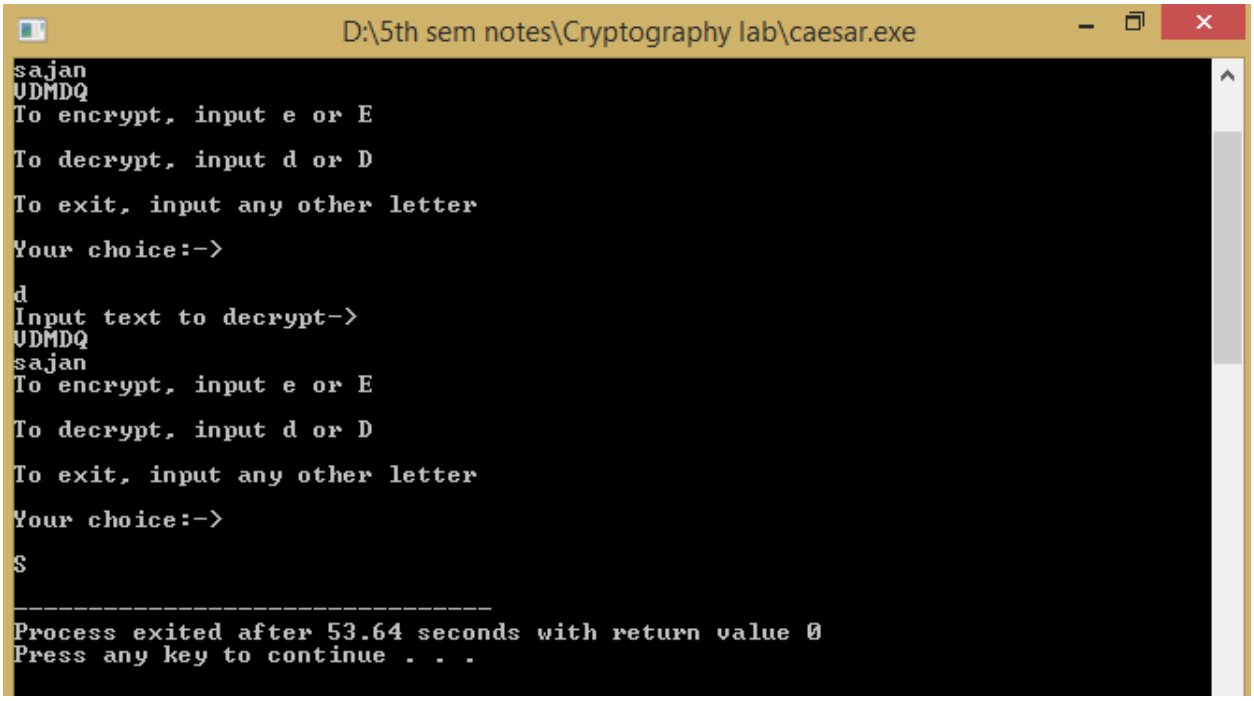
1. Write a program to implement Caesar Cipher.

```
#include <stdio.h>
#include <ctype.h>
 #define MAXSIZE 1024
 void encrypt(char*);
 void decrypt(char*);
 int menu();
 int main(void) {
 char c, choice[2], s[MAXSIZE];
 while (1) {
 menu();
 gets(choice);
 if ((choice[0] == 'e') || (choice[0] == 'E')) {
 puts("Input text to encrypt->");
 gets(s);
 encrypt(s);
      } else if ((choice[0] == 'd') || (choice[0] == 'D')) {
         puts("Input text to decrypt->");
         gets(s);
         decrypt(s);
      } else
         break;
   }
 return 0;
}
 void encrypt(char*str) {
   int n = 0;
   char *p = str, q[MAXSIZE];
   while (*p) {
    if (islower(*p)) {
         if ((*p >= 'a') && (*p < 'x'))
            q[n] = toupper(*p + (char) 3);
         else if (*p == 'x')
            q[n] = 'A';
         else if (*p == 'y')
            q[n] = 'B';
         else
            q[n] = 'C';
      } else {
         q[n] = *p;
      }
```

```
        n++;
        p++;
    }
    q[n++] = '\0';
    puts(q);
}
 void decrypt(char*str) {
    int n = 0;
    char *p = str, q[MAXSIZE];
 while (*p) {
        if (isupper(*p)) {
            if ((*p >= 'D') && (*p <= 'Z'))
                q[n] = tolower(*p - (char) 3);
            else if (*p == 'A')
                q[n] = 'x';
            else if (*p == 'B')
                q[n] = 'y';
            else
                q[n] = 'z';
        } else {
            q[n] = *p;
        }
        n++;
        p++;
    }
    q[n++] = '\0';
    puts(q);
}
 int menu() {
  puts("To encrypt, input e or E\n");
puts("To decrypt, input d or D\n");
puts("To exit, input any other letter\n");
puts("Your choice:->\n");
return 0;
}
```

Output:

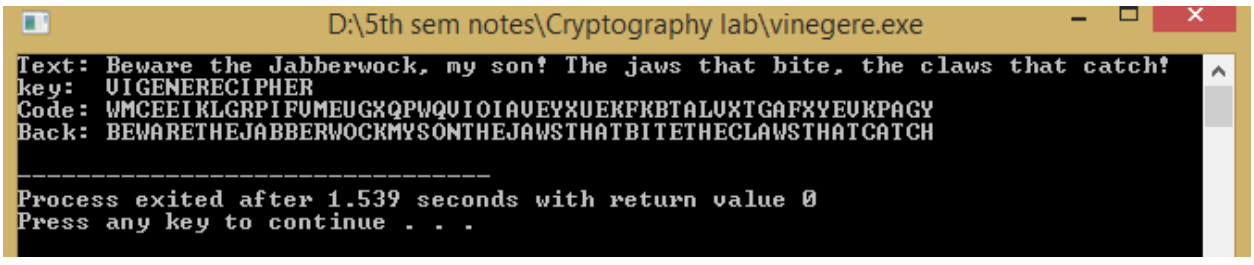


2. Write a program to implement vigenere cipher.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

void upper_case(char *src) {
   while (*src != '\0') {
      if (islower(*src))
         *src &= ~0x20;
      src++;
   }
}

char* encipher(const char *src, char *key, int is_encode) {
   int i, klen, slen;
   char *dest;

   dest = strdup(src);
   upper_case(dest);
   upper_case(key);
```

```
    for (i = 0, slen = 0; dest[slen] != '\0'; slen++)
        if (isupper(dest[slen]))
            dest[i++] = dest[slen];

    dest[slen = i] = '\0'; /* null pad it, make it safe to use */

    klen = strlen(key);
    for (i = 0; i < slen; i++) {
        if (!isupper(dest[i]))
            continue;
        dest[i] = 'A' + (is_encode ? dest[i] - 'A' + key[i % klen] - 'A'
                : dest[i] - key[i % klen] + 26) % 26;
    }

    return dest;
}

int main() {
    const char *str = "Beware the Jabberwock, my son! The jaws that bite, "
        "the claws that catch!";
    const char *cod, *dec;
    char key[] = "VIGENERECIPHER";
    printf("Text: %s\n", str);
    printf("key:  %s\n", key);
    cod = encipher(str, key, 1);
    printf("Code: %s\n", cod);
    dec = encipher(cod, key, 0);
    printf("Back: %s\n", dec);
    return 0;
}
```

Output:



3. Write a program to implement play fair cipher.

```
#include<stdio.h>

int check(char table[5][5], char k) {
   int i, j;
   for (i = 0; i < 5; ++i)
      for (j = 0; j < 5; ++j) {
         if (table[i][j] == k)
            return 0;
      }
   return 1;
}

int main() {
   int i, j, key_len;
   char table[5][5];
   for (i = 0; i < 5; ++i)
      for (j = 0; j < 5; ++j)
         table[i][j] = '0';

   printf("**********Playfair Cipher**********\n\n");

   printf("Enter the length of the Key. ");
   scanf("%d", &key_len);

   char key[key_len];

   printf("Enter the Key. ");
   for (i = -1; i < key_len; ++i) {
      scanf("%c", &key[i]);
      if (key[i] == 'j')
         key[i] = 'i';
   }

   int flag;
   int count = 0;

   // inserting the key into the table
   for (i = 0; i < 5; ++i) {
      for (j = 0; j < 5; ++j) {
         flag = 0;
         while (flag != 1) {
            if (count > key_len)
```

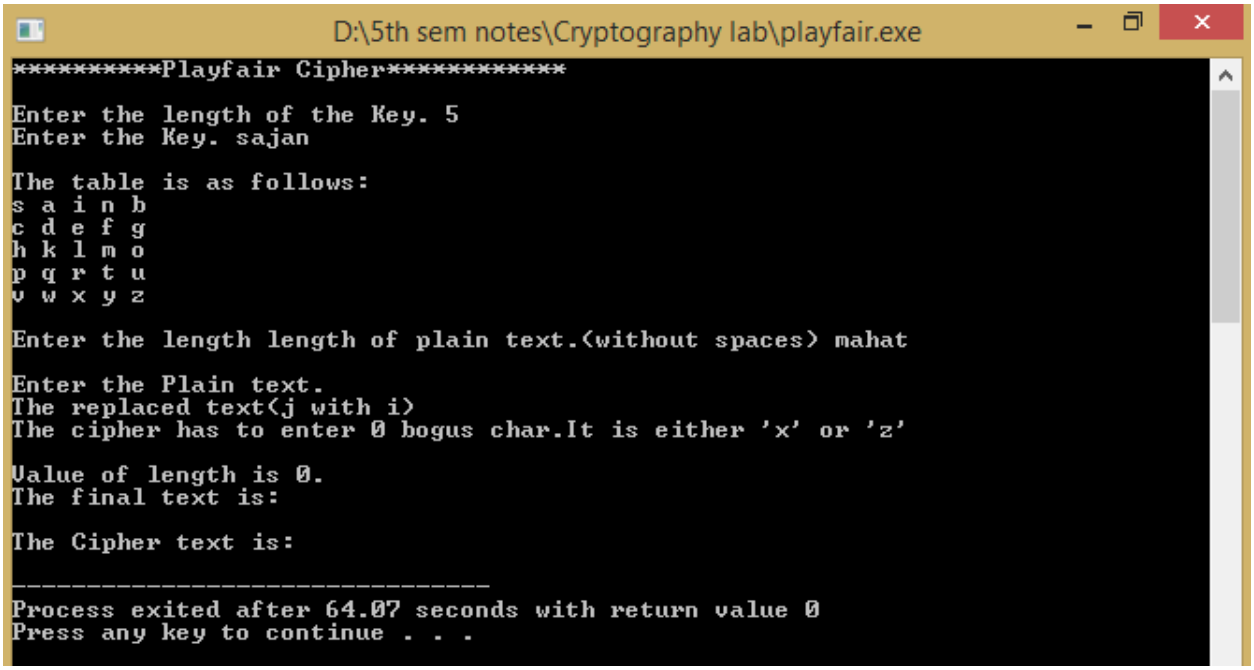
```
            goto l1;

         flag = check(table, key[count]);
         ++count;
      }// end of while
      table[i][j] = key[(count - 1)];
   }// end of inner for
}// end of outer for


l1: printf("\n");

int val = 97;
//inserting other alphabets
for (i = 0; i < 5; ++i) {
   for (j = 0; j < 5; ++j) {
      if (table[i][j] >= 97 && table[i][j] <= 123) {
      } else {
         flag = 0;
         while (flag != 1) {
            if ('j' == (char) val)
               ++val;
            flag = check(table, (char) val);
            ++val;
         }// end of while
         table[i][j] = (char) (val - 1);
      }//end of else
   }// end of inner for
}// end of outer for

printf("The table is as follows:\n");
for (i = 0; i < 5; ++i) {
   for (j = 0; j < 5; ++j) {
      printf("%c ", table[i][j]);
   }
   printf("\n");
}

int l = 0;
printf("\nEnter the length length of plain text.(without spaces) ");
scanf("%d", &l);
```

```
printf("\nEnter the Plain text. ");
char p[l];
for (i = -1; i < l; ++i) {
   scanf("%c", &p[i]);
}

for (i = -1; i < l; ++i) {
   if (p[i] == 'j')
      p[i] = 'i';
}

printf("\nThe replaced text(j with i)");
for (i = -1; i < l; ++i)
   printf("%c ", p[i]);

count = 0;
for (i = -1; i < l; ++i) {
   if (p[i] == p[i + 1])
      count = count + 1;
}

printf("\nThe cipher has to enter %d bogus char.It is either 'x' or 'z'\n",
      count);

int length = 0;
if ((l + count) % 2 != 0)
   length = (l + count + 1);
else
   length = (l + count);

printf("\nValue of length is %d.\n", length);
char p1[length];

//inserting bogus characters.
char temp1;
int count1 = 0;
for (i = -1; i < l; ++i) {
   p1[count1] = p[i];
   if (p[i] == p[i + 1]) {
      count1 = count1 + 1;
      if (p[i] == 'x')
         p1[count1] = 'z';
```

```
      else
         p1[count1] = 'x';
   }
   count1 = count1 + 1;
}

//checking for length

char bogus;
if ((l + count) % 2 != 0) {
   if (p1[length - 1] == 'x')
      p1[length] = 'z';
   else
      p1[length] = 'x';
}

printf("The final text is:");
for (i = 0; i <= length; ++i)
   printf("%c ", p1[i]);

char cipher_text[length];
int r1, r2, c1, c2;
int k1;

for (k1 = 1; k1 <= length; ++k1) {
   for (i = 0; i < 5; ++i) {
      for (j = 0; j < 5; ++j) {
         if (table[i][j] == p1[k1]) {
            r1 = i;
            c1 = j;
         } else if (table[i][j] == p1[k1 + 1]) {
            r2 = i;
            c2 = j;
         }
      }//end of for with j
   }//end of for with i

   if (r1 == r2) {
      cipher_text[k1] = table[r1][(c1 + 1) % 5];
      cipher_text[k1 + 1] = table[r1][(c2 + 1) % 5];
   }
```

```
      else if (c1 == c2) {
         cipher_text[k1] = table[(r1 + 1) % 5][c1];
         cipher_text[k1 + 1] = table[(r2 + 1) % 5][c1];
      } else {
         cipher_text[k1] = table[r1][c2];
         cipher_text[k1 + 1] = table[r2][c1];
      }

      k1 = k1 + 1;
   }//end of for with k1

   printf("\n\nThe Cipher text is:\n ");
   for (i = 1; i <= length; ++i)
      printf("%c ", cipher_text[i]);

}
```

Output:



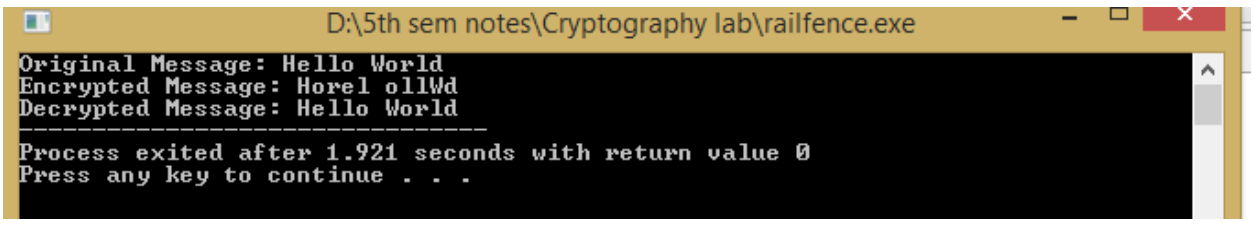
4. Write a program to implement railfence cipher.

```
#include<stdio.h>
#include<string.h>

void encryptMsg(char msg[], int key){
```

```
   int msgLen = strlen(msg), i, j, k = -1, row = 0, col = 0;
   char railMatrix[key][msgLen];

   for(i = 0; i < key; ++i)
      for(j = 0; j < msgLen; ++j)
         railMatrix[i][j] = '\n';

   for(i = 0; i < msgLen; ++i){
      railMatrix[row][col++] = msg[i];

      if(row == 0 || row == key-1)
         k= k * (-1);

      row = row + k;
   }

   printf("\nEncrypted Message: ");

   for(i = 0; i < key; ++i)
      for(j = 0; j < msgLen; ++j)
         if(railMatrix[i][j] != '\n')
            printf("%c", railMatrix[i][j]);
}

void decryptMsg(char enMsg[], int key){
   int msgLen = strlen(enMsg), i, j, k = -1, row = 0, col = 0, m = 0;
   char railMatrix[key][msgLen];

   for(i = 0; i < key; ++i)
      for(j = 0; j < msgLen; ++j)
         railMatrix[i][j] = '\n';

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

      railMatrix[row][col++] = '*';

      if(row == 0 || row == key-1)
         k= k * (-1);

      row = row + k;
   }
```

```
    for(i = 0; i < key; ++i)
        for(j = 0; j < msgLen; ++j)
            if(railMatrix[i][j] == '*')
                railMatrix[i][j] = enMsg[m++];

    row = col = 0;
    k = -1;

    printf("\nDecrypted Message: ");

    for(i = 0; i < msgLen; ++i){
        printf("%c", railMatrix[row][col++]);

        if(row == 0 || row == key-1)
            k= k * (-1);

        row = row + k;
    }
}

int main(){
    char msg[] = "Hello World";
    char enMsg[] = "Horel ollWd";
    int key = 3;

    printf("Original Message: %s", msg);

    encryptMsg(msg, key);
    decryptMsg(enMsg, key);

    return 0;
}
```

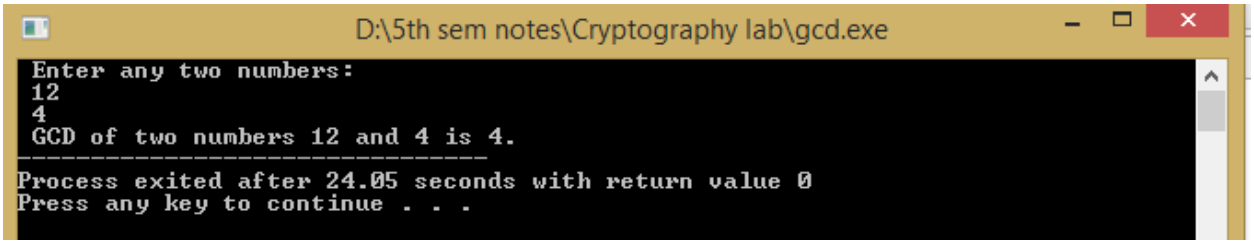
Output:



5. Write a program to compute GCD of two integers.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   // declare the variables
   int n1, n2, i, GCD_Num;
   printf ( " Enter any two numbers: \n ");
   scanf ( "%d %d", &n1, &n2);

   // use for loop
   for( i = 1; i <= n1 && i <= n2; ++i)
   {
      if (n1 % i ==0 && n2 % i == 0)
         GCD_Num = i; /* if n1 and n2 is completely divisible by i, the divisible number will be the
GCD_Num */
   }
   // print the GCD of two numbers
   printf (" GCD of two numbers %d and %d is %d.", n1, n2, GCD_Num);
   return 0;
}
```

Output:

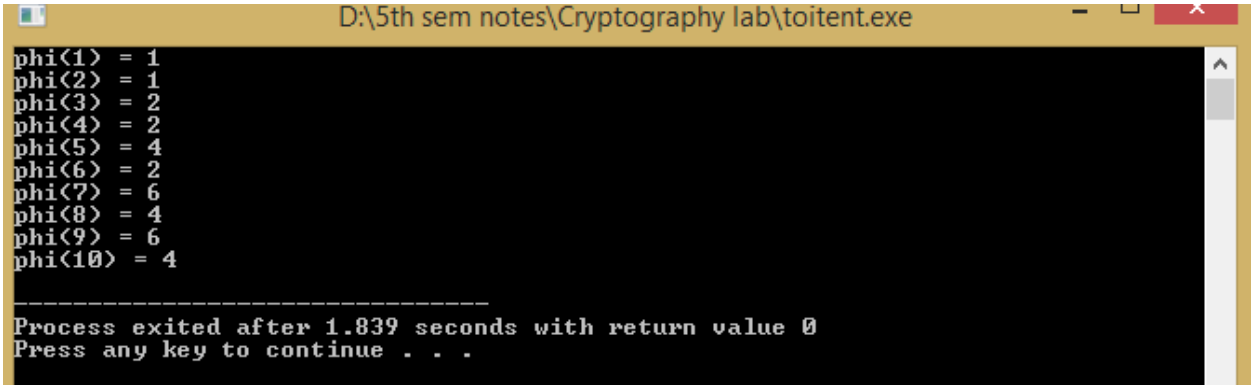


6. Write a program to compute Totient of a number.

```
// C program to calculate Euler's Totient Function
#include <stdio.h>

int phi(int n)
{
   int result = n; // Initialize result as n

   // Consider all prime factors of n and subtract their
   // multiples from result
```

```
    for (int p = 2; p * p <= n; ++p) {

        // Check if p is a prime factor.
        if (n % p == 0) {

            // If yes, then update n and result
            while (n % p == 0)
                n /= p;
            result -= result / p;
        }
    }

    // If n has a prime factor greater than sqrt(n)
    // (There can be at-most one such prime factor)
    if (n > 1)
        result -= result / n;
    return result;
}

// Driver program to test above function
int main()
{
    int n;
    for (n = 1; n <= 10; n++)
        printf("phi(%d) = %d\n", n, phi(n));
    return 0;
}
```

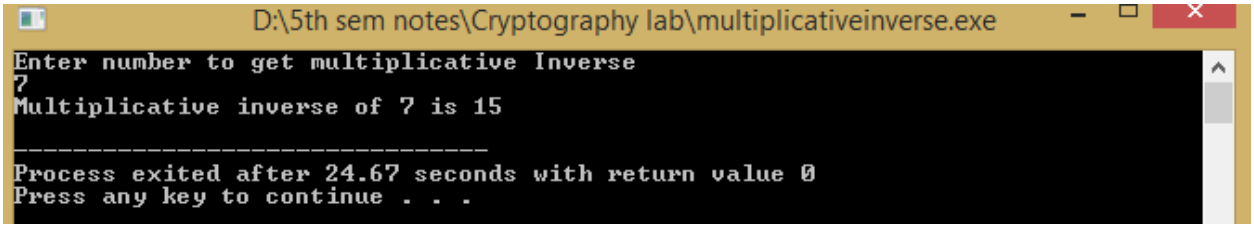
Output:



7. Write a program to compute multiplicative inverse of an integer.

```
#include<stdio.h>
main()
{
 int i,num,MI;
 printf("Enter number to get multiplicative Inverse\n");
 scanf("%d",&num);
 for(i=1;i<=num;i++)
 {
  MI=((i*26)+1);
  if(MI%num==0)
  {
   break;
  }
 }
 MI=MI/num;
 printf("Multiplicative inverse of %d is %d\n",num,MI);

}
```

Output:

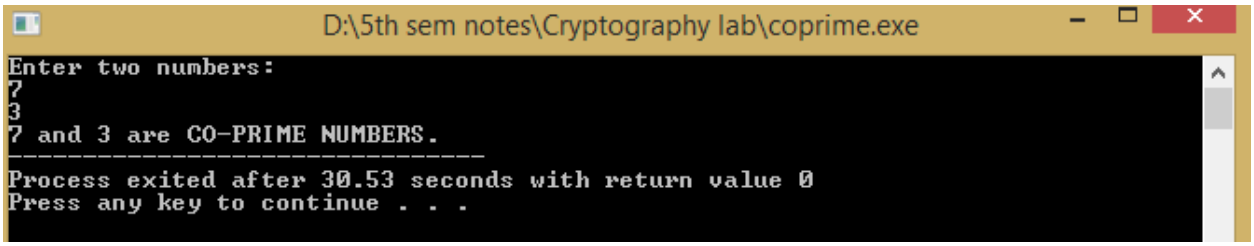


8. Write a program to check whether a given numbers are coprime or not.

```
#include<stdio.h>
#include<conio.h>

int main()
{
        int num1, num2, hcf, i;
        printf("Enter two numbers:\n");
        scanf("%d%d", &num1, &num2);
```

```
        // Finding HCF
        for(i=1;i<=num1;i++)
        {
         if(num1%i==0 && num2%i==0)
         {
          hcf = i;
         }
        }

        // Making Decision
        if(hcf == 1)
        {
         printf("%d and %d are CO-PRIME NUMBERS.", num1, num2);
        }
        else
        {
         printf("%d and %d are NOT CO-PRIME NUMBERS.", num1, num2);
        }
        getch();
        return(0);
}
```

Output:

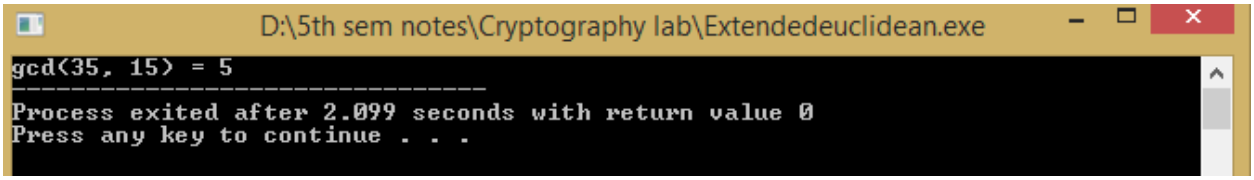


9. Write a program to implement Extended Euclidean algorithm.

```
// C program to demonstrate working of extended
// Euclidean Algorithm
#include <stdio.h>

// C function for extended Euclidean Algorithm
int gcdExtended(int a, int b, int* x, int* y)
{
```

```
    // Base Case
    if (a == 0) {
        *x = 0;
        *y = 1;
        return b;
    }

    int x1, y1; // To store results of recursive call
    int gcd = gcdExtended(b % a, a, &x1, &y1);

    // Update x and y using results of recursive
    // call
    *x = y1 - (b / a) * x1;
    *y = x1;

    return gcd;
}

// Driver Program
int main()
{
    int x, y;
    int a = 35, b = 15;
    int g = gcdExtended(a, b, &x, &y);
    printf("gcd(%d, %d) = %d", a, b, g);
    return 0;
}
```

Output:

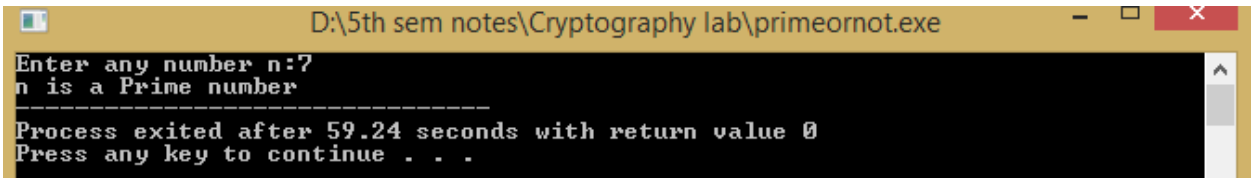


10. Write a program to check whether a given number is prime or not.

```
#include <stdio.h>

main() {
  int n, i, c = 0;
```

```
 printf("Enter any number n:");
 scanf("%d", &n);

 //logic
 for (i = 1; i <= n; i++) {
    if (n % i == 0) {
      c++;
    }
 }

 if (c == 2) {
     printf("n is a Prime number");
 }
 else {
      printf("n is not a Prime number");
 }
 return 0;
}
```

Output:



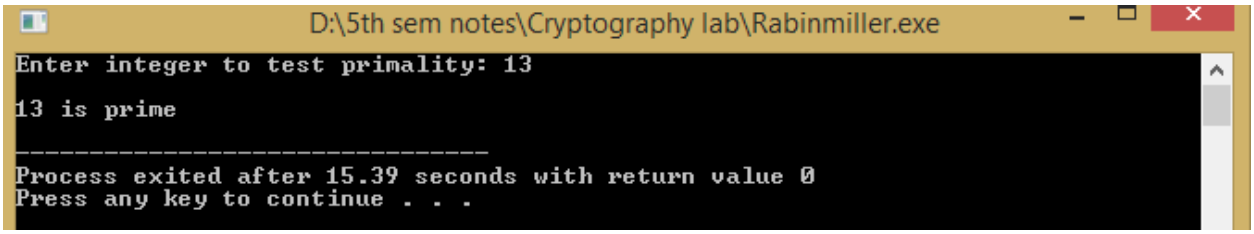
11. Write a program to perform primality checking using Rabin-Miller algorithm.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
/*
 * calculates (a * b) % c taking into account that a * b might overflow
 */
long long mulmod(long long a, long long b, long long mod)
{
   long long x = 0,y = a % mod;
   while (b > 0)
   {
      if (b % 2 == 1)
      {
```

```
            x = (x + y) % mod;
        }
        y = (y * 2) % mod;
        b /= 2;
    }
    return x % mod;
}
/*
 * modular exponentiation
 */
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;
        y = (y * y) % mod;
        exponent = exponent / 2;
    }
    return x % mod;
}

/*
 * Miller-Rabin Primality test, iteration signifies the accuracy
 */
int Miller(long long p,int iteration)
{

    int i;
    long long s;
    if (p < 2)
    {
        return 0;
    }
    if (p != 2 && p % 2==0)
    {
        return 0;
    }
     s = p - 1;
    while (s % 2 == 0)
```

```
    {
        s /= 2;
    }
    for (i = 0; i < iteration; i++)
    {
        long long a = rand() % (p - 1) + 1, temp = s;
        long long mod = modulo(a, temp, p);
        while (temp != p - 1 && mod != 1 && mod != p - 1)
        {
            mod = mulmod(mod, mod, p);
            temp *= 2;
        }
        if (mod != p - 1 && temp % 2 == 0)
        {
            return 0;
        }
    }
    return 1;
}
//Main
int main()
{
    int iteration = 5;
    long long num;
    printf("Enter integer to test primality: ");
    scanf("%lld", &num);
    if ( Miller( num, iteration))
        printf("\n%lld is prime\n", num);
    else
        printf("\n%lld is not prime\n", num);
    return 0;
}
```

Output:

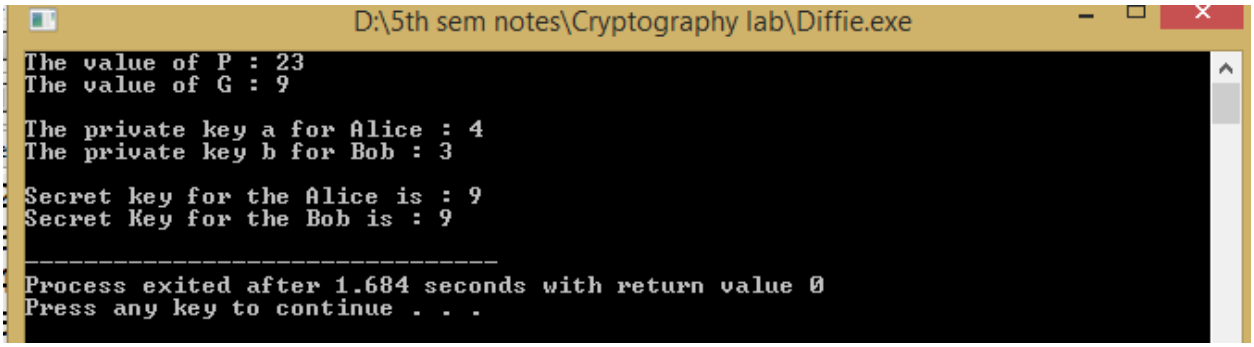


12. Write a program to implement Diffie-Hellman algorithm.

```
#include<stdio.h>
#include<math.h>
long long int power(long long int a, long long int b,
                                    long long int P)
{
        if (b == 1)
                return a;

        else
                return (((long long int)pow(a, b)) % P);
}

//Driver program
int main()
{
        long long int P, G, x, a, y, b, ka, kb;

        // Both the persons will be agreed upon the
                // public keys G and P
        P = 23; // A prime number P is taken
        printf("The value of P : %lld\n", P);

        G = 9; // A primitive root for P, G is taken
        printf("The value of G : %lld\n\n", G);

        // Alice will choose the private key a
        a = 4; // a is the chosen private key
        printf("The private key a for Alice : %lld\n", a);
        x = power(G, a, P); // gets the generated key

        // Bob will choose the private key b
        b = 3; // b is the chosen private key
        printf("The private key b for Bob : %lld\n\n", b);
        y = power(G, b, P); // gets the generated key

        // Generating the secret key after the exchange
                // of keys
        ka = power(y, a, P); // Secret key for Alice
```

```
        kb = power(x, b, P); // Secret key for Bob

        printf("Secret key for the Alice is : %lld\n", ka);
        printf("Secret Key for the Bob is : %lld\n", kb);

        return 0;
}
```

Output:



13. Write a program to perform key exchange and encryption-decryption using RSA algorithm.

```
#include<stdio.h>
#include<math.h>

//to find gcd
int gcd(int a, int h)
{
   int temp;
   while(1)
   {
      temp = a%h;
      if(temp==0)
      return h;
      a = h;
      h = temp;
   }
}

int main()
{
   //2 random prime numbers
```

```
double p = 3;
double q = 7;
double n=p*q;
double count;

double totient = (p-1)*(q-1);

//public key
//e stands for encrypt
double e=2;

//for checking co-prime which satisfies e>1
while(e<totient){
count = gcd(e,totient);
if(count==1)
   break;
else
   e++;
}

//private key
//d stands for decrypt
double d;

//k can be any arbitrary value
double k = 2;

//choosing d such that it satisfies d*e = 1 + k * totient
d = (1 + (k*totient))/e;
double msg = 12;
double c = pow(msg,e);
double m = pow(c,d);
c=fmod(c,n);
m=fmod(m,n);

printf("Message data = %lf",msg);
printf("\np = %lf",p);
printf("\nq = %lf",q);
printf("\nn = pq = %lf",n);
printf("\ntotient = %lf",totient);
printf("\ne = %lf",e);
printf("\nd = %lf",d);
```

```
    printf("\nEncrypted data = %lf",c);
    printf("\nOriginal Message Sent = %lf",m);

    return 0;
}
```

Output: