QUESTION 1: PRODUCT CIPHER

Code:

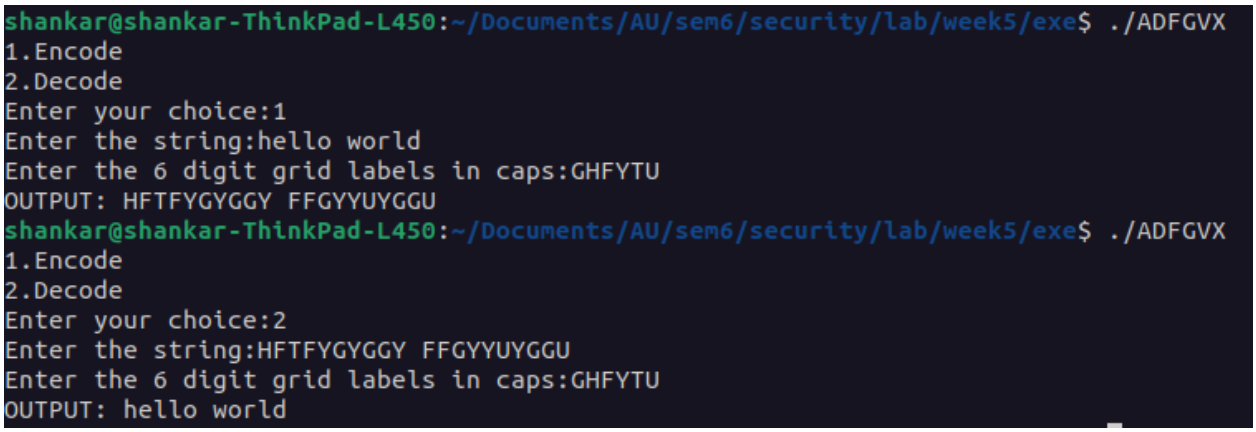
```
#include <string>
#include <iostream>
using namespace std;

class ADFGVX
{
public:

    string grid_val = "ai2o0d1bh6mstnwcq4lg7vyrf5e3x29pjk8u";

    string grid_label;

    string plain_text;
    string cipher_text;
    string input;
    int i, option, length;

    int getInput();
    void encode();
    void decode();
    void display(string);
    string getEncodeChar(char ch);
    string getDecodeChar(char row, char col);
};

int ADFGVX ::getInput()
{

    cout << "1.Encode\n2.Decode\nEnter your choice:";
    cin >> option;
    getchar();

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

    cout << "Enter the 6 digit grid labels in caps:";
```

```
    getline(cin, grid_label);

    if (option == 1)
    {
        plain_text = input;
    }
    else
    {
        cipher_text = input;
    }

    length = input.length();

    return (option);
}

string ADFGVX ::getEncodeChar(char ch)
{
    size_t loc = grid_val.find(ch);

    string temp = "";

    temp += grid_label[loc / 6];
    temp += grid_label[loc % 6];

    return temp;
}

void ADFGVX ::encode()
{
    for (i = 0; i < length; ++i)
    {
        if (plain_text[i] == ' ')
        {
            cipher_text += ' ';
        }
        else
        {
            cipher_text += getEncodeChar(plain_text[i]);
        }
    }
}

string ADFGVX ::getDecodeChar(char row, char col)
```

```
{

    string temp = "";
    size_t row_loc = grid_label.find(row);
    size_t col_loc = grid_label.find(col);

    temp += grid_val[row_loc * 6 + col_loc];

    return temp;
}

void ADFGVX ::decode()
{
    for (i = 0; i <= length; i += 2)
    {
        if (cipher_text[i] == ' ')
        {
            plain_text += ' ';
            i++;
        }

        plain_text += getDecodeChar(cipher_text[i], cipher_text[i + 1]);
    }
}

void ADFGVX ::display(string output)
{
    cout << "OUTPUT: " << output << "\n";
}

int main()
{
    ADFGVX obj;
    int option = obj.getInput();

    if (option == 1)
    {
        obj.encode();
        obj.display(obj.cipher_text);
    }
    else
    {
        obj.decode();
        obj.display(obj.plain_text);
```

```
    }

    return (1);
}
```

Output:



QUESTION 2: RAILFENCE CIPHER

Code:

```
#include <string>
#include <iostream>
#include <cmath>

using namespace std;



class Railfence
{
public:
```

```
    string plain_text;
    string cipher_text;
    string input;

    int i, j, option, length, jump_val[4] = {0,0,0,0};


    int getInput();
    void encode();
    void decode();
    void display(string);
    string getEncodeChar(int index, int shift);
    void getDecodeChar();
};

int Railfence ::getInput()
{

    cout << "1.Encode\n2.Decode\nEnter your choice:";
    cin >> option;
    getchar();

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


    if (option == 1)
    {
        plain_text = input;
    }
    else
    {
        cipher_text = input;
    }

    length = input.length();

    return (option);
}

string Railfence ::getEncodeChar(int index, int shift)
{

    string temp = "";
```

```
    for (j = 0; j < length - index; j += shift)
    {
        temp += plain_text[index + j];
    }

    return (temp);
}

void Railfence ::encode()
{
    cipher_text += getEncodeChar(0,4);
    cipher_text += getEncodeChar(1,2);
    cipher_text += getEncodeChar(2,4);
}

void Railfence ::getDecodeChar()
{

    jump_val[0] += 1;
    jump_val[2] += 1;

    jump_val[1] -= 1;
    jump_val[3] -= 1;
}

void Railfence ::decode()
{

    jump_val[0] = ceil(length/4.0);
    jump_val[1] = ceil( (length-1) / 2.0);
    jump_val[2] = -(jump_val[1]-1);
    jump_val[3] = -(jump_val[0]);

    j = 0;

    for( i = 0; i<length; ++i){
        plain_text += cipher_text[j];

        if( i%4 == 0 && i>0){
            getDecodeChar();
        }

        j += jump_val[i%4];
```

```
    }
}

void Railfence ::display(string output)
{
    cout <<"OUTPUT: "<<output << "\n";
}

int main()
{
    Railfence obj;
    int option = obj.getInput();

    if (option == 1)
    {
        obj.encode();
        obj.display(obj.cipher_text);
    }
    else
    {
        obj.decode();
        obj.display(obj.plain_text);
    }

    return (1);
}
```

Output:

```
shankar@shankar-ThinkPad-L450:~/Documents/AU/sem6/security/lab/week5/exe$ ./railfence
1.Encode
2.Decode
Enter your choice:1
Enter the string:helloworldiamshank
OUTPUT: holmnelwrdasakloih
shankar@shankar-ThinkPad-L450:~/Documents/AU/sem6/security/lab/week5/exe$ ./railfence
1.Encode
2.Decode
Enter your choice:2
Enter the string:holmnelwrdasakloih
OUTPUT: helloworldiamshank
```

QUESTION 3: COLUMNAR CIPHER

Code:

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

class Columnar
{
public:
    string key;

    string plain_text;
    string cipher_text;
    string ans_text;
    string input;
    int i, j, option, length, height, key_length;

    void usage();
    void setMode();
    int getInput();
    bool key_gen();
    string getKey();

    void encode(string plain_text, string key);
    string getEncodeChar(size_t loc);

    void decode();
    string getDecodeChar(int offset);

    void computeDoubleKey();
    void brute_preprocess();
    void display(string, string);
};

void Columnar::usage()
{
    cout << "Usage:\n Input length should be an integer multiple of key length\n The key for brute force is the desired output that is to be matched (i.e. the plaintext)\n\n";
```

```
   cout << " Enter your choice:1\n Enter the string:helloworldiamshank\n Enter the key:432561\n
OUTPUT: wakllhershomldaoin\n\n";
   cout << " Enter your choice:2\n Enter the string:wakllhershomldaoin\n Enter the key:432561\n
OUTPUT: helloworldiamshank\n\n";
   cout << " Enter your choice:3\n Enter the string:ttnaaptmtsuoaodwcoixknlypetz\n Enter the
key:attackpostponeduntiltwoamxyz\n OUTPUT:\n Plain text   = attackpostponeduntiltwoamxyz\n
Desired text = attackpostponeduntiltwoamxyz\n Key is       = 4312567\n Tries taken  =
2399\n\n";
}

string Columnar :: getKey(){
   string inp;
   cout << "\nEnter the "<<ans_text<<":";
   getline(cin, inp);

   return(inp);
}


void Columnar :: setMode(){
   if(option < 3){
      ans_text = "key";
   }
   else{
      ans_text = "ans_text";
   }


}
int Columnar ::getInput()
{
   cout << "1.Encode\n2.Decode\n3.Just brute it\n4.Double Transposition\n Enter your choice:";
   cin >> option;
   getchar();

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

   setMode();
   key = getKey();

   if (option == 1)
   {
      plain_text = input;
```

```
    }
    else
    {
        cipher_text = input;
    }

    length = input.length();
    if (option >= 3)
    {
        ans_text = key;
    }
    else
    {
        key_length = key.length();
        height = length / key_length;
    }

    return (option);
}

string Columnar ::getEncodeChar(size_t loc)
{

    string temp = "";

    for (j = 0; j < length; j += key_length)
    {
        temp += plain_text[loc + j];
    }

    return (temp);
}

void Columnar ::encode(string plain_text, string key)
{
    cipher_text = "";
    char ch = '1';
    for (i = 0; i < key_length; ++i)
    {
        size_t loc = key.find(ch);
        cipher_text += getEncodeChar(loc);

        ch += 1;
    }
```

```
}

string Columnar ::getDecodeChar(int offset)
{

    string temp = "";

    for (i = 0; i < key_length; ++i)
    {
        temp += cipher_text[height * (key[i] - '1') + offset];
    }
    return temp;
}

void Columnar ::decode()
{
   plain_text = "";
    if (option < 3)
    {
        for (j = 0; j < height; ++j)
        {
            plain_text += getDecodeChar(j);
        }
    }
    else
    {
        int count = 0;
        while (key_gen())
        {
            for (j = 0; j < height; ++j)
            {
                plain_text += getDecodeChar(j);
            }
            if (plain_text.compare(ans_text) == 0)
            {
                plain_text  = "\nPlain text   = "  + plain_text;
                plain_text += "\nDesired text = " + ans_text;
                plain_text += "\nKey is       = " + key;
                plain_text += "\nTries taken  = " + to_string(count);

                return;
            }

            count++;
```

```
            plain_text = "";
        }
        plain_text = "Un-matchable as the key falls within no permutations of the input text";
    }
}

bool Columnar ::key_gen()
{
    bool flag = next_permutation(key.begin(), key.end());
    return flag;
}

void Columnar::brute_preprocess()
{
    key = "";
    for (i = 1; i < length / 2; i++)
    {
        if (length % i == 0)
        {
            height = i;
            if (key_length == height)
            {
                height = j;
                key = "";
                for (i = 0; i < key_length; i++)
                    key += (i + '1');
                return;
            }
            key_length = length / i;
            j = height;
        }
    }
    height = j;
    for (i = 0; i < key_length; i++)
        key += (i + '1');
}

void Columnar :: computeDoubleKey(){
    plain_text = key;
    display("cipher text round 1", plain_text);

    encode(plain_text, getKey());
    plain_text = cipher_text.substr(0, key.length());
    display("cipher text round 1", plain_text);
```

```
    encode(plain_text, getKey());
    cipher_text = cipher_text.substr(0, key.length());
    display("cipher text round 2", plain_text);
}

void Columnar ::display(string output_name, string output)
{
    cout << "\n" + output_name + ": " << output << "\n";
}

int main()
{
    Columnar obj;
    int option = obj.getInput();

    if (option == 1)
    {
        obj.encode(obj.plain_text, obj.key);
        obj.display("cipher text", obj.cipher_text);
    }
    else
    {
        if (option >= 3)
        {
            obj.brute_preprocess();
            if(option == 4){
                obj.option = 1;

                obj.input = obj.ans_text;
                obj.setMode();
                obj.computeDoubleKey();

                obj.ans_text = obj.input;
                obj.option = 4;
            }
        }
        obj.decode();
        obj.display("result", obj.plain_text);
    }

    return (1);
}
```

Output:

```
shankar@shankar-ThinkPad-L450:~/Documents/AU/sem6/security/lab/week5/exe$ ./brute
1.Encode
2.Decode
3.Just brute it
4.Double Transposition
 Enter your choice:1
Enter the string:helloworldiamshank

Enter the key:432561

cipher text: wakllhershomldaoin
shankar@shankar-ThinkPad-L450:~/Documents/AU/sem6/security/lab/week5/exe$ ./brute
1.Encode
2.Decode
3.Just brute it
4.Double Transposition
 Enter your choice:2
Enter the string:wakllhershomldaoin

Enter the key:432561

result: helloworldiamshank
```

```
shankar@shankar-ThinkPad-L450:~/Documents/AU/sem6/
1.Encode
2.Decode
3.Just brute it
4.Double Transposition
 Enter your choice:3
Enter the string:ttnaaptmtsuoaodwcoixknlypetz

Enter the ans_text:attackpostponeduntiltwoamxyz

result:
Plain text    = attackpostponeduntiltwoamxyz
Desired text = attackpostponeduntiltwoamxyz
Key is        = 4312567
Tries taken   = 2399
```

```
shankar@shankar-ThinkPad-L450:~/Documents/AU/sem
1.Encode
2.Decode
3.Just brute it
4.Double Transposition
 Enter your choice:4
Enter the string:1234567

Enter the ans_text:1234567

cipher text round 1: 1234567

Enter the key:1762345

cipher text round 1: 1456732

Enter the key:3425671

cipher text round 2: 1456732

result:
Plain text    = 1234567
Desired text = 1234567
Key is        = 3174256
Tries taken   = 1541
```