Practical – 1

AIM: A. Write a C program that contains a string (char pointer) with a value 'Hello. The program should XOR, AND and OR each character in this string with 0 and displays the result.

- **Code:**

```
#include <iostream>
#include <string>
using namespace std;
int main() {
   string str = "Hello World";
   int var_xor = 0, var_and = 0, var_or = 0;
   cout << "XOR, AND, OR operations with 0" << endl;
   for (int i = 0; i < str.length(); i++) {
      cout << str[i] << " (" << int(str[i]) << ") ";
      cout << "XOR " << var_xor << ": " << (str[i] ^ var_xor) << ", ";
      cout << "AND " << var_and << ": " << (str[i] & var_and) << ", ";
      cout << "OR " << var_or << ": " << (str[i] | var_or) << endl;
   }
   return 0;
}
```

- **Output:**

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\CS
XOR, AND, OR operations with 0
H (72) XOR 0: 72, AND 0: 0, OR 0: 72
e (101) XOR 0: 101, AND 0: 0, OR 0: 101
l (108) XOR 0: 108, AND 0: 0, OR 0: 108
l (108) XOR 0: 108, AND 0: 0, OR 0: 108
o (111) XOR 0: 111, AND 0: 0, OR 0: 111
  (32) XOR 0: 32, AND 0: 0, OR 0: 32
W (87) XOR 0: 87, AND 0: 0, OR 0: 87
o (111) XOR 0: 111, AND 0: 0, OR 0: 111
r (114) XOR 0: 114, AND 0: 0, OR 0: 114
l (108) XOR 0: 108, AND 0: 0, OR 0: 108
d (100) XOR 0: 100, AND 0: 0, OR 0: 100
PS D:\CSE\CSE_github\SEM 6\CNS>
```

B. Write a C program that contains a string (char pointer) with a value 'Hello World'. The program should XOR, AND and OR each character in this string with 127 and displays the result.

- **Code:**

```
#include <iostream>
#include <string>
using namespace std;
int main() {
   string str = "Hello World";
   int var_xor = 127, var_and = 127, var_or = 127;
   cout << "XOR, AND, OR operations with 127" << endl;
   for (int i = 0; i < str.length(); i++) {
      cout << str[i] << " (" << int(str[i]) << ") ";
      cout << "XOR " << var_xor << ": " << (str[i] ^ var_xor) << ", ";
      cout << "AND " << var_and << ": " << (str[i] & var_and) << ", ";
      cout << "OR " << var_or << ": " << (str[i] | var_or) << endl;
   }
   return 0;
}
```

- **Output:**

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\CSE\CSE_github\SE
XOR, AND, OR operations with 127
H (72) XOR 127: 55, AND 127: 72, OR 127: 127
e (101) XOR 127: 26, AND 127: 101, OR 127: 127
l (108) XOR 127: 19, AND 127: 108, OR 127: 127
l (108) XOR 127: 19, AND 127: 108, OR 127: 127
o (111) XOR 127: 16, AND 127: 111, OR 127: 127
  (32) XOR 127: 95, AND 127: 32, OR 127: 127
W (87) XOR 127: 40, AND 127: 87, OR 127: 127
o (111) XOR 127: 16, AND 127: 111, OR 127: 127
r (114) XOR 127: 13, AND 127: 114, OR 127: 127
l (108) XOR 127: 19, AND 127: 108, OR 127: 127
d (100) XOR 127: 27, AND 127: 100, OR 127: 127
PS D:\CSE\CSE_github\SEM 6\CNS>
```

C. Write a C program that contains a string (char pointer) with a value 'Hello World'. The program should bitwise OR, left shift and right shift each character in this string and displays the result.

- **Code:**

```
#include <iostream>
#include <string>
using namespace std;
int main() {
   string str = "Hello World";
   int var_or = 127;
   cout << "Bitwise OR, Left Shift, and Right Shift operations" << endl;
   for (int i = 0; i < str.length(); i++) {
      cout << str[i] << " (" << int(str[i]) << ") ";
      cout << "OR " << var_or << ": " << (str[i] | var_or) << ", ";
      cout << "Left Shift 1: " << (str[i] << 1) << ", ";
      cout << "Right Shift 1: " << (str[i] >> 1) << endl;
   }
   return 0;
}
```

- **Output:**

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\CSE\CSE_github\SEM 6\CNS
Bitwise OR, Left Shift, and Right Shift operations
H (72) OR 127: 127, Left Shift 1: 144, Right Shift 1: 36
e (101) OR 127: 127, Left Shift 1: 202, Right Shift 1: 50
l (108) OR 127: 127, Left Shift 1: 216, Right Shift 1: 54
l (108) OR 127: 127, Left Shift 1: 216, Right Shift 1: 54
o (111) OR 127: 127, Left Shift 1: 222, Right Shift 1: 55
  (32) OR 127: 127, Left Shift 1: 64, Right Shift 1: 16
W (87) OR 127: 127, Left Shift 1: 174, Right Shift 1: 43
o (111) OR 127: 127, Left Shift 1: 222, Right Shift 1: 55
r (114) OR 127: 127, Left Shift 1: 228, Right Shift 1: 57
l (108) OR 127: 127, Left Shift 1: 216, Right Shift 1: 54
d (100) OR 127: 127, Left Shift 1: 200, Right Shift 1: 50
PS D:\CSE\CSE_github\SEM 6\CNS>
```

Practical – 2

AIM: To implement Caesar Cipher Encryption - Decryption

- **Code:**

```
#include <iostream>
#include <string>
using namespace std;
string encryption(string str, int key) {
   string encrypted_str = "";
   for (int i = 0; i < str.length(); i++) {
      char c = str.at(i);
      if(c == ' ') {
      } else {
         if (isalpha(c)) {
            if (islower(c)) {
               c = (c - 'a' + key) % 26 + 'a';
            } else if (isupper(c)) {
               c = (c - 'A' + key) % 26 + 'A';
            }
         } else {
            int temp = (int) c;
            temp += key;
            c = (char) temp;
         }
      }
         encrypted_str += c;
   }
   return encrypted_str;
}
string description(string encrypted_str, int key) {
   string str = "";
   for (int i = 0; i < encrypted_str.length(); i++) {
      char c = encrypted_str.at(i);
      if(c == ' ') {
      } else {
         if (isalpha(c)) {
            if (islower(c)) {
               c = (c - 'a' - key + 26) % 26 + 'a';
            } else if (isupper(c)) {
               c = (c - 'A' - key + 26) % 26 + 'A';
            }
         } else {
            int temp = (int) c;
```

```
            temp -= key;
            c = (char) temp;
          }
        }
          str += c;
      }
      return str;
    }
    int main() {
      string str = "hello world";
      unsigned int key = 5;
      cout << "Enter any text" << endl;
      getline(cin, str);
      cout << "Enter key" << endl;
      cin >> key;
      string encrypted_str = encryption(str, key);
      string description_str = description(encrypted_str, key);
      cout << "text: " << str << endl;
      cout << "Encrypted text: " << encrypted_str << endl;
      cout << "descripted text: " << description_str << endl;
      return 0;
    }
```

- **Output:**

```
PS D:\CSE\CSE_github\SEM 6\CNS>
Enter any text
RIAUHAS
Enter key
RJ
text: RIAUHAS
Encrypted text: RIAUHAS
descripted text: RIAUHAS
```

Practical – 3

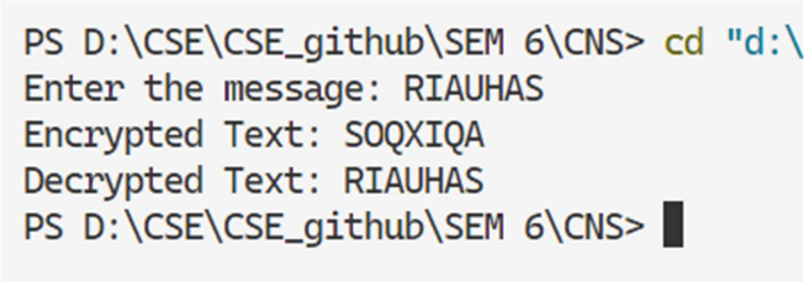
AIM: To implement Mono-alphabetic Cipher Encryption – Decryption.

- **Code:**

```
#include <iostream>
#include <string>
using namespace std;
string encrypt_text(string plaintext, string key) {
    string normal = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    string ciphertext = "";
    for (int i = 0; i < plaintext.length(); i++) {
        char ch = plaintext[i];
        if (ch >= 'A' && ch <= 'Z') {
            int index = ch - 'A';
            ciphertext += key[index];
        }
        else if (ch >= 'a' && ch <= 'z') {
            int index = ch - 'a';
            ciphertext += tolower(key[index]);
        }
        else {
            ciphertext += ch;
        }
    }
    return ciphertext;
}
string decrypt_cipher(string ciphertext, string key) {
    string normal = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    string plaintext = "";
    for (int i = 0; i < ciphertext.length(); i++) {
        char ch = ciphertext[i];
        if (ch >= 'A' && ch <= 'Z') {
            int index = key.find(ch);
            plaintext += normal[index];
        }
        else if (ch >= 'a' && ch <= 'z') {
            int index = key.find(toupper(ch));
            plaintext += tolower(normal[index]);
        }
        else {
            plaintext += ch;
        }
    }
```

```
    return plaintext;
}
int main() {
    string key = "QWERTYUIOPLKJHGFDSAZXCVBNM";
    string message;
    cout << "Enter the message: ";
    getline(cin, message);
    string encryptedText = encrypt_text(message, key);
    string decryptedText = decrypt_cipher(encryptedText, key);
    cout << "Encrypted Text: " << encryptedText << endl;
    cout << "Decrypted Text: " << decryptedText << endl;
    return 0;
}
```

- **Output:**



Practical – 4

AIM: To implement Hill Cipher Encryption

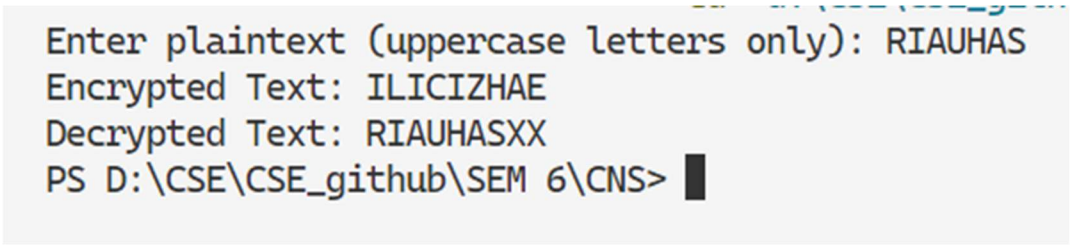
- Code:

```
#include <iostream>
#include <cstring>
using namespace std;
const int SIZE = 3;
void multiplyMatrix(int key[SIZE][SIZE], int text[SIZE], int result[SIZE]) {
   for (int i = 0; i < SIZE; i++) {
      result[i] = 0;
      for (int j = 0; j < SIZE; j++) {
         result[i] += key[i][j] * text[j];
      }
      result[i] = (result[i] % 26 + 26) % 26;
   }
}
int modInverse(int a, int m) {
   a = a % m;
   for (int x = 1; x < m; x++) {
      if ((a * x) % m == 1) {
         return x;
      }
   }
   return -1;
}
void findInverseMatrix(int key[SIZE][SIZE], int inverseKey[SIZE][SIZE]) {
   int determinant = key[0][0] * (key[1][1] * key[2][2] - key[1][2] * key[2][1]) -
                     key[0][1] * (key[1][0] * key[2][2] - key[1][2] * key[2][0]) +
                     key[0][2] * (key[1][0] * key[2][1] - key[1][1] * key[2][0]);
   determinant = (determinant % 26 + 26) % 26;
   int determinantInverse = modInverse(determinant, 26);
   if (determinantInverse == -1) {
      cout << "Key matrix is not invertible.\n";
      return;
   }
   int adjoint[SIZE][SIZE] = {
      {(key[1][1] * key[2][2] - key[1][2] * key[2][1]), -(key[0][1] * key[2][2] - key[0][2] *
key[2][1]), (key[0][1] * key[1][2] - key[0][2] * key[1][1])},
      {-(key[1][0] * key[2][2] - key[1][2] * key[2][0]), (key[0][0] * key[2][2] - key[0][2] *
key[2][0]), -(key[0][0] * key[1][2] - key[0][2] * key[1][0])},
      {(key[1][0] * key[2][1] - key[1][1] * key[2][0]), -(key[0][0] * key[2][1] - key[0][1] *
key[2][0]), (key[0][0] * key[1][1] - key[0][1] * key[1][0])}
```

```
    };
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            inverseKey[i][j] = (adjoint[i][j] * determinantInverse) % 26;
            if (inverseKey[i][j] < 0) {
                inverseKey[i][j] += 26;
            }
        }
    }
}
void encrypt_text(char plaintext[], char ciphertext[]) {
    int key[SIZE][SIZE] = {{6, 24, 1}, {13, 16, 10}, {20, 17, 15}};
    int textBlock[SIZE], encryptedBlock[SIZE];
    int len = strlen(plaintext);
    while (len % SIZE != 0) {
        plaintext[len] = 'X';
        len++;
        plaintext[len] = '\0';
    }
    cout << "Encrypted Text: ";
    for (int i = 0; i < len; i += SIZE) {
        for (int j = 0; j < SIZE; j++) {
            textBlock[j] = plaintext[i + j] - 'A';
        }
        multiplyMatrix(key, textBlock, encryptedBlock);
        for (int j = 0; j < SIZE; j++) {
            ciphertext[i + j] = (char)(encryptedBlock[j] + 'A');
            cout << ciphertext[i + j];
        }
    }
    ciphertext[len] = '\0'; // Null terminate the ciphertext
    cout << endl;
}
void decrypt_cipher(char ciphertext[]) {
    int key[SIZE][SIZE] = {{6, 24, 1}, {13, 16, 10}, {20, 17, 15}};
    int inverseKey[SIZE][SIZE];
    findInverseMatrix(key, inverseKey);
    int textBlock[SIZE], decryptedBlock[SIZE];
    int len = strlen(ciphertext);
    cout << "Decrypted Text: ";
    for (int i = 0; i < len; i += SIZE) {
        for (int j = 0; j < SIZE; j++) {
            textBlock[j] = ciphertext[i + j] - 'A';
        }
```

```
            multiplyMatrix(inverseKey, textBlock, decryptedBlock);
            for (int j = 0; j < SIZE; j++) {
                cout << (char)(decryptedBlock[j] + 'A');
            }
        }
        cout << endl;
    }
    int main() {
        char plaintext[100], ciphertext[100];
        cout << "Enter plaintext (uppercase letters only): ";
        cin >> plaintext;
        encrypt_text(plaintext, ciphertext);
        decrypt_cipher(ciphertext);
        return 0;
    }
```

- Output:



Practical – 5

AIM: To implement Poly-alphabetic Cipher (Vigener Cipher) Technique

- Code:

```
#include <iostream>
#include <cstring>
using namespace std;
void generateKey(const char* text, const char* key, char* newKey) {
    int textLen = strlen(text), keyLen = strlen(key);
    for (int i = 0, j = 0; i < textLen; i++, j++) {
        if (j == keyLen) {
            j = 0;
        }
        newKey[i] = key[j];
    }
    newKey[textLen] = '\0';
}
void encrypt_text(const char* text, const char* key, char* encryptedText) {
    int len = strlen(text);
    for (int i = 0; i < len; i++) {
        encryptedText[i] = ((text[i] + key[i]) % 26) + 'A';
    }
    encryptedText[len] = '\0';
}
void decrypt_cipher(const char* encryptedText, const char* key, char* decryptedText) {
    int len = strlen(encryptedText);
    for (int i = 0; i < len; i++) {
        decryptedText[i] = (((encryptedText[i] - key[i]) + 26) % 26) + 'A';
    }
    decryptedText[len] = '\0';
}
int main() {
    char text[100], key[100], newKey[100], encryptedText[100], decryptedText[100];
    cout << "Enter text (uppercase letters only): ";
    cin >> text;
    cout << "Enter key (uppercase letters only): ";
    cin >> key;
    generateKey(text, key, newKey);
    encrypt_text(text, newKey, encryptedText);
    decrypt_cipher(encryptedText, newKey, decryptedText);
    cout << "Encrypted Text: " << encryptedText << endl;
    cout << "Decrypted Text: " << decryptedText << endl;
    return 0;
```

```
}
```

- Output:

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\CSE\CSE_
Enter text (uppercase letters only): RIAUHAS
Enter key (uppercase letters only): RJ
Encrypted Text: IRRDYJJ
Decrypted Text: RIAUHAS
PS D:\CSE\CSE_github\SEM 6\CNS>
```

Practical – 6

AIM: To implement Play-Fair Cipher Technique.

- **Code:**

```
#include <iostream>
#include <string>
using namespace std;
const int SIZE = 5;
void generateKeyMatrix(const string& key, char keyMatrix[SIZE][SIZE]) {
    bool used[26] = {false};
    string refinedKey = "";
    for (char ch : key) {
        if (ch >= 'a' && ch <= 'z') ch -= 32;
        if (ch == 'J') ch = 'I';
        if (ch >= 'A' && ch <= 'Z' && !used[ch - 'A']) {
            refinedKey += ch;
            used[ch - 'A'] = true;
        }
    }
    for (char ch = 'A'; ch <= 'Z'; ++ch) {
        if (ch == 'J') continue;
        if (!used[ch - 'A']) {
            refinedKey += ch;
            used[ch - 'A'] = true;
        }
    }
    int index = 0;
    for (int i = 0; i < SIZE; ++i) {
        for (int j = 0; j < SIZE; ++j) {
            keyMatrix[i][j] = refinedKey[index++];
        }
    }
}
void findPosition(char keyMatrix[SIZE][SIZE], char ch, int &row, int &col) {
    if (ch == 'J') ch = 'I';
    for (int i = 0; i < SIZE; ++i) {
        for (int j = 0; j < SIZE; ++j) {
            if (keyMatrix[i][j] == ch) {
                row = i;
                col = j;
                return;
```

```
            }
        }
    }
}
string prepareText(const string& text) {
    string result = "";
    for (char ch : text) {
        if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z')) {
            if (ch >= 'a' && ch <= 'z') ch -= 32;
            result += ch;
        }
    }
    string processed = "";
    for (size_t i = 0; i < result.length(); ++i) {
        processed += result[i];
        if (i + 1 < result.length() && result[i] == result[i + 1]) {
            processed += 'X';
        }
    }
    if (processed.length() % 2 != 0) {
        processed += 'X';
    }
    return processed;
}
string encrypt(const string& text, char keyMatrix[SIZE][SIZE]) {
    string preparedText = prepareText(text);
    string encryptedText = "";
    for (size_t i = 0; i < preparedText.length(); i += 2) {
        char first = preparedText[i];
        char second = preparedText[i + 1];
        int row1, col1, row2, col2;
        findPosition(keyMatrix, first, row1, col1);
        findPosition(keyMatrix, second, row2, col2);
        if (row1 == row2) {
            encryptedText += keyMatrix[row1][(col1 + 1) % SIZE];
            encryptedText += keyMatrix[row2][(col2 + 1) % SIZE];
        } else if (col1 == col2) {
            encryptedText += keyMatrix[(row1 + 1) % SIZE][col1];
            encryptedText += keyMatrix[(row2 + 1) % SIZE][col2];
        } else {
            encryptedText += keyMatrix[row1][col2];
            encryptedText += keyMatrix[row2][col1];
        }
    }
```

```
    return encryptedText;
}
string decrypt(const string& text, char keyMatrix[SIZE][SIZE]) {
    string decryptedText = "";
    for (size_t i = 0; i < text.length(); i += 2) {
        char first = text[i];
        char second = text[i + 1];
        int row1, col1, row2, col2;
        findPosition(keyMatrix, first, row1, col1);
        findPosition(keyMatrix, second, row2, col2);
        if (row1 == row2) {
            decryptedText += keyMatrix[row1][(col1 - 1 + SIZE) % SIZE];
            decryptedText += keyMatrix[row2][(col2 - 1 + SIZE) % SIZE];
        } else if (col1 == col2) {
            decryptedText += keyMatrix[(row1 - 1 + SIZE) % SIZE][col1];
            decryptedText += keyMatrix[(row2 - 1 + SIZE) % SIZE][col2];
        } else {
            decryptedText += keyMatrix[row1][col2];
            decryptedText += keyMatrix[row2][col1];
        }
    }
    return decryptedText;
}
int main() {
    string key, text;
    char keyMatrix[SIZE][SIZE];
    cout << "Enter key: ";
    cin >> key;
    generateKeyMatrix(key, keyMatrix);
    cout << "Enter text to encrypt: ";
    cin >> text;
    string encryptedText = encrypt(text, keyMatrix);
    cout << "Encrypted Text: " << encryptedText << endl;
    string decryptedText = decrypt(encryptedText, keyMatrix);
    cout << "Decrypted Text: " << decryptedText << endl;
    return 0;
}
```

- **Output:**

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\
Enter key: RJ
Enter text to encrypt: RIAUHAS
Encrypted Text: IACSFCXA
Decrypted Text: RIAUHASX
PS D:\CSE\CSE_github\SEM 6\CNS>
```

Practical – 7

AIM: Write a program to implement Rail-Fence, Simple columnar Encryption Technique.

- **Code:**

```
#include <iostream>
#include <cstring>
using namespace std;
string railFenceEncrypt(string text, int key) {
    char rail[key][text.length()];
    memset(rail, ' ', sizeof(rail));
    int row = 0, direction = 1;
    for (int i = 0; i < text.length(); i++) {
        rail[row][i] = text[i];
        row += direction;
        if (row == key - 1 || row == 0) direction *= -1;
    }
    string encryptedText = "";
    for (int i = 0; i < key; i++) {
        for (int j = 0; j < text.length(); j++) {
            if (rail[i][j] != ' ') encryptedText += rail[i][j];
        }
    }
    return encryptedText;
}
string railFenceDecrypt(string cipher, int key) {
    char rail[key][cipher.length()];
    memset(rail, ' ', sizeof(rail));
    int row = 0, direction = 1;
    for (int i = 0; i < cipher.length(); i++) {
        rail[row][i] = '*';
        row += direction;
        if (row == key - 1 || row == 0) direction *= -1;
    }
    int index = 0;
    for (int i = 0; i < key; i++) {
        for (int j = 0; j < cipher.length(); j++) {
            if (rail[i][j] == '*' && index < cipher.length()) {
                rail[i][j] = cipher[index++];
            }
        }
    }
    string decryptedText = "";
```

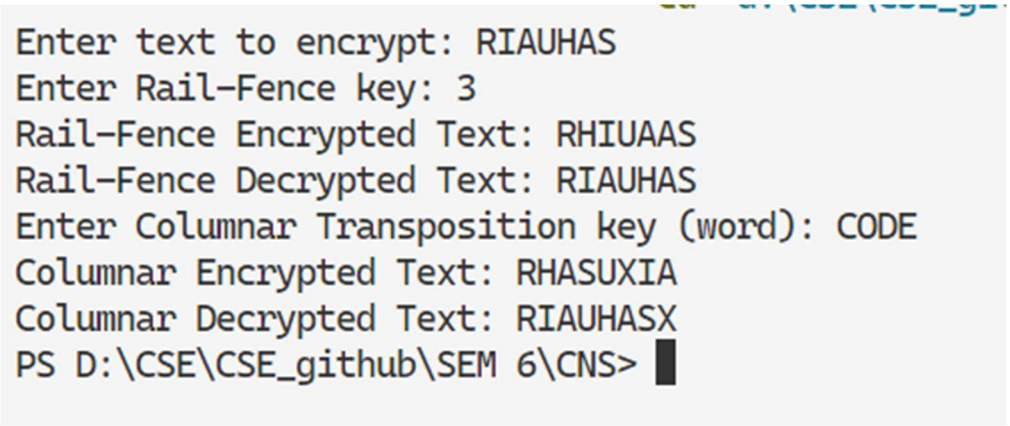
```
    row = 0, direction = 1;
    for (int i = 0; i < cipher.length(); i++) {
        decryptedText += rail[row][i];
        row += direction;
        if (row == key - 1 || row == 0) direction *= -1;
    }
    return decryptedText;
}
void sortKey(string key, int keyOrder[]) {
    int len = key.length();
    char tempKey[len];
    for (int i = 0; i < len; i++) tempKey[i] = key[i];
    for (int i = 0; i < len; i++) {
        int minIdx = i;
        for (int j = i + 1; j < len; j++) {
            if (tempKey[j] < tempKey[minIdx]) {
                minIdx = j;
            }
        }
        swap(tempKey[i], tempKey[minIdx]);
        swap(keyOrder[i], keyOrder[minIdx]);
    }
}
string columnarEncrypt(string text, string key) {
    int keyLen = key.length();
    int textLen = text.length();
    int numRows = (textLen + keyLen - 1) / keyLen;
    char grid[numRows][keyLen];
    int index = 0;
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < keyLen; j++) {
            grid[i][j] = (index < textLen) ? text[index++] : 'X';
        }
    }
    int keyOrder[keyLen];
    for (int i = 0; i < keyLen; i++) keyOrder[i] = i;
    sortKey(key, keyOrder);

    string encryptedText = "";
    for (int i = 0; i < keyLen; i++) {
        int col = keyOrder[i];
        for (int j = 0; j < numRows; j++) {
            encryptedText += grid[j][col];
        }
```

```
    }
    return encryptedText;
}
string columnarDecrypt(string cipher, string key) {
    int keyLen = key.length();
    int cipherLen = cipher.length();
    int numRows = (cipherLen + keyLen - 1) / keyLen;
    char grid[numRows][keyLen];
    int keyOrder[keyLen];
    for (int i = 0; i < keyLen; i++) keyOrder[i] = i;
    sortKey(key, keyOrder);
    int index = 0;
    for (int i = 0; i < keyLen; i++) {
        int col = keyOrder[i];
        for (int j = 0; j < numRows; j++) {
            grid[j][col] = cipher[index++];
        }
    }
    string decryptedText = "";
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < keyLen; j++) {
            decryptedText += grid[i][j];
        }
    }
    return decryptedText;
}
int main() {
    string text, key;
    int railKey;
    cout << "Enter text to encrypt: ";
    cin >> text;
    cout << "Enter Rail-Fence key: ";
    cin >> railKey;
    string railCipher = railFenceEncrypt(text, railKey);
    cout << "Rail-Fence Encrypted Text: " << railCipher << endl;
    string railDecrypted = railFenceDecrypt(railCipher, railKey);
    cout << "Rail-Fence Decrypted Text: " << railDecrypted << endl;
    cout << "Enter Columnar Transposition key (word): ";
    cin >> key;
    string columnCipher = columnarEncrypt(text, key);
    cout << "Columnar Encrypted Text: " << columnCipher << endl;
    string columnDecrypted = columnarDecrypt(columnCipher, key);
    cout << "Columnar Decrypted Text: " << columnDecrypted << endl;
    return 0;
```

```
}
```

- **Output:**


Enter text to encrypt: RIAUHAS
Enter Rail-Fence key: 3
Rail-Fence Encrypted Text: RHIUAAS
Rail-Fence Decrypted Text: RIAUHAS
Enter Columnar Transposition key (word): CODE
Columnar Encrypted Text: RHASUXIA
Columnar Decrypted Text: RIAUHASX
PS D:\CSE\CSE_github\SEM 6\CNS>

Practical – 8

AIM: To implement the S-DES algorithm for data encryption.

- **Code:**

```
#include <iostream>
#include <string>
using namespace std;
int P10[] = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
int P8[] = {6, 3, 7, 4, 8, 5, 10, 9};
int IP[] = {2, 6, 3, 1, 4, 8, 5, 7};
int IP_inv[] = {4, 1, 3, 5, 7, 2, 8, 6};
int EP[] = {4, 1, 2, 3, 2, 3, 4, 1};
int P4[] = {2, 4, 3, 1};
int S0[4][4] = {
   {1, 0, 3, 2}, {3, 2, 1, 0}, {0, 2, 1, 3}, {3, 1, 3, 2}};
int S1[4][4] = {
   {0, 1, 2, 3}, {2, 0, 1, 3}, {3, 0, 1, 0}, {2, 1, 0, 3}};
string permute(string input, int* table, int size) {
   string output = "";
   for (int i = 0; i < size; i++) {
      output += input[table[i] - 1];
   }
   return output;
}
string leftShift(string key, int shifts) {
   return key.substr(shifts) + key.substr(0, shifts);
}
string generateKey(string key, bool first) {
   key = permute(key, P10, 10);
   key = leftShift(key.substr(0, 5), first ? 1 : 2) + leftShift(key.substr(5, 5), first ? 1 : 2);
   return permute(key, P8, 8);
}
string xorOperation(string a, string b) {
   string result = "";
   for (size_t i = 0; i < a.length(); i++) {
      result += (a[i] == b[i]) ? '0' : '1';
   }
   return result;
}
string sBox(string input, int S[4][4]) {
   int row = (input[0] - '0') * 2 + (input[3] - '0');
   int col = (input[1] - '0') * 2 + (input[2] - '0');
   int val = S[row][col];
```

```
    return string(1, '0' + (val / 2)) + string(1, '0' + (val % 2));
}
string fk(string input, string key) {
    string left = input.substr(0, 4);
    string right = input.substr(4, 4);
    string expandedRight = permute(right, EP, 8);
    string xored = xorOperation(expandedRight, key);
    string sboxOut = sBox(xored.substr(0, 4), S0) + sBox(xored.substr(4, 4), S1);
    string permuted = permute(sboxOut, P4, 4);
    return xorOperation(left, permuted) + right;
}
string swapHalves(string input) {
    return input.substr(4, 4) + input.substr(0, 4);
}
string encrypt(string plaintext, string key) {
    string K1 = generateKey(key, true);
    string K2 = generateKey(key, false);
    string permutedText = permute(plaintext, IP, 8);
    string firstRound = fk(permutedText, K1);
    string swapped = swapHalves(firstRound);
    string secondRound = fk(swapped, K2);
    return permute(secondRound, IP_inv, 8);
}
string decrypt(string ciphertext, string key) {
    string K1 = generateKey(key, true);
    string K2 = generateKey(key, false);
    string permutedText = permute(ciphertext, IP, 8);
    string firstRound = fk(permutedText, K2);
    string swapped = swapHalves(firstRound);
    string secondRound = fk(swapped, K1);
    return permute(secondRound, IP_inv, 8);
}
int main() {
    string key, plaintext;
    cout << "Enter 10-bit key: ";
    cin >> key;
    cout << "Enter 8-bit plaintext: ";
    cin >> plaintext;
    string ciphertext = encrypt(plaintext, key);
    cout << "Encrypted Text: " << ciphertext << endl;
    string decryptedText = decrypt(ciphertext, key);
    cout << "Decrypted Text: " << decryptedText << endl;
    return 0;
}
```

- **Output:**

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\
Enter 10-bit key: 1010000010
Enter 8-bit plaintext: 11010101
Encrypted Text: 11100011
Decrypted Text: 11010101
PS D:\CSE\CSE_github\SEM 6\CNS>
```

Practical – 9

AIM: Write a program to implement RSA asymmetric (public key and private key) - Encryption

- **Code:**

```
#include <iostream>
#include <string>
#include <cmath>
using namespace std;
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;
        }
        exp = exp >> 1;
        base = (base * base) % mod;
    }
    return result;
}
long long simpleHash(const string& message, long long n) {
    long long hash = 0;
    for (char c : message) {
        hash = (hash * 31 + c) % n;
    }
    return hash;
}
long long gcdExtended(long long a, long long b, long long* x, long long* y) {
    if (a == 0) {
        *x = 0;
        *y = 1;
        return b;
    }
    long long x1, y1;
    long long gcd = gcdExtended(b % a, a, &x1, &y1);
    *x = y1 - (b / a) * x1;
    *y = x1;
    return gcd;
}
long long modInverse(long long e, long long phi) {
    long long x, y;
    long long g = gcdExtended(e, phi, &x, &y);
```

```
    if (g != 1) {
        cerr << "Modular inverse doesn't exist";
        exit(EXIT_FAILURE);
    }
    return (x % phi + phi) % phi;
}
void generateKeys(long long& e, long long& d, long long& n) {
    long long p = 1009;
    long long q = 1013;
    n = p * q;
    long long phi = (p - 1) * (q - 1);
    e = 65537;
    d = modInverse(e, phi);
}
long long signMessage(long long hash, long long d, long long n) {
    return modExp(hash, d, n);
}
bool verifySignature(long long hash, long long signature, long long e, long long n) {
    long long decryptedHash = modExp(signature, e, n);
    return hash == decryptedHash;
}
int main() {
    string message;
    cout << "Enter message: ";
    getline(cin, message);
    long long e, d, n;
    generateKeys(e, d, n);
    long long hashValue = simpleHash(message, n);
    long long signature = signMessage(hashValue, d, n);
    cout << "Original Hash: " << hashValue << endl;
    cout << "Signature: " << signature << endl;
    if (verifySignature(hashValue, signature, e, n)) {
        cout << "Signature verified successfully!" << endl;
    } else {
        cout << "Signature verification failed!" << endl;
    }
    return 0;
}
```

- **Output:**

```
PS D:\CSE\CSE_github\SEM 6\CNS> cd "d:\C
Enter message: RIAUHAS
Original Hash: 553453
Signature: 961365
Signature verified successfully!
PS D:\CSE\CSE_github\SEM 6\CNS>
```