**Must do Pattern Problems before starting DSA**

[Strivers A2Z-DSA Course | DSA Playlist | Placements - YouTube](https://www.youtube.com/playlist?list=PLgUwDviBIf0oF6QL8m22w1hIDC1vJ_BHz)

**Approach:**

There are 4 general rules for solving a pattern-based question:

1. We always use nested loops for printing the patterns. For the outer loop, we count the number of lines/rows and loop for them.
2. Next, for the inner loop, we focus on the number of columns and somehow connect them to the rows by forming a logic such that for each row we get the required number of columns to be printed.
3. We print the ‘\*’ inside the inner loop.
4. Observe symmetry in the pattern or check if a pattern is a combination of two or more similar patterns.

In this particular problem, we run the outer loop for N times since we have N rows to be printed, the inner loop also runs for N times as we have to print N stars in each row. This way we get a rectangular star pattern (square) with an equal number of rows and columns.

**Pattern-1: Rectangular Star Pattern**

190

**Problem Statement:** Given an integer **N,**print the following pattern.



**Examples:**

**Example 1:**

**Input:** N = 3

**Output:**

**\* \* \***

**\* \* \***

**\* \* \***

**Example 2:**

**Input:** N = 6

**Output**:

**\* \* \* \* \* \***

**\* \* \* \* \* \***

**\* \* \* \* \* \***

**\* \* \* \* \* \***

**\* \* \* \* \* \***

**\* \* \* \* \* \***

**Solution**

Code:

#include <bits/stdc++.h>

using namespace std;

void pattern1(int N)

{

// This is the outer loop which will loop for the rows.

for (int i = 0; i < N; i++)

{

// This is the inner loop which here, loops for the columns

// as we have to print a rectangular pattern.

for (int j = 0; j < N; j++)

{

cout << "\* ";

}

// As soon as N stars are printed, we move to the

// next row and give a line break otherwise all stars

// would get printed in 1 line.

cout << endl;

}

}

int main()

{

// Here, we have taken the value of N as 5.

// We can also take input from the user.

int N = 5;

pattern1(N);

return 0;

}

**pattern-2: Right-Angled Triangle Pattern**

**Problem Statement:** Given an integer **N,**print the following pattern :



Here, N = 5.

**Examples**:

**Input Format:** N = 3

**Result:**

\*

\* \*

\* \* \*

**Input Format:** N = 6

**Result:**

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

**Solution :**

#include <bits/stdc++.h>

using namespace std;

void pattern2(int N)

{

// This is the outer loop which will loop for the rows.

for (int i = 0; i < N; i++)

{

// This is the inner loop which loops for the columns

// no. of columns = row number for each line here.

for (int j = 0; j <=i; j++)

{

cout << "\* ";

}

// As soon as stars for each iteration are printed, we move to the

// next row and give a line break otherwise all stars

// would get printed in 1 line.

cout << endl;

}

}

int main()

{

// Here, we have taken the value of N as 5.

// We can also take input from the user.

int N = 5;

pattern2(N);

return 0;

}

**Pattern - 10: Half Diamond Star Pattern**

**Problem Statement:** Given an integer **N,**print the following pattern :



Here, N = 5.

**Examples**:

**Input Format**: N = 3

**Result**:

\*

\*\*

\*\*\*

\*\*

\*

**Input Format**: N = 6

**Result**:

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

**Solution:**

#include <bits/stdc++.h>

using namespace std;

void pattern10(int N)

{

// Outer loop for number of rows.

for(int i=1;i<=2\*N-1;i++){

// stars would be equal to the row no. uptill first half

int stars = I;

// for the second half of the rotated triangle.

if(i>N) stars = 2\*N-I;

// for printing the stars in each row.

for(int j=1;j<=stars;j++){

cout<<"\*";

}

// As soon as the stars for each iteration are printed, we move to the

// next row and give a line break otherwise all stars

// would get printed in 1 line.

cout<<endl;

}

}

int main()

{

// Here, we have taken the value of N as 5.

// We can also take input from the user.

int N = 5;

pattern10(N);

return 0;

}

**Pattern - 11: Binary Number Triangle Pattern**

**Problem Statement:** Given an integer **N,**print the following pattern :



Here, N = 5.

**Examples**:

**Input Format**: N = 3

**Result**:

1

01

101

**Input Format**: N = 6

**Result**:

1

01

101

0101

10101

010101

**Solution**

#include <bits/stdc++.h>

using namespace std;

void pattern11(int N)

{

// First row starts by printing a single 1.

int start =1;

// Outer loop for the no. of rows

for(int i=0;i<N;i++){

// if the row index is even then 1 is printed first

// in that row.

if(i%2 ==0) start = 1;

// if odd, then the first 0 will be printed in that row.

else start = 0;

// We alternatively print 1's and 0's in each row by using

// the inner for loop.

for(int j=0;j<=i;j++){

cout<<start;

start = 1-start;

}

// As soon as the numbers for each iteration are printed, we move to the

// next row and give a line break otherwise all numbers

// would get printed in 1 line.

cout<<endl;

}

}

int main()

{

// Here, we have taken the value of N as 5.

// We can also take input from the user.

int N = 5;

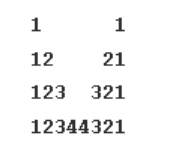
pattern11(N);

return 0;

}

**Pattern - 12: Number Crown Pattern**

**Problem Statement:** Given an integer **N,**print the following pattern :



Here, N = 5.

**Examples**:

**Input Format**: N = 3

**Result**:

1 1

12 21

123321

**Input Format**: N = 6

**Result**:

1 1

12 21

12 321

1234 4321

12345 54321

123456654321

**Solution**

cout << "Enter the number";

int n;

cin >> n;

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

// print numbers

for(int j = 1 ; j <= i; j++){

cout << j;

}

// print numbers of spaces

for(int j = 1; j <= 2\*(n-i) ; j++){

cout << " ";

}

// print numbers

for(int j = i ; j>=1 ; j-- ){

cout << j;

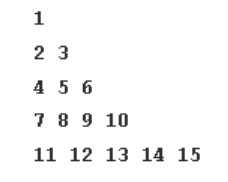
}

cout << endl;

}

**Pattern - 13: Increasing Number Triangle Pattern**

**Problem Statement:** Given an integer **N,**print the following pattern :



Here, N = 5.

**Examples**:

**Input Format**: N = 3

**Result**:

1

2 3

4 5 6

**Input Format**: N = 6

**Result**:

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

16 17 18 19 20 21

**Solution**

cout << "Enter any number";

    int n;

    cin  >> n;

    int count = 1;

    for(int i = 1 ; i<=n ; i++){

        for(int j = 1 ; j<=i;  j++){

            cout << count ;

            count++;

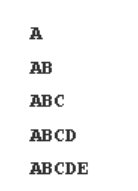
        }

        cout << endl;

    }

**Pattern-14: Increasing Letter Triangle Pattern**

**Problem Statement:** Given an integer **N,**print the following pattern :



Here, N = 5.

**Examples**:

**Input Format**: N = 3

**Result**:

A

A B

A B C

**Input Format**: N = 6

**Result**:

A

A B

A B C

A B C D

A B C D E

A B C D E F

**Solution**

cout << "Enter any number";

    int n ;

    cin >> n;

    for(int i =0 ; i<n;i++){

        for(char ch = 'A'; ch <='A'+i ; ch++){

            cout << ch << " ";

        }

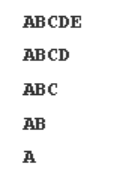
        cout << endl;

    }

**Pattern-15: Reverse Letter Triangle Pattern**

00

**Problem Statement:** Given an integer **N,**print the following pattern :



Here, N = 5.

**Examples**:

**Input Format**: N = 3

**Result**:

A B C

A B

A

**Input Format**: N = 6

**Result**:

A B C D E F

A B C D E

A B C D

A B C

A B

A

**Solution**

  // cout << "Enter any number";

    // int n;

    // cin >> n;

    // for(int i = 0; i<n ;i++){

    //     for(char ch='A';ch < 'A' + (n-i);ch++){

    //         cout << ch;

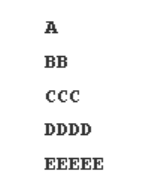
    //     }

    //     cout << endl;

    // }

**Pattern - 16: Alpha-Ramp Pattern**

**Problem Statement:** Given an integer **N,**print the following pattern :



Here, N = 5.

**Examples**:

**Input Format**: N = 3

**Result**:

A

B B

C C C

**Input Format**: N = 6

**Result**:

A

B B

C C C

D D D D

E E E E E

F F F F F F

**Solution**

cout << "Enter any number";

    int n;

    cin >> n;

    for(char ch ='A'; ch< 'A'+ n ; ch++){

        for(int j = 'A';j<=ch ;j++){

            cout << ch;

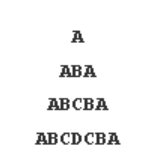
        }

        cout << endl;

    }

**Pattern - 17: Alpha-Hill Pattern**

**Problem Statement:** Given an integer **N,**print the following pattern :



Here, N = 4.

**Examples**:

**Input Format**: N = 3

**Result**:

A

ABA

ABCBA

**Input Format**: N = 6

**Result**:

A

ABA

ABCBA

ABCDCBA

ABCDEDCBA

ABCDEFEDCBA

**Solution**

 cout << "Enter any number";

    int n;

    cin >> n;

    for(int i =0; i<n;i++){

        for(int j=0 ; j<=n-i-1;j++){

            cout << " ";

        }

        char ch = 'A';

        int breakpoint = (2\*i+1) /2;

        for(int j = 1 ; j<=2\*i+1; j++){

            cout << ch;

            if(j<=breakpoint) ch++;

            else ch--;

        }

        cout << endl;

    }

}

**Pattern-18: Alpha-Triangle Pattern**

10

**Problem Statement:** Given an integer **N,**print the following pattern :



Here, N = 5.

**Examples**:

**Input Format**: N = 3

**Result**:

C

B C

A B C

**Input Format**: N = 6

**Result**:

F

E F

D E F

C D E F

B C D E F

A B C D E F

**Solution**

cout << "Enter sny number";

    int n;

    cin >> n;

    for(int i = 0 ; i<n ; i++){

        char ch = 'A';

        ch = ch+n-i-1;

        for(int j = 0 ; j<=i ; j++ ){

            cout << ch;

            ch++;

        }

        cout << endl;

    }

**Standard Template library(STL)**

STL is one of the unique abilities of C++ which makes it stand out from every other programming language. STL stands for standard template library which contains a lot of pre-defined templates in terms of containers and classes which makes it very easy for developers or programmers to implement different data structures easily without having to write complete code and worry about space-time complexities.

If you dive a little deeper into STL, you will have to understand everything about templates and how they work, which is one of the most power full tools when it comes to C++ programming language.

However, in this tutorial we will stick to some of the most popular STL containers and algorithms, and its useful functions which is used by programmers very frequently in day to day programming.

* [unordered\_set in C++ STL](https://takeuforward.org/c/unordered-set-in-c-stl/)
* [Vector in C++ STL](https://takeuforward.org/c/vector-in-c-stl/)
* [Set in C++ STL](https://takeuforward.org/c/set-in-c-stl/)
* [unordered\_multiset in C++ STL](https://takeuforward.org/c/c-stl/unordered-multiset-in-c-stl/)
* [multiset in C++ STL](https://takeuforward.org/c/c-stl/multiset-in-c-stl/)
* [unordered\_map in C++ STL](https://takeuforward.org/c/c-stl/unordered_map-in-c-stl/)
* [map in C++ STL](https://takeuforward.org/c/c-stl/map-in-c-stl/)
* [unordered\_multimap in C++ STL](https://takeuforward.org/c/c-stl/unordered_multimap-in-c-stl/)
* [queue in C++ STL](https://takeuforward.org/c/c-stl/queue-in-c-stl/)
* [stack in C++ STL](https://takeuforward.org/c/c-stl/stack-in-c-stl/)
* [deque in C++ STL](https://takeuforward.org/c/c-stl/deque-in-c-stl/)
* [priority\_queue in C++ STL](https://takeuforward.org/c/c-stl/priority_queue-in-c-stl/)
* [multimap in C++ STL](https://takeuforward.org/c/c-stl/multimap-in-c-stl/)
* [list in C++ STL](https://takeuforward.org/c/c-stl/list-in-c-stl/)
* [next\_permutation in STL](https://takeuforward.org/c/c-stl/next_permutation-in-c-stl/)
* [\_\_builtin\_popcount() in STL](https://takeuforward.org/c/c-stl/builtin-popcount-in-stl-cpp-count-set-bits/)
* [sort() in C++ STL](https://takeuforward.org/c/c-stl/sort-in-c-stl/)
* [min\_element() in C++ STL](https://takeuforward.org/c/c-stl/min-element-in-c-stl/)
* [max\_element() in C++ STL](https://takeuforward.org/c/c-stl/max-element-in-c-stl/)

1. **Pair**

Is a simple container defined in the **<utility>** header of the C++ Standard Library. It allows you to store a pair of values, which can be of different types.

e.g.

#include<bits/stdc++.h>

Using namespace std;

int main() {

pair<int, double> myPair = {5, 3.14};

cout << "First element: " << myPair.first << std::endl;

cout << "Second element: " << myPair.second << std::endl;

return 0;

}

**Nested pair property**

Declare pair inside pair and accessing it

**pair<int,pair<int,int>> p1 = {1,{2,3}};**

**cout << p1.first <<p1.second.first  << p1.second.second;**

**Pair type Array**

Declare pair type array and accessing the elements

pair <int,int> arr[]={{1,2},{3,4}}; cout << arr[1].second;

**Vector**

Vectors in STL are basically dynamic arrays that have the ability to change size whenever elements are added or deleted from them. Vector elements can be easily accessed and traversed using iterators. A vector stores elements in contiguous memory locations.

Syntax:

vector<object\_type> variable\_name;

Example:

vector<int> v1;

vector<char> v2;

vector<string> v3;

Most used functions in Vector:

begin() - it returns an iterator pointing to the first element of the vector.

auto iterator = itr;

itr = v1.begin();

end() - it returns an iterator pointing to the element theoretically after the last element of the vector.

auto iterator = itr;

itr = v1.end();

push\_back() - it accepts a parameter and insert the element passed in the parameter in the vectors, the element is inserted at the end.

vector<int> v1;

v1.push\_back(1);

v1.push\_back(2);

insert() - it is used to insert an element at a specified position.

auto it= v1.begin();

v1.insert(it,5); //inserting 5 at the beginning

erase() - it is used to delete a specific element

vector<int> v1;

auto it= v1.begin();

v1.erase(it);// erasing the first element

pop\_back() - it deletes the last element and returns it to the calling function.

v1.pop\_back();

front() - it returns a reference to the first element of the vector.

v1.front();

back() - it returns a reference to the last element of the vector.

v1.back();

clear() - deletes all the elements from the vector.

v1.clear();

empty() - to check if the vector is empty or not.

v1.empty();

size() - returns the size of the vector

v1.size();

**List**

list<int> myList = {1, 2, 3, 4, 5};

**Common Operations:**

1. **Insertion and Deletion**:
   * **push\_back(value)**: Adds an element to the end.
   * **push\_front(value)**: Adds an element to the beginning.
   * **insert(iterator, value)**: Inserts an element before the specified iterator position.
   * **pop\_back()**: Removes the last element.
   * **pop\_front()**: Removes the first element.
   * **erase(iterator)**: Removes the element at the specified iterator position or a range of elements.
   * **clear()**: Removes all elements.
2. **Traversal**:
   * **begin()**, **end()**: Return iterators to the beginning and end of the list, respectively.
   * **rbegin()**, **rend()**: Return reverse iterators to the beginning and end of the list.
3. **Modifiers**:
   * **resize(new\_size)**: Resizes the list to contain the specified number of elements.
   * **splice(position, other)**: Moves elements from another list into this list.
   * **swap(other)**: Swaps the contents of two lists.
4. **Capacity**:
   * **size()**: Returns the number of elements in the list.
   * **empty()**: Checks if the list is empty.

**Stack**

stack<int> myStack;

**Common Operations:**

1. **Push**:
   * **push(value)**: Adds an element to the top of the stack.
2. **Pop**:
   * **pop()**: Removes the top element from the stack.
3. **Top**:
   * **top()**: Returns a reference to the top element of the stack without removing it.
4. **Size and Empty Check**:
   * **size()**: Returns the number of elements in the stack.
   * **empty()**: Checks if the stack is empty.