**RNS INSTITUTE OF TECHNOLOGY**

**CRYPTOGRAPHY ASSIGNMENT 2019**

**TOPICS:**

**CHINESE REMIDER THEOREM**

**FERMAT’S THEOREM**

**PRIMALITY TEST**

**EULER’S THEOREM**

**By-**

**Vivek B A (1RN16EC178)**

**Keshav V Bharadwaj (1RN16EC164)**

**Vivek Urankar (1RN16EC181)**

**Chiranjit R Patel (1RN16EC182)**

**Chinese Reminder Theorem**

**#include<stdio.h>**

**int crtx(int n[], int r[], int k)**

**{**

**int x = 1;**

**while(1)**

**{**

**int j; // Check if remainder of x % num[j] is rem[j]**

**for (j=0; j<k; j++ )**

**if (x%n[j] != r[j])**

**break;**

**// printf("%d ",x);**

**if (j == k) // If all remainder match,found x**

**return x;**

**x++; // Else try next numner**

**// printf("%d \n",x);**

**}**

**return x;**

**}**

**int main(void)**

**{**

**int y=0;**

**int r[] = {5, 3, 10}; //Remainders**

**int n[] = {7,11,13}; //NUMBERS: (gcd is 1 between them)**

**int k = sizeof(n)/sizeof(n[0]); //memory allocated**

**//printf("%ld",sizeof(n));**

**y=crtx(n,r,k);**

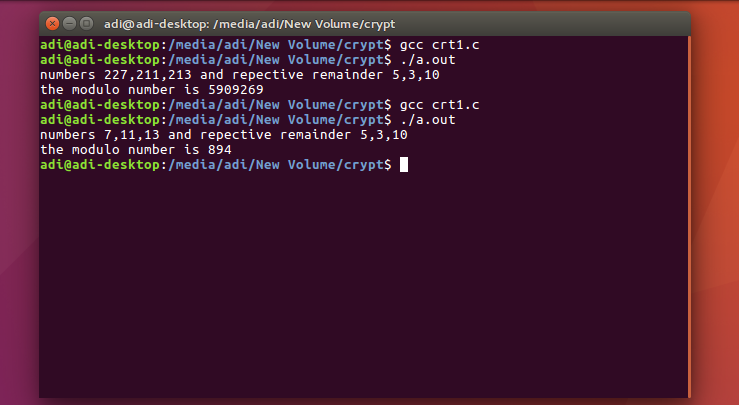
**printf("numbers %d,%d,%d and repective remainder %d,%d,%d \n",n[0],n[1],n[2],r[0],r[1],r[2]);**

**printf("the modulo number is %d \n",y);**

**return 0;**

**}**

**CRT output:**



**Fermat’s Mod Inverse**

**#include <stdio.h>**

**int gcd;**

**int pw(int x,unsigned int y,unsigned int m);**

**int gcdeuclid(int n1,int n2) // Finding the GCD of two numbers**

**{**

**for(int i=1; i <= n1 && i <= n2; ++i)**

**{**

**if(n1%i==0 && n2%i==0)**

**gcd = i;**

**}**

**//printf("%d",gcd);**

**return gcd;**

**}**

**void modinv(int a,int m)**

**{**

**if (gcdeuclid(a, m) != 1)**

**{**

**printf("%d^-1 mod %d \n",a,m);**

**printf("no inv \n");**

**}**

**else**

**{**

**printf("%d^-1 mod %d \n",a,m);**

**printf("mul inv %d \n", pw(a,m-2,m)); // mod inv is a^(m-2) mod m**

**}**

**}**

**int pw(int x,unsigned int y,unsigned int m) //compute x^y under mod m**

**{**

**if (y == 0)**

**return 1;**

**int p = pw(x, y / 2, m) % m;**

**p = (p \* p) % m;**

**return (y % 2 == 0) ? p : (x \* p) % m;**

**}**

**int main()**

**{**

**int a,m; // Enter the inverse number and modulo**

**printf("Enter inverse and modulo");**

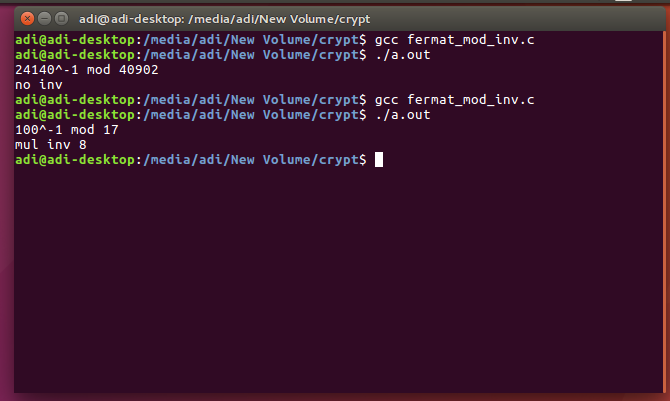
**scanf("%d,%d",&a,&m);**

**modinv(a, m);**

**return 0;**

**}**

**Fermat’s mod inverse output**

****

**Algorithm**

**1. Input the values ‘a’ and ‘m’.**

**2. GCD of (a,m) = 1**

**3. Find a-1**

**4. Perform inverse a-1 ≡ a m-2 (mod m)**

**5. Print result of inverse of a.**

**Fermat Prime**

**#include <stdio.h>**

**#include <stdlib.h>**

**long long modulo(long long base,long long exponent,long long mod)**

**{**

**long long x = 1;**

**long long y = base;**

**while (exponent > 0)**

**{**

**if (exponent % 2 == 1)**

**x = (x \* y) % mod; // Finding the modulo till the**

**y = (y \* y) % mod; //exponent is nonnegtive**

**exponent = exponent / 2;**

**}**

**return x % mod;**

**}**

**long long fermat(long long n, long long accu)**

**{**

**long long i;**

**if (n==1)**

**return 0;**

**for (i=0;i<accu;i++) // the the function for the given**

**{**

**long long a = rand()%(n-1)+1; // accuracy number of time**

**if (modulo(a,n-1,n)!=1)**

**return 0;**

**}**

**return 1;**

**}**

**long long main()**

**{**

**long long accu= 100; // accuracy**

**long long n;**

**printf("Enter number \n");**

**scanf("%lld",&n); // number in long long format**

**if (fermat(n, accu) == 1)**

**printf("prime \n");**

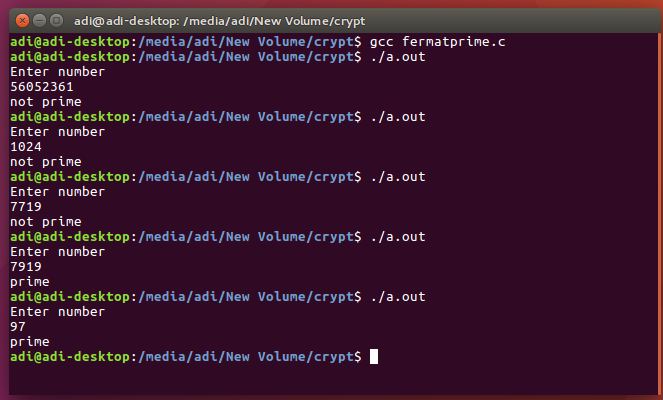
**else**

**printf("not prime \n");**

**return 0;**

**}**

**Fermat prime output**

****

**Algorithm**

**1. Input a number (‘n’)**

**2. Pick a random number ‘a’ which is less than ‘n’ (a<n)**

**3. Compute an-1 mod n, if the result is ‘1’, and then it’s probably a prime**

**4. If the result is not ‘1’ then return not a prime.**

**5. Repeat the above steps for more accuracy in prediction of prime number. The accuracy is input from the user, a value >0.**

**Totient function using Euler**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <math.h>**

**int gcd(int a, int b) //gcd of a and b**

**{**

**if (a == 0)**

**return b;**

**return gcd(b % a, a);**

**}**

**int phi(int num) //euler totient function**

**{**

**int totient = 1;**

**for (int i = 2; i < num; i++)**

**if (gcd(i, num) == 1)**

**totient++;**

**return totient;**

**}**

**int euler (int num, int power, int totient, int modulo)**

**{**

**int modp,eulertotient=0,x;**

**if(power>totient)**

**{**

**modp=power%totient; // reducing the power based on**

**printf("%d \n",modp); // num^totient mod modulo = 1**

**x=(int)(pow(num,modp)); // taking the reamining difference power**

**eulertotient=x%modulo; // the final module**

**}**

**else**

**{**

**printf("NA");**

**}**

**return eulertotient;**

**}**

**int main()**

**{**

**int num,res=0,power,totient,modulo;**

**printf("enter values \n");**

**scanf("%d,%d,%d",&num,&power,&modulo);**

**totient=phi(modulo);**

**printf("phi(%d)=%d\n",num,totient); // totient function**

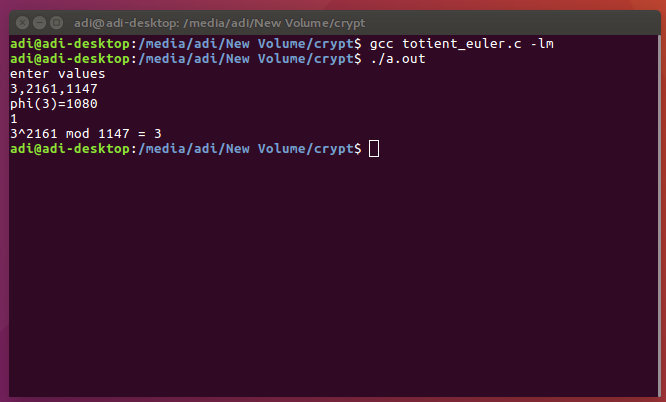
**res=euler(num,power,totient,modulo);**

**printf("%d^%d mod %d = %d \n",num,power,modulo,res); // final result**

**return 0;**

**}**

**Totient output**

****

**Fermat modular arithmetic**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <math.h>**

**int gcd(int a, int b) //gcd of a and b**

**{**

**if (a == 0)**

**return b;**

**return gcd(b % a, a);**

**}**

**int fermat(int a,int power, int m)**

**{**

**int modp,x,fermat\_mod;**

**if (gcd(a, m) == 1)**

**{**

**modp=power%(m-1);**

**x=(int)(pow(a,modp));**

**fermat\_mod=x%m;**

**}**

**else**

**printf("NA");**

**return fermat\_mod;**

**}**

**int main()**

**{**

**int a, m, power, res;**

**printf("Enter the number, power,and prime modulo\n");**

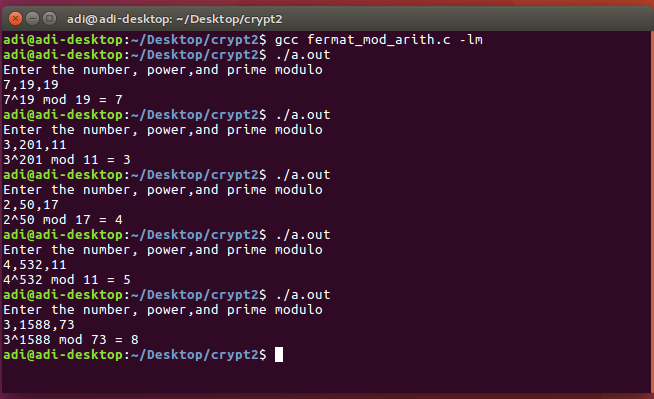
**scanf("%d,%d,%d",&a,&power,&m);**

**res=fermat(a,power,m);**

**printf("%d^%d mod %d = %d \n",a,power,m,res);**

**}**

**Fermat modular output**



Variable inverse

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

int gcd(int a, int b) //gcd of a and b

{

if (a == 0)

return b;

return gcd(b % a, a);

}

int fermatinv(int res,int power, int m)

{

int modp,x,fermatinv\_mod,i;

modp=power%(m-1);

for(i=0;i<m;i++)

{

if(((int)(pow(i,modp))%m==res))

{

fermatinv\_mod=i;

if(gcd(fermatinv\_mod,m)==1)

return fermatinv\_mod;

else

printf("NA");

}

}

return 0;

}

int main()

{

int a, m, power, res;

printf("Enter the result, power,and prime modulo (x^p mod n = res, find x?)\n");

scanf("%d,%d,%d",&res,&power,&m);

a=fermatinv(res,power,m);

printf("%d^%d mod %d = %d \n giving x as %d \n",a,power,m,res,a);

}

**Variable inverse output**

