > 1) {

        for (int j = 1; j <= k; j++) {
            cout << "**";
        }
        for (int j = 1; j <= inner_space; j++) {
            cout << " ";
        }

        for (int j = 1; j <= k; j++) {
            cout << "**";
        }
        inner_space += 2;
        --k;
        cout << endl;
    }

}
return 0;
}

```

Pattern 5: (Pascal Triangle)

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

Solution:

```
#include <iostream>
```

```
int main() {
    int n;
```

```

std::cout << "Enter the number of rows for Pascal's Triangle: ";
std::cin >> n;

for (int i = 0; i < n; i++) {
    int coefficient = 1;

    for (int j = 0; j <= i; j++) {
        if (j > 0) {
            coefficient = coefficient * (i - j + 1) / j;
        }
        std::cout << coefficient << " ";
    }
    std::cout << std::endl;
}

return 0;
}

```

Find HCF/GCD

```

#include <iostream>
using namespace std;

int main() {
    int n1, n2, hcf;
    cout << "Enter two numbers: ";
    cin >> n1 >> n2;

    // swapping variables n1 and n2 if n2 is greater than n1.
    if ( n2 > n1) {
        int temp = n2;
        n2 = n1;
        n1 = temp;
    }

    for (int i = 1; i <= n2; ++i) {
        if (n1 % i == 0 && n2 % i == 0) {
            hcf = i;
        }
    }
    cout << "HCF = " << hcf;
    return 0;
}

```

ARRAYS

Array 1:

Write a C++ program that takes an array of characters as input and reorders the elements in such a way that lowercase letters are moved to the beginning of the array while keeping the original order of other characters intact.

Solution:

```
#include <iostream>
using namespace std;

void processNum(char nums[], int n) {
    int i = -1, j = 0;
    for (;j < n; ++j)
    {
        if (nums[j] >= 'a' && nums[j] <= 'z')
        {
            int temp = nums[i + 1];
            nums[i + 1] = nums[j];
            nums[j] = temp;
            ++i;
        }
    }
}

int main() {
    char nums[] = { 'A','a', 'B', 'c', 'd', 'C', 'D', 'b'};

    int n = 8;

    cout << "Original Array: ";
    for (int i = 0; i < n; i++)
    {
        cout << nums[i] << " ";
    }
    processNum(nums, n);

    cout << "\nProcessed array : ";
```

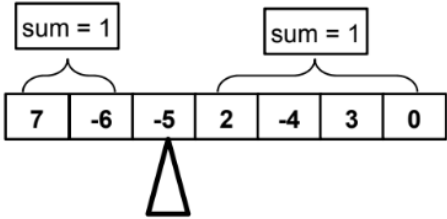
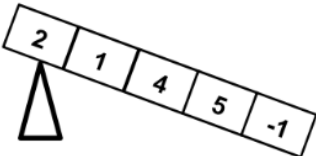
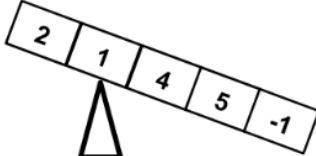
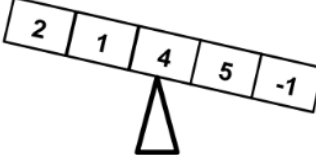
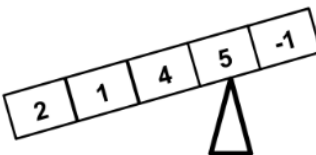
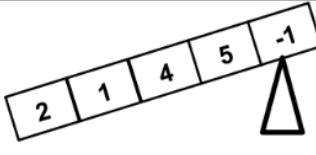
```

for (int i = 0; i < n; i++)
{
    cout << nums[i] << " ";
}
return 0;
}

```

Array 2:

Q4. [10] Write a C++ function called **findEquilibrium** that accepts an integer array A of size n and finds the *equilibrium index* of the array if it exists. An index k of an array is said to be an equilibrium index if the sum of all the elements on the left of k is equal to the sum of all the elements on the right of k . Precisely, k is an equilibrium index of A if and only if $\sum_{i=0}^{k-1} A[i] = \sum_{i=k+1}^{n-1} A[i]$. For clarity, consider this example below. If the index is not found, the function returns -1.

Equilibrium Index Found	Equilibrium Index not Found
	 [0] is not equilibrium index
	 [1] is not equilibrium index
	 [2] is not equilibrium index
	 [3] is not equilibrium index
	 [4] is not equilibrium index
findEquilibrium returns 2 since equilibrium found at index number 2.	findEquilibrium returns -1 since equilibrium not found at any index 0, 1, 2, 3 and 4.

Solution 1:

```

int main() {
    int str[] = { 7,-6,-5,2,-4,3,0};

    int length = sizeof(str)/sizeof(str[0]);
    for (int i = 0; i < length; i++)
    {
        int leftSum = 0, rightSum = 0;

```



```

    for (int j = 0; j < i; j++)
    {
        leftSum += str[j];
    }
    for (int j = i+1; j < length; j++)
    {
        rightSum += str[j];
    }
    cout << "str[i] = " << str[i] << ", left sum = " << leftSum << ", right sum = " << rightSum <<
endl;
}

return 0;
}

```

Solution 2:

```

#include <iostream>

int findEquilibrium(int A[], int n) {
    if (n == 0) {
        return -1; // Array is empty, so no equilibrium index.
    }

    long long prefixSum[n]; // Use a long long to avoid integer overflow.
    long long suffixSum[n];

    prefixSum[0] = A[0];
    for (int i = 1; i < n; i++) {
        prefixSum[i] = prefixSum[i - 1] + A[i];
    }

    suffixSum[n - 1] = A[n - 1];
    for (int i = n - 2; i >= 0; i--) {
        suffixSum[i] = suffixSum[i + 1] + A[i];
    }

    for (int i = 0; i < n; i++) {
        if (prefixSum[i] == suffixSum[i]) {
            return i; // Equilibrium index found.
        }
    }

    return -1; // No equilibrium index found.}

```

```

int main() {
    int A[] = { 7,-6,-5,2,-4,3,0};
    int n = sizeof(A) / sizeof(A[0]);
    int equilibriumIndex = findEquilibrium(A, n);

    if (equilibriumIndex != -1) {
        std::cout << "Equilibrium index is at position " << equilibriumIndex << std::endl;
    } else {
        std::cout << "No equilibrium index found." << std::endl;
    }

    return 0;
}

```

Array 3:

Write a C++ function which takes 1-D array of integers and its size as input and it removes all of the Armstrong numbers from this array. (May require shifting)

A positive integer is called an Armstrong number, if the sum of cubes of individual digits of number is equal to that number itself. For example

Number **153** is Armstrong because sum of cubes of individual digits $(1 * 1 * 1 + 5 * 5 * 5 + 3 * 3 * 3) = 153$
 Number **12** is not Armstrong because sum of cubes of individual digits $(1 * 1 * 1 + 2 * 2 * 2) = 9$

Solution:

```

#include <iostream>
using namespace std;
int main()
{
    int n = 12;
    int ans = 0;
    while (n != 0)
    {
        int x;
        x = (n % 10);
        cout << x;
        n /= 10;
        ans += x * x * x;
    }
    cout << endl << ans;
    if (ans == n)
        cout << "\nit is an armstrong!";
    else
        cout << "\nit is not an armstrong";
    return 0;
}

```

Array 4:

Write a program that takes a character array from the user as input and removes duplicate words from it. After removing the duplicate words, it should also sort all the remaining words.

You have to write two functions ***removeDuplicates()*** and ***sortWords()***.

- ***removeDuplicates()*** should remove all the duplicate words from the passed array,
- ***sortWords()*** should sort all the words

You need to take the input in *main()* and pass that character array to both of these two functions.

Example#1

Input:	<i>Hello there hello how are you YOU</i>
Output after removing duplicates:	<i>hello there how are you</i>
Output after sorting words:	<i>are hello how there you</i>

Example#2

Input:	<i>The more you practice the more you become better</i>
Output after removing duplicates:	<i>the more you practice become better</i>
Output after sorting words:	<i>become better more practice the you</i>

Note: You are not allowed to use any of the string library functions.

Array 5:

You are given two arrays *nums1* and *nums2* consisting of positive integers.

You have to replace all the 0's in both arrays with strictly positive integers such that the sum of elements of both arrays becomes equal.

Return the minimum equal sum you can obtain, or -1 if it is impossible.

Examples:

Input: *nums1* = [3,2,0,1,0], *nums2* = [6,5,0]

Output: 12

Input: *nums1* = [2,0,2,0], *nums2* = [1,4]

Output: -1

Solution:

```
int s1=0, s2=0;
int c1=0, c2=0;

for(int val:nums1){
    s1 += val;
    c1 += (val==0);
}
for(int val:nums2){
```

```

    s2 += val;
    c2 += (val==0);
}

if(c1==0 && s1<s2+c2) return -1;
if(c2==0 && s2<s1+c1) return -1;

output = max(s1+c1,s2+c2);

```

Array 4:

Write a program that takes N integer arrays (*of varying sizes*) as input. You have to ensure that user enter these arrays in ascending order, if user enters incorrectly display a prompt to read input in correct format. Write a C++ program to produce an array that merges elements of all arrays in descending order, but it also needs to remove duplicates.

Example:

```

Array A_1: {1, 3, 5, 6}
Array A_2: {1, 2, 4, 8}
Array A_3: {5, 6, 7, 8}
Array A_4: {23, 24, 94, 108}
Array A_5: {1, 2, 2, 23, 24, 67, 1234}

Merged array: {1234, 108, 94, 67, 24, 23, 8, 7, 6, 5, 4, 3, 2, 1}

```

Note: Final array should be created/merged in sorted order.

Array 5:

There are various problems with input validation, especially when reading numeric data types using `cin`. Although, `cin.fail()` gives us some sort of solution but still it cannot handle all scenarios, for example, 123abc is considered a valid input because 123 is read where as it an alphanumeric input.

You are required to write a program that read input as a character array/string and validate it. Table 1 shows valid and invalid input for different numeric data types.

Table 1: Numeric data types

Date type	Valid input	Invalid input
Integers	Only digits 1 2132 -3121	123abc 123.123 aqwe121 _12 +121
Float	12.23f -12.23f	123abc 123.123
Double	12.23 -12.23	123abc 123.123asdasda 12.2f

Functions:

Function 1:

```
# include <iostream>
using namespace std;
f_1();
f_2(int,int);
f_3();
f_4(int);

main()
{
    int x=6;           //Line 1
    f_1();             //Line 2
    f_3();             //Line 3
    f_4(x);            //Line 4
}
f_1()
{
    int x=7, int y=5;
    f_2(x,y);
}
f_2(int x, int y)
{
    cout<<"X is "<<x<<" and Y is "<<y<<" in function f_1()."<<endl;
    f_3();
}
f_3()
{
    static int x=5;
    cout<<"A static X found in one of the functions with the value
"<<x<<"."<<endl;
    x++;
    f_4(x);
}
f_4(int x)
{
    cout<<"Value of X in f_4() is "<<x<<"."<<endl;
}
}
```

```
X is 7 and Y is 5 in function f_2().
A static X found in one of the functions with the value 5.
Value of X in f_4() is 6.
A static X found in one of the functions with the value 6.
Value of X in f_4() is 7.
Value of X in f_4() is 6.
```

Function 2:

```
int mystery(int x, int n)
{
    int val;
    val = 1;
    if (n >= 0)
    {
        if (n % 3 > 1)
            val = val * x;
        else
            val = val * 2;
    }
    return val;
}

void main()
{
    cout << "The mysterious value is: " << mystery(10, 3);
}
```

Output Questions:

1:

```
int main(){

    int suite = 5 ;
    switch ( suite ) ;
    {
        case 0+5 ;
        cout<< "\nClub"  ;
        case 1+5 ;
        cout<< "\nDiamond"  ;
    }
    return 0;
}
```

2:

```
int x=10;

void main()
{
    int x=20;
    cout<<::x++;
    cout<<x+::x;
}
```
