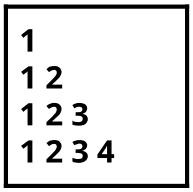
Patterns

Introduction

Patterns are a good application of loops. These will help you get yourself a better clarity over implementing loops.

Suppose, we want to print the following pattern:



Solution:

- ➔ Here, you can see that we have 4 rows.
- ➔ Now, moving to find the generic number of columns, it's clear that the i-th row (where i = 1, 2, ...) has i columns. Hence, the number of columns for i-th row is i.
- ➔ Visually, it's also clear that we want to print numbers starting from 1 to the row-th number. Like for the first row, we print numbers starting from 1 and ending at 1, for second row numbers are starting from 1 and ending at 2 and so on...

The above approach can be generalised in the following way:

- We start to figure out the number of rows that our pattern requires.
- Now, we should know how many columns do we have to print in the generic i-th row.

- Once, we have figured out the number of rows and columns, then we should focus on what to print.

There are two popular types of patterns-related questions that are usually posed:

- Square Pattern
- Triangular Pattern

Coding a pattern:

Now, starting to code this pattern. It's clear that we have to start with a loop for running over the row. So let's first begin with writing the basic structure and the loop for the row:

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

int main() {

        int n = 4;    //Number of rows taken input

        int i = 1;    // for iterating over rows starting from the first row

        while(i < n) {

                i++;      // for iterating over each row
        }
}
```

Here, we have iterated over each row.

Now for printing the above pattern, we also need to iterate over columns and that too, on each row.

For this, we will run another loop inside the while() loop. Let's see that also...

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

int main() {

        int n = 4;                      //Number of rows taken input

        int i = 1;                      // for iterating over rows starting from the first row

        while(i < n) {

                int j = 1;              // for iterating over columns
                while(j <= i) {         // since for each column row, we will be iterating
                                        //  over each column
                        cout << j;
                        j = j + 1;      // as we have discussed above that i-th row will
                                        // have i columns
                }
                cout << endl;
                i++;                    // for iterating over each row
        }
}
```

This will result in the above pattern that we discussed. These types of patterns are known as **triangular patterns** as the shape resembles a triangle.

Similarly, there are other patterns too like:

```
11111
11111
11111
11111
11111
```

This type of pattern is **square Pattern.** Try coding it out yourself...

Once, you have tried it, you can check your solution with the one given below:

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

int main() {

        int n = 5;

        int i = 0;
        while(i <= n) {

                int j = 1;
                while(j <= n) {
                        cout << 1;
                        j = j + 1;
                }
                cout << endl;
                i = i + 1;
        }
}
```

Like these integral valued patterns there are character valued patterns too. The only difference is that here we are printing the character values.

For example:

Given below are three patterns of the characters. They are very similar to what we discussed above.

******	abcde	A
******	abcde	AB
******	abcde	ABC
******	abcde	ABCD
******	abcde	ABCDE

Practice Examples:

Print the following patterns:

1)

```
55555
45555
34555
23455
12345
```

2)

```
ABCDE
ABCD
ABC
AB
A
```

3)

```
12344321
123**321
12****21
1******1
```

Try these yourselves and in case, you find yourself stuck then following is the approach provided...

Solution 1

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

int main() {

   int i = 5, j, k;
   while(i >= 1) {
         k = i;
         j = 1;
         while(j <= 5) {
                  if(k <= 5) {
                        cout << k;
                  }
                  else {
                        cout << 5;
                  }
                  k = k + 1;
                  j = j + 1;
         }
         cout << endl;
         i = i - 1;
    }
}
```

Solution 2

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

int main() {

         int i = 1, j;
         while(i <= 5) {
                  j = 5 - i + 1;
                  int k = 1;
                  while(k <= j) {
                           cout<<(char)(64 + k);
                           k = k + 1;
                  }
                  cout << endl;
                  i = i + 1;
         }
}
```

Solution 3

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

int main() {

        int i = 4, j;
        while(i >= 1) {

                j = 1;
                while(j <= 4) {
                        if(j <= i) {
                                cout << j;
                        }
                        else {
                                cout << "*";
                        }
                        j = j + 1;
                }

                while(j >= 1) {
                        if(j <= i) {
                                cout << j;
                        }
                        else {
                                cout << "*";
                        }
                        j = j - 1;
                }
                i = i - 1;
                cout << endl;
        }
}
```

Advanced patterns

Here, we are gonna study the patterns that include a form of mirror images at some part of the program. We'll be solving these like we did in the "Patterns 1" portion by following the same three steps of identifying the number of rows, then identifying the columns and finally what to print.

So directly moving on to programming questions:

Example 1: Print the following pattern:

```
 1  2  3  4 5
16           6
15           7
14           8
13 12 11 10 9
```

Try this out, in case you are stuck, just checkout the code below for the same.

Note: The pattern may not be totally similar, meaning it may be a bit shifted due to the difference in the number of digits of a 2-digit number and a 1-digit number.

Solution:

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

int main() {

   int i = 1, k = 6, l = 13, m = 16;
        while(i <= 5)
        {
                int j = 1;
                while(j <= 5)
                {
                        if(i == 1)
                                cout << j <<" ";
                         else if(j == 5)
                                cout << k++ << " ";
                        else if(i == 5)
```

```
                cout << i-- << " ";
            else if(j == 1)
                cout << m-- << " ";
            else
                cout<< "  ";
            j++;
        }
        cout << endl;
        i++;
    }
}
```

Example 2: Print the following pattern:

```
    1
   123
  12345
 1234567
123456789
 1234567
  12345
   123
    1
```

Try this out, in case you are stuck, just checkout the code below for the same.

Hint: Here you can see that there are generally two different patterns that you have to make: one spreading in the downwards directions and other one in continuation, shrinking towards the bottom.

Approach: Let's first try to make the downwards growing pattern i.e., upto row number 5. It is clearly visible that each row has 2n-1 numbers (where n = 1, 2, 3, 4, 5). Now, as we know the number of rows, columns in each row and the general formula that our pattern follows, it can be easily coded.

Moving on to the lower triangle of the pattern, if we start counting these rows in reverse order starting from number 4 down to number 1, each row contains 2n-1 numbers over here also, and can be similarly done using the upper triangle pattern but in reverse order.

Solution:

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

int main() {
   int i = 1;
   while (i <= 5)                  // Starting our first part of the pattern.
   {
         int j = i;
         while (j < 5)
         {
               cout<<" ";
               j++;
         }
         int k = 1;
         while(k < 2*i)            // upto k = 2n-1
         {
               cout << k;
               k++;
         }
         i++;
         cout << endl;
   }

   i = 4;                          // Starting our second pattern in reverse order.
   while (i > 0)
   {
         int j = 5;
         while (j > i)
         {
               cout << " ";
               j--;
         }
         int k = 1;
         while (k < 2*i)
         {
                  cout << k;
                  k++;
         }
         cout << endl;
         i--;
   }
```

```
}
```

You have already practised most of these questions earlier. So now directly moving on to some of the practice questions.

PRACTICE: Print the following programs

```
1    1
 2  2
  3 3
   4
  3 3
 2  2
1    1
```

Solutions:

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

int main()
{
   int i = 1, k=1;
   int m[7][7]={0};       // Initialises all the elements of the array equal to 0, you will
                          //   study about arrays in your further sessions.
   while (i <= 7)
   {
         int j = 1;
         while (j <= 7)
         {
               if (j == i || 8-i == j)
                     m[i-1][j-1]=k;
               j++;
         }
         if (i < 4)
               k++;
         else
               --k;
         i++;
   }

   i = 0;
   while (i < 7)
```

```
  {
        int j = 0;

        while (j < 7)
        {
                if(m[i][j]==0)
                        cout << " ";
                else
                        cout << m[i][j];
                j++;
        }
        i++;
        cout << endl;
  }
}
```

For more questions, visit the link:

http://cbasicprogram.blogspot.com/2012/04/number-patterns.html