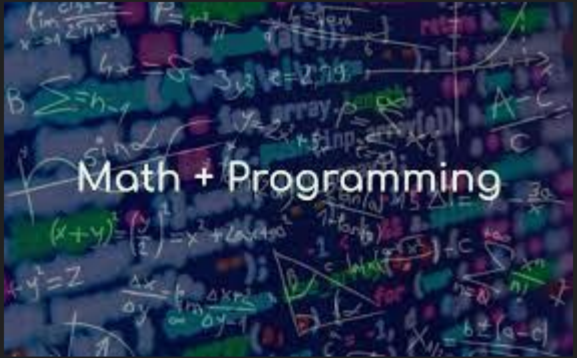
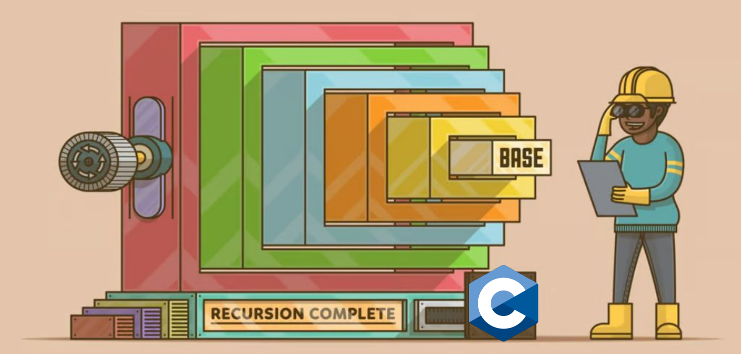




Basic Maths & Recursion

In C

Math Operations

% - Modulus Operator, gives out remainder of 2 numbers

a%b==remainder when b divides a

a=b*q+r

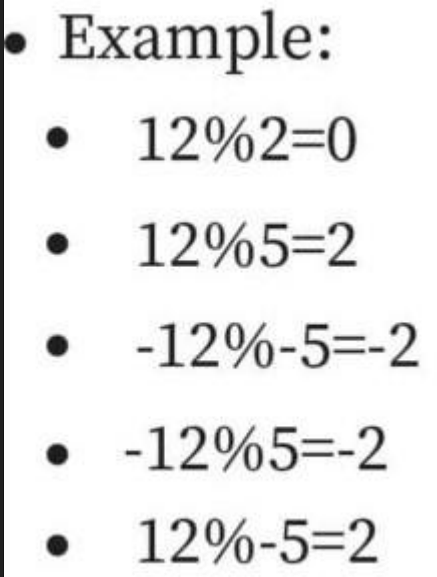
To extract last digit of a number we do num%10

/ - Division operator, outputs just the integer value

Example: 5/2 = 2

47/10=4

^ - It is used for XOR operation not power



- Example:
  - 12%2=0
  - 12%5=2
  - -12%-5=-2
  - -12%5=-2
  - 12%-5=2

Counting Digits

Set Up Your Counter

First off, grab a counter called "count" and set it to zero. It's like your scoreboard for how many digits you find.

Go Through Each Digit Adventure

Picture your number as a treasure map. You're going on an adventure to each spot (digit) on the map.

Every time you find a spot (digit), make a mark on your scoreboard (increase the count by 1).

Update the Map

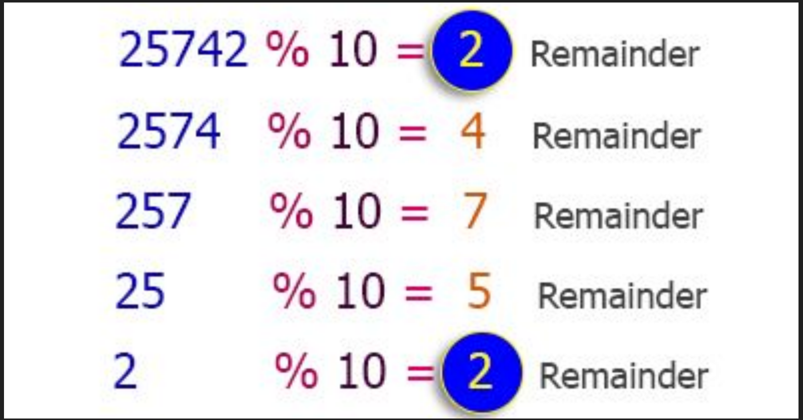
After marking down a spot, erase it from your treasure map. Pretend you're uncovering the map, bit by bit.

Keep doing this until you've explored every spot on your map.

Final Scoreboard Tally

Look at your scoreboard. The number you see there is like the treasure you found! It's the total count of digits in your original number.

Counting Digits



25742 % 10 = 2 Remainder

2574 % 10 = 4 Remainder

257 % 10 = 7 Remainder

25 % 10 = 5 Remainder

2 % 10 = 2 Remainder

Counting Digits

```
#include <stdio.h>
int main() {
    int number = 1234567;
    int count = 0;  // Step 1
    while (number != 0) {  // Step 2
        int digit = number % 10;  // Extract the last digit
        count++;  // Increment the count
        number /= 10;  // Remove the last digit
    }
    printf("The number of digits is %d\n", count);  // Step 4
    return 0;
}
```

Input : 1234567

Output : 7

Thala for a reason

Reversing a number

Get Ready to Flip

- Imagine you have a special tool, let's call it "reverse Number," ready to flip your digits.

Start Flipping

- Think of your number as a series of digits. Begin from the end, pick each digit, and place it in the reversed Number tool.

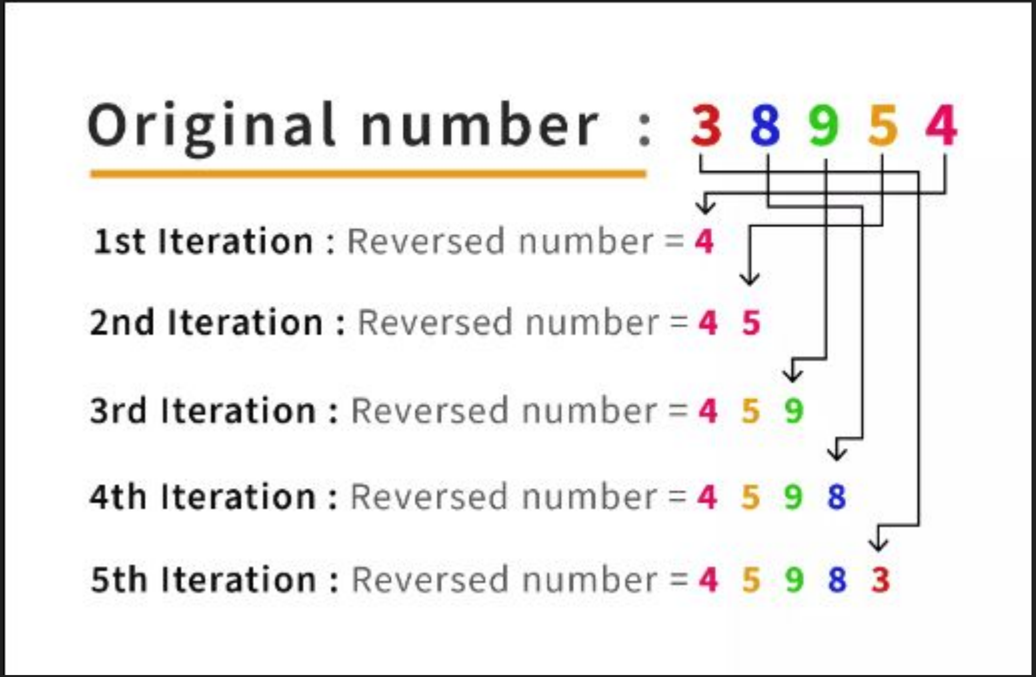
Keep Flipping, Keep Stacking

- Move through each digit, stacking them in the reversed Number tool one by one until you've gone through all of them.

Voila! Reversed Number Ready

- Look at the reversed Number tool. That's your reversed number, all flipped and ready to go!

Reversing a number



Original number : 3 8 9 5 4

1st Iteration : Reversed number = 4

2nd Iteration : Reversed number = 4 5

3rd Iteration : Reversed number = 4 5 9

4th Iteration : Reversed number = 4 5 9 8

5th Iteration : Reversed number = 4 5 9 8 3

Reversing a number

```
#include<stdio.h>
 int main()
{
int n, reverse=0, rem;
printf("Enter a number: ");
  scanf("%d", &n);
  while(n!=0)
  {
     rem=n%10;
     reverse=reverse*10+rem;
     n=n/10;
  }
  printf("Reversed Number: %d",reverse);
return 0;
}
```

Input : 123

Output : 321

Palindrome Number

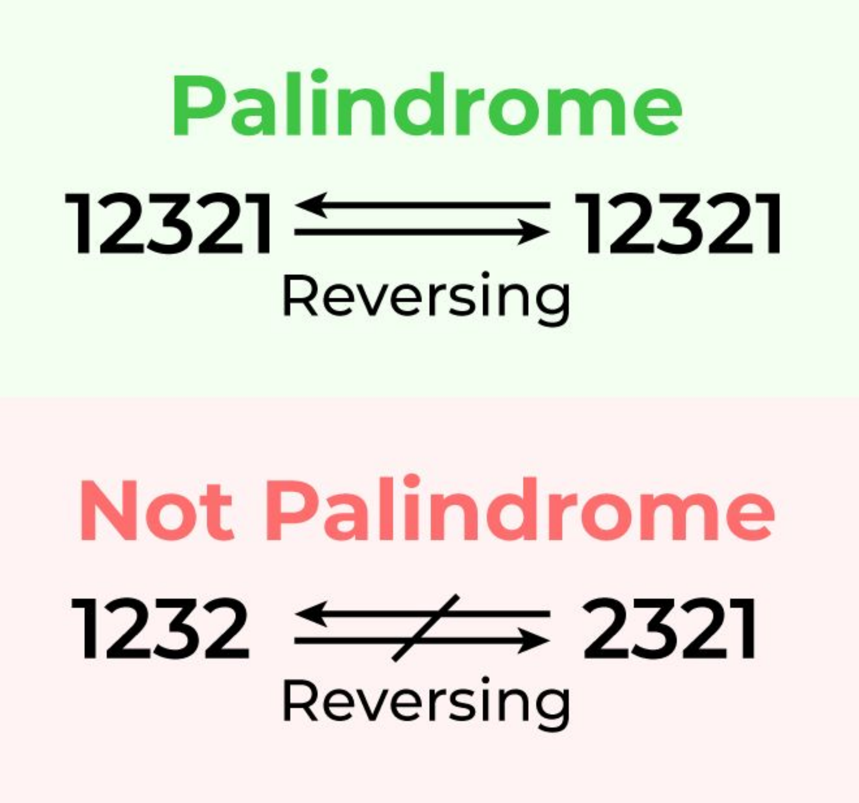
Doing this only for numbers in c.

Reverse the number

- Reverse the input using the previous logic

Checking

- Use Conditional operators to check the reversed and input



Palindrome Number

```
#include<stdio.h>
int main()
{
int n,r,sum=0,temp;
printf("enter the number=");
scanf("%d",&n);
temp=n;
while(n>0)
{
r=n%10;
sum=(sum*10)+r;
n=n/10;
}
if(temp==sum)
printf("palindrome number ");
else
printf("not palindrome");
return 0;
}
```

Input : 123

Output: Not Palindrome

Input 121

Output : Palindrome

Prime number

Set Up Your Prime Checker

- Get ready to unleash your prime checker! Imagine it as a superhero ready to determine if a number is prime.

The Detective Work Begins

- Start by checking if the number is 2. If it is, it's a prime! Your prime checker is quick to identify the first prime superhero.

More Detective Work

- If the number isn't 2, and it's an even number, your prime checker confidently declares it's not a prime. Even numbers (except 2) don't make the cut.

Odd Numbers Under Investigation

- For odd numbers (greater than 2), let your prime checker investigate from 3 up to the square root of the number.
- If the number is divisible evenly by any of these investigating numbers, it's not prime! The prime checker exposes the culprits.

Verdict Time

- After the investigation, if the number survived without any divisors, your prime checker proudly declares it as a prime number!

Prime number

```
#include <stdio.h>
#include <math.h>
int main() {
  int number;
  printf("Enter a number: ");
  scanf("%d", &number);
  int isPrime = 1;  // Assume it's prime
  if (number == 2) {
    // 2 is a prime number
    isPrime = 1;
  } else if (number % 2 == 0 || number == 1) {
    // Even numbers (except 2) and 1 are not prime
    isPrime = 0;
  } else {
    // Check odd numbers for divisors
    int limit = number/2;;
    for (int i = 3; i <= limit; i += 2) {
      if (number % i == 0) {
        // Found a divisor, not a prime
        isPrime = 0;
        break;
      }
    }
  }
  if (isPrime) {
    printf("%d is a prime number!\n", number);
  } else {
    printf("%d is not a prime number.\n", number);
  }
  return 0;
}
```

Input : 2

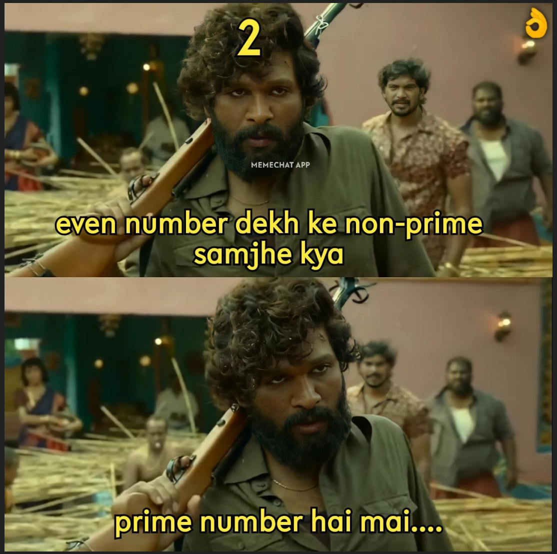
Output : Prime

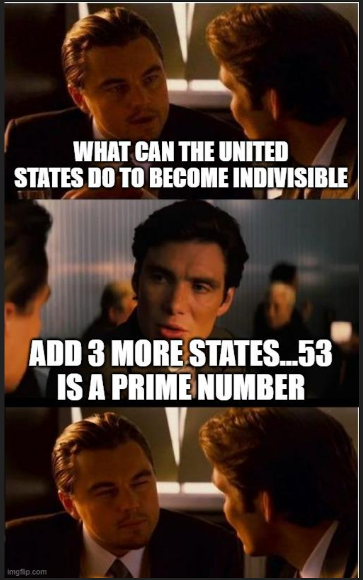
Input : 9

Output : Not Prime

Input : 6

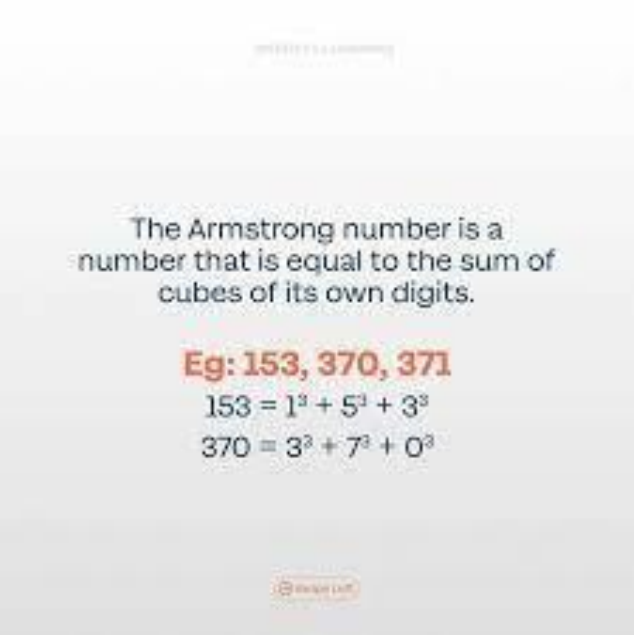
Output: Not Prime




WHAT CAN THE UNITED STATES DO TO BECOME INDIVISIBLE
ADD 3 MORE STATES...53 IS A PRIME NUMBER
imgflip.com

Armstrong Number

1. Find number of digits
2. Now in a loop
   - Find the `remainder` (last digit) (using %)
   - Add the `remainder` raised to the power of `n` to the `result`.
   - Update `originalNumber` by dividing it by 10.
3. Now use conditional operator to check the number and result



The Armstrong number is a number that is equal to the sum of cubes of its own digits.

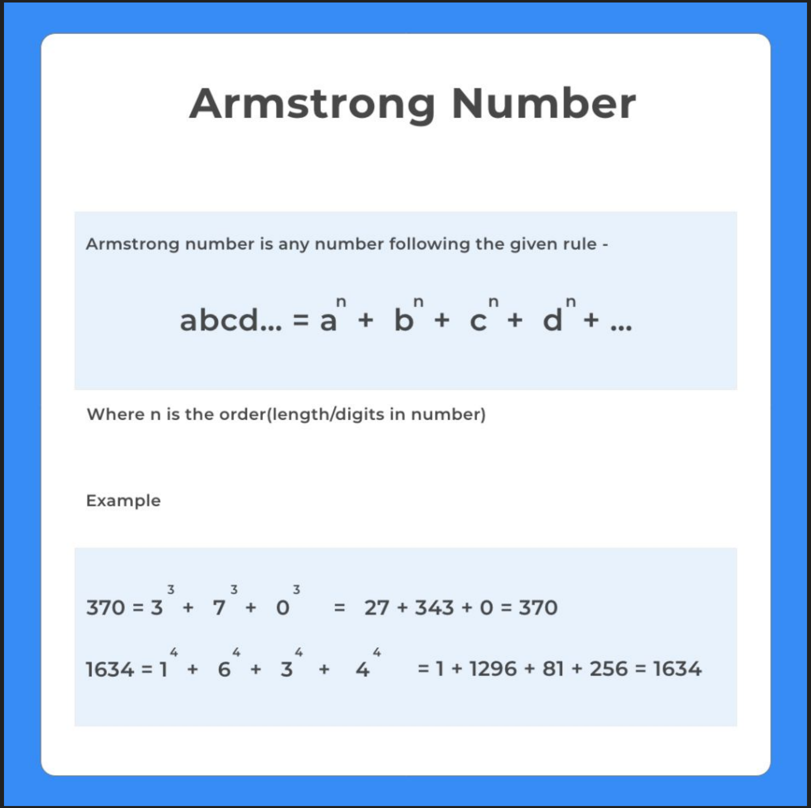
Eg: 153, 370, 371

$153 = 1^3 + 5^3 + 3^3$

$370 = 3^3 + 7^3 + 0^3$

Armstrong Number

```
#include <stdio.h>
#include <math.h>
int main() {
 int number, originalNumber, remainder, n = 0, result = 0;
    printf("Enter a number: ");
    scanf("%d", &number);
    originalNumber = number;
    while (originalNumber != 0) {
        originalNumber /= 10;
        ++n;
    }
    originalNumber = number;
    while (originalNumber != 0) {
        remainder = originalNumber % 10;
        result += pow(remainder, n);
        originalNumber /= 10;
    }
    if (result == number)
        printf("%d is an Armstrong number.\n", number);
    else
        printf("%d is not an Armstrong number.\n", number);
    return 0;
}
```



Armstrong Number

Armstrong number is any number following the given rule -

$$abcd... = a^n + b^n + c^n + d^n + ...$$

Where n is the order(length/digits in number)

Example

$370 = 3^3 + 7^3 + 0^3 = 27 + 343 + 0 = 370$

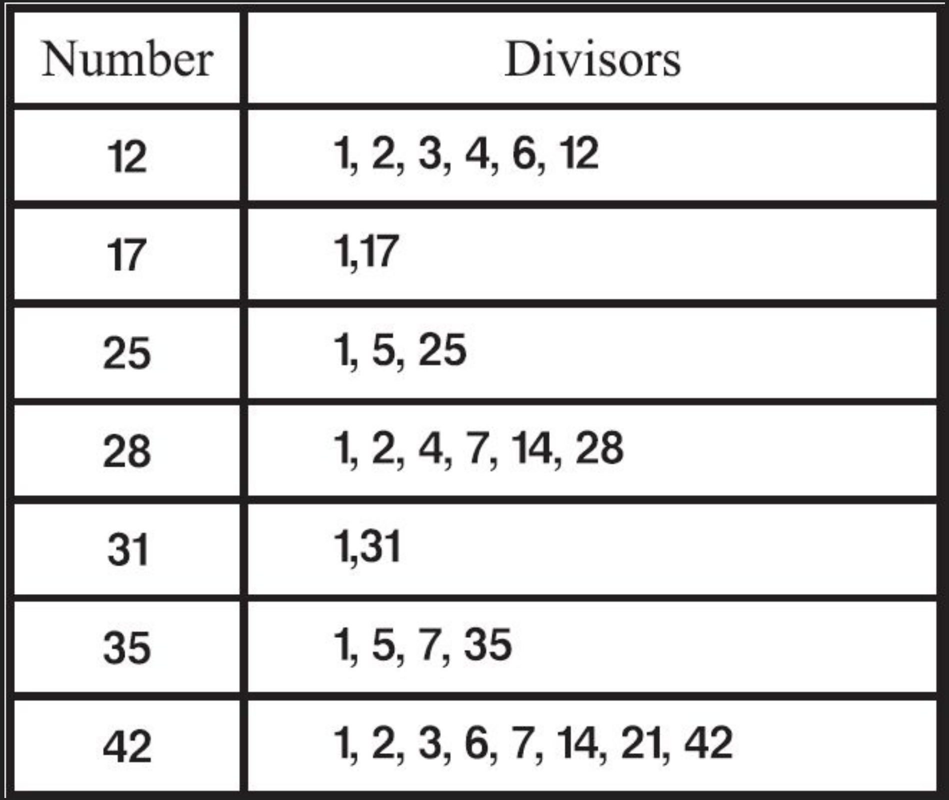
$1634 = 1^4 + 6^4 + 3^4 + 4^4 = 1 + 1296 + 81 + 256 = 1634$

Divisor

A divisor is a number that exactly divides another number, yielding a whole number quotient and leaving no remainder. For example, in the division of 12 by 3, both 3 and 4 are divisors of 12.

Algorithm

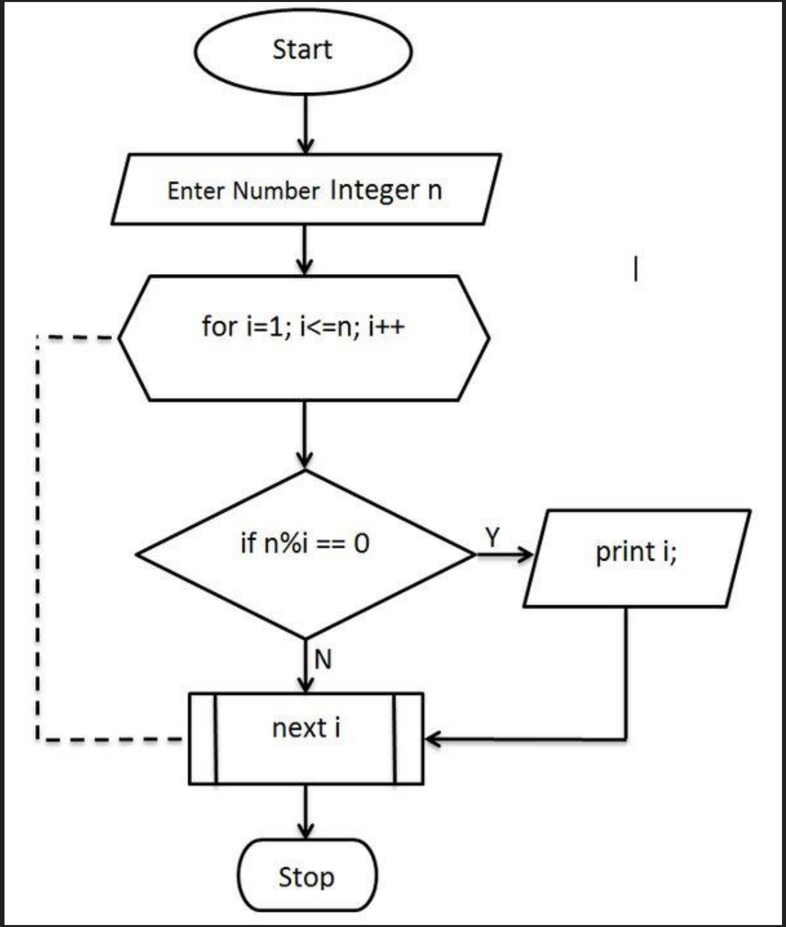
- 1 and number itself are divisors
- Using loop to iterate from 2 to num/2
- Using % to find remainder
- So using a loop we can print the divisors



Number	Divisors
12	1, 2, 3, 4, 6, 12
17	1,17
25	1, 5, 25
28	1, 2, 4, 7, 14, 28
31	1,31
35	1, 5, 7, 35
42	1, 2, 3, 6, 7, 14, 21, 42

Divisor

```
#include<iostream>
int main(){
        int n;
        cin>>n;
        for(int i=1;i<=n/2;i++){
                if(n%i==0)
                        cout<<i<<" ";
        }
        cout<<n<<endl;
}
```

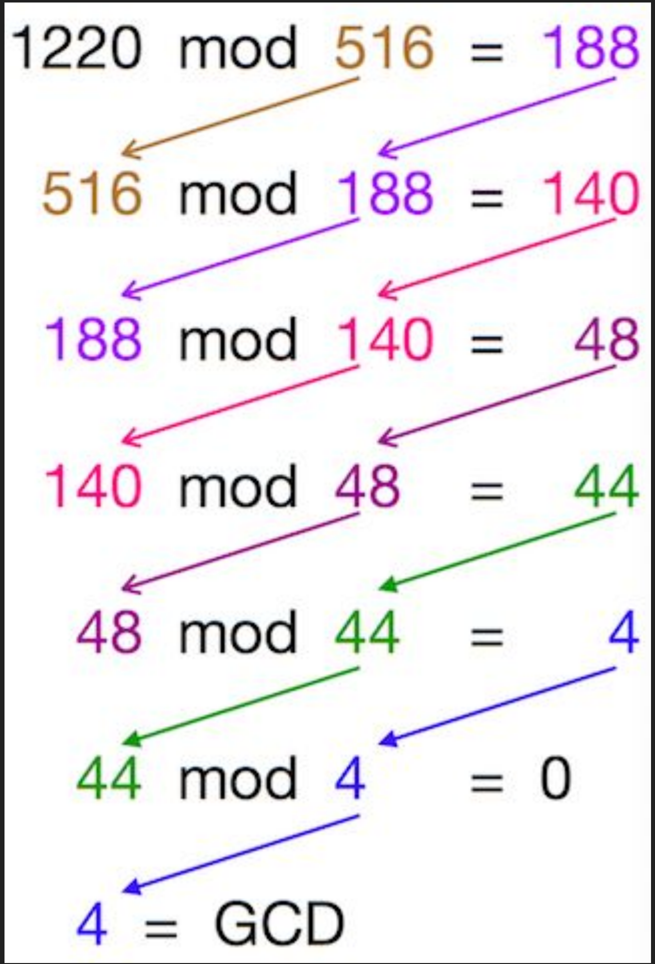


GCD

Start with Number

- Let us take 2 numbers a,b such that a>b

```
gcd(a, b):
    while b ≠ 0:
        remainder = a mod b
        a = b
        b = remainder
    return a
```



1220 mod 516 = 188

516 mod 188 = 140

188 mod 140 = 48

140 mod 48 = 44

48 mod 44 = 4

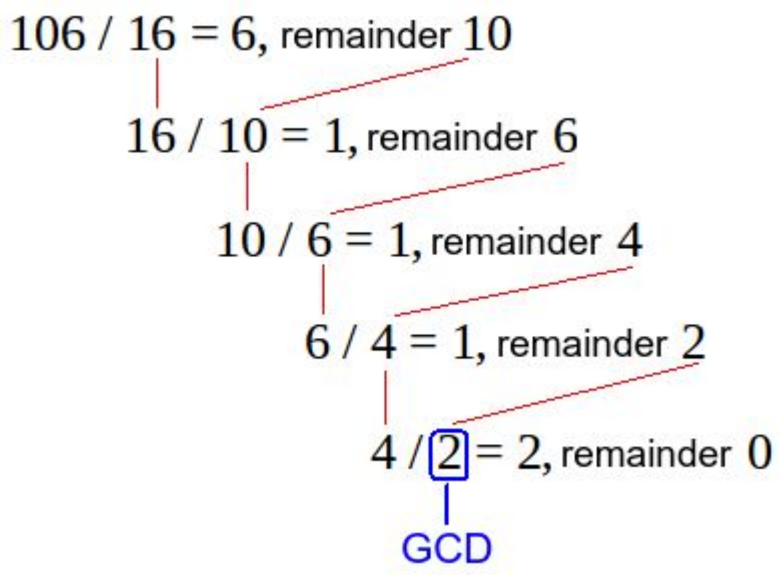
44 mod 4 = 0

4 = GCD

GCD

```
#include <stdio.h>

int main() {
   int num1, num2, remainder;
   printf("Enter the first number: ");
   scanf("%d", &num1);
   printf("Enter the second number: ");
   scanf("%d", &num2);
   while (num2 != 0) {
      remainder = num1 % num2;
      num1 = num2;
      num2 = remainder;
   }
   printf("The GCD of %d and %d is: %d\n", num1, num2,
num1);
   return 0;
}
```



106 / 16 = 6, remainder 10

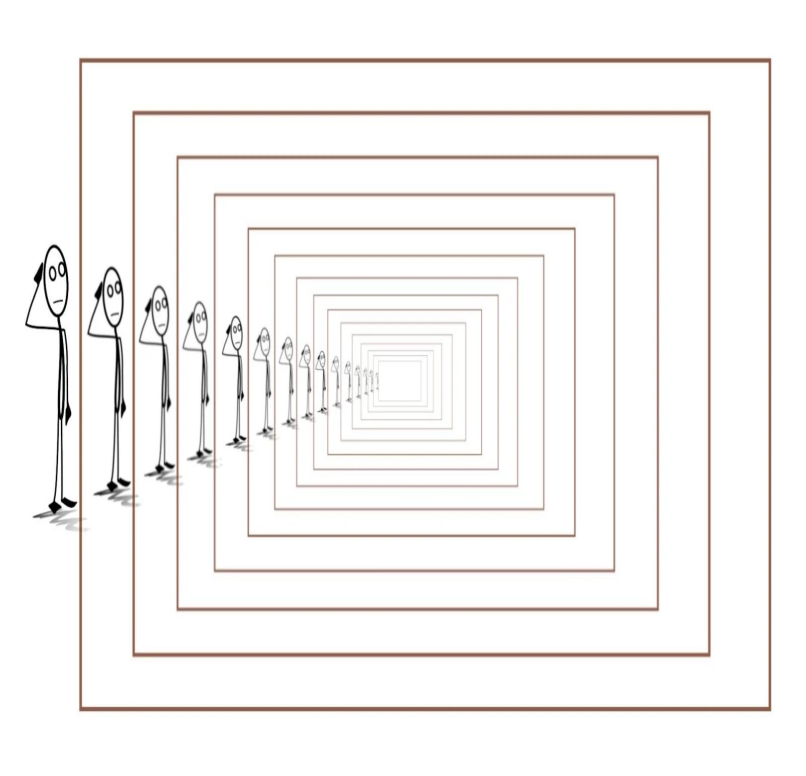
16 / 10 = 1, remainder 6

10 / 6 = 1, remainder 4

6 / 4 = 1, remainder 2

4 / $\boxed{2}$ = 2, remainder 0

Recursion



Certain Terms

- Base Case -
  - The base case is like a safety net in a recursive function. It's a condition that tells the function when to stop calling itself.
- Static variable -
  - A static variable is a special kind of variable that retains its value between function calls. It doesn't get reset every time the function is called.
- Recursive call -
  - A recursive call is when a function calls itself. This is a key concept in recursion, where a problem is solved by solving a smaller instance of the same problem.

Printing numbers N to 1 using recursion

- The `printNumbers` function is a recursive function that prints the current number and then calls itself with the next smaller number.
- The base case checks if `n` is less than or equal to 0, and if true, it returns, stopping the recursion.
- The `main` function calls `printNumbers` with an example starting number of 5.

o/p-

```
5   4    3    2    1
```

```c
#include <stdio.h>
void printNumbers(int n) {
       if (n <= 0)    //BASE CASE
           return;
      else {
              printf("%d\t", n);
              printNumbers(n - 1);    //RECURSIVE CALL
      }
}
int main() {
   int startNumber = 5;
   printNumbers(startNumber);
   return 0;
}
```

How to print 1 to N numbers using recursion?

- How & where will you write the recursive call?
- What will be your base case?

```
void print1_n(int n){
    if(n==0)
        return;
    else{
        print1_n(n-1);
        printf(“%d”,&n);
    }
}
```

Sum of N numbers

1. Input: Take the value of n as input from the user.
2. Base Case: If n is equal to 0, then the sum of the first 0 natural numbers is 0. Return 0.
3. Recursive Case: If n is greater than 0, then the sum of the first n natural numbers is calculated as follows:

   Add n to the sum of the first (n−1) natural numbers. This is done by making a recursive call to the function with n−1.

4. Output: Print or return the calculated sum.

Sum of N numbers

```
#include <stdio.h>

int calculateSum(int n) {
       if (n == 0) {
        return 0;
    } else {
        return n + calculateSum(n - 1);
    }
}

int main() {
    int n;

    printf("Enter a positive integer (n): ");
    scanf("%d", &n);
    int sum = calculateSum(n);
    printf("The sum till %d numbers is: %d\n", n, sum);
    return 0;
}
```

Factorial of N

1. Input: Take the value of n as input from the user.
2. Base Case: If n is equal to 0, then the factorial of 0 is 1. Return 1.
3. Recursive Case: If n is greater than 0, then the factorial of n is calculated as follows:

   Multiply n with the factorial of (n−1). This is done by making a recursive call to the function with n−1.

4. Output: Print or return the calculated factorial.

Factorial of N

```
#include <stdio.h>

int factorial(int n) {
    if (n == 0 || n == 1)        // Base Case
        return 1;
    else                        // Recursive Case
        return n * factorial(n - 1);
}

int main() {
    int n;
    printf("Enter a non-negative integer: ");
    scanf("%d", &n);
    printf("Factorial of %d is: %d\n", n, factorial(n));
}
```

Reversing

1. Base Case:
   - The base case is when num becomes 0.
   - When num becomes 0, the function returns the accumulated sum, which contains the reversed number.
2. Recursive Case:
   - Calculate the last digit of num (rem = num % 10).
   - Update the sum by multiplying it by 10 and adding the last digit (sum = sum * 10 + rem).
   - Update num by removing the last digit (num = num / 10).
   - Make a recursive call to the reverse function with the updated num.
   - The recursion continues until the base case is reached.

Reversing

```
#include <stdio.h>

int reverse(int num){
  static int sum=0;    //will store reversed number
  int rem;
  if(num==0){
            return sum;
  }
  else{
    rem=num%10;
    sum=sum*10+rem;
    return reverse(num/10);
   }
}
```

```
int main(){
  int num;

  printf("Enter any number:");
  scanf("%d",&num);

  printf("The reverse of entered number is %d",reverse(num));
  return 0;
}
```

Fibonacci

1. Input: A non-negative integer n.
2. Base Cases:
    - If n is 0, return 0.
    - If n is 1, return 1.
3. Recursive Step:

   Return Fibonacci(n-1) + Fibonacci(n-2).
4. Output: The nth Fibonacci number.

Fibonacci

```
#include<stdio.h>

int fibonacci(int n) {
    if (n == 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
    int n;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    printf("Fibonacci(%d) = %d\n", n, fibonacci(n));
}
```