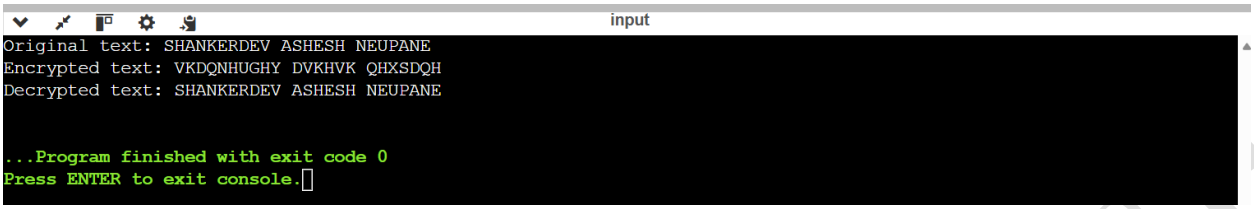
Encrypt and Decrypt the plaintext "SHANKERDEV ASHESH NEUPANE" using Caesar Cipher.

Source Code:

```
#include <stdio.h>
#include <ctype.h>
// Function for both encryption and decryption
void caesar(char text[], int key, int mode) {
    // mode = 1 → encrypt, mode = -1 → decrypt
    for (int i = 0; text[i] != '\0'; i++) {
        char ch = text[i];
        if (isupper(ch)) {
            text[i] = ((ch - 'A' + mode * key + 26) % 26) + 'A';
        }
        else if (islower(ch)) {
            text[i] = ((ch - 'a' + mode * key + 26) % 26) + 'a';
        }
    }
}
int main() {
    char text[] = "SHANKERDEV ASHESH NEUPANE";
    int key = 3;
    printf("Original text: %s\n", text);
    // Encrypt
    caesar(text, key, 1);
    printf("Encrypted text: %s\n", text);
    // Decrypt
    caesar(text, key, -1);
    printf("Decrypted text: %s\n", text);
    return 0;
}
```

Output Screen:



Consider a secure communication between you and your classmate using the Diffie Hellman key exchange algorithm. Assume:

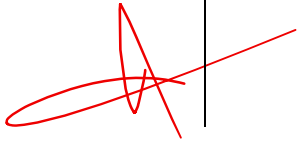
a) ***Your private key is your TU roll number: (XA)***
b) ***Your classmate's private key (XB) = 15***
c) ***Prime number (q) = 47***
d) ***Primitive root (α) = 5***

Write a C program to:

a) ***Generate public keys***
b) ***Exchange keys***
c) ***Compute the shared secret key***
d) ***Verify that both parties obtain the same key***

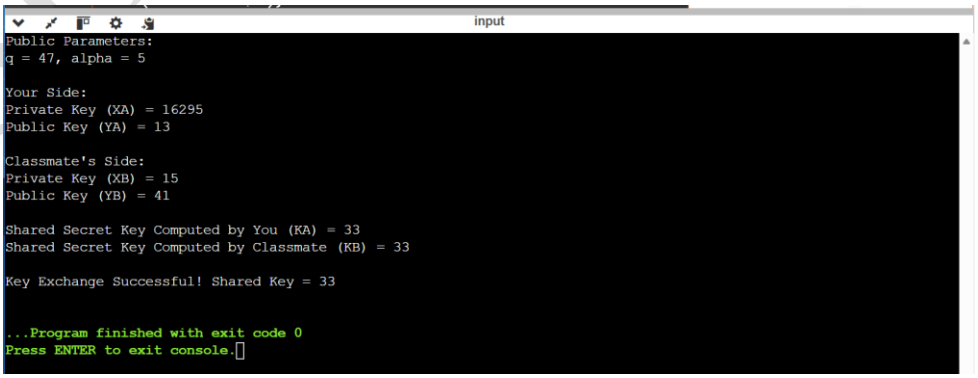
Source Code:

```
#include <stdio.h>
// Fast Modular Exponentiation
long long modExp(long long base, long long exp, long long mod) {
    long long result = 1;
    base = base % mod;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp = exp / 2;
    }
    return result;
}
int main() {
    // Given Parameters
    long long q = 47;       // Prime number
    long long alpha = 5;    // Primitive root
    // Private Keys
    long long XA = 16295;   // My TU Roll Number
    long long XB = 15;      // Classmate's private key
```



```
    // Public Keys
    long long YA = modExp(alpha, XA, q); // My public key
    long long YB = modExp(alpha, XB, q); // Classmate's public key
    // Shared Secret Key
    long long KA = modExp(YB, XA, q); // Computed by me
    long long KB = modExp(YA, XB, q); // Computed by classmate
    // Output
    printf("Public Parameters:\n");
    printf("q = %lld, alpha = %lld\n\n", q, alpha);
    printf("Your Side:\n");
    printf("Private Key (XA) = %lld\n", XA);
    printf("Public Key (YA) = %lld\n\n", YA);
    printf("Classmate's Side:\n");
    printf("Private Key (XB) = %lld\n", XB);
    printf("Public Key (YB) = %lld\n\n", YB);
    printf("Shared Secret Key Computed by You (KA) = %lld\n", KA);
    printf("Shared Secret Key Computed by Classmate (KB) = %lld\n", KB);
    if (KA == KB)
        printf("\nKey Exchange Successful! Shared Key = %lld\n", KA);
    else
        printf("\nKey Exchange Failed!\n");
    return 0;
}
```

Output Screen:



Consider secure communication using the RSA encryption algorithm. Given:

a) ***Prime numbers: p = 3, q = 47***
b) ***Public exponent: e = 5 (coprime with φ(n))***

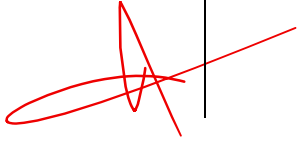
Tasks:

a) ***Compute n and phi (n)***
b) ***Find private key d***
c) ***Take your TU roll number as message (M)***
d) ***Encrypt the message***
e) ***Decrypt the ciphertext and verify original message***

Source Code:

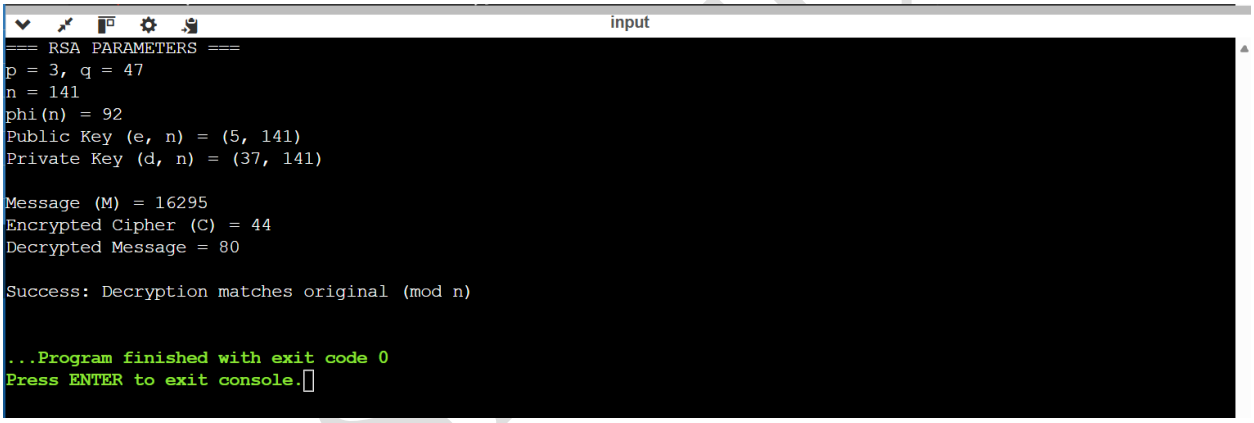
```
#include <stdio.h>

// Fast modular exponentiation
long long modExp(long long base, long long exp, long long mod) {
    long long result = 1;
    base = base % mod;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp = exp / 2;
    }
    return result;
}
// GCD function
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
```

```
}
// Find modular inverse (brute force method)
int findD(int e, int phi) {
    int d = 1;
    while ((d * e) % phi != 1) {
        d++;
    }
    return d;
}
int main() {
    // Given values
    int p = 3, q = 47;
    int e = 5;
    int M = 16295;   // TU Roll Number (message)
    int n = p * q;
    int phi = (p - 1) * (q - 1);
    // Validate e
    if (gcd(e, phi) != 1) {
        printf("Invalid e! Not coprime with phi(n)\n");
        return 1;
    }
    // Compute private key
    int d = findD(e, phi);
    // Encrypt
    long long C = modExp(M % n, e, n);
    // Decrypt
    long long decrypted = modExp(C, d, n);
    // Output
    printf("=== RSA PARAMETERS ===\n");
    printf("p = %d, q = %d\n", p, q);
    printf("n = %d\n", n);
    printf("phi(n) = %d\n", phi);
```



```
    printf("Public Key (e, n) = (%d, %d)\n", e, n);
    printf("Private Key (d, n) = (%d, %d)\n\n", d, n);
    printf("Message (M) = %d\n", M);
    printf("Encrypted Cipher (C) = %lld\n", C);
    printf("Decrypted Message = %lld\n", decrypted);
    if (decrypted == M % n)
        printf("\nSuccess: Decryption matches original (mod n)\n");
    else
        printf("\nFailure: Decryption mismatch\n");
    return 0;
}
```

Output Screen:



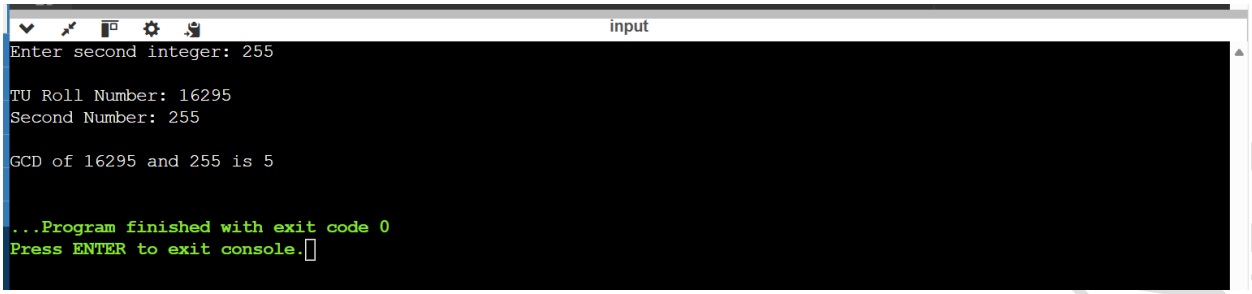
Write a C program to compute the Greatest Common Divisor (GCD) of two integers using the Euclidean algorithm. The program should:

a) ***Take two integers as input from the user, assume one input should be your TU roll number.***
b) ***Compute their GCD using a function***
c) ***Display the result***

Source Code:

```
#include <stdio.h>
// Function to compute GCD using Euclidean algorithm
int gcd(int a, int b) {
    while (b != 0) {
        int remainder = a % b;
        a = b;
        b = remainder;
    }
    return a;
}
int main() {
    int num1, num2;
    // TU Roll Number used as one input
    int tu_roll_number = 16295;
    printf("Enter second integer: ");
    scanf("%d", &num2);
    num1 = tu_roll_number;
    printf("\nTU Roll Number: %d", num1);
    printf("\nSecond Number: %d\n", num2);
    int result = gcd(num1, num2);
    printf("\nGCD of %d and %d is %d\n", num1, num2, result);
    return 0;
}
```

Output Screen:



Write a C program to showcase key generation in DES.

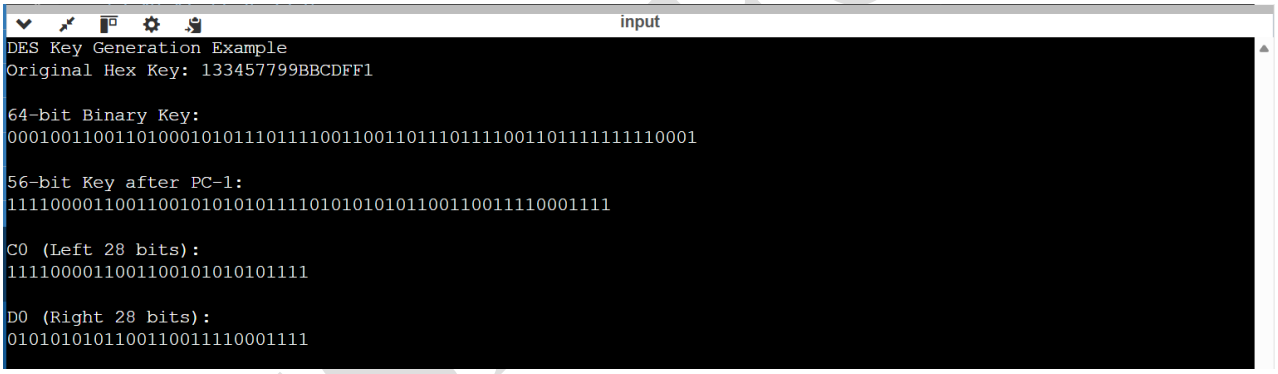
Source Code:

```
#include <stdio.h>
// PC-1 table (64-bit key → 56-bit key)
int PC1[56] = {
    57,49,41,33,25,17,9,
    1,58,50,42,34,26,18,
    10,2,59,51,43,35,27,
    19,11,3,60,52,44,36,
    63,55,47,39,31,23,15,
    7,62,54,46,38,30,22,
    14,6,61,53,45,37,29,
    21,13,5,28,20,12,4
};
// Convert hex character to 4-bit binary
void hexToBin(char hex, int bin[4]) {
    int val;
    if (hex >= '0' && hex <= '9')
        val = hex - '0';
    else
        val = hex - 'A' + 10;
    for (int i = 3; i >= 0; i--) {
        bin[i] = val % 2;
        val /= 2;
    }
}
// Convert full hex key to 64-bit binary
void convertToBinary(char hexKey[], int binKey[]) {
    int k = 0;
    for (int i = 0; i < 16; i++) {
        int temp[4];
```

```
        hexToBin(hexKey[i], temp);
        for (int j = 0; j < 4; j++) {
            binKey[k++] = temp[j];
        }
    }
}
// Apply PC-1 permutation
void applyPC1(int input[64], int output[56]) {
    for (int i = 0; i < 56; i++) {
        output[i] = input[PC1[i] - 1];
    }
}
// Print binary array
void printBits(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d", arr[i]);
    printf("\n");
}
int main() {
    char hexKey[17] = "133457799BBCDFF1";
    int key64[64];
    int key56[56];
    int C0[28], D0[28];
    printf("DES Key Generation Example\n");
    printf("Original Hex Key: %s\n\n", hexKey);
    // Step 1: Convert hex to binary
    convertToBinary(hexKey, key64);
    printf("64-bit Binary Key:\n");
    printBits(key64, 64);
    // Step 2: Apply PC-1 permutation
    applyPC1(key64, key56);
    printf("\n56-bit Key after PC-1:\n");
```

```
    printBits(key56, 56);
    // Step 3: Split into C0 and D0
    for (int i = 0; i < 28; i++) {
        C0[i] = key56[i];
        D0[i] = key56[i + 28];
    }
    printf("\nC0 (Left 28 bits):\n");
    printBits(C0, 28);
    printf("\nD0 (Right 28 bits):\n");
    printBits(D0, 28);
    return 0;
}
```

Output Screen:



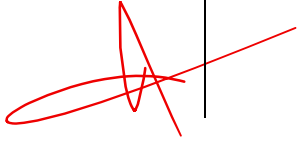
Write a C program that generate hash value of your name using MD4 hashing algorithm.

Source Code:

```
#include <stdio.h>
#include <string.h>
#include <stdint.h>
// Left rotate
#define LEFTROTATE(x, c) (((x) << (c)) | ((x) >> (32 - (c))))
// MD4 functions
#define F(x,y,z) ((x & y) | (~x & z))
#define G(x,y,z) ((x & y) | (x & z) | (y & z))
#define H(x,y,z) (x ^ y ^ z)
// Process one 512-bit block
void md4_transform(uint32_t state[4], uint8_t block[64]) {
    uint32_t a = state[0], b = state[1], c = state[2], d = state[3];
    uint32_t X[16];
    // Copy block into 16 words
    for (int i = 0; i < 16; i++) {
        X[i] = (uint32_t)block[i*4] |
               ((uint32_t)block[i*4+1] << 8) |
               ((uint32_t)block[i*4+2] << 16) |
               ((uint32_t)block[i*4+3] << 24);
    }
    // Round 1
    a = LEFTROTATE(a + F(b,c,d) + X[0], 3);
    d = LEFTROTATE(d + F(a,b,c) + X[1], 7);
    c = LEFTROTATE(c + F(d,a,b) + X[2], 11);
    b = LEFTROTATE(b + F(c,d,a) + X[3], 19);
    a = LEFTROTATE(a + F(b,c,d) + X[4], 3);
    d = LEFTROTATE(d + F(a,b,c) + X[5], 7);
    c = LEFTROTATE(c + F(d,a,b) + X[6], 11);
```

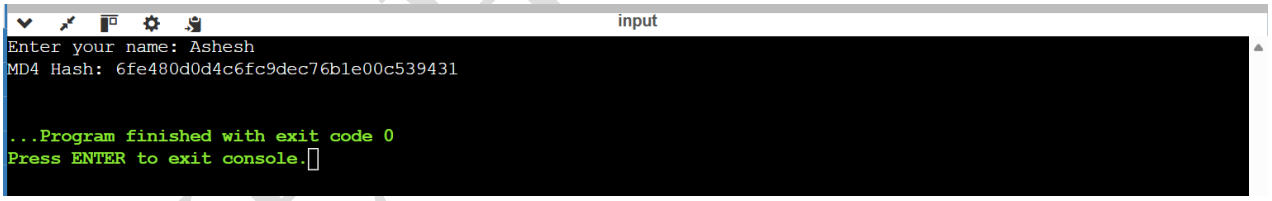
```
b = LEFTROTATE(b + F(c,d,a) + X[7], 19);
a = LEFTROTATE(a + F(b,c,d) + X[8], 3);
d = LEFTROTATE(d + F(a,b,c) + X[9], 7);
c = LEFTROTATE(c + F(d,a,b) + X[10], 11);
b = LEFTROTATE(b + F(c,d,a) + X[11], 19);
a = LEFTROTATE(a + F(b,c,d) + X[12], 3);
d = LEFTROTATE(d + F(a,b,c) + X[13], 7);
c = LEFTROTATE(c + F(d,a,b) + X[14], 11);
b = LEFTROTATE(b + F(c,d,a) + X[15], 19);
// Round 2
a = LEFTROTATE(a + G(b,c,d) + X[0] + 0x5A827999, 3);
d = LEFTROTATE(d + G(a,b,c) + X[4] + 0x5A827999, 5);
c = LEFTROTATE(c + G(d,a,b) + X[8] + 0x5A827999, 9);
b = LEFTROTATE(b + G(c,d,a) + X[12] + 0x5A827999, 13);
a = LEFTROTATE(a + G(b,c,d) + X[1] + 0x5A827999, 3);
d = LEFTROTATE(d + G(a,b,c) + X[5] + 0x5A827999, 5);
c = LEFTROTATE(c + G(d,a,b) + X[9] + 0x5A827999, 9);
b = LEFTROTATE(b + G(c,d,a) + X[13] + 0x5A827999, 13);
a = LEFTROTATE(a + G(b,c,d) + X[2] + 0x5A827999, 3);
d = LEFTROTATE(d + G(a,b,c) + X[6] + 0x5A827999, 5);
c = LEFTROTATE(c + G(d,a,b) + X[10] + 0x5A827999, 9);
b = LEFTROTATE(b + G(c,d,a) + X[14] + 0x5A827999, 13);
a = LEFTROTATE(a + G(b,c,d) + X[3] + 0x5A827999, 3);
d = LEFTROTATE(d + G(a,b,c) + X[7] + 0x5A827999, 5);
c = LEFTROTATE(c + G(d,a,b) + X[11] + 0x5A827999, 9);
b = LEFTROTATE(b + G(c,d,a) + X[15] + 0x5A827999, 13);
// Round 3
a = LEFTROTATE(a + H(b,c,d) + X[0] + 0x6ED9EBA1, 3);
d = LEFTROTATE(d + H(a,b,c) + X[8] + 0x6ED9EBA1, 9);
c = LEFTROTATE(c + H(d,a,b) + X[4] + 0x6ED9EBA1, 11);
b = LEFTROTATE(b + H(c,d,a) + X[12] + 0x6ED9EBA1, 15);
a = LEFTROTATE(a + H(b,c,d) + X[2] + 0x6ED9EBA1, 3);
```

```
    d = LEFTROTATE(d + H(a,b,c) + X[10] + 0x6ED9EBA1, 9);
    c = LEFTROTATE(c + H(d,a,b) + X[6] + 0x6ED9EBA1, 11);
    b = LEFTROTATE(b + H(c,d,a) + X[14] + 0x6ED9EBA1, 15);
    a = LEFTROTATE(a + H(b,c,d) + X[1] + 0x6ED9EBA1, 3);
    d = LEFTROTATE(d + H(a,b,c) + X[9] + 0x6ED9EBA1, 9);
    c = LEFTROTATE(c + H(d,a,b) + X[5] + 0x6ED9EBA1, 11);
    b = LEFTROTATE(b + H(c,d,a) + X[13] + 0x6ED9EBA1, 15);
    a = LEFTROTATE(a + H(b,c,d) + X[3] + 0x6ED9EBA1, 3);
    d = LEFTROTATE(d + H(a,b,c) + X[11] + 0x6ED9EBA1, 9);
    c = LEFTROTATE(c + H(d,a,b) + X[7] + 0x6ED9EBA1, 11);
    b = LEFTROTATE(b + H(c,d,a) + X[15] + 0x6ED9EBA1, 15);
    // Add back to state
    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
}
// Main MD4 function (simplified for short messages)
void md4(uint8_t *msg, int len) {
    uint32_t state[4] = {
        0x67452301,
        0xEFCDAB89,
        0x98BADCFE,
        0x10325476
    };
    uint8_t block[64] = {0};
    // Copy message
    memcpy(block, msg, len);
    // Padding
    block[len] = 0x80;
    // Append length (bits)
    uint64_t bitLen = len * 8;
```



```
    memcpy(block + 56, &bitLen, 8);
    // Process block
    md4_transform(state, block);
    // Output
    printf("MD4 Hash: ");
    for (int i = 0; i < 4; i++) {
        printf("%08x", state[i]);
    }
    printf("\n");
}
int main() {
    char input[100];
    printf("Enter your name: ");
    scanf("%s", input);
    md4((uint8_t*)input, strlen(input));
    return 0;
}
```

Output Screen:



Write a C program to encrypt the text "SHANKERDEV" using the key "teach" using the Vigenere Cipher.

Source Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main() {
    char text[] = "SHANKERDEV";
    char key[] = "teach";
    char cipher[100];
    int textLen = strlen(text);
    int keyLen = strlen(key);
    // Convert key to uppercase for consistency
    for (int i = 0; i < keyLen; i++) {
        key[i] = toupper(key[i]);
    }
    for (int i = 0; i < textLen; i++) {
        char t = text[i];
        char k = key[i % keyLen];
        // Encrypt using Vigenere formula
        cipher[i] = ((t - 'A') + (k - 'A')) % 26 + 'A';
    }
    cipher[textLen] = '\0';
    printf("Plain Text: %s\n", text);
    printf("Key: %s\n", key);
    printf("Cipher Text: %s\n", cipher);
    return 0;
}
```


Output Screen: