# LAB 3

**AIM: Implementation of Monte-Carlo Type of Randomized Primality Testing Algorithm.**

- Fermat's Primality Testing is Monte-Carlo Algorithm to check for Large Prime Numbers in computational feasible way.

  If n is a prime number, then for every a, 1<a<n-1.$a^{n-1}$ mod n =1. (a<a<n-1

- **Observation :**
- If n is a prime, then it is always true that $a^{n-1}$ mod n=1.
- But Sometimes, even if n is not prime (composite), then also for some values of 'a', we are getting $a^{n-1}$ mod n=1.(i.e. n=15)

- **Randomized Primality Testing Algorithm using Fermat's Test :**

**<u>CODE:</u>**

```
#include<bits/stdc++.h>
using namespace std;
long long gcd(long a, long long b)
{
   if(a < b)
   {
         return gcd(b, a);
   }
   else if(a%b == 0)
   {
         return b;
   }
   else
   {
```

```
        return gcd(b, a%b);
    }

}

long long power(long long  a, unsigned long long n, long long p)
{
    long long res = 1;
    a = a % p;

    while (n > 0)
    {

        if (n & 1)
        {
            res = (res*a) % p;
        }
        n = n>>1;
        a = (a*a) % p;
    }
    return res;
}

bool isPrime(long long n,long long k)
{
    if(n<=1 || n==4)
    {
        return false;
    }
    if(n<=3)
    {
        return true ;
    }
    while(k>0)
    {
        long long a;
        a=2+rand()%(n-4);
        if(gcd(n,a)!=1)
        {
            return false;
```

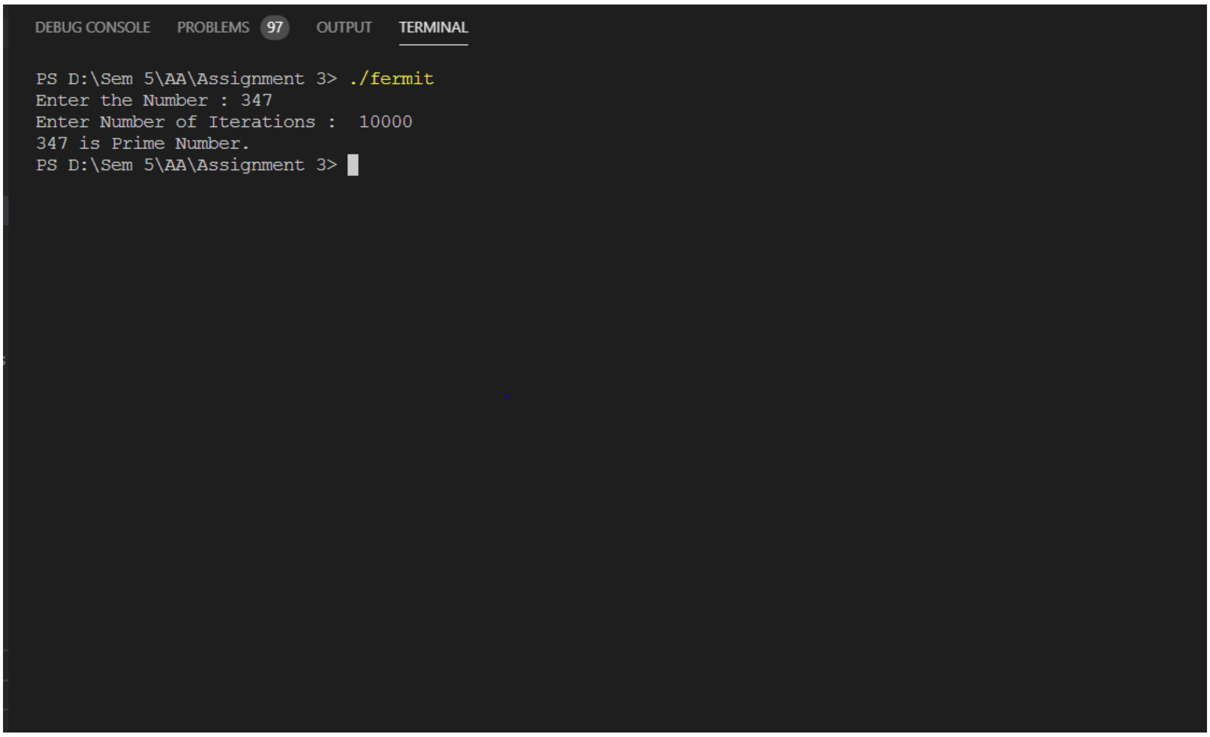
```
        }
        if(power(a,n-1,n)!=1)
        {
            return false;
        }
        k--;
    }
    return true;
}

int main()
{
    long long n,k;
    cout << "Enter the Number : ";
    cin >> n;


    cout << "Enter Number of Iterations :  ";
    cin >> k;

    bool prime=isPrime(n,k);
    if(prime==true)
    {
        cout << n << " is Prime Number.";
    }
    else
    {
        cout << n << " is Composite Number.";
    }
    return 0;
}
```

## OUTPUT:




- **Time Complexity:**
  O(klogn)