Date:22.11.2022

**Final Year B. Tech., Sem VII 2022-23**

**Cryptography And Network Security Lab**

**Assignment submission**

**PRN No: 2019BTECS00022**

**Full name: Prachi R. Chobhare**

**Batch: B6**

**Assignment: 11**

**Title of assignment: Implementation of Prime Factorization of Coprime Number**

**Title:**

Implementation of Prime Factorization of Coprime Number

**Aim:**

To develop and implement Prime Factorization of Coprime Number

**Theory:**

* In mathematics, the Euclidean algorithm, or Euclid's algorithm, is an efficient method for computing the greatest common divisor (GCD) of two integers (numbers), the largest number that divides them both without a remainder.
* The Euclidean algorithm is based on the principle that the greatest common divisor of two numbers does not change if the larger number is replaced by its difference with the smaller number. For example, 21 is the GCD of 252 and 105 (as 252 = 21 × 12 and 105 = 21 × 5), and the same number 21 is also the GCD of 105 and 252 − 105 = 147.
* By using the extended Euclidean algorithm, the GCD can be expressed as a linear combination of the two original numbers, that is the sum of the two numbers, each multiplied by an integer (for example, 21 = 5 × 105 + (−2) × 252). The fact that the GCD can always be expressed in this way is known as Bézout's identity.
* The Euclidean algorithm has many theoretical and practical applications. It is used for reducing fractions to their simplest form and for performing division in modular arithmetic. Computations using this algorithm form part of the cryptographic protocols that are used to secure internet communications, and in methods for breaking these cryptosystems by factoring large composite numbers.
* In arithmetic and computer programming, the extended Euclidean algorithm is an extension to the Euclidean algorithm, and computes, in addition to the greatest common divisor (gcd) of integers a and b, also the coefficients of Bézout's identity, which are integers x and y such that



* The extended Euclidean algorithm is particularly useful when a and b are coprime. With that provision, x is the modular multiplicative inverse of a modulo b, and y is the modular multiplicative inverse of b modulo a.

**Implementation of Prime Factorization of Coprime Number**

**Code:**

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef vector<long long> vl;

string longDivision(string number, ll divisor)

{

// As result can be very large store it in string

string ans;

// Find prefix of number that is larger

// than divisor.

ll idx = 0;

ll temp = number[idx] - '0';

while (temp < divisor)

temp = temp \* 10 + (number[++idx] - '0');

// Repeatedly divide divisor with temp. After

// every division, update temp to include one

// more digit.

while (number.size() > idx) {

// Store result in answer i.e. temp / divisor

ans += (temp / divisor) + '0';

// Take next digit of number

temp = (temp % divisor) \* 10 + number[++idx] - '0';

}

// If divisor is greater than number

if (ans.length() == 0)

return "0";

// else return ans

return ans;

}

string multiply(string num1, string num2)

{

int len1 = num1.size();

int len2 = num2.size();

if (len1 == 0 || len2 == 0)

return "0";

// will keep the result number in vector

// in reverse order

vector<int> result(len1 + len2, 0);

// Below two indexes are used to find positions

// in result.

int i\_n1 = 0;

int i\_n2 = 0;

// Go from right to left in num1

for (int i = len1 - 1; i >= 0; i--)

{

int carry = 0;

int n1 = num1[i] - '0';

// To shift position to left after every

// multiplication of a digit in num2

i\_n2 = 0;

// Go from right to left in num2

for (int j = len2 - 1; j >= 0; j--)

{

// Take current digit of second number

int n2 = num2[j] - '0';

// Multiply with current digit of first number

// and add result to previously stored result

// at current position.

int sum = n1 \* n2 + result[i\_n1 + i\_n2] + carry;

// Carry for next iteration

carry = sum / 10;

// Store result

result[i\_n1 + i\_n2] = sum % 10;

i\_n2++;

}

// store carry in next cell

if (carry > 0)

result[i\_n1 + i\_n2] += carry;

// To shift position to left after every

// multiplication of a digit in num1.

i\_n1++;

}

// ignore '0's from the right

int i = result.size() - 1;

while (i >= 0 && result[i] == 0)

i--;

// If all were '0's - means either both or

// one of num1 or num2 were '0'

if (i == -1)

return "0";

// generate the result string

string s = "";

while (i >= 0)

s += to\_string(result[i--]);

return s;

}

ll isPrime(ll n)

{

// Corner case

if (n <= 1)

return 0;

// Check from 2 to square root of n

for (ll i = 2; i <= sqrt(n); i++)

if (n % i == 0)

return 0;

return 1;

}

int main()

{

ll t = 1;

//cin >> t;

string s;

cout<<"Enter the Number:\n";

cin >> s;

ll till = 100000;

for (ll i = 1; i < till; i++)

{

//cout << i << endl;

if (isPrime(i) == 0)

{

continue;

}

ll first = i;

string fs = to\_string(first);

string x = longDivision(s, i);

if (multiply(fs, x) != s)

continue;

cout << "\*\*" << endl;

cout << first << endl;

cout << x << endl;

cout << endl;

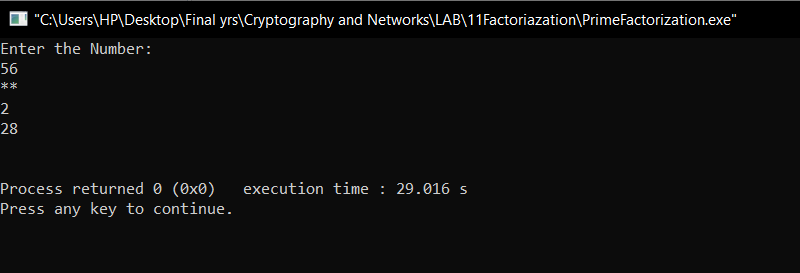
break;

}

return 0;

}

**Output:**



**Conclusion:**

Performed the experiment successfully.

The Euclidean and Extended Euclidean algorithm are used to find the GCD of numbers and the Multiplicative inverse of two coprime numbers respectively.