**Caesar Cipher** (Basic Substitution Cipher)

**Monoalphabetic Cipher** (Simple Substitution)

**Playfair Cipher** (Digraph Substitution)

**Hill Cipher** (Matrix-Based Cipher)

**Vigenère Cipher** (Polyalphabetic Cipher)

**DES (Data Encryption Standard)**

**AES (Advanced Encryption Standard)**

**Diffie-Hellman Key Exchange**

**MD5 Hashing**

**SHA-1 Hashing**

**SHA-256 Hashing**

**ElGamal Encryption**

**Blowfish Cipher**

## 1. Caesar Cipher

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

// Function to encrypt or decrypt using Caesar Cipher
void caesarCipher(char *text, int shift) {
    for (int i = 0; text[i] != '\0'; i++) {
        if (text[i] >= 'A' && text[i] <= 'Z') {
            text[i] = ((text[i] - 'A' + shift + 26) % 26) + 'A';
        }
        else if (text[i] >= 'a' && text[i] <= 'z') {
            text[i] = ((text[i] - 'a' + shift + 26) % 26) + 'a';
        }
    }
}

int main() {
    char text[100];
    int shift;

    // Input from user
    printf("Enter text: ");
    fgets(text, sizeof(text), stdin);
    text[strcspn(text, "\n")] = 0;  // Remove newline character

    printf("Enter shift value (1-25): ");
    scanf("%d", &shift);

    // Encryption
    caesarCipher(text, shift);
    printf("Encrypted: %s\n", text);

    // Decryption
    caesarCipher(text, -shift);
    printf("Decrypted: %s\n", text);

    return 0;
}
```

---

## 2. Vigenère Cipher

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

```
// Vigenère Cipher encryption function
void vigenereCipher(char *text, char *key, int encrypt) {
    int textLen = strlen(text), keyLen = strlen(key);
    for (int i = 0, j = 0; i < textLen; i++) {
        if (isalpha(text[i])) {
            char base = isupper(text[i]) ? 'A' : 'a';
            int shift = (key[j % keyLen] - 'A') * (encrypt ? 1 : -1);  // Encrypt or Decrypt
            text[i] = ((text[i] - base + shift + 26) % 26) + base;
            j++;  // Only move key index for letters
        }
    }
}

int main() {
    char text[100], key[100];

    // User input
    printf("Enter text: ");
    fgets(text, sizeof(text), stdin);
    text[strcspn(text, "\n")] = 0;  // Remove newline

    printf("Enter key: ");
    fgets(key, sizeof(key), stdin);
    key[strcspn(key, "\n")] = 0;

    // Convert key to uppercase
    for (int i = 0; key[i]; i++) {
        key[i] = toupper(key[i]);
    }

    // Encryption
    vigenereCipher(text, key, 1);
    printf("Encrypted: %s\n", text);

    // Decryption
    vigenereCipher(text, key, 0);
    printf("Decrypted: %s\n", text);

    return 0;
}
```

---

### 3. Hill Cipher (2x2 Matrix)

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

#define MOD 26

// Hill Cipher Encryption Function
void hillCipher(int key[2][2], char text[]) {
    int len = strlen(text);

    // If odd length, add a padding character 'X'
    if (len % 2 != 0) {
        text[len] = 'X';
        text[len + 1] = '\0';
        len++;
    }

    printf("Encrypted Text: ");
    for (int i = 0; i < len; i += 2) {
        int p1 = text[i] - 'A';
        int p2 = text[i + 1] - 'A';

        // Matrix multiplication
        int c1 = (key[0][0] * p1 + key[0][1] * p2) % MOD;
        int c2 = (key[1][0] * p1 + key[1][1] * p2) % MOD;

        // Convert numbers back to characters
        printf("%c%c", c1 + 'A', c2 + 'A');
    }
    printf("\n");
}

int main() {
    // Given key matrix
    int key[2][2] = {{9, 4}, {5, 7}};

    char text[] = "MEETMEATTHEUSUALPLACEATTENRATHERTHANEIGHTOCLOCK";

    // Convert to uppercase and remove spaces (already done in input)
    hillCipher(key, text);

    return 0;
}
```

MONO alphabetic cipher

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

// Substitution key for encryption (A-Z mapped to a shuffled alphabet)
char key[26] = "QWERTYUIOPASDFGHJKLZXCVBNM";

// Function to encrypt using Monoalphabetic Cipher
void monoalphabeticEncrypt(char text[]) {
    int len = strlen(text);
    printf("Encrypted Text: ");
    for (int i = 0; i < len; i++) {
        if (isalpha(text[i])) {
            char upper = toupper(text[i]);  // Convert to uppercase
            printf("%c", key[upper - 'A']); // Substitute with key mapping
        } else {
            printf("%c", text[i]); // Keep spaces or special characters unchanged
        }
    }
    printf("\n");
}
```

```
int main() {

    char text[] = "MEET ME AT THE USUAL PLACE AT TEN RATHER THAN
EIGHT OCLOCK";

    monoalphabeticEncrypt(text);

    return 0;

}
```

DES:

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

void xorEncrypt(char *text, char *key, int length) {
    int keyLen = strlen(key);
    for (int i = 0; i < length; i++) {
        text[i] ^= key[i % keyLen];
    }
}

void printHex(char *text, int length) {
    for (int i = 0; i < length; i++) {
        printf("%02X ", (unsigned char)text[i]);
    }
    printf("\n");
}

int main() {
    char text[] = "HELLO";
    char key[] = "KEY12";
    int length = strlen(text);

    // Encrypt
    xorEncrypt(text, key, length);
    printf("Encrypted (Hex): ");
    printHex(text, length);

    // Decrypt
    xorEncrypt(text, key, length);
    printf("Decrypted: %s\n", text);

    return 0;
}
```

## 4. RSA Algorithm (Key Generation & Encryption)

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

long long powerMod(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

int main() {
    long long p = 61, q = 53, n = p * q, e = 17, d = 2753;
    long long plaintext = 65, ciphertext, decrypted;

    ciphertext = powerMod(plaintext, e, n);
    printf("Encrypted: %lld\n", ciphertext);

    decrypted = powerMod(ciphertext, d, n);
    printf("Decrypted: %lld\n", decrypted);

    return 0;
}
```

---

## 5. SHA-3 (Simplified Version)

```
#include <stdio.h>
#include <string.h>
#include <openssl/sha.h>

void sha3Hash(char *text) {
    unsigned char hash[SHA512_DIGEST_LENGTH];
    SHA512((unsigned char*)text, strlen(text), hash);
    printf("SHA-3 Hash: ");
    for (int i = 0; i < SHA512_DIGEST_LENGTH; i++)
        printf("%02x", hash[i]);
    printf("\n");
```

```
}

int main() {
    char text[] = "HELLO";
    sha3Hash(text);
    return 0;
}
```

Here are the remaining cryptographic algorithms in simple C code:

---

## 6. DES (Data Encryption Standard)

*DES requires an external library like OpenSSL, but here is a simplified version using XOR (not the real DES algorithm but demonstrates basic symmetric encryption).*

```
#include <stdio.h>

#include <string.h>


void xorEncrypt(char *text, char *key) {

    int keyLen = strlen(key);

    for (int i = 0; text[i] != '\0'; i++) {

        text[i] ^= key[i % keyLen];

    }

}


int main() {

    char text[] = "HELLO";

    char key[] = "KEY12";


    xorEncrypt(text, key);

    printf("Encrypted: %s\n", text);
```

```
    xorEncrypt(text, key); // Decrypting
    printf("Decrypted: %s\n", text);
    return 0;
}
```

*For real DES, OpenSSL is needed (`#include <openssl/des.h>`). Let me know if you need it.*

---

## 7. AES (Advanced Encryption Standard)

*Similar to DES, AES requires OpenSSL. Below is a simple XOR-based approach for demonstration.*

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

void xorAES(char *text, char *key) {
    int keyLen = strlen(key);
    for (int i = 0; text[i] != '\0'; i++) {
        text[i] ^= key[i % keyLen];
    }
}

int main() {
    char text[] = "HELLO";
    char key[] = "AESKEY";

    xorAES(text, key);
    printf("Encrypted: %s\n", text);
```

```
    xorAES(text, key); // Decrypting
    printf("Decrypted: %s\n", text);
    return 0;
}
```

*For real AES, OpenSSL is required (`#include <openssl/aes.h>`). Let me know if you need it.*

---

## 8. Diffie-Hellman Key Exchange

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

long long powerMod(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

int main() {
    long long p = 23, g = 5; // Public values
    long long a = 6, b = 15; // Private keys
```

```
    long long A = powerMod(g, a, p); // A's public key
    long long B = powerMod(g, b, p); // B's public key

    long long sharedA = powerMod(B, a, p); // Shared secret for A
    long long sharedB = powerMod(A, b, p); // Shared secret for B

    printf("Shared Key: %lld (A) and %lld (B)\n", sharedA, sharedB);
    return 0;
}
```

---

## 9. MD5 Hashing

```
#include <stdio.h>
#include <string.h>
#include <openssl/md5.h>

void md5Hash(char *text) {
    unsigned char hash[MD5_DIGEST_LENGTH];
    MD5((unsigned char*)text, strlen(text), hash);

    printf("MD5 Hash: ");
    for (int i = 0; i < MD5_DIGEST_LENGTH; i++)
        printf("%02x", hash[i]);
    printf("\n");
}
```

```
int main() {
    char text[] = "HELLO";
    md5Hash(text);
    return 0;
}
```

---

## 10. SHA-1 Hashing

```
#include <stdio.h>
#include <string.h>
#include <openssl/sha.h>

void sha1Hash(char *text) {
    unsigned char hash[SHA_DIGEST_LENGTH];
    SHA1((unsigned char*)text, strlen(text), hash);

    printf("SHA-1 Hash: ");
    for (int i = 0; i < SHA_DIGEST_LENGTH; i++)
        printf("%02x", hash[i]);
    printf("\n");
}

int main() {
    char text[] = "HELLO";
    sha1Hash(text);
```

```
    return 0;
}
```

---

## 11. SHA-256 Hashing

```
#include <stdio.h>
#include <string.h>
#include <openssl/sha.h>

void sha256Hash(char *text) {
    unsigned char hash[SHA256_DIGEST_LENGTH];
    SHA256((unsigned char*)text, strlen(text), hash);

    printf("SHA-256 Hash: ");
    for (int i = 0; i < SHA256_DIGEST_LENGTH; i++)
        printf("%02x", hash[i]);
    printf("\n");
}

int main() {
    char text[] = "HELLO";
    sha256Hash(text);
    return 0;
}
```

---

## 12. ElGamal Encryption

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

long long powerMod(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}

int main() {
    long long p = 23, g = 5, x = 6; // Private key
    long long y = powerMod(g, x, p); // Public key

    long long k = 15, m = 10; // Random k and message
    long long c1 = powerMod(g, k, p);
    long long c2 = (powerMod(y, k, p) * m) % p;

    long long decrypt = (c2 * powerMod(c1, p - 1 - x, p)) % p;

    printf("Encrypted: (%lld, %lld)\n", c1, c2);
```

```
    printf("Decrypted: %lld\n", decrypt);
    return 0;
}
```

---

## 13. Blowfish (XOR-Based Alternative)

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

void blowfishEncrypt(char *text, char *key) {
    int keyLen = strlen(key);
    for (int i = 0; text[i] != '\0'; i++) {
        text[i] ^= key[i % keyLen];
    }
}

int main() {
    char text[] = "HELLO";
    char key[] = "BLOWFISH";

    blowfishEncrypt(text, key);
    printf("Encrypted: %s\n", text);

    blowfishEncrypt(text, key); // Decrypting
    printf("Decrypted: %s\n", text);
    return 0;
```

```
}
```