**Theory:**

Given a prime number n, the task is to find its primitive root under modulo n. The primitive root of a prime number n is an integer r between [1, n-1] such that the values of r^x (mod n) where x is in the range [0, n-2] are different. -1 if n is a non-prime number.

Example:

n = 7

r = 3 then,

30 (mod 7) = 1

31 (mod 7) = 3

32 (mod 7) = 2

33 (mod 7) = 6

34 (mod 7) = 4

35 (mod 7) = 5

**Programming Language: C**

**IDE: VS-Code**

**Code :**

#include <stdio.h>

#include <stdbool.h>

#include <math.h>

#include <string.h>

#include <stdlib.h>

bool isPrime(int n) {

int i;

if (n <= 1) return false;

if (n <= 3) return true;

if (n%2 == 0 || n%3 == 0) return false;

for (i = 5; i <= sqrt(n); i = i + 6)

if (n%i == 0 || n%(i+2) == 0)

return false;

return true;

}

int power(int x, unsigned int y, int p) {

int res = 1;

x = x % p;

while (y > 0) {

if (y & 1)

res = (res\*x) % p;

y = y >> 1;

x = (x\*x) % p;

}

return res;

}

void findPrimefactors(int\* s, int\* size, int n) {

int i;

while (n % 2 == 0) {

s[(\*size)++] = 2;

n = n / 2;

}

for (i = 3; i <= sqrt(n); i = i + 2) {

while (n % i == 0) {

s[(\*size)++] = i;

n = n / i;

}

}

if (n > 2)

s[(\*size)++] = n;

}

int findPrimitive(int n) {

int r, i;

int s[20];

int size = 0;

if (!isPrime(n)) return -1;

int phi = n - 1;

findPrimefactors(s, &size, phi);

for (r = 1; r <= phi; r++) {

bool flag = true;

for (i = 0; i < size; i++) {

if (power(r, phi / s[i], n) == 1) {

flag = false;

break;

}

}

if (flag == true)

return r;

}

return -1;

}

int main() {

int n;

printf("Enter the number: ");

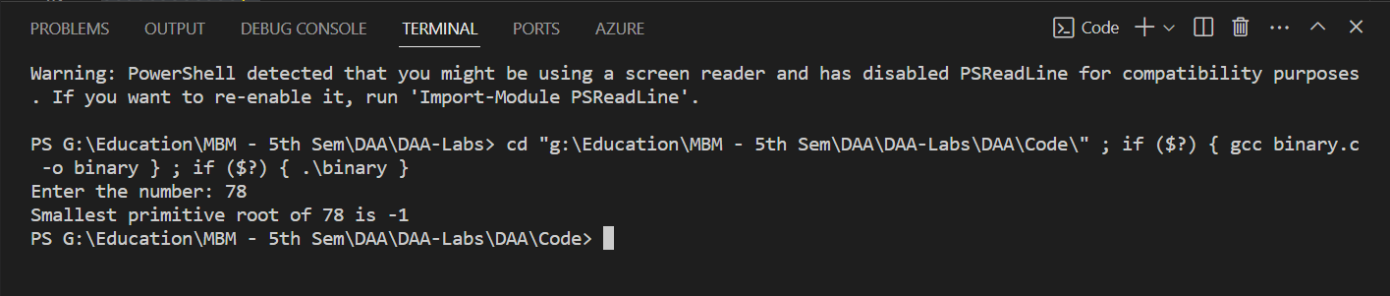
scanf("%d", &n);

printf("Smallest primitive root of %d is %d\n", n, findPrimitive(n));

return 0;

}

**Output**

****

**A computer screen shot of a black background

Description automatically generated**