List of loop programming exercises

1. Write a C program to print all natural numbers from 1 to n. - using while loop

   There are various ways to print n numbers. For this post I am concentrating on for loop to print natural numbers.

   Step by step descriptive logic to print natural numbers from 1 to n.

   1. Input upper limit to print natural number from user. Store it in some variable say N.
   2. Run a for loop from 1 to N with 1 increment. The loop structure should be like for(i=1; i<=N; i++). At this point you might be thinking of various things such as.

      Why starting from 1? Because we need to print natural numbers from 1.

      Why going till N? Because we need to print natural numbers up to N.

      Why increment loop counter by 1? Because difference between two natural numbers is 1. Therefore if n is one natural number then next natural number is given by n+1.

   3. Inside the loop body print the value of i. You might think, why print value of i inside loop? Because we need to print natural numbers from 1 to N and from loop structure it is clear that i will iterate from 1 to N. So to print from 1 to N print the value of i.

2. Write a C program to print all natural numbers in reverse (from n to 1). - using while loop

   Logic to print natural numbers in reverse is almost similar to printing natural numbers from 1 to n.

   Step by step descriptive logic to print natural numbers in reverse.

   1. Input start limit from user. Store it in some variable say start.
   2. Run a loop from start to 1 and decrement 1 in each iteration. The loop structure should look like for(i=start; i>=1; i--).

      Let me first answer few question popping in your head right now.

      Why we need to initialize loop from start? Because the first value we need to print is start.

      Why use i>=1 why not i<=1? Because the loop we are constructing is in decrementing order with numbers to print must be greater than or equal to 1.

      Why decrement the value of i instead of increment? Because we are running downwards (reverse), from start to 1.

   3. Inside the loop body print the value of i.

3. Write a C program to print all alphabets and their code numbers from a to z. - using while loop

   Printing alphabets in C, is little trick. If you are good at basic data types and literals then this is an easy drill for you.

   Internally C represent every character using ASCII character code. *ASCII* is a fixed integer value for each global printable or non-printable characters.

   For example - ASCII value of a=97, b=98, A=65 etc. Therefore, you can treat characters in C as integer and can perform all basic arithmetic operations on character.

   Step by step descriptive logic to print alphabets.

   1. Declare a character variable, say ch.
   2. Initialize loop counter variable from ch = 'a', that goes till ch <= 'z', increment the loop by 1 in each iteration. The loop structure should look like for(ch='a'; ch<='z'; ch++).
   3. Inside the loop body print the value of ch.

4. Write a C program to print all even numbers between 1 to 100. - using while loop

   First let us talk about the easiest way to print even numbers. If I ask you to list all even numbers from 1 to 100 what will be your immediate step? You will probably start from 1 check if its even number then add it to the even list otherwise not.

   Step by step descriptive logic to print all even number between 1 to n using if condition.

   1. Input upper limit to the even numbers from user. Store it in some variable say N.
   2. Run a loop from 1, that runs till N, increment the loop counter by 1 in each iteration. The loop structure should look like for(i=1; i<=N; i++).
   3. Inside the loop body check even/odd condition. If the current number i is divisible by 2 then i is even. Means if(i % 2 == 0), then print the value of i.

5. Write a C program to print all odd number between 1 to 100.

   Logic to print odd numbers is similar to logic to print even numbers.

   Step by step descriptive logic to print odd numbers from 1 to n.

   1. Input upper limit to print odd number from user. Store it in some variable say N.
   2. Run a loop from 1 to N, increment loop counter by 1 in each iteration. The loop structure should look like for(i=1; i<=N; i++).
   3. Inside the loop body check odd condition i.e. if a number is exactly divisible by 2 then it is odd. Which is if(i % 2 != 0) then, print the value of i.

6. Write a C program to find sum of all natural numbers between 1 to n.

Step by step descriptive logic to find sum of n natural numbers.

1. Input upper limit to find sum of natural numbers. Store it in some variable say N.
2. Initialize another variable to store sum of numbers say sum = 0.
3. In order to find sum we need to iterate through all natural numbers between 1 to n. Initialize a loop from 1 to N, increment loop counter by 1 for each iteration. The loop structure should look like for(i=1; i<=N; i++).
4. Inside the loop add previous value of sum with i. Which is sum = sum + i.
5. Finally after loop print the value of sum.

7. Write a C program to find sum of all even numbers between 1 to n.

Step by step descriptive logic to find sum of even numbers.

1. Input upper limit to find sum of even number. Store it in some variable say N.
2. Initialize another variable to store sum with 0 say sum = 0.
3. To find sum of even numbers we need to iterate through even numbers from 1 to n. Initialize a loop from 2 to N and increment 2 on each iteration. The loop structure should look like for(i=2; i<=N; i+=2).
4. Inside the loop body add previous value of sum with i i.e. sum = sum + i.
5. After loop print final value of sum.

8. Write a C program to find sum of all odd numbers between 1 to n.

Step by step descriptive logic to find sum of odd numbers between 1 to n.

1. Input upper limit to find sum of odd numbers from user. Store it in some variable say N.
2. Initialize other variable to store sum say sum = 0.
3. To find sum of odd numbers we must iterate through all odd numbers between 1 to n. Run a loop from 1 to N, increment 1 in each iteration. The loop structure must look similar to for(i=1; i<=N; i++).
4. Inside the loop add sum to the current value of i i.e. sum = sum + i.
5. Print the final value of sum.

9. Write a C program to print multiplication table of any number.

Step by step descriptive logic to print multiplication table.

1. Input a number from user to generate multiplication table. Store it in some variable say num.
2. To print multiplication table we need to iterate from 1 to 10. Run a loop from 1 to 10, increment 1 on each iteration. The loop structure should look like for(i=1; i<=10; i++).
3. Inside loop generate multiplication table using num * i and print in specified format.

10. Write a C program to count number of digits in a number.

First logic is the easiest and is the common to think. It uses loop to count number of digits. To count number of digits divide the given number by 10 till number is greater than 0. For each iteration increment the value of some count variable.

Step by step descriptive logic to count number of digits in given integer using loop.

1. Input a number from user. Store it in some variable say num.
2. Initialize another variable to store total digits say digit = 0.
3. If num > 0 then increment count by 1 i.e. count++.
4. Divide num by 10 to remove last digit of the given number i.e. num = num / 10.
5. Repeat step 3 to 4 till num > 0 or num != 0.

11. Write a C program to find first and last digit of a number.

Step by step descriptive logic to find first and last digit of a number without loop.

1. Input a number from user. Store it in some variable say num.
2. Find last digit using modulo division by 10 i.e. lastDigit = num % 10.
3. To find first digit we have simple formula firstDigit = n / pow(10, digits - 1). Where digits is total number of digits in given number.

12. Write a C program to find sum of first and last digit of a number.

Step by step descriptive logic to find sum of first and last digit using loop.

1. Input a number from user. Store it in some variable say num.
2. To find last digit of given number we modulo divide the given number by 10. Which is lastDigit = num % 10.
3. To find first digit we divide the given number by 10 till num is greater than 0.
4. Finally calculate sum of first and last digit i.e. sum = firstDigit + lastDigit.

13. Write a C program to calculate sum of digits of a number.

The main idea to find sum of digits can be divided in three steps.

1. Extract last digit of the given number.
2. Add the extracted last digit to sum.
3. Remove last digit from given number. As it is processed and not required any more.

If you repeat above three steps till the number becomes 0. Finally you will be left with sum of digits.

Step by step descriptive logic to find sum of digits of a given number.

1. Input a number from user. Store it in some variable say num.
2. Find last digit of the number. To get last digit modulo division the number by 10 i.e. lastDigit = num % 10.
3. Add last digit found above to sum i.e. sum = sum + lastDigit.
4. Remove last digit from number by dividing the number by 10 i.e. num = num / 10.
5. Repeat step 2-4 till number becomes 0. Finally you will be left with the sum of digits in sum.

14. Write a C program to calculate product of digits of a number except zero.

Logic to find product of digits is exactly similar to sum of digits. If you are done with previous program sum of digits, you can easily think logic for this program. If not read below logic to find product of digits.

I have divided the logic to calculate product of digits in three steps.

1. Extract last digit of the given number.
2. Multiply the extracted last digit with product.
3. Remove the last digit by dividing number by 10.

Step by step descriptive logic to find product of digits of a given number.

1. Input a number from user. Store it in some variable say num.
2. Initialize another variable to store product i.e. product = 1. Now, you may think why I have initialized product with 1 why not 0? This is because we are performing multiplication operation not summation. Multiplying a number with 1 returns same, so as summation with 0 returns same. Hence, I have initialized product with 1. Also be sure to initialize the product with 0 if num is 0.
3. Find last digit of number by performing modulo division by 10 i.e. lastDigit = num % 10.
4. Multiply last digit found above with product i.e. product = product * lastDigit.
5. Remove last digit by dividing the number by 10 i.e. num = num / 10.
6. Repeat step 3-5 till number becomes 0. Finally you will be left with product of digits in product variable.

15. Write a C program to enter a number and print its reverse.

Step by step descriptive logic to find reverse of a number.

1. Input a number from user to find reverse. Store it in some variable say num.
2. Declare and initialize another variable to store reverse of num, say reverse = 0.
3. Extract last digit of the given number by performing modulo division. Store the last digit to some variable say lastDigit = num % 10.
4. Increase the place value of reverse by one. To increase place value multiply reverse variable by 10 i.e. reverse = reverse * 10.
5. Add lastDigit to reverse i.e. reverse = reverse + lastDigit.
6. Since last digit of num is processed hence, remove last digit of num. To remove last digit divide num by 10 i.e. num = num / 10.
7. Repeat step 3 to 6 till num is not equal to (or greater than) zero.

16. Write a C program to check whether a number is palindrome or not.

What is Palindrome number?

Palindrome number is such number which when reversed is equal to the original number. For example: 121, 12321, 1001 etc.

Logic to check palindrome number

Step by step descriptive logic to check palindrome number.

1. Input a number from user. Store it in some variable say num.
2. Find reverse of the given number. Store it in some variable say reverse.
3. Compare num with reverse. If both are same then the number is palindrome otherwise not.

17. Write a C program to enter a number and print it in words.

Step by step descriptive logic to convert number in words.

1. Input number from user. Store it in some variable say num.
2. Extract last digit of given number by performing modulo division by 10. Store the result in a variable say digit = num % 10.
3. Switch the value of digit found above. Since there are 10 possible values of digit i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 hence, write 10 cases. Print corresponding word for each case.
4. Remove last digit from num by dividing it by 10 i.e. num = num / 10.
5. Repeat step 2 to 4 till number becomes 0.

18. Write a C program to print all ASCII character with their values.

Step by step descriptive logic to print ASCII value of all characters.

1. Declare an integer variable i as loop counter.
2. Run a loop from 0 to 255, increment by 1 in each iteration. The loop structure must look like for(i=0; i<=255; i++).

   Why iterate from 0 to 255? Because total number of ASCII characters is 256 (non-printable + printable + extended ASCII) and values are in between 0 to 255.

3. Inside the loop print character representation of the given integer.

19. Write a C program to find power of a number using for loop.

1. Input base and exponents from user. Store it in two variables say base and expo.
2. Declare and initialize another variable to store power say power = 1.
3. Run a loop from 1 to expo, increment loop counter by 1 in each iteration. The loop structure must look similar to for(i=1; i<=expo; i++).
4. For each iteration inside loop multiply power with num i.e. power = power * num.
5. Finally after loop you are left with power in power variable.

20. Write a C program to find all factors of a number.

1. Input number from user. Store it in some variable say num.
2. Run a loop from 1 to num, increment 1 in each iteration. The loop structure should look like for(i=1; i<=num; i++).
3. For each iteration inside loop check current counter loop variable i is a factor of num or not. To check factor we check divisibility of number by performing modulo division i.e. if(num % i == 0) then i is a factor of num.

   If i is a factor of num then print the value of i.

21. Write a C program to calculate factorial of a number.

Step by step descriptive logic to find factorial of a number.

1. Input a number from user. Store it in some variable say num.
2. Initialize another variable that will store factorial say fact = 1.

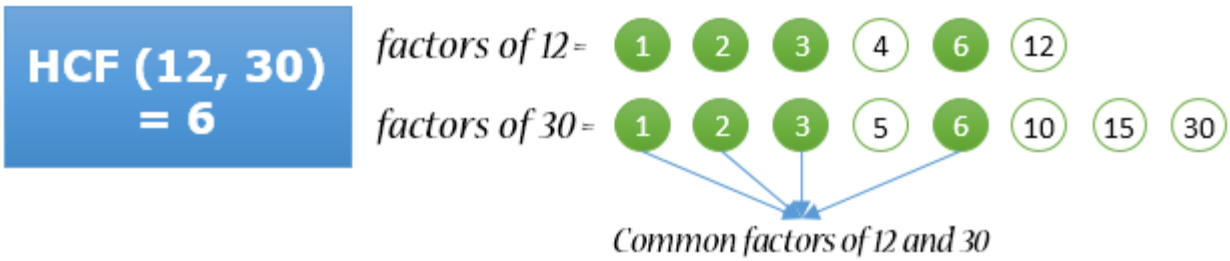
   Why initialize fact with 1 not with 0? This is because you need to perform multiplication operation not summation. Multiplying 1 by any number results same, same as summation of 0 and any other number results same.

3. Run a loop from 1 to num, increment 1 in each iteration. The loop structure should look like for(i=1; i<=num; i++).
4. Multiply the current loop counter value i.e. i with fact. Which is fact = fact * i.

22. Write a C program to find HCF (GCD) of two numbers.

What is HCF?

HCF (Highest Common Factor) is the greatest number that divides exactly two or more numbers. HCF is also known as GCD (Greatest Common Divisor) or GCF (Greatest Common Factor).



Logic to find HCF of two numbers

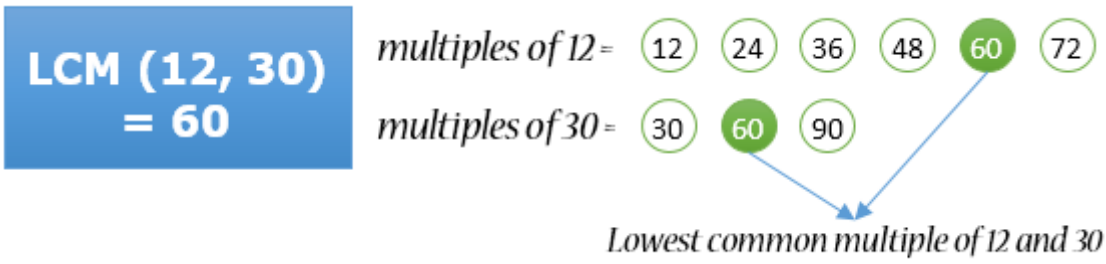
Step by step descriptive logic to find HCF.

1. Input two numbers from user. Store them in some variable say num1 and num2.
2. Declare and initialize a variable to hold hcf i.e. hcf = 1.
3. Find minimum between the given two numbers. Store the result in some variable say min = (num1<num2) ? num1 : num2;.
4. Run a loop from 1 to min, increment loop by 1 in each iteration. The loop structure should look like for(i=1; i<=min; i++).
5. Inside the loop check if i is a factor of two numbers i.e. if i exactly divides the given two numbers num1 and num2 then set i as HCF i.e. hcf = i.

23. Write a C program to find LCM of two numbers.

What is LCM?

LCM is a smallest positive integer that exactly divides two or more numbers. For Example



Logic to find LCM of two numbers

Step by step descriptive logic to find LCM of two numbers.

1. Input two numbers from user. Store them in some variable say num1 and num2.
2. Find maximum between two numbers. Store the result in some variable, say max. Maximum is used to generate next multiple which must be common to both.
3. If max is exactly divisible by both numbers. Then you got your answer, store max to some variable say lcm = max. If LCM is found then terminate from loop using break keyword.
4. If max is not divisible by both numbers. Then generate next multiple of max.
5. Repeat steps 2 to 3 step till LCM is found.

24. Write a C program to check whether a number is Prime number or not.

Logic to check prime number

There are several efficient algorithms for prime test. For this post I am implementing the simplest and easiest algorithm for beginners. If the number is divisible by any number in between 2 to n-1. Then it is composite number otherwise prime.

Step by step descriptive logic to check prime number.

1. Input a number from user. Store it in some variable say num.
2. Declare and initialize another variable say isPrime = 1. isPrime variable is used as a notification or flag variable. Assigning 0 means number is composite and 1 means prime.
3. Run a loop from 2 to num/2, increment 1 in each iteration. The loop structure should be like for(i=2; i<=num/2; i++).
4. Check, divisibility of the number i.e. if(num%i == 0) then, the number is not prime.

   Set isPrime = 0 indicating number is not prime and terminate from loop.

5. Outside the loop check the current value of isPrime. According to our assumption if it is equal to 1 then the number is prime otherwise composite.

25. Write a C program to print all Prime numbers between 1 to n.

Logic to print prime numbers between 1 to n

Step by step descriptive logic to print all prime numbers between 1 to n.

1. Input upper limit to print prime numbers from user. Store it in some variable say end.
2. Run a loop from 2 to end, increment 1 in each iteration. The loop structure should be like for(i=2; i<=end; i++).
3. Inside the loop for each iteration print value of i if it is prime number.

26. Write a C program to find sum of all prime numbers between 1 to n.

What is Prime number?

Prime numbers are positive integers greater than 1 that has only two divisors 1 and the number itself. For example: 2, 3, 5, 7, 11 are the first 5 prime numbers.

Logic to find sum of prime numbers between 1 to n

Step by step descriptive logic to find sum of prime numbers between 1 to n.

1. Input upper limit to find sum of prime from user. Store it in some variable say end.
2. Initialize another variable sum = 0 to store sum of prime numbers.
3. Run a loop from 2 to end, incrementing 1 in each iteration. The loop structure should look like for(i=2; i<=end; i++).
4. Inside the loop check if loop counter variable is prime or not. If i is prime then add i to sum i.e. sum = sum + i.
5. Finally after loop print the resultant value of sum.

27. Write a C program to find all prime factors of a number.

What is Prime factor?

Factors of a number that are prime numbers are called as Prime factors of that number. For example: 2 and 5 are the prime factors of 10.

Logic to check prime factors of a number

Step by step descriptive logic to find prime factors.

1. Input a number from user. Store it in some variable say num.
2. Run a loop from 2 to num/2, increment 1 in each iteration. The loop structure should look like for(i=2; i<=num/2; i++).

   You may think why loop from 2 to num/2? Because prime number starts from 2 and any factor of a number n is always less than n/2.

3. Inside the loop, first check if i is a factor of num or not. If it is a factor then check it is prime or not.

   Print the value of i if it is prime and a factor of num.

28. Write a C program to check whether a number is Armstrong number or not.

What is Armstrong number?

An *Armstrong number* is a n-digit number that is equal to the sum of the n^{th} power of its digits. For example - $6 = 6^1 = 6$

$371 = 3^3 + 7^3 + 1^3 = 371$

Logic to check Armstrong number

Step by step descriptive logic to check Armstrong number.

1. Input a number from user. Store it in some variable say num. Make a temporary copy of the value to some another variable for calculation purposes, say originalNum = num.
2. Count total digits in the given number, store result in a variable say digits.
3. Initialize another variable to store the sum of power of its digits ,say sum = 0.
4. Run a loop till num > 0. The loop structure should look like while(num > 0).
5. Inside the loop, find last digit of num. Store it in a variable say lastDigit = num % 10.
6. Now comes the real calculation to find sum of power of digits. Perform sum = sum + pow(lastDigit, digits).
7. Since the last digit of num is processed. Hence, remove last digit by performing num = num / 10.
8. After loop check if(originalNum == sum), then it is Armstrong number otherwise not.

29. Write a C program to check whether a number is Perfect number or not.

What is Perfect number?

Perfect number is a positive integer which is equal to the sum of its proper positive divisors.
For example: 6 is the first perfect number
Proper divisors of 6 are 1, 2, 3
Sum of its proper divisors = 1 + 2 + 3 = 6.
Hence 6 is a perfect number.

Logic to check Perfect number

Step by step descriptive logic to check Perfect number.

1. Input a number from user. Store it in some variable say num.
2. Initialize another variable to store sum of proper positive divisors, say sum = 0.
3. Run a loop from 1 to num/2, increment 1 in each iteration. The loop structure should look like for(i=1; i<=num/2; i++).

   Why iterating from 1 to num/2, why not till num? Because a number does not have any proper positive divisor greater than num/2.

4. Inside the loop if current number i.e. i is proper positive divisor of num, then add it to sum.

   Learn - Program to check divisibility of a number.
5. Finally, check if the sum of proper positive divisors equals to the original number. Then, the given number is Perfect number otherwise not.

30. Write a C program to print all Perfect numbers between 1 to n.

What is Perfect number?

Perfect number is a positive integer which is equal to the sum of its proper positive divisors.

For example: 6 is the first perfect number
Proper divisors of 6 are 1, 2, 3
Sum of its proper divisors = 1 + 2 + 3 = 6.
Hence 6 is a perfect number.

Logic to find all Perfect number between 1 to n

Step by step descriptive logic to find Perfect numbers from 1 to n.

1. Input upper limit from user to find Perfect numbers. Store it in a variable say end.
2. Run a loop from 1 to end, increment 1 in each iteration. The loop structure should look like for(i=1; i<=end; i++).
3. For each iteration inside loop print the value of i if it is a Perfect number.

31. Write a C program to check whether a number is Strong number or not.

What is Strong number?

Strong number is a special number whose sum of factorial of digits is equal to the original number.
For example: 145 is strong number. Since, 1! + 4! + 5! = 145

Logic to check Strong number

Step by step descriptive logic to check strong number.

1. Input a number from user to check for strong number. Store this in a variable say num. Copy it to a temporary variable for calculations purposes, say originalNum = num.
2. Initialize another variable to store sum of factorial of digits, say sum = 0.
3. Find last digit of the given number num. Store the result in a variable say lastDigit = num % 10.
4. Find factorial of lastDigit. Store factorial in a variable say fact.
5. Add factorial to sum i.e. sum = sum + fact.
6. Remove last digit from num as it is not needed further.
7. Repeat steps 3 to 6 till num > 0.
8. After loop check condition for strong number. If sum == originalNum, then the given number is Strong number otherwise not.

32. Write a C program to print all Strong numbers between 1 to n.

What is a Strong number?

Strong number is a special number whose sum of factorial of digits is equal to the original number. For example: 145 is strong number. Since, 1! + 4! + 5! = 145

Logic to print Strong numbers between 1 to n

Step by step descriptive logic to print strong numbers from 1 to n.

1. Input upper limit to print strong number from user. Store it in a variable say end.
2. Run a loop from 1 to end, increment 1 in each iteration. Structure of the loop should be similar to for(i=1; i<=end; i++).
3. For each iteration inside loop check i for strong number. Print the value of i if it is a strong number.

33. Write a C program to print Fibonacci series up to n terms.

What is Fibonacci series?

Fibonacci series is a series of numbers where the current number is the sum of previous two terms. For Example: 0, 1, 1, 2, 3, 5, 8, 13, 21, ... , (n-1th + n-2th)

Logic to print Fibonacci series upto n terms

Step by step descriptive logic to print n Fibonacci terms.

1. Input number of Fibonacci terms to print from user. Store it in a variable say terms.
2. Declare and initialize three variables, I call it as Fibonacci magic initialization. a=0, b=1 and c=0.

   Here c is the current term, b is the n-1^{th} term and a is n-2^{th} term.

3. Run a loop from 1 to terms, increment loop counter by 1. The loop structure should look like for(i=1; i<=term; i++). It will iterate through n terms
4. Inside the loop copy the value of n-1^{th} term to n-2^{th} term i.e. a = b.

   Next, copy the value of n^{th} to n-1^{th} term b = c.

   Finally compute the new term by adding previous two terms i.e. c = a + b.

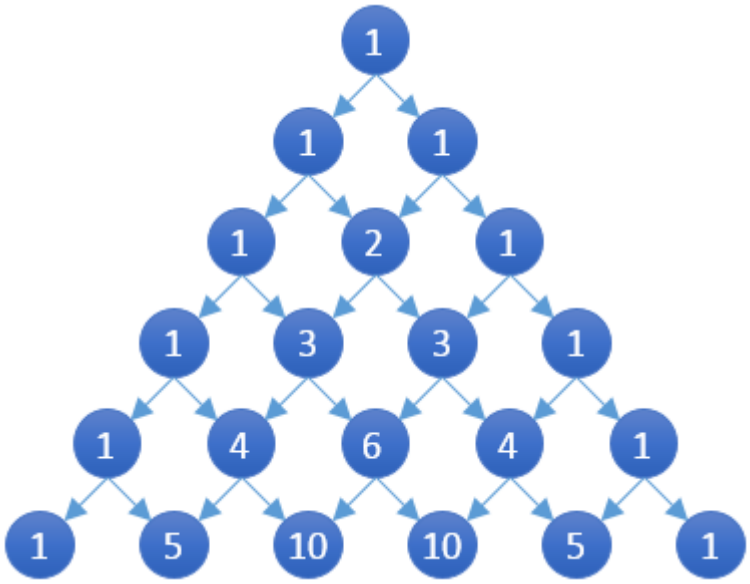
5. Print the value of current Fibonacci term i.e. c.

34. Write a C program to print Pascal triangle upto n rows.

Pascal Triangle

Pascal triangle is a triangular number pattern named after famous mathematician Blaise Pascal.

For example Pascal triangle with 6 rows.



Logic to print pascal triangle

To find n^{th} term of a pascal triangle we use following formula. $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Where n is row number and k is term of that row.

Step by step descriptive logic to print pascal triangle.

1. Input number of rows to print from user. Store it in a variable say num.
2. To iterate through rows, run a loop from 0 to num, increment 1 in each iteration. The loop structure should look like for(n=0; n<num; n++).
3. Inside the outer loop run another loop to print terms of a row. Initialize the loop from 0 that goes to n, increment 1 in each iteration.
4. Inside the inner loop use formula term = fact(n) / (fact(k) * fact(n-k)); to print current term of pascal triangle.