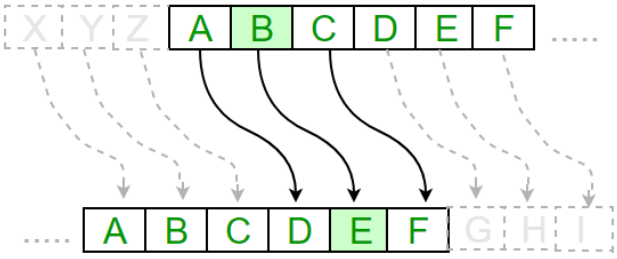
Experiment 1

Aim: To implement Caesar Cipher encryption algorithm

Theory:

It is one of the earliest and simplest methods of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter with a fixed number of positions down the alphabet. Thus to cipher a given text we need an integer value, known as a shift which indicates the number of positions each letter of the text has been moved down.



Code:

```
import sys
def encrypt(text, s):
    result = ""
    for char in text:
        if char == " ":
            result += char
        elif (char.isupper()):
            result += chr ((ord (char) + s-65) % 26 + 65)
        else:
            result += chr ((ord (char) + s - 97) % 26 + 97)
    return result

def runner():
    while 1:
        print(" ")
        print("1. Encryption")
        print("2. Decryption")
        print("3. Exit")
        choice = int(input("Enter choice: "))
```

```
        print(" ")
        if (choice == 3):
            sys.exit("")
        elif (choice == 1 or choice == 2):
            u_input = input("Enter message: ") if (choice == 1) else input("Enter cipher text: ")
            u_key = int(input("Enter the key: "))
            if(choice == 1):
                output = encrypt(u_input, u_key)
                print ("Result is: " + output)
            elif(choice == 2):
                output = encrypt(u_input, 26-u_key)
                print("Result is: " + output)
        else:
            print("Invalid input")
runner()
```

Output:

```
yashwalia@Yashs-MacBook-Air crypto % python caesar.py

1. Encryption
2. Decryption
3. Exit
Enter choice: 1

Enter message: abcde
Enter the key: 1
Result is: bcdef

1. Encryption
2. Decryption
3. Exit
Enter choice: 2

Enter cipher text: bcdef
Enter the key: 1
Result is: abcde

1. Encryption
2. Decryption
3. Exit
Enter choice: 3


yashwalia@Yashs-MacBook-Air crypto %
```

Experiment 2

Aim: To implement the monoalphabetic cipher encryption algorithm

Theory:

A mono-alphabetic cipher (aka simple substitution cipher) is a substitution cipher where each letter of the plain text is replaced with another letter of the alphabet. It uses a fixed key which consists of the 26 letters of a “shuffled alphabet”.
This type of cipher is a form of symmetric encryption as the same key can be used to both encrypt and decrypt a message.

Code:

```
#include<iostream>
#include<string>
#include<unordered_map>
using namespace std;
char decrypt(unordered_map<char,char> map,char a){
    auto iterate = map.begin();
    char c = '/';
    while(iterate!=map.end()){
        if(iterate->second == a){
            c = iterate->first;
        }
        iterate++;
    }
    return c;
}
int main(){
    unordered_map<char,char> mapKey;
    string a = "attack postponed to tomorrow and do not use our secret paper until further
information";
    string assigned;
    string assignedkey;
    string key = "the quick brown fox jumps over the lazy dog";
    int index = 0;
    cout<<"Message: "<<a<<endl;
    cout<<"Key: "<<key<<endl;
```

```
    for(int i =0;i<a.length();i++){
        if(a[i] == ' '){
            continue;
        }
        if(assigned.find(a[i])!=-1){
            continue;
        }
        else{
            assigned+=a[i];
            if(key[index] == ' '||assignedkey.find(key[index])!=-1){
                index++;
            }
            mapKey[a[i]] = key[index];
            assignedkey+= key[index];
            index++;
        }
    }
   cout<<endl;
    cout<<"Encrypted Text: "<<endl;
    cout<<endl;
    for(int i = 0;i<a.length();i++){
        if(a[i] == ' '){
            continue;
        }
        a[i] = mapKey[a[i]];
    }
    cout<<a<<endl;
    for(int i = 0;i<a.length();i++){
        if(a[i] == ' '){
            continue;
        }
        char c = decrypt(mapKey,a[i]);
        a[i] = c;
    }
    cout<<endl;
    cout<<"Decrypted Text: "<<endl;
    cout<<endl;
    cout<<a;
}
```

Output:

```
Message: attack postponed to tomorrow and do not use our secret paper until further information
Key: the quick brown fox jumps over the lazy dog

Encrypted Text:

thhteq uichuikbr hi hioiwwin tkr ri kih fcb ifw cbewbh utubw fkhxj mfwhpbw xkmiwothxik

Decrypted Text:

attack postponed to tomorrow and do not use our secret paper until further information
Process finished with exit code 0
```

Experiment 3

Aim: To implement the playfair cipher encryption algorithm

Theory:

The Playfair cipher was the first practical digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher.
The Algorithm consists of 2 steps:

1. Generate the key Square(5×5):

The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I. The initial letters in the key square are the unique letters of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

2. Algorithm to encrypt the plain text: The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

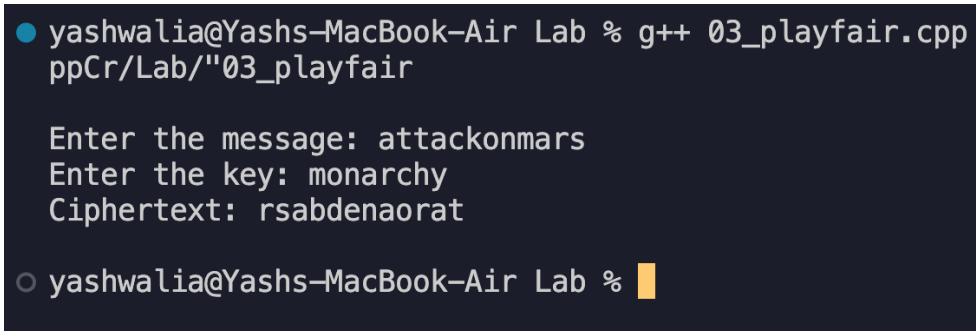
Code:

```
#include<iostream>
#include<string>
#include<vector>
#include<map>

using namespace std;
int main() {
  int i, j, k, n;
  cout << "\nEnter the message: ";
  string s, origin;

  getline(cin, origin);
  cout << "Enter the key: ";
  string key;
  cin >> key;
  for (i = 0; i < origin.size(); i++) {
    if (origin[i] != ' ')
      s += origin[i];
```

```
}
vector < vector < char > > a(5, vector < char > (5, ' '));
n = 5;
map < char, int > mp;
k = 0;
int pi, pj;
for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++) {
    while (mp[key[k]] > 0 && k < key.size()) {
      k++;
    }
    if (k < key.size()) {
      a[i][j] = key[k];
      mp[key[k]]++;
      pi = i;
      pj = j;
    }
    if (k == key.size())
      break;
  }

  if (k == key.size())
    break;
}
k = 0;
for (; i < n; i++) {
  for (; j < n; j++) {
    while (mp[char(k + 'a')] > 0 && k < 26) {
      k++;
    }
    if (char(k + 'a') == 'j') {
      j--;
      k++;
      continue;
    }
    if (k < 26) {
      a[i][j] = char(k + 'a');
      mp[char(k + 'a')]++;
    }
  }
```

```
    j = 0;
  }

  string ans;
  if (s.size() % 2 == 1)
    s += "x";
  for (i = 0; i < s.size() - 1; i++) {
    if (s[i] == s[i + 1])

      s[i + 1] = 'x';
  }

  map < char, pair < int, int > > mp2;

  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      mp2[a[i][j]] = make_pair(i, j);
    }
  }

  for (i = 0; i < s.size() - 1; i += 2) {
    int y1 = mp2[s[i]].first;
    int x1 = mp2[s[i]].second;
    int y2 = mp2[s[i + 1]].first;
    int x2 = mp2[s[i + 1]].second;
    if (y1 == y2) {
      ans += a[y1][(x1 + 1) % 5];
      ans += a[y1][(x2 + 1) % 5];
    } else if (x1 == x2) {
      ans += a[(y1 + 1) % 5][x1];
      ans += a[(y2 + 1) % 5][x2];
    } else {
      ans += a[y1][x2];
      ans += a[y2][x1];
    }
  }
  cout <<"Ciphertext: " << ans << "\n\n";
  return 0;
}
```

Output:


yashwalia@Yashs-MacBook-Air Lab % g++ 03_playfair.cpp
ppCr/Lab/"03_playfair
Enter the message: attackonmars
Enter the key: monarchy
Ciphertext: rsabdenaorat
yashwalia@Yashs-MacBook-Air Lab %

Experiment 4

Aim: To implement the hill cipher encryption algorithm

Theory:

The Hill cipher is a polygraphic substitution cipher based on linear algebra.Each letter is represented by a number modulo 26.
To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26.
The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).

Code:

```
import math

def getKeyMatrix(key, kdim):
    k = 0
    keyMatrix = [[0] * kdim for i in range(kdim)]

    for i in range(kdim):
        for j in range(kdim):
            keyMatrix[i][j] = ord(key[k]) % 65
            k += 1
    return keyMatrix

def encrypt(messageVector, keyMatrix, keyDim, d=0):
    cipherMatrix = [[0] for i in range(keyDim)]
    for i in range(keyDim):
        for j in range(1):
            cipherMatrix[i][j] = 0
            for x in range(keyDim):
                cipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
            if d == 0:
                cipherMatrix[i][j] = cipherMatrix[i][j] % 26
    return cipherMatrix

def HillCipher(message, key, keyDim):
    keyMatrix = getKeyMatrix(key, keyDim)
```

```
    messageVector = [[0] for i in range(keyDim)]
    CipherText = []

    for i in range(keyDim):
        messageVector[i][0] = ord(message[i]) % 65

    cipherMatrix = encrypt(messageVector, keyMatrix, keyDim)
    for i in range(keyDim):
        CipherText.append(chr(cipherMatrix[i][0] + 65))

    return CipherText

def main():
    msg = "".join(
        (input("\nEnter the message to be encrypted: ").upper()).split(" "))
    key = input("Enter the key: ").upper()
    keyDim = (int(math.sqrt(len(key))))

    if keyDim == 0 or keyDim ** 2 != len(key):
        print("Invalid Key")
    else:
        rem = len(msg) % keyDim
        if rem != 0:
            msg += "X" * (keyDim - rem)
        msgList = [msg[i:i + keyDim] for i in range(0, len(msg), keyDim)]
        cipherList = []
        for ms in msgList:
            out = HillCipher(ms, key, keyDim)
            cipherList.append(out)

        FinalCipher = ""
        for x in cipherList:
            FinalCipher += "".join(x)
            FinalCipher += " "
        print("Final Ciphertext: ", FinalCipher)
        print(" ")

if __name__ == "__main__":
    main()
```

Output:

```
yashwalia@Yashs-MacBook-Air Lab % python temp/04_hill.py

Enter the message to be encrypted: the quick brown fox
Enter the key: gybnqkurp
Message is:  THEQUICKBROWNFOXXX
After division, message is:  THE QUI CKB ROW NFO XXX
Final Ciphertext:  AJN MKA TOR SPY EZJ LNA

yashwalia@Yashs-MacBook-Air Lab %
```

```
yashwalia@Yashs-MacBook-Air Lab % python temp/04_hill.py

Enter the message to be encrypted: the quick brown fox
Enter the key: hill
Message is:  THEQUICKBROWNFOX
After division, message is:  TH EQ UI CK BR OW NF OX
Final Ciphertext:  HA AM WW QC NQ OG BQ WR

yashwalia@Yashs-MacBook-Air Lab %
```

Experiment 5

Aim: To implement the Vernam Cipher encryption algorithm

Theory:

Vernam Cipher is a method of encrypting alphabetic text. It is one of the Substitution techniques for converting plain text into cipher text. In this mechanism we assign a number to each character of the Plain-Text, like (a = 0, b = 1, c = 2, … z = 25).
The key is taken to encrypt the plain text whose length should be equal to the length of the plain text.

Encryption Algorithm:
Assign a number to each character of the plain-text and the key according to alphabetical order.
Bitwise XOR both the number (Corresponding plain-text character number and Key character number).
Subtract the number from 26 if the resulting number is greater than or equal to 26, if it isn't then leave it.

Code:

```
plaintext = input("\nEnter Plaintext: ").upper()
otp = input("Enter the Key/One Time Pad: ").upper()
if len(plaintext) != len(otp):
    print("Invalid Key")
    print(" ")
    quit()
plaintextList = []
otpList = []

for c1 in plaintext:
    temp = ord(c1) - 65
    plaintextList.append(temp)

for c2 in otp:
    temp = ord(c2) - 65
    otpList.append(temp)

for number in plaintextList:
    if (number < 0):
```

```
        plaintextList.remove(number)

for number in otpList:
    if (number < 0):
        otpList.remove(number)

print("Plaintext: ", plaintextList)
print("Key: ", otpList)

tempList = []
for i in range(len(plaintextList)):
    tempList.append(plaintextList[i] ^ otpList[i])

ciphertextList = []
for item in tempList:
    if(item > 25):
        item -= 26
        ciphertextList.append(item)
    else:
        ciphertextList.append(item)

print("Cipher Matrix: ", ciphertextList)

ct = ""
for number in ciphertextList:
    number += 65
    temp = chr(number)
    tempstr = str(temp)
    ct = ct + tempstr

print("Ciphertext: ", ct.lower())
print(" ")
```

Output:

```
yashwalia@Yashs-MacBook-Air Lab % python vernam/ver.py

Enter Plaintext: kumbaya
Enter the Key/One Time Pad: polygon
Plaintext:  [10, 20, 12, 1, 0, 24, 0]
Key:  [15, 14, 11, 24, 6, 14, 13]
Cipher Matrix:  [5, 0, 7, 25, 6, 22, 13]
Ciphertext:  fahzgwn
```

Experiment 6

Aim: To implement the One Time Pad encryption algorithm

Theory:

One-Time Pad is the only algorithm that is truly unbreakable and can be used for low-bandwidth channels requiring very high security(ex. for military uses).

To encrypt every new message requires a new key of the same length as the new message in one-time pad.

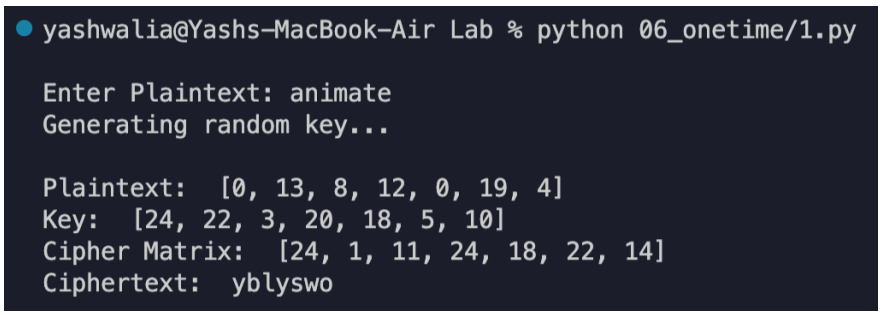
The ciphertext generated by the One-Time pad is random, so it does not have any statistical relation with the plain text.

Code:

```
import random
import string

plaintext = input("\nEnter Plaintext: ").upper()
print("Generating random key...")
print(" ")

otp = ''.join((random.choice(string.ascii_uppercase)
            for x in range(len(plaintext))))

plaintextList = []
otpList = []

for c1 in plaintext:
   temp = ord(c1) - 65
   plaintextList.append(temp)

for c2 in otp:
   temp = ord(c2) - 65
   otpList.append(temp)

print("Plaintext: ", plaintextList)
print("Key: ", otpList)
```

```
tempList = []
for i in range(len(plaintextList)):
    tempList.append(plaintextList[i] ^ otpList[i])

ciphertextList = []
for item in tempList:
    if(item > 25):
        item -= 26
        ciphertextList.append(item)
    else:
        ciphertextList.append(item)

print("Cipher Matrix: ", ciphertextList)

ct = ""
for number in ciphertextList:
    number += 65
    temp = chr(number)
    tempstr = str(temp)
    ct = ct + tempstr

print("Ciphertext: ", ct.lower())
print(" ")
```

Output:



Experiment 7

Aim: To implement the Rail Fence encryption algorithm

Theory:

The rail fence cipher (also called a zigzag cipher) is an easy to apply form of transposition cipher. It derives its name from the way in which it is encoded.
It jumbles up the order of the letters of a message in a quick convenient way. It also has the security of a key to make it a little bit harder to break.
The Rail Fence cipher works by writing the message on alternate lines across the page, and then reading off each line in turn.
For example, the plaintext "defend the east wall" is written as shown below, with all spaces removed.

D		F		N		T		E		A		T		A		L
	E		E		D		H		E		S		W		L	

The ciphertext is then read off by writing the top row first, followed by the bottom row, to get "DFNTEATALEEDHESWL".

Code:

```
def encryptRailFence(text, key):
    rail = [['\n' for i in range(len(text))]
            for j in range(key)]

    dir_down = False
    row, col = 0, 0

    for i in range(len(text)):
        if (row == 0) or (row == key - 1):
            dir_down = not dir_down

        rail[row][col] = text[i]
        col += 1

        if dir_down:
            row += 1
        else:
            row -= 1
```

```
    result = []
    for i in range(key):
        for j in range(len(text)):
            if rail[i][j] != '\n':
                result.append(rail[i][j])
    return("" . join(result))


def decryptRailFence(cipher, key):

    rail = [['\n' for i in range(len(cipher))]
            for j in range(key)]

    dir_down = None
    row, col = 0, 0

    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key - 1:
            dir_down = False

        rail[row][col] = '*'
        col += 1

        if dir_down:
            row += 1
        else:
            row -= 1

    index = 0
    for i in range(key):
        for j in range(len(cipher)):
            if ((rail[i][j] == '*') and
               (index < len(cipher))):
                rail[i][j] = cipher[index]
                index += 1

    result = []
    row, col = 0, 0
```

```
    for i in range(len(cipher)):
        if row == 0:
            dir_down = True
        if row == key-1:
            dir_down = False

        if (rail[row][col] != '*'):
            result.append(rail[row][col])
            col += 1

        if dir_down:
            row += 1
        else:
            row -= 1
    return("".join(result))

def main():
    plaintext = input("\nEnter Plaintext: ")
    rails = int(input("Enter no. of rails: "))
    out = encryptRailFence(plaintext, rails)
    print("Ciphertext: ", out)
    out2 = decryptRailFence(out, rails)
    print("After deciphering: ", out2)
    print(" ")

main()
```

Output:

```
yashwalia@Yashs-MacBook-Air Lab % python 07_rail/1.py

Enter Plaintext: defend the east wall
Enter no. of rails: 2
Ciphertext:  dfn h atwleedtees al
After deciphering:  defend the east wall

yashwalia@Yashs-MacBook-Air Lab % python 07_rail/1.py

Enter Plaintext: defend the east wall
Enter no. of rails: 3
Ciphertext:  dnhaweedtees alf  tl
After deciphering:  defend the east wall

yashwalia@Yashs-MacBook-Air Lab % python 07_rail/1.py

Enter Plaintext: defend the east wall
Enter no. of rails: 4
Ciphertext:  d aledtesalfnh twee
After deciphering:  defend the east wall
```

Experiment 8

Aim: To implement Row Column Transposition Algorithm

Theory:

The Row Columnar Transposition Cipher is a form of transposition cipher just like Rail Fence Cipher. Row Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one as per the key provided.
For Encryption, the message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order. For Decryption, the recipient must work out the column lengths by dividing the message length by the key length. Then, write the message out in columns again, then re- order the columns by reforming the key word.

Code:

```
#include<bits/stdc++.h>
using namespace std;

string const key = "dcabefg";
map < int, int > keyMap;
void setPermutationOrder() {
  for (int i = 0; i < key.length(); i++) {
    keyMap[key[i]] = i;
  }
}
// Encryption
string encryptMessage(string msg) {
  int row, col, j;
  string cipher = "";
  col = key.length();
  row = msg.length() / col;
  if (msg.length() % col)
    DATE: 12 / 09 / 2022
  row += 1;
  char matrix[row][col];
  for (int i = 0, k = 0; i < row; i++) {
    for (int j = 0; j < col;) {
      if (msg[k] == '\0') {
```

```
      matrix[i][j] = '_';
      j++;
     }
     if (isalpha(msg[k]) || msg[k] == ' ') {
      matrix[i][j] = msg[k];
      j++;
     }
     k++;
    }
   }
   for (map < int, int > ::iterator ii = keyMap.begin(); ii != keyMap.end(); ++ii) {
    j = ii -> second;
    for (int i = 0; i < row; i++) {}
   }
   return cipher;
}
if (isalpha(matrix[i][j]) || matrix[i][j] == ' ' || matrix[i][j] == '_') cipher += matrix[i][j];

// Decryption
string decryptMessage(string cipher) {
  int col = key.length();
  int row = cipher.length() / col;
  char cipherMat[row][col];
  for (int j = 0, k = 0; j < col; j++)
    for (int i = 0; i < row; i++)
      cipherMat[i][j] = cipher[k++];
  int index = 0;
  for (map < int, int > ::iterator ii = keyMap.begin(); ii != keyMap.end(); ++ii)
    ii -> second = index++;
  char decCipher[row][col];
  map < int, int > ::iterator ii = keyMap.begin();
  int k = 0;
  for (int l = 0, j; key[l] != '\0'; k++) {
    j = keyMap[key[l++]];
    for (int i = 0; i < row; i++) {
      decCipher[i][k] = cipherMat[i][j];
    }
  }
  string msg = "";
  for (int i = 0; i < row; i++) {
```

```
    for (int j = 0; j < col; j++) {
      if (decCipher[i][j] != '_')

        msg += decCipher[i][j];
    }
  }
  return msg;
}
int main(void) {
  string msg = "killcoronavirusattwelveamtomorrow";
  cout << "Original Message: " << msg << endl;
  setPermutationOrder();
  string cipher = encryptMessage(msg);
  cout << "Encrypted Message: " << cipher << endl;
  cout << "Decrypted Message: " << decryptMessage(cipher) << endl;
  return 0;
}
```

Output:

```
Original Message: killcoronavirusattwelveamtomorrow
Encrypted Message: latarlvtmoinaerkosvociwtworeo_rulm_
Decrypted Message: killcoronavirusattwelveamtomorrow
```

Experiment 9

Aim: To implement DES Encryption algorithm

Code:

```
#include<iostream>
#include<bitset>
#include<math.h>
using namespace std;
int ls1[10];

int ip[8] = {1,5,2,0,3,7,4,6};
int ipin[8] = {3,0,2,4,6,1,7,5};
int ep[8] = {3,0,1,2,1,2,3,0};
int p4[4] = {1,3,2,0};

string s0[4][4]={ "01","00","11","10","11","10","01","00", "00","10","01","11",
"11","01","11","10"};
string s1[4][4] = { "00","01","10","11",
                    "10","00","01","11",
                    "11","00","01","00",
                    "10","01","00","11"};
int *keyGeneration2(){
  int p10[10] = {2,4,1,6,3,9,0,8,7,5};
  int p8[8] = {5,2,6,3,7,4,9,8};
  int ls2[10];
  int left2[5];
  int right2[5];
  for (int i = 0; i < 5; i++) {
    left2[i] = ls1[i];
  }
  for (int i = 5; i < 10; i++) {
    right2[i - 5] = ls1[i];
  }
  cout << endl;
  for (int i = 0; i < 10; i++) {
    if (i < 5) {
      if (i == 3) {
        ls2[i] = left2[0];
```

```
      } else if (i == 4) {
        ls2[i] = left2[1];
      } else {
        Date: 31 / 10 / 2022

        ls2[i] = left2[i + 2];
      }
    } else {
      if (i == 8) {
        ls2[i] = right2[0];
      } else if (i == 9) {
        ls2[i] = right2[1];
      } else {
        ls2[i] = right2[(i + 2) - 5];
      }
    }
  }
  int * key2 = new int[8];
  for (int i = 0; i < 8; i++) {
    key2[i] = ls2[p8[i]];
  }
  return key2;
}
int * keyGeneration1(int * keyMain) {
  int p10[10] = {2,4,1,6,3,9,0,8,7,5};
  int p8[8] = {5,2,6,3,7,4,9,8};
  int left[5];
  int right[5];
  pair < int * , int * > keys;
  for (int i = 0; i < 5; i++) {
    left[i] = keyMain[p10[i]];
  }
  for (int i = 0; i < 5; i++) {
    right[i] = keyMain[p10[5 + i]];
  }
  for (int i = 0; i < 10; i++) {
    if (i < 5) {
      if (i == 4) {
        ls1[i] = left[0];
      } else {
```

```
      ls1[i] = left[i + 1];
     }
    } else {
     if (i == 9) {
      ls1[i] = right[0];
     } else {
      ls1[i] = right[(i + 1) - 5];
     }
    }

  }
  int * key1 = new int[8];
  for (int i = 0; i < 8; i++) {
   key1[i] = ls1[p8[i]];
  }
  return key1;
}
int * round1(int * pt, int * key1) {
  int left[4], right[4];
  for (int i = 0; i < 4; i++) {
   left[i] = pt[ip[i]];
  }
  for (int i = 4; i < 8; i++) {
   right[i - 4] = pt[ip[i]];
  }
  int getEP[8];
  for (int i = 0; i < 8; i++) {
   getEP[i] = right[ep[i]];
  }
  for (int i = 0; i < 8; i++) {
   getEP[i] = getEP[i] ^ key1[i];
  }
  for (int i = 0; i < 4; i++) {
   left[i] = getEP[i];
  }
  for (int i = 4; i < 8; i++) {
   right[i - 4] = getEP[i];
  }
  int rowleft[2] = {
   left[0],
```

```
  left[3]
};
int colleft[2] = {
  left[1],
  left[2]
};
int row = 0, col = 0;
for (int i = 0; i < 2; i++) {
  if (rowleft[i] == 0) {
    continue;
  } else {
    row += pow(2, 1 - i);
  }
}
for (int i = 0; i < 2; i++) {
  if (colleft[i] == 0) {
    continue;
  } else {
    col += pow(2, 1 - i);
  }
}
string a = s0[row][col];
int op[2];
for (int i = 0; i < a.length(); i++) {

  op[i] = (int) a[i] - '0';
}
int rowright[2] = {
  right[0],
  right[3]
};
int colright[2] = {
  right[1],
  right[2]
};
int rrow = 0, rcol = 0;
for (int i = 0; i < 2; i++) {
  if (rowright[i] == 0) {
    continue;
  } else {
```

```
    rrow += pow(2, 1 - i);
  }
}
for (int i = 0; i < 2; i++) {
  if (colright[i] == 0) {
    continue;
  } else {
    rcol += pow(2, 1 - i);
  }
}
string b = s1[rrow][rcol];
int op2[2];
for (int i = 0; i < b.length(); i++) {
  op2[i] = (int) b[i] - '0';
}
int fop[4];
for (int i = 0; i < 4; i++) {
  if (i < 2) {
    fop[i]
  } else {
    fop[i]
  }
}
int newfop[4];
for (int i = 0; i < 4; i++) {
  newfop[i] = fop[p4[i]];
}
for (int i = 0; i < 4; i++) {
  left[i] = pt[ip[i]];
}
int oxor[4];
for (int i = 0; i < 4; i++) {
  oxor[i] = newfop[i] ^ left[i];
}
int * finalOp = new int[8];
for (int i = 4; i < 8; i++) {
  right[i - 4] = pt[ip[i]];
} = op[i]; = op2[i - 2];

for (int i = 0; i < 8; i++) {
```

```
    if (i < 4) {
      finalOp[i] = right[i];
    } else {
      finalOp[i] = oxor[i - 4];
    }
  }
  return finalOp;
}
int * round2(int * r1, int * key2) {
  int getEP[8];
  int left[4] = {
    r1[0],
    r1[1],
    r1[2],
    r1[3]
  };
  int right[4] = {
    r1[4],
    r1[5],
    r1[6],
    r1[7]
  };
  for (int i = 0; i < 8; i++) {
    getEP[i] = right[ep[i]];
  }
  int getXor[8];
  for (int i = 0; i < 8; i++) {
    getXor[i] = getEP[i] ^ key2[i];
  }
  int rowleft[2] = {
    getXor[0],
    getXor[3]
  };
  int colleft[2] = {
    getXor[1],
    getXor[2]
  };
  int row = 0, col = 0;
  for (int i = 0; i < 2; i++) {
    if (rowleft[i] == 0) {
```

```
    continue;
  } else {
    row += pow(2, 1 - i);
  }
}
for (int i = 0; i < 2; i++) {
  if (colleft[i] == 0) {
    continue;
  } else {
    col += pow(2, 1 - i);
  }
}
string a = s0[row][col];
int op[2];
cout << endl;
for (int i = 0; i < a.length(); i++) {
  op[i] = (int) a[i] - '0';
}
int rowright[2] = {
  getXor[4],
  getXor[7]
};
int colright[2] = {
  getXor[5],
  getXor[6]
};
int rrow = 0, rcol = 0;
for (int i = 0; i < 2; i++) {

  if (rowright[i] == 0) {
    continue;
  } else {
    rrow += pow(2, 1 - i);
  }
}
for (int i = 0; i < 2; i++) {
  if (colright[i] == 0) {
    continue;
  } else {
    rcol += pow(2, 1 - i);
```

```
    }
  }
  string b = s1[rrow][rcol];
  int op2[2];
  for (int i = 0; i < b.length(); i++) {
    op2[i] = (int) b[i] - '0';
  }
  int fop[4] = {
    op[0],
    op[1],
    op2[0],
    op2[1]
  };
  int p4op[4];
  for (int i = 0; i < 4; i++) {
    p4op[i] = fop[p4[i]];
  }
  int * finalOp = new int[8];
  for (int i = 0; i < 8; i++) {
    if (i < 4) {
      finalOp[i] = left[i] ^ p4op[i];
    } else {
      finalOp[i] = right[i - 4];
    }
  }
  return finalOp;
}
int main() {
  int keyMain[10];
  int pt[8];
  cout << "PLEASE ENTER THE KEY: " << endl;
  for (int i = 0; i < 10; i++) {
    cin >> keyMain[i];
  }
  cout << "PLEASE ENTER THE PLAIN TEXT: ";
  cout << endl;
  for (int i = 0; i < 8; i++) {
    cin >> pt[i];
  }
  int * key1 = keyGeneration1(keyMain);
```

```
  int * key2 = keyGeneration2();

  cout << "FIRST KEY IS: ";
  for (int i = 0; i < 8; i++) {
    cout << key1[i];
  }
  cout << endl;
  cout << "SECOND KEY IS: ";
  for (int i = 0; i < 8; i++) {
    cout << key2[i];
  }
  cout << endl;
  int * r1 = round1(pt, key1);
  int * r2 = round2(r1, key2);
  int cipher[8];
  for (int i = 0; i < 8; i++) {
    cipher[i] = r2[ipin[i]];
  }
  cout << "THE ENCRYPTED CIPHER TEXT IS: ";
  for (int i: cipher) {
    cout << i;
  }
}
```

Output:

```
PLEASE ENTER THE KEY:
1
0
1
0
0
0
0
0
1
0
PLEASE ENTER THE PLAIN TEXT:
0
1
1
1
0
0
1
0

FIRST KEY IS: 10100100
SECOND KEY IS: 01000011

THE ENCRYPTED CIPHER TEXT IS: 01110111
```

Experiment 10

Aim: To implement the RSA encryption algorithm

Code:

```
#include<iostream>
#include<bits/stdc++.h>
#include<math.h>
using namespace std;

double encrypt(int m, pair < int, int > publicKey, int n) {
  double cipher;
  cipher = fmod(pow(m, publicKey.first), n);
  return cipher;
}
double decrypt(double c, pair < int, int > privateKey, int n) {
  double m;
  m = fmod(pow(c, privateKey.first), n);
  return m;
}
int gcd(int a, int b) {
  if (a == 0) return b;
  if (b == 0) return a;
  if (a == b) return a;
  if (a > b)
    return gcd(a - b, b);
  return gcd(a, b - a);
}
bool checkPrime(int n) {
  if (n <= 1) {
    return false;
  }
  for (int i = 2; i < sqrt(n); i++) {
    if (n % i == 0) {
      return false;
    }
    Date: 07 / 11 / 2022

  }
```

```
  return true;
}
int main() {
  int p, q;
  cout << "SELECT FIRST RANDOM PRIME NUMBER: ";
  cin >> p;
  cout << endl;
  cout << "SELECT SECOND RANDOM PRIME NUMBER: ";
  cin >> q;
  if (!checkPrime(q) || !checkPrime(p)) {
    cout << "WRONG NUMBERS SELECTED";
    return 0;
  }
  int n = p * q;
  int phi = (p - 1) * (q - 1);
  int e;
  for (int i = 2; i < phi; i++) {
    if (gcd(i, phi) == 1) {
      e = i;
      break;
    }
  }
  int d;
  for (int i = 1; i <= phi; i++) {
    if ((e * i) % phi == 1) {
      d = i;
      break;
    }
  }
  cout << endl;
  pair < int, int > publicKey;
  publicKey.first = e;
  publicKey.second = n;
  pair < int, int > privateKey;
  privateKey.first = d;
  privateKey.second = n;
  cout << "PLEASE ENTER THE MESSAGE IN INTEGER FORM: ";
  int m;

  cin >> m;
```

```
  cout << endl;
  double cipher = encrypt(m, publicKey, n);
  cout << "THE ENCRYPTED MESSAGE IS: " << cipher;
  cout << endl;
  cout << "THE DECRYPTED MESSAGE IS: " << decrypt(cipher, privateKey, n);
}
```

Output:

```
SELECT FIRST RANDOM PRIME NUMBER: 3

SELECT SECOND RANDOM PRIME NUMBER: 11

PLEASE ENTER THE MESSAGE IN INTEGER FORM: 24

THE ENCRYPTED MESSAGE IS: 30
THE DECRYPTED MESSAGE IS: 24
```