1. Write a program that contains a string (char pointer) with a value ‘Hello world’. The program should XOR each character in this string with 0 and display the result.

```
#include <stdio.h>
int main() {
    char *str = "Hello world";
    char result[50]; // buffer to store XOR result
    int i = 0;
    // XOR each character with 0
    while (str[i] != '\0') {
        result[i] = str[i] ^ 0; // XOR with 0
        i++;
    }
    result[i] = '\0'; // Null-terminate the result string
    // Display the result
    printf("Original String: %s\n", str);
    printf("XOR with 0 Result: %s\n", result);
    return 0;
}
```

Output:

```
Original String: Hello world
XOR with 0 Result: Hello world
```

2. Write a program that contains a string (char pointer) with a value ‘Hello world’. The program should AND or and XOR each character in this string with 127 and display the result.

```
#include <stdio.h>
int main() {
    char *str = "Hello world";
    char andResult[50];
    char xorResult[50];
    int i = 0;
    while (str[i] != '\0') {
        andResult[i] = str[i] & 127;   // AND with 127
        xorResult[i] = str[i] ^ 127;   // XOR with 127
        i++;
    }
    andResult[i] = '\0';
    xorResult[i] = '\0';
    printf("Original String : %s\n", str);
    printf("AND with 127    : %s\n", andResult);
    printf("XOR with 127    : %s\n", xorResult);
    return 0;
}
```

Output:

```
Original String : Hello world
AND with 127    : Hello world
XOR with 127    : 3z??p?0p?~?s
```

3. Write a program to perform encryption and decryption using the following algorithms a) Ceaser cipher b) Substitution cipher c) Hill Cipher

Caesar Cipher Program

```
import java.util.*;   // Import utility package for Scanner
public class CaesarCipher {
   public static void main(String args[]) {
      // Create Scanner object to read input
      Scanner sc = new Scanner(System.in);

      // Ask user for input string (plaintext)
      System.out.print("Enter String to Encrypt: ");
      String word = sc.nextLine();   // Read the string from user
      // Ask user for shift value (key K)
      System.out.print("Enter K: ");
      int k = sc.nextInt();   // Read integer key
      // ---------------- Encryption ----------------
      String encryptedWord = "";  // To store encrypted text
      // Loop through each character of the word
      for (int i = 0; i < word.length(); i++) {
         // Convert character into 0–25 range (assuming lowercase a–z)
         int r = (int)(word.charAt(i)) - 97;
         // Apply Caesar Cipher formula: (position + k) mod 26
         r = (r + k) % 26;
         // Convert back to character by adding 'a' (97 in ASCII)
         encryptedWord += (char)(r + 97);
      }
      // Print the encrypted text
      System.out.println("Encrypted Text : " + encryptedWord);
      // ---------------- Decryption ----------------
      String decryptedWord = "";  // To store decrypted text
      // Loop through each character of encrypted text
      for (int i = 0; i < encryptedWord.length(); i++) {
         // Convert character into 0–25 range
         int r = (int)(encryptedWord.charAt(i)) - 97;
         // Reverse Caesar Cipher: (position - k) mod 26
         // If negative, add 26 to wrap around
         r = (r - k) >= 0 ? (r - k) : 26 + (r - k);
         // Convert back to character
         decryptedWord += (char)(r + 97);
      }
      // Print the decrypted text
      System.out.println("Decrypted Text : " + decryptedWord);
```

```
        // Close Scanner object
        sc.close();
    }
}
```

Output:
Enter String to Encrypt: hello
Enter K: 3
Encrypted Text : khoor
Decrypted Text : hello

Substitution Cipher Program

```
import java.util.*;   // Import Java utility package for Scanner
public class SubstitutionCipher {
    public static void main(String args[]) {
        // Create Scanner object to read input from user
        Scanner sc = new Scanner(System.in);
        // Ask user to enter a word for encryption
        System.out.print("Enter a word to encrypt (lowercase letters only): ");
        String word = sc.nextLine();   // Read input word
        // Strings to store encrypted and decrypted results
        String encryptedWord = "";
        String decryptedWord = "";
        // ----------------- Encryption -----------------
        // Loop through each character of the input word
        for (int i = 0; i < word.length(); i++) {
            char ch = word.charAt(i);   // Get the character at position i
            // Check if character is a lowercase alphabet
            if (ch >= 'a' && ch <= 'z') {
                int pos = ch - 'a';          // Find alphabet position (0–25)
                int newPos = 25 - pos;       // Mirror position (Atbash rule: a↔z, b↔y, etc.)
                char enc = (char) (newPos + 'a'); // Convert back to character
                encryptedWord += enc;        // Append encrypted char
            } else {
                // If not a letter (like space or number), keep it unchanged
                encryptedWord += ch;
            }
        }
        // ----------------- Decryption -----------------
        // Loop through each character of the encrypted word
        for (int i = 0; i < encryptedWord.length(); i++) {
            char ch = encryptedWord.charAt(i);   // Get the character
            // Again check if it's a lowercase alphabet
            if (ch >= 'a' && ch <= 'z') {
                int pos = ch - 'a';          // Find position (0–25)
                int newPos = 25 - pos;       // Apply same Atbash rule again
                char dec = (char) (newPos + 'a'); // Convert back to character
                decryptedWord += dec;        // Append decrypted char
            } else {
```

```
            // If not a letter, keep unchanged
            decryptedWord += ch;
        }
    }
    // ---------- Display Results ----------
    System.out.println("Encrypted Word: " + encryptedWord); // Print encrypted text
    System.out.println("Decrypted Word: " + decryptedWord); // Print decrypted text
    // Close the Scanner object to prevent memory leak
    sc.close();
  }
}
```

Output:
Enter a word to encrypt (lowercase letters only): hello
Encrypted Word: svool
Decrypted Word: hello

Hill Cipher Program

```
 import java.util.*;   // Import Scanner class for input
public class HillCipher {
  // Function to calculate determinant of 2x2 matrix (mod 26)
  static int determinant(int[][] m) {
    // det = ad - bc
    int det = (m[0][0]*m[1][1] - m[0][1]*m[1][0]) % 26;
    if (det < 0) det += 26;   // ensure positive result
    return det;
  }
  // Function to find modular inverse of determinant mod 26
  static int modInverse(int det) {
    // brute force search: find x such that (det * x) % 26 == 1
    for (int i = 1; i < 26; i++) {
      if ((det * i) % 26 == 1) return i;
    }
    return -1; // if no modular inverse exists
  }
  // Function to calculate adjoint of 2x2 matrix
  static int[][] adjoint(int[][] m) {
    int[][] adj = new int[2][2];
    // For 2x2, adjoint = [[d, -b], [-c, a]]
    adj[0][0] =  m[1][1];
    adj[0][1] = -m[0][1];
    adj[1][0] = -m[1][0];
    adj[1][1] =  m[0][0];
    return adj;
  }
  // Multiply 2x2 matrix with 2x1 vector mod 26
  static int[] multiply(int[][] m, int[] v) {
    int[] result = new int[2];
    for (int i = 0; i < 2; i++) {
```

```
            // Multiply row of matrix with vector
            result[i] = (m[i][0]*v[0] + m[i][1]*v[1]) % 26;
            if (result[i] < 0) result[i] += 26;   // keep result positive
        }
        return result;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);  // Scanner for input
        // Step 1: Input 2x2 Key Matrix
        int[][] key = new int[2][2];
        System.out.println("Enter 2x2 key matrix (must be invertible mod 26):");
        for (int i=0; i<2; i++) {       // loop rows
            for (int j=0; j<2; j++) {   // loop columns
                key[i][j] = sc.nextInt();   // read each value
            }
        }
        // Step 2: Input plaintext (must be 2 characters long)
        sc.nextLine(); // clear input buffer
        System.out.print("Enter plaintext (2 letters, lowercase): ");
        String word = sc.nextLine().toLowerCase();  // read input word
        if (word.length() != 2) {   // validation
            System.out.println("Plaintext must be exactly 2 characters!");
            return;
        }
        // Convert plaintext into numeric vector (a=0, b=1, …, z=25)
        int[] textVec = new int[2];
        for (int i=0; i<2; i++) {
            textVec[i] = word.charAt(i) - 'a';
        }
        // Step 3: Encrypt → C = K × P mod 26
        int[] encryptedVec = multiply(key, textVec);
        StringBuilder encrypted = new StringBuilder();
        for (int val : encryptedVec) encrypted.append((char)(val + 'a'));
        System.out.println("Encrypted Text: " + encrypted);
        // Step 4: Decrypt
        int det = determinant(key);      // find determinant of key
        int invDet = modInverse(det);    // find modular inverse of determinant
        if (invDet == -1) {              // if no inverse exists
            System.out.println("Key is not invertible. Decryption not possible.");
            return;
        }
        int[][] adj = adjoint(key);      // find adjoint of key
        // Compute inverse key matrix = (invDet × adj) mod 26
        int[][] invKey = new int[2][2];
        for (int i=0; i<2; i++) {
            for (int j=0; j<2; j++) {
                int val = adj[i][j] * invDet;  // multiply adjoint by invDet
                val %= 26;                     // take mod 26
                if (val < 0) val += 26;        // ensure positive
                invKey[i][j] = val;            // store in inverse key matrix
```

```
            }
        }

        // Decrypt: P = K⁻¹ × C mod 26
        int[] decryptedVec = multiply(invKey, encryptedVec);
        StringBuilder decrypted = new StringBuilder();
        for (int val : decryptedVec) decrypted.append((char)(val + 'a'));
        System.out.println("Decrypted Text: " + decrypted);
        sc.close(); // close scanner
    }
}
```

Output:

```
Enter 2x2 key matrix (row by row):
3 3
2 5
Enter plaintext (2 letters, lowercase): hi
Encrypted Text: rc
Decrypted Text: hi
```

4. Write a program to implement the DES algorithm

```
import java.util.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class DES{
    public static void main(String args[]) throws Exception{
        Scanner sc = new Scanner(System.in);

        // take message to encrypt
        System.out.print("Enter message to encrypt: ");
        String msg = sc.nextLine();
        byte[] message = msg.getBytes();

        // take custom key and prepare key
        System.out.print("Enter custom key: ");
        String key = sc.nextLine();
        byte[] keyData = key.getBytes();
        DESKeySpec secretKey = new DESKeySpec(keyData);
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
        SecretKey keyN = keyFactory.generateSecret(secretKey);

        // ENCRYPTION
        Cipher cipher = Cipher.getInstance("DES");
        cipher.init(Cipher.ENCRYPT_MODE, keyN);
        byte[] encrypted = cipher.doFinal(message);

        //DECRYPTION
        cipher.init(Cipher.DECRYPT_MODE, keyN);
```

```
        byte[] decrypted = cipher.doFinal(encrypted);
        String decryptedMsg = new String(decrypted);

        // OUTPUT
        System.out.println("Message: " + msg);
        System.out.println("Encrypted: " + encrypted);
        System.out.println("Decrypted: " + decryptedMsg);

    }
}
```

Output:
Enter message to encrypt: Tomorrow is the last day to submit
Enter custom key: it is very urgent
Message: Tomorrow is the last day to submit
Encrypted: [B@47d90b9e
Decrypted: Tomorrow is the last day to submit

5. Write a program to implement the Blowfish algorithm.

```
import java.util.*;                         // For Scanner input
import java.security.GeneralSecurityException; // For handling crypto exceptions
import javax.crypto.*;                      // For Cipher
import javax.crypto.spec.SecretKeySpec;     // For SecretKeySpec

public class BlowFish {
    public static void main(String args[]) throws GeneralSecurityException {
        Scanner sc = new Scanner(System.in); // Create Scanner object

        // ----- INPUT: Message -----
        System.out.print("Enter message to encrypt: ");
        String msg = sc.nextLine();         // Read plaintext
        byte[] message = msg.getBytes();    // Convert to byte array

        // ----- INPUT: Key -----
        System.out.print("Enter custom key: ");
        String key = sc.nextLine();         // Read key
        byte[] keyData = key.getBytes();    // Convert key to byte array

        // Create a SecretKeySpec for Blowfish using the key bytes
        // Valid key size: 32 bits (4 bytes) to 448 bits (56 bytes)
        SecretKeySpec secretKey = new SecretKeySpec(keyData, "BlowFish");

        // ----- ENCRYPTION -----
        Cipher cipher = Cipher.getInstance("BlowFish"); // Create Cipher for Blowfish
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);    // Initialize for encryption
        byte[] encrypted = cipher.doFinal(message);     // Encrypt message

        // ----- DECRYPTION -----
        cipher.init(Cipher.DECRYPT_MODE, secretKey);    // Re-initialize for decryption
```

```
        byte[] decrypted = cipher.doFinal(encrypted);   // Decrypt ciphertext
        String decryptedMsg = new String(decrypted);    // Convert back to String

        // ----- OUTPUT -----
        System.out.println("Message   : " + msg);        // Original plaintext
        System.out.println("Encrypted : " + encrypted);  // Encrypted bytes (not readable!)
        System.out.println("Decrypted : " + decryptedMsg);// Decrypted text
    }
}
```

Output:
Enter message to encrypt: Tomorrow is the last day to submit
Enter custom key: t is very urgent
Message   : Tomorrow is the last day to submit
Encrypted : [B@5cee5251
Decrypted : Tomorrow is the last day to submit

6. Write the RC4 logic in Java Using Java cryptography; encrypt the text “Hello world” using RC4. Create your own key using Java key tool.

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Base64;

public class RC4Encryption {

    public static void main(String[] args) {
        try {
            // Step 1: Generate a secret key for RC4
            KeyGenerator keyGen = KeyGenerator.getInstance("RC4");
            keyGen.init(128); // Set key size to 128 bits
            SecretKey secretKey = keyGen.generateKey();

            // Step 2: Convert the key to a readable format (optional, for demonstration)
            String encodedKey = Base64.getEncoder().encodeToString(secretKey.getEncoded());
            System.out.println("Generated Secret Key (Base64): " + encodedKey);

            // Step 3: Initialize the Cipher for encryption
            Cipher cipher = Cipher.getInstance("RC4");
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);

            // Step 4: Encrypt the plaintext
            String plaintext = "Hello world";
            byte[] encryptedBytes = cipher.doFinal(plaintext.getBytes());
            String encryptedText = Base64.getEncoder().encodeToString(encryptedBytes);
            System.out.println("Encrypted Text: " + encryptedText);

            // Step 5: Initialize the Cipher for decryption
```

```
            cipher.init(Cipher.DECRYPT_MODE, secretKey);

            // Step 6: Decrypt the ciphertext
            byte[] decryptedBytes = cipher.doFinal(Base64.getDecoder().decode(encryptedText));
            String decryptedText = new String(decryptedBytes);
            System.out.println("Decrypted Text: " + decryptedText);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

OUTPUT:
Generated Secret Key (Base64): VvoQvLvSJ7mAHo/h6QEooQ==
Encrypted Text: X55P/ErVY7TTi4A=
Decrypted Text: Hello world

7. Write a program to implement the RSA algorithm.

```
import java.math.*;
import java.util.*;
public class Main {
  public static int getGCD(int mod, int num) {
    // If the mod is zero, return the num
    if (mod == 0)
      return num;
    else
      // recursive function call
      return getGCD(num % mod, mod);
  }
  public static void main(String args[]) {
    int d = 0, e; // Intialization
    int message = 88; // number message
    int prime1 = 11; // 1st prime number p
    int prime2 = 17; // 2nd prime number q
    int primeMul = prime1 * prime2; // performing operations
    int primeMul1 = (prime1 - 1) * (prime2 - 1);
    System.out.println("primeMul1 is equal to : " + primeMul1 + "\n");
    // Finding the valid public key
    for (e = 2; e < primeMul1; e++) {
      // Here e is a public key
      if (getGCD(e, primeMul1) == 1) {
        break;
      }
    }
    // Printing the public key
    System.out.println("Public key e is = " + e);
    // Calculating the private key
    for (int m = 0; m <= 9; m++) {
```

```
        // get the value of temp
        int temp = 1 + (m * primeMul1);
        // private key
        if (temp % e == 0) {
          d = temp / e;
          break;
        }
      }
      System.out.println("d is : " + d);
      double cipher;
      BigInteger d_message;
      // getting the cipher text
      cipher = (Math.pow(message, e)) % primeMul;
      System.out.println("Cipher text is : " + cipher);
      // Int to BigInteger
      BigInteger bigN = BigInteger.valueOf(primeMul);
      // Float to bigINt
      BigInteger bigC = BigDecimal.valueOf(cipher).toBigInteger();
      // decrypting the message
      d_message = (bigC.pow(d)).mod(bigN);
      // print decrypted message
      System.out.println("Decrypted text is : " + d_message);
    }
}
```

Output:
primeMul1 is equal to : 160

Public key e is = 3
d is : 107
Cipher text is : 44.0
Decrypted text is : 88

8. Write a java program to Implement the Diffie-Hellman Key Exchange mechanism

```
import java.math.BigInteger;   // For large number arithmetic
import java.security.SecureRandom;  // For secure random number generation

public class DiffieHellman {

    // Secure random number generator (for private keys)
    private final static SecureRandom random = new SecureRandom();

    // Common (publicly agreed) values
    private BigInteger agreedPrimeP;         // Prime modulus (p)
    private BigInteger agreedPrimitiveRootG; // Primitive root or base (g)

    // Individual (per user) values
    private BigInteger publicKey;  // Public key = g^privateKey mod p
    private BigInteger privateKey; // Private key (randomly chosen)
```

```
/**
 * Constructor: initializes base (g), modulus (p), and generates keys
 * @param keySize  number of bits for private key (e.g., 512)
 * @param base     the agreed primitive root g
 * @param modulus  the agreed prime number p
 */
public DiffieHellman(int keySize, BigInteger base, BigInteger modulus) {
   agreedPrimeP = modulus;          // Prime modulus
   agreedPrimitiveRootG = base;     // Base (generator)

   // Generate a random private key
   privateKey = new BigInteger(keySize, random);

   // Compute the corresponding public key: g^privateKey mod p
   publicKey = agreedPrimitiveRootG.modPow(privateKey, agreedPrimeP);
}

// Method to return the public key (to share with other party)
BigInteger getPublicKey() {
   return publicKey;
}

// Method to compute the shared secret key using other party's public key
BigInteger getSharedKey(BigInteger receivedPublicKey) {
   // sharedKey = (receivedPublicKey ^ privateKey) mod p
   return receivedPublicKey.modPow(privateKey, agreedPrimeP);
}

public static void main(String args[]) {

   // ----- Step 1: Agree on public values -----
   // Both users (Alice and Bob) agree on:
   // - A prime number p
   // - A base (primitive root) g
   // These can be shared publicly.
   BigInteger primeP = new BigInteger("23");   // Example small prime
   BigInteger baseG = new BigInteger("19");    // Example primitive root

   // ----- Step 2: Create Diffie-Hellman key pairs for Alice and Bob -----
   DiffieHellman alice = new DiffieHellman(512, baseG, primeP);
   DiffieHellman bob = new DiffieHellman(512, baseG, primeP);

   // ----- Step 3: Exchange public keys -----
   BigInteger alicePublicKey = alice.getPublicKey();  // Alice sends to Bob
   BigInteger bobPublicKey = bob.getPublicKey();      // Bob sends to Alice

   System.out.println("Alice Public Key: " + alicePublicKey);
   System.out.println("Bob Public Key  : " + bobPublicKey);
```

```
        // ----- Step 4: Each computes shared secret key -----
        BigInteger aliceSharedSecretKey = alice.getSharedKey(bobPublicKey); // Using Bob's
public key
        BigInteger bobSharedSecretKey = bob.getSharedKey(alicePublicKey);   // Using Alice's
public key

        // ----- Step 5: Verify both secrets are identical -----
        if (aliceSharedSecretKey.equals(bobSharedSecretKey)) {
            System.out.println("\n Shared secret keys match!");
            System.out.println("Shared Secret Key = " + aliceSharedSecretKey);
        } else {
            System.out.println("\n Shared secret keys do not match!");
        }
    }
}
```

Output:
Alice Public Key: 2
Bob Public Key : 7

Shared secret keys match!
Shared Secret Key = 3

9. Calculate the message digest of a text using the SHA-1 algorithm in JAVA.

```
import java.security.MessageDigest;   // For message digest algorithms
import java.util.Scanner;             // For user input
import java.util.Base64;              // For Base64 encoding

public class SHA1Example {
    public static void main(String[] args) {
        try {
            // Create Scanner object to read text from user
            Scanner sc = new Scanner(System.in);

            // Step 1: Input text message
            System.out.print("Enter text to compute SHA-1 hash: ");
            String inputText = sc.nextLine();

            // Step 2: Create MessageDigest instance for SHA-1
            MessageDigest md = MessageDigest.getInstance("SHA-1");

            // Step 3: Pass the input text bytes to MessageDigest
            md.update(inputText.getBytes());

            // Step 4: Compute the hash (message digest)
            byte[] digestBytes = md.digest();

            // Step 5: Convert the hash bytes to a readable hexadecimal string
            StringBuilder hexHash = new StringBuilder();
```

```
            for (byte b : digestBytes) {
                hexHash.append(String.format("%02x", b)); // format each byte as hex
            }

            // Step 6: Display results
            System.out.println("\n--- SHA-1 Message Digest ---");
            System.out.println("Input Text  : " + inputText);
            System.out.println("Digest (Hex): " + hexHash.toString());
            System.out.println("Digest (Base64): " +
Base64.getEncoder().encodeToString(digestBytes));

            sc.close();

        } catch (Exception e) {
            System.err.println("Error while computing SHA-1: " + e.getMessage());
        }
    }
}
```

OUTPUT:

```
Enter text to compute SHA-1 hash: Hello World

--- SHA-1 Message Digest ---
Input Text  : Hello World
Digest (Hex): 0a4d55a8d778e5022fab701977c5d840bbc486d0
Digest (Base64): Ck1VqNd45QIvq3AZd8XYQLvEhtA=
```

10. Calculate the message digest of a text using the MD5 algorithm in JAVA

```
import java.security.MessageDigest;   // For message digest algorithms
import java.util.Scanner;             // For user input
import java.util.Base64;              // For Base64 encoding

public class MD5Example {
    public static void main(String[] args) {
        try {
            // Step 1: Read text input from the user
            Scanner sc = new Scanner(System.in);
            System.out.print("Enter text to compute MD5 hash: ");
            String inputText = sc.nextLine();

            // Step 2: Create MessageDigest instance for MD5
            MessageDigest md = MessageDigest.getInstance("MD5");

            // Step 3: Feed the input text bytes into the digest object
            md.update(inputText.getBytes());

            // Step 4: Compute the hash (message digest)
            byte[] digestBytes = md.digest();
```

```
            // Step 5: Convert the digest bytes into a readable Hex string
            StringBuilder hexHash = new StringBuilder();
            for (byte b : digestBytes) {
                hexHash.append(String.format("%02x", b)); // format each byte as two hex digits
            }

            // Step 6: Display the results
            System.out.println("\n--- MD5 Message Digest ---");
            System.out.println("Input Text       : " + inputText);
            System.out.println("Digest (Hex)     : " + hexHash.toString());
            System.out.println("Digest (Base64)  : " +
Base64.getEncoder().encodeToString(digestBytes));

            sc.close();
        } catch (Exception e) {
            System.err.println("Error while computing MD5: " + e.getMessage());
        }
    }
}
```

Output:

```
Enter text to compute MD5 hash: Hello World

--- MD5 Message Digest ---
Input Text       : Hello World
Digest (Hex)     : b10a8db164e0754105b7a99be72e3fe5
Digest (Base64)  : sQqNsWTgdUEFt6mb5y4/5Q==
```