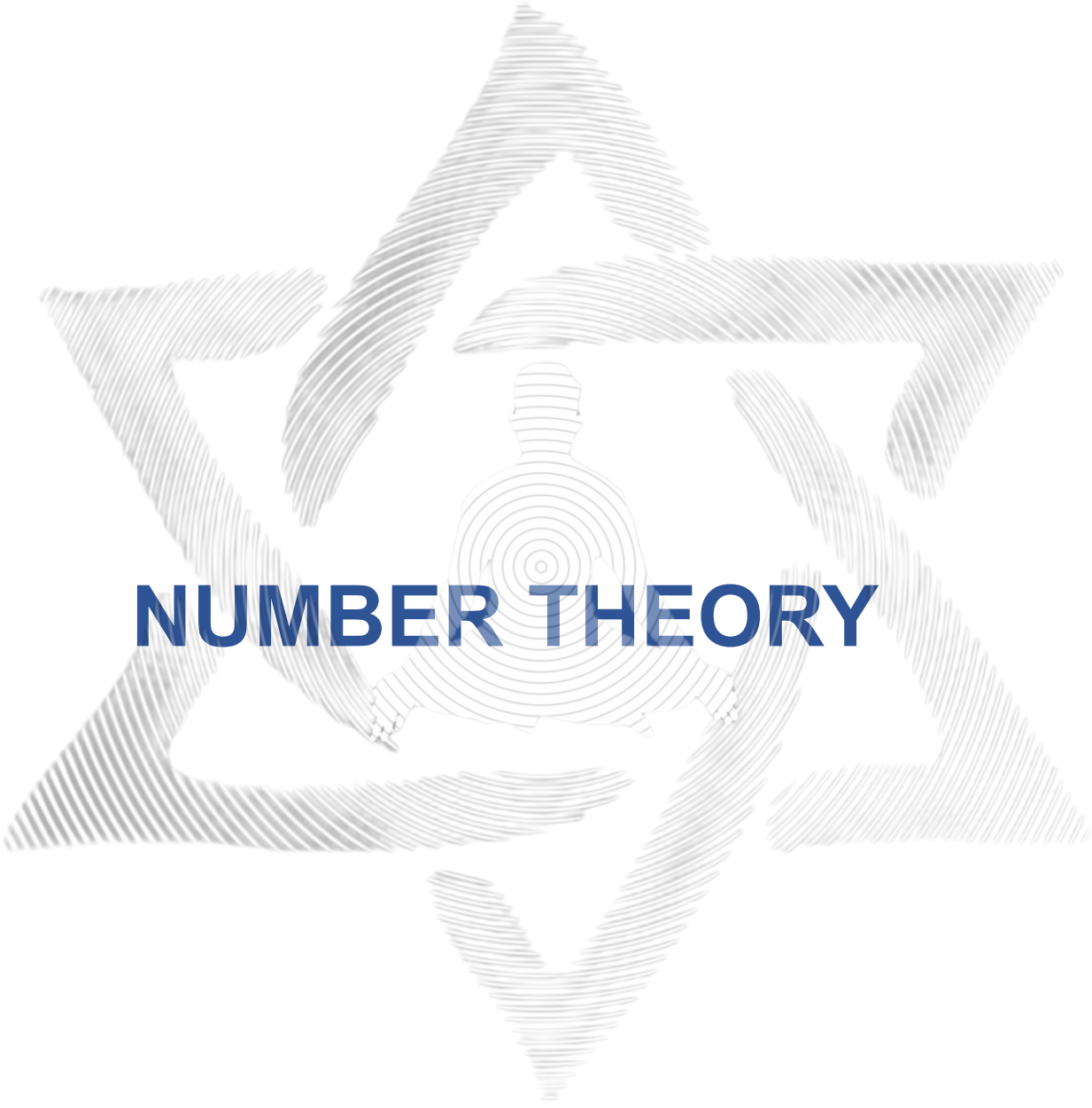


# NUMBER THEORY

# LESSON – 3

**Topics to be covered:**

1. **Segmented Sieve**
2. **Sum of all Divisors**
3. **Smith number**
4. **Fibonacci Series**
5. **Zeckendorf's Theorem (Non-Neighbouring Fibonacci Representation)**
6. **Catalan Numbers**

## 1. Segmented Sieve

**Statement -** Print Prime Numbers between 1000 (lower limit)LL and 2000(Upper Limit) UL.

**Approach -**

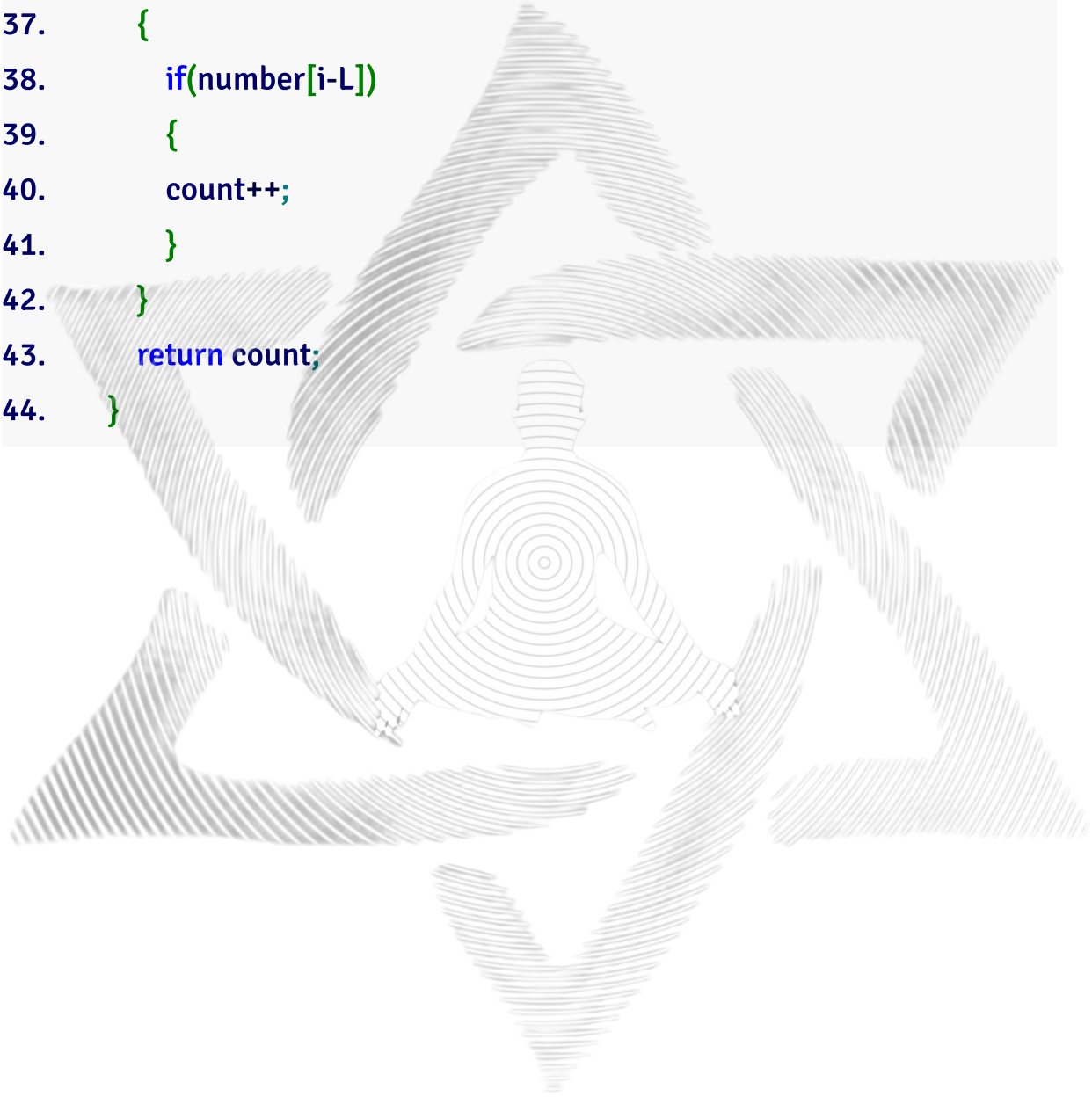
1. Make a boolean array PrimeFactors [ ] of size √UL +1
2. Make an boolean array Numbers[ ] of size UP- LL +1
3. Mark all the prime Numbers in PrimeFactors as True (statement 1).
4. Mark all the Multiples of PrimeFactors as False in Array 2

Code :-

```
1. int SegmentedSieve(int L, int R) {
2.     if(L<=1)
3.     L=2;
4.     int n=sqrt(R);
5.     bool prime[n+1];
```

```
6.      memset(prime,true,sizeof(prime));
7.      bool number[R-L+1];
8.      memset(number,true,sizeof(number));
9.      for(int i=2;i*i<=n;i++)
10.        {
11.          if(prime[i])
12.          {
13.            for(int j=i*i;j<=n;j+=i)
14.            prime[j]=false;
15.          }
16.        }
17.
18.        for(int i=2;i<=n;i++)
19.        {
20.          if(prime[i])
21.          {
22.            int initial = (L/i)*i;
23.            if(initial<L)
24.            initial+=i;
25.            if(initial==i)
26.            initial+=i;
27.            for(int j=initial;j<=R;j+=i)
28.            {
29.             number[j-L]=false;
30.            }
31.          }
32.        }
```

```
33.
34.
35.         int count=0;
36.         for(int i=L;i<=R;i++)
37.         {
38.           if(number[i-L])
39.           {
40.           count++;
41.           }
42.         }
43.         return count;
44.       }
```



## 2. Sum Of All Divisors of Number

**Statement - To find the sum of all divisors of a number N.**

**Approach 1 :-**

**----------find factors upto root(N) and add Pair of the divisors to sum.**

**Code:-**

```
1. int divSum(int num)
2. {
3.    // Final result of summation of divisors
4.    int result = 0;
5.    if(num == 1) // there will be no proper divisor
6.      return result;
7.    // find all divisors which divides 'num'
8.    for (int i=2; i<=sqrt(num); i++)
9.    {
10.         // if 'i' is divisor of 'num'
11.         if (num%i==0)
12.         {
13.             // if both divisors are same then add
14.             // it only once else add both
15.             if (i==(num/i))
16.                 result += i;
17.             else
18.                 result += (i + num/i);
19.         }
20.     }
```

```
21.
22.     // Add 1 to the result as 1 is also a divisor
23.     return (result + 1);
24. }
```

**Approach 2:- Use mathematics to reduce Time of program.**

**Mathematics: -**

**$n=p1^a * p2^b * p3^c$**

**Sum of All factors $=(1+p1+p1^2+p1^3+ - - - - - - p1^a)*(1+p2+p2^2+p2^3+ - - - - - - p2^b)*(1+p3+p3^2+p3^3+ - - - - - - p3^a)$**

**Code:-**

```
1.  int sumofFactors(int n)
2.  {
3.      // Traversing through all prime factors.
4.      int res = 1;
5.      for (int i = 2; i <= sqrt(n); i++)
6.      {
7.
8.
9.          int curr_sum = 1;
10.         int curr_term = 1;
11.         while (n % i == 0) {
12.
```

```
13.             // THE BELOW STATEMENT MAKES
14.             // IT BETTER THAN ABOVE METHOD
15.             //  AS WE REDUCE VALUE OF n.
16.             n = n / i;
17.
18.             curr_term *= i;
19.             curr_sum += curr_term;
20.         }
21.
22.         res *= curr_sum;
23.     }
24.
25.     // This condition is to handle
26.     // the case when n is a prime
27.     // number greater than 2.
28.     if (n >= 2)
29.         res *= (1 + n);
30.
31.     return res;
32. }
```

## 3. Smith number

**Given a number n, the task is to find out whether this number is smith or not. A Smith Number is a composite number whose sum of digits is equal to the sum of digits in its prime factorization.**

```
Input   : n = 4                          Output : Yes
Input    : n = 666                       Output   : Yes
```

### Code:

```
1.  bool isSmith(int n)
2.  {
3.    int original_no = n;
4.    int pDigitSum = 0;
5.    for (int i = 0; primes[i] <= n/2; i++)
6.    {
7.       while (n % primes[i] == 0)
8.       {
9.          int p = primes[i];
10.            n = n/p;
11.            while (p > 0)
12.            {
13.               pDigitSum += (p % 10);
14.               p = p/10;
15.            }
16.         }
17.      }
18.      if (n != 1 && n != original_no)
```

```
19.     {
20.         while (n > 0)
21.         {
22.             pDigitSum = pDigitSum + n%10;
23.             n = n/10;
24.         }
25.     }
26.     int sumDigits = 0;
27.     while (original_no > 0)
28.     {
29.         sumDigits = sumDigits + original_no % 10;
30.         original_no = original_no/10;
31.     }
32.     return (pDigitSum == sumDigits);
33. }
34.
```

# 4. Fibonacci numbers

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, .....

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation.

$$Fn = Fn-1 + Fn-2$$

with seed values

$$F0 = 0 \text{ and } F1 = 1.$$

**Q: Given a number n,print n-th fibonacci number.**

**Method 1:(using dynamic programming approach)**

**Code:**

```
#include <iostream>
int fib(int n)
{
  int f[n + 2];
  int i;
  f[0] = 0;
  f[1] = 1;
  for(i = 2; i <= n; i++)
  {
      f[i] = f[i - 1] + f[i - 2];
    }
    return f[n];
  }
```

```
14.
15.   Method 2:(using space optimized method)
16.   Code:
17.   int fib(int n)
18.   {
19.     int a = 0, b = 1, c, i;
20.     if( n == 0)
21.        return a;
22.     for(i = 2; i <= n; i++)
23.     {
24.       c = a + b;
25.       a = b;
26.       b = c;
27.     }
28.     return b;
29.   }
```

**Method 3:(using direct formula)**

$F_n = \{[(\sqrt{5} + 1)/2] \wedge n\} / \sqrt{5}$

**Code:**

```
int fib(int n) {
  double phi = (1 + sqrt(5)) / 2;
  return round(pow(phi, n) / sqrt(5));
}
```

**Application:**

**It is used in nth catlan numbers,Bell number(no of way to partition a number), binomial coefficient , permutation coefficient, gold mine problem,etc.**
**It is also used in architecture, where it is known as golden ratio.**

## 5.Zeckendorf's Theorem (Non-Neighbouring Fibonacci Representation)

**It states that every positive integer can be written uniquely as a sum of distinct non-neighbouring Fibonacci numbers. Two Fibonacci numbers are neighbours if they one come after the other in Fibonacci Sequence (0, 1, 1, 2, 3, 5, ..). For example, 3 and 5 are neighbours, but 2 and 5 are not.**

**Input: 10** **output: 8 2**
**Input: 30** **output: 21 8 1**

**Approach:**
**While n >= 0**
**a) Find the greatest Fibonacci Number smaller than n.**
**Let this number be 'f'. Print 'f'**
**b) n = n - f**

**Code:**

```
1. #int nearestSmallerEqFib(int n)
2. {
3.    if (n == 0 || n == 1)
4.
5.       return n;
```

```
6.    int f1 = 0, f2 = 1, f3 = 1;
7.    while (f3 <= n)
8.    {
9.       f1 = f2;
10.          f2 = f3;
11.          f3 = f1 + f2;
12.       }
13.       return f2;
14.    }
15.    void printFibRepresntation(int n)
16.    {
17.       while (n > 0)
18.       {
19.          int f = nearestSmallerEqFib(n);
20.          cout << f << " ";
21.          n = n - f;
22.       }
23.    }
```

# 6.Catalan Number

Catalan numbers are a sequence of natural numbers that occurs in many interesting counting problems like the following.

1. Count the number of expressions containing n pairs of parentheses which are correctly matched. For n = 3, possible expressions are ((())), ()(()), ()()(), (())(), (()()).
2. Count the number of possible Binary Search Trees with n keys
3. Count the number of full binary trees (A rooted binary tree is full if every vertex has either two children or no children) with n+1 leaves.
4. Given a number n, return the number of ways you can draw n chords in a circle with 2 x n points such that no 2 chords intersect.

Catlan number = $({}_{2n}C_n)(n+1)$
Method 1:(using binomial coefficient)

```
1.
2.  unsigned long int binomialCoeff(unsigned int n, unsigned int k)
3.  {
4.     unsigned long int res = 1;
5.     if (k > n - k)
6.        k = n - k;
7.    for (int i = 0; i < k; ++i) {
8.        res *= (n - i);
9.        res /= (i + 1);
10.       }
```

```
11.     return res;
12.   }
13.   unsigned long int catalan(unsigned int n)
14.   {
15.     unsigned long int c = binomialCoeff(2 * n, n);
16.     return c / (n + 1);
17.   }
18.   Method 2:(using dynamic solution)
19.   unsigned long int catalanDP(unsigned int n)
20.   {   unsigned long int catalan[n + 1];
21.      catalan[0] = catalan[1] = 1;
22.     for (int i = 2; i <= n; i++) {
23.        catalan[i] = 0;
24.        for (int j = 0; j < i; j++)
25.           catalan[i] += catalan[j] * catalan[i - j - 1];
26.     }
27.     return catalan[n];
28.   }
29.
```