ENCRYPTION & DECRYPTION

GROUP MEMBERS

Ghulam Qadir

Muqeem Ahmed

Noor Malik

Sara Tariq

Khudema Haroon

MAY 29

Logo Name

Code For Encryption and Decryption:

```
#include <iostream>
#include <string>
#include <cctype>
using namespace std;

// Function Prototypes

string caesar_encrypt(string ptxt, int key);

string caesar_decrypt(string ctxt, int key);

string substitution_encrypt(string ptxt, int key);

void substitution_decrypt(string ctxt, int key);

string playfair_encrypt(string ptxt, char arr[5][5]);

string playfair_decrypt(string ctxt, char arr[5][5]);

string hill_encrypt(string ptxt, int kmatrix[3][3]);

string vigenere_encrypt(string ptxt, string key);

string vigenere_decrypt(string ctxt, string key);

string sorting_key(string key);

void cipher_text(string key, char sortk[], string plaintext);

int main()
{
   int choice;

   while (true)
         {
      cout<<"-------------------------------------------------"<<endl;
```

```
            cout << "\nChoose the encryption/decryption method:\n";
cout << "1. Caesar Cipher\n";
cout << "2. Substitution Cipher\n";
cout << "3. Playfair Cipher\n";
cout << "4. Hill Cipher\n";
cout << "5. Vigenère Cipher\n";
cout << "6. Exit\n"<<endl;
cin >> choice;

cin.ignore();


if (choice == 0) break;

string ptxt, ctxt;
int key;

switch (choice)
            {
   case 1:
                        {
                                cout<<"\n-----------------------------------------------"<<endl;
      cout << "\nEnter the plain text: ";
      getline(cin, ptxt);

      cout << "Enter the key: ";
      cin >> key;

      ctxt = caesar_encrypt(ptxt, key);

      cout << "Cipher text is: " << ctxt << endl;
      cout << "Decrypted text is: " << caesar_decrypt(ctxt, key) << endl;

      break;
   }

   case 2:
                        {
      cout<<"\n-----------------------------------------------"<<endl;
```

```
                cout << "Enter the plain text: ";
    getline(cin, ptxt);

    cout << "Enter the key: ";
    cin >> key;

    ctxt = substitution_encrypt(ptxt, key);

    cout << "Cipher text is: " << ctxt << endl;

    for (int i = 0; i < 26; i++)
                {
        substitution_decrypt(ctxt, i);
    }

    break;
}

case 3:
            {
    char arr[5][5] =
                {
        {'H', 'O', 'C', 'K', 'E'},
        {'Y', 'A', 'B', 'D', 'F'},
        {'G', 'I', 'L', 'M', 'N'},
        {'P', 'Q', 'R', 'S', 'T'},
        {'U', 'V', 'W', 'X', 'Z'}
    };

    cout<<"\n-----------------------------------------------"<<endl;
                cout << "Enter the plain text without spaces: ";
    getline(cin, ptxt);

    // Preprocess plaintext
    for (char &c : ptxt)
                {
        c = toupper(c);
        if (c == 'J') c = 'I';
    }
```

```
    // Insert 'X' between identical pairs and at the end if needed
    for (int i = 0; i < ptxt.length(); i += 2)
                                {
        if (i + 1 == ptxt.length())
                                        {
            ptxt += 'X';
        }

                                        else if (ptxt[i] == ptxt[i + 1])
                                        {
            ptxt.insert(i + 1, 1, 'X');
        }
    }

    ctxt = playfair_encrypt(ptxt, arr);

    cout << "Cipher text is: " << ctxt << endl;
    cout << "Decrypted text is: " << playfair_decrypt(ctxt, arr) << endl;

    break;
}

case 4:
                        {
    int kmatrix[3][3] =
                                {
        {6, 24, 1},
        {13, 16, 10},
        {20, 17, 15}
    };

    cout<<"\n-----------------------------------------------"<<endl;
                                cout << "Enter the plaintext (IN uppercase and word only): ";
    cin >> ptxt;

    ctxt = hill_encrypt(ptxt, kmatrix);

    cout << "Ciphertext is: " << ctxt << endl;
```

```
            break;
        }

        case 5:
                        {
            string key;

                                cout<<"\n-----------------------------------------------"<<endl;
                                cout << "Enter the plaintext: ";
            getline(cin, ptxt);

            cout << "Enter the key: ";
            getline(cin, key);

            ctxt = vigenere_encrypt(ptxt, key);

            cout << "Cipher text is: " << ctxt << endl;
            cout << "Decrypted text is: " << vigenere_decrypt(ctxt, key) << endl;

            break;
        }



        case 6:
            cout<<"\n-----------------------------------------------"<<endl;
                                cout << "Exiting the Program . Thank you !" << endl;
                                exit (0);

        default:
            cout<<"\nYou have entered invalid Number.";
      }
    }
    return 0;
}

// Implementations of the function prototypes...
```

```
// Caesar Cipher
string caesar_encrypt(string ptxt, int key)
{
   for (int i = 0; i < ptxt.length(); i++)
         {
      if (isalpha(ptxt[i]))
                  {
         char base = isupper(ptxt[i]) ? 'A' : 'a';
         ptxt[i] = base + (ptxt[i] - base + key) % 26;
      }
   }

   return ptxt;
}

string caesar_decrypt(string ctxt, int key)
{
   for (int i = 0; i < ctxt.length(); i++)
         {
      if (isalpha(ctxt[i]))
                  {
         char base = isupper(ctxt[i]) ? 'A' : 'a';
         ctxt[i] = base + (ctxt[i] - base + 26 - key) % 26;
      }
   }

   return ctxt;
}

// Substitution Cipher
string substitution_encrypt(string ptxt, int key)
{
   for (int i = 0; i < ptxt.length(); i++)
         {
      if (isalpha(ptxt[i]))
                  {
         char base = isupper(ptxt[i]) ? 'A' : 'a';
         ptxt[i] = base + (ptxt[i] - base + key) % 26;
      }
```

```
    }

    return ptxt;
}

void substitution_decrypt(string ctxt, int key)
{
    for (int i = 0; i < ctxt.length(); i++)
            {
        if (isalpha(ctxt[i]))
                    {
            char base = isupper(ctxt[i]) ? 'A' : 'a';
            ctxt[i] = base + (ctxt[i] - base + 26 - key) % 26;
        }
    }

    cout << key + 1 << ") Cipher text decrypted: " << ctxt << endl;
}

// Playfair Cipher
string playfair_encrypt(string ptxt, char arr[5][5])
{
    string ctxt;

    for (int k = 0; k < ptxt.length(); k += 2)
            {
        char char1 = ptxt[k];
        char char2 = ptxt[k + 1];
        int row1 = -1, row2 = -1, col1 = -1, col2 = -1;

        for (int i = 0; i < 5; i++)
                    {
            for (int j = 0; j < 5; j++)
                                {
                if (arr[i][j] == char1)
                                            {
                    row1 = i;
                    col1 = j;
                }
```

```
            if (arr[i][j] == char2)
                                        {
                row2 = i;
                col2 = j;
            }
        }
    }

    if (row1 == row2)
                    {
        ctxt += arr[row1][(col1 + 1) % 5];
        ctxt += arr[row2][(col2 + 1) % 5];
    }

                    else if (col1 == col2)
                    {
        ctxt += arr[(row1 + 1) % 5][col1];
        ctxt += arr[(row2 + 1) % 5][col2];
    }

                    else
                    {
        ctxt += arr[row1][col2];
        ctxt += arr[row2][col1];
    }
  }

  return ctxt;
}

string playfair_decrypt(string ctxt, char arr[5][5])
{
  string ptxt;

  for (int k = 0; k < ctxt.length(); k += 2)
          {
    char char1 = toupper(ctxt[k]);
    char char2 = toupper(ctxt[k + 1]);
```

```
    int row1 = -1, row2 = -1, col1 = -1, col2 = -1;

    for (int i = 0; i < 5; i++)
                {
       for (int j = 0; j < 5; j++)
                        {
          if (arr[i][j] == char1)
                                {
             row1 = i;
             col1 = j;
          }

          if (arr[i][j] == char2)
                                {
             row2 = i;
             col2 = j;
          }
       }
    }

    if (row1 == row2)
                {
       ptxt += arr[row1][(col1 + 4) % 5];
       ptxt += arr[row2][(col2 + 4) % 5];
    }

                else if (col1 == col2)
                {
       ptxt += arr[(row1 + 4) % 5][col1];
       ptxt += arr[(row2 + 4) % 5][col2];
    }

                else
                {
       ptxt += arr[row1][col2];
       ptxt += arr[row2][col1];
    }
 }
```

```
    if (!ptxt.empty() && ptxt.back() == 'X')
        {
      ptxt.pop_back();
    }

    return ptxt;
}

// Hill Cipher
string hill_encrypt(string ptxt, int kmatrix[3][3])
{
    int Ksize = 3;
    int Bsize = 3;
    string ctxt = "";

    if (ptxt.length() % Ksize == 1)
        {
      ptxt += "XX";
    }

        else if (ptxt.length() % Ksize == 2)
        {
      ptxt += "X";
    }

    for (int i = 0; i < ptxt.length(); i += Bsize)
        {
      int portion[Bsize];

      for (int j = 0; j < Bsize; j++)
                {
        portion[j] = ptxt[i + j] - 'A';
      }

      for (int k = 0; k < Ksize; k++)
                {
        int sum = 0;

        for (int l = 0; l < Ksize; l++)
```

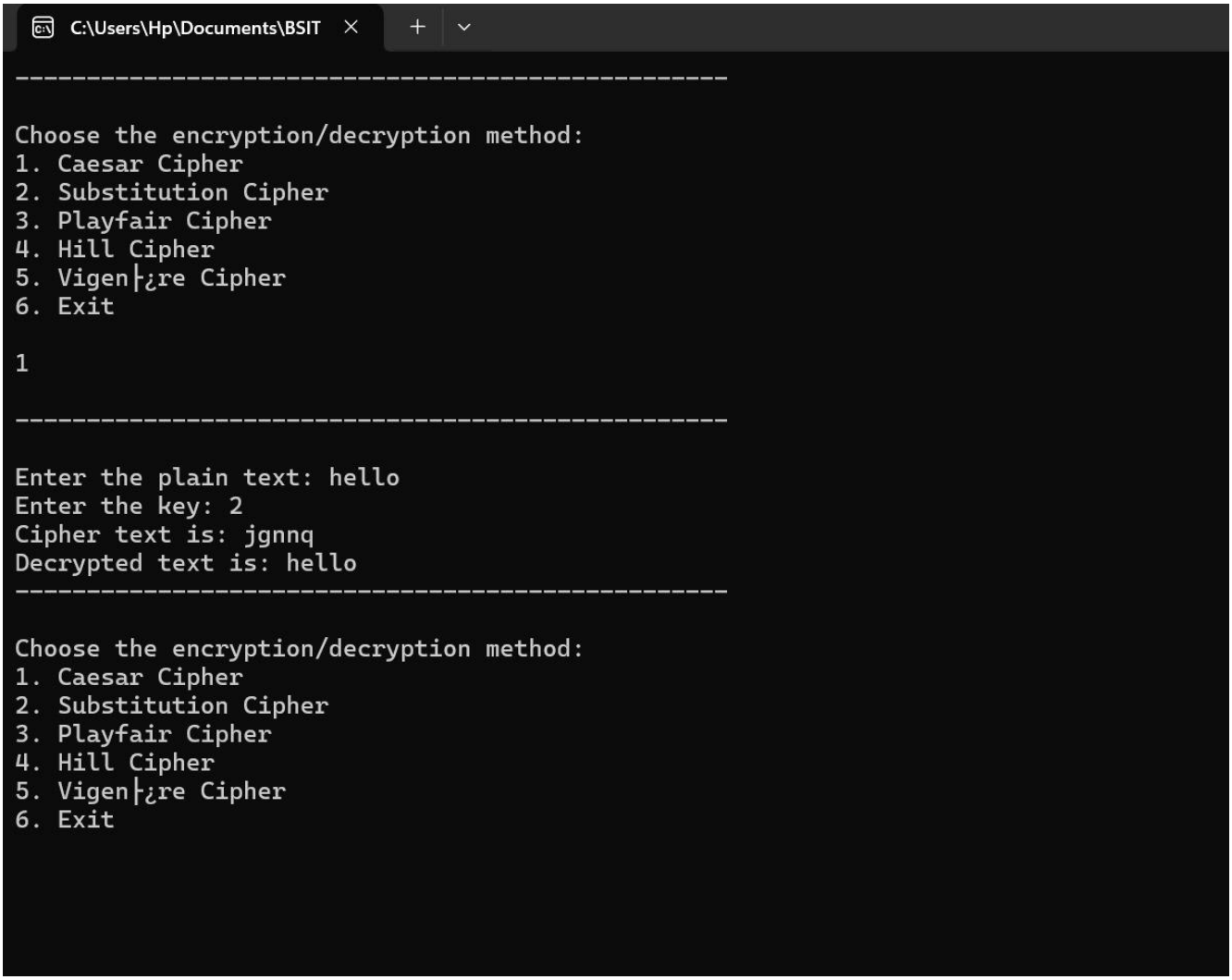
```
                        {
            sum += kmatrix[k][l] * portion[l];
        }

        ctxt += ((sum % 26) + 'A');
    }
  }

  return ctxt;
}

// Vigenère Cipher
string vigenere_encrypt(string ptxt, string key)
{
  string ctxt;
  int keylength = key.length();

  for (int i = 0, j = 0; i < ptxt.length(); i++, j++)
       {
    if (isalpha(ptxt[i]))
              {
        char base = isupper(ptxt[i]) ? 'A' : 'a';
        char encrychar = (ptxt[i] - base + tolower(key[j % keylength]) - 'a') % 26 + base;
        ctxt += encrychar;
    }

              else
              {
        ctxt += ptxt[i];
    }
  }

  return ctxt;
}

string vigenere_decrypt(string ctxt, string key)
{
  string ptxt;
  int keylength = key.length();
```

```
   for (int i = 0, j = 0; i < ctxt.length(); i++, j++)
          {
      if (isalpha(ctxt[i]))
                  {
         char base = isupper(ctxt[i]) ? 'A' : 'a';
         char decrychar = (ctxt[i] - base - (tolower(key[j % keylength]) - 'a') + 26) % 26 + base;
         ptxt += decrychar;
      }

                  else {

         ptxt += ctxt[i];
      }
   }
   return ptxt;
}


void cipher_text(string key, char sortk[], string plaintext)
{
   cout << "\nPlain text matrix will be:" << endl;
   char ptxt[(plaintext.length() + key.length()) / key.length()][key.length()];

   int k = 0;

   for (int i = 0; i < (plaintext.length() + key.length()) / key.length(); i++)
          {
      for (int j = 0; j < key.length(); j++)
                  {
         if (k < plaintext.length())
                        {
            ptxt[i][j] = plaintext[k];
            k++;
         }

                        else
                        {
            ptxt[i][j] = ' ';
```

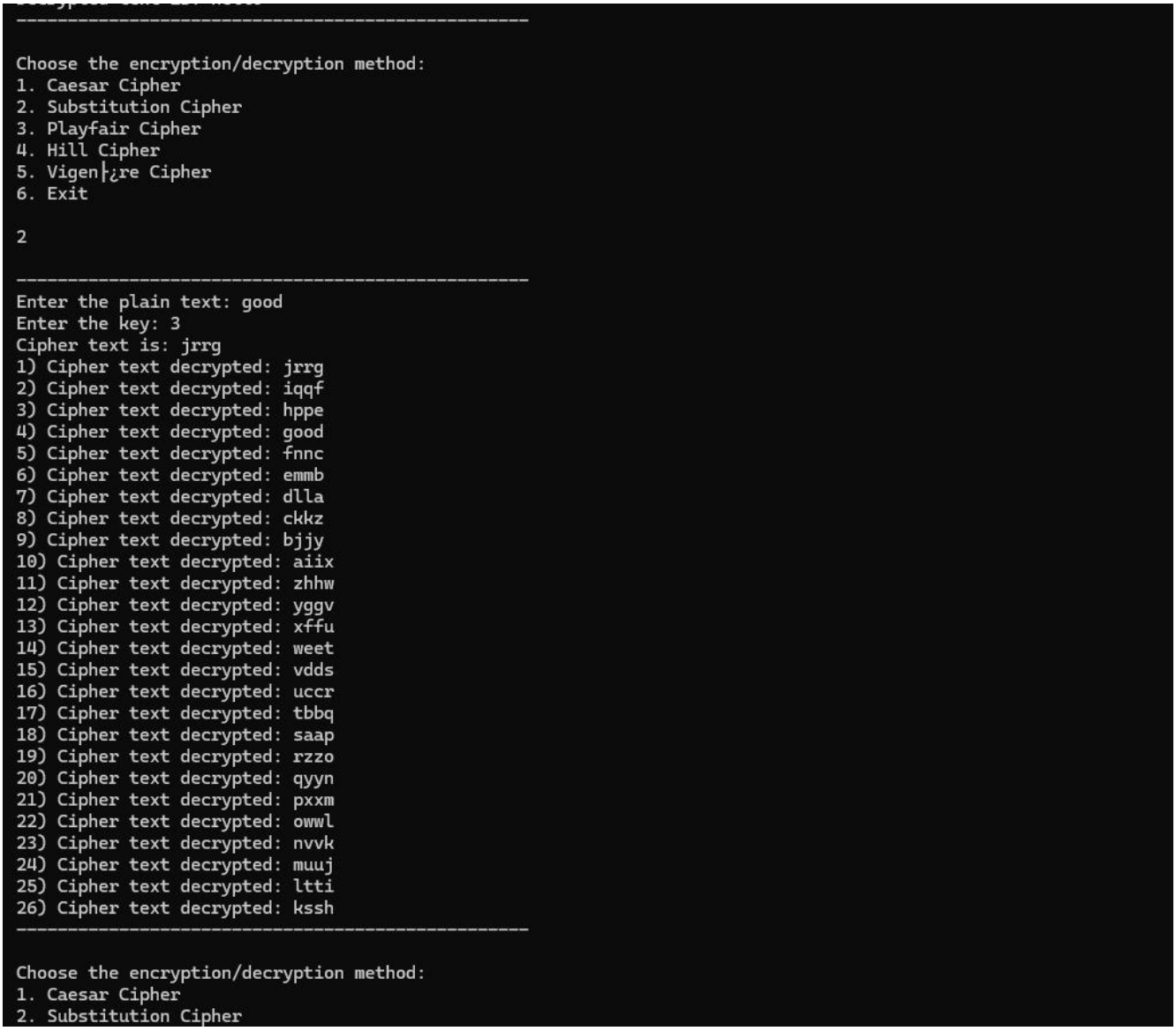
```
        }

        cout << ptxt[i][j] << " ";
      }

      cout << endl;
   }


   cout << endl;
}
```

OUTPUTS

Ceaser Cipher:



Substitution Cipher:



Playfair Cipher:

```
25) Cipher text decrypted: ltti
26) Cipher text decrypted: kssh
--------------------------------------------------

Choose the encryption/decryption method:
1. Caesar Cipher
2. Substitution Cipher
3. Playfair Cipher
4. Hill Cipher
5. Vigen├¿re Cipher
6. Exit

3

--------------------------------------------------
Enter the plain text without spaces: bullet
Cipher text is: YWMWNCSZ
Decrypted text is: BULXLET
--------------------------------------------------

Choose the encryption/decryption method:
```

Hill Cipher:

```
Decrypted text is: BULXLET
--------------------------------------------------

Choose the encryption/decryption method:
1. Caesar Cipher
2. Substitution Cipher
3. Playfair Cipher
4. Hill Cipher
5. Vigen├¿re Cipher
6. Exit

4

--------------------------------------------------
Enter the plaintext (IN uppercase and word only): CAR
Ciphertext is: DOJ
--------------------------------------------------

Choose the encryption/decryption method:
```

Vigenere Cipher:

Exit Program:

```
Decrypted text is: focus
------------------------------------------------

Choose the encryption/decryption method:
1. Caesar Cipher
2. Substitution Cipher
3. Playfair Cipher
4. Hill Cipher
5. Vigen├¿re Cipher
6. Exit

6

------------------------------------------------
Exiting the Program . Thank you !

--------------------------------
Process exited after 265.9 seconds with return value 0
Press any key to continue . . .
```