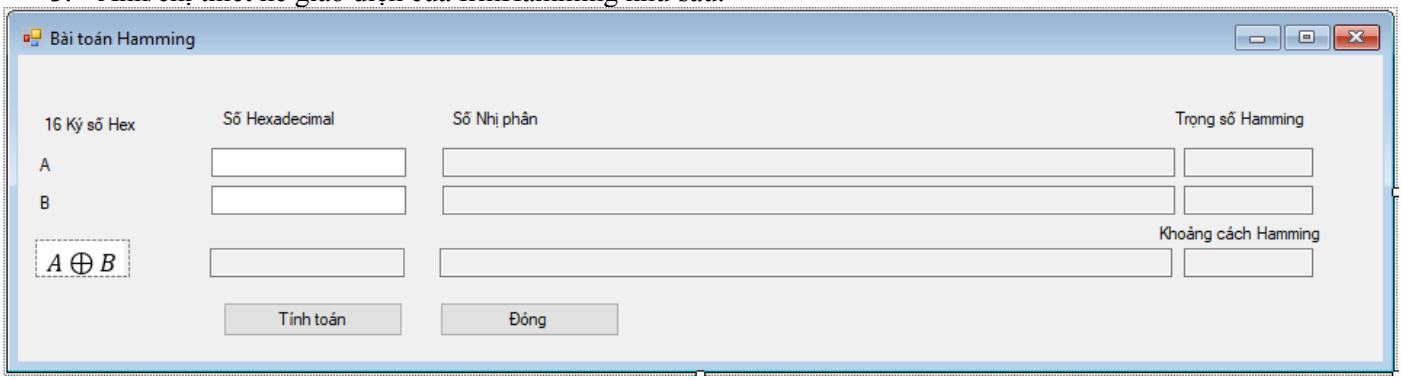
Bài thực hành 2
Khoảng cách Hamming,
các phép Mã hóa và giải mã DES đơn giản

Bài 1: **Khoảng cách Hamming**

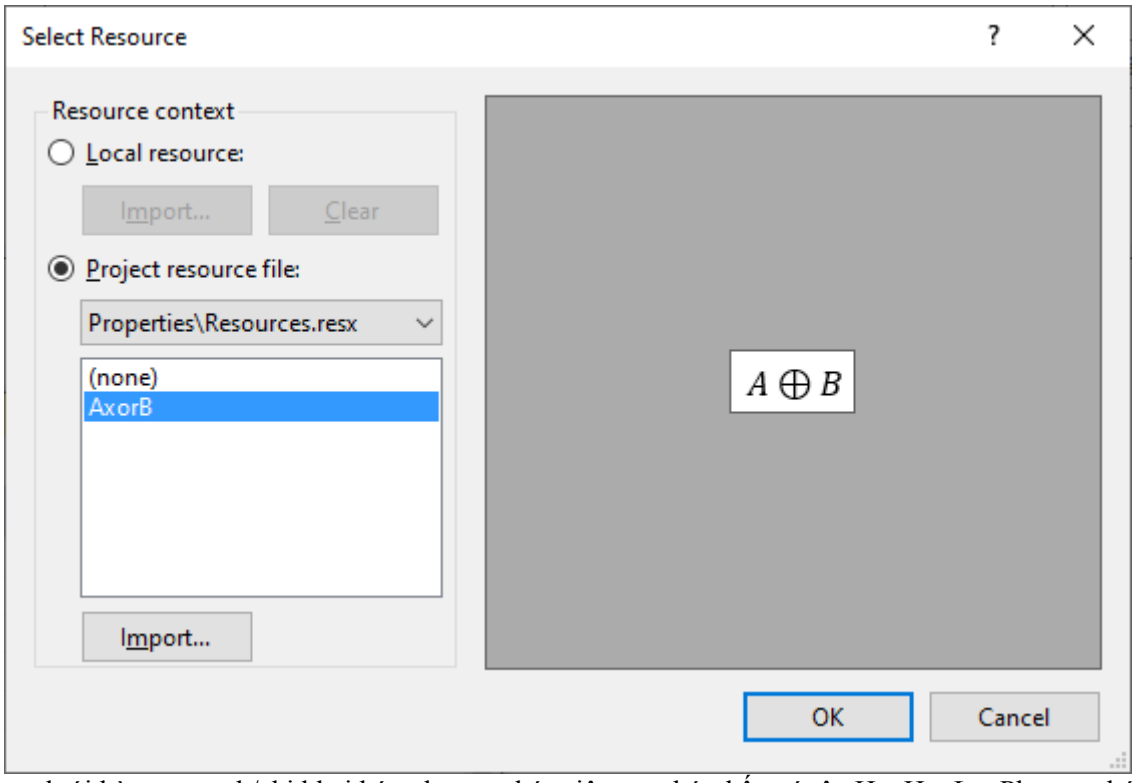
1. Anh/chị tạo một đề án C# mới, có tên Hamming.
2. Anh/chị đổi tên form1 thành frmHamming, và đổi tiêu đề thành Bài toán Hamming.
3. Anh/chị thiết kế giao diện của frmHamming như sau:



STT	Tên đối tượng	Loại đối tượng	Text	Các thuộc tính khác	Vị trí
1	lbl16KySoHex	Label	16 ký số Hex		Dòng đầu tiên
2	lblSoHex	Label	Số Hexadecimal		Bên phải lbl16KySoHex
3	lblSoNhiPhan	Label	Số nhị phân		Bên phải lblSoHex
4	lblTrongSoHamming	Label	Trọng số Hamming		Bên phải lblSoNhiPhan
5	lblSoThuNhat	Label	A		Ngay dưới lbl16KySoHex
6	txtAHexStr	TextBox	(để trống)		Bên phải lblSoThuNhat
7	txtABinStr	TextBox	(để trống)	ReadOnly=True	Bên phải txtAHexStr
8	txtATrongSo	TextBox	(để trống)	ReadOnly=True	Bên phải txtABinStr
9	lblSoThuHai	Label	B		Ngay dưới lblSoThuNhat
10	txtBHexStr	TextBox	(để trống)		Bên phải lblSoThuHai
11	txtBBinStr	TextBox	(để trống)	ReadOnly=True	Bên phải txtBHexStr
12	txtBTrongSo	TextBox	(để trống)	ReadOnly=True	Bên phải txtBBinStr
13	lblKhoangCach	Label	Khoảng cách Hamming		Ngay dưới txtBTrongSo
14	picAxorB	PictureBox		Image=AxorB.bmp	Cách 1 hàng phia dưới lblSoThuHai
15	txtAxorBHexStr	TextBox	(để trống)	ReadOnly=True	Bên phải picAxorB
16	txtAxorBBinStr	TextBox	(để trống)	ReadOnly=True	Bên phải txtAxorBHexStr
17	txtKhoangCach	TextBox	(để trống)	ReadOnly=True	Bên phải txtAxorBinStr và ngay dưới lblKhoangCach
18	btnTinhToan	Button	Tính toán	Enabled=False	Ngay dưới txtAxorHexStr

19	btmDong	Button	Đóng		Bên phải btnTinhToan

Ghi chú: Tập tin AxorB.bmp có sẵn chứa hình ảnh $A \oplus B$ cần được chép vào thư mục con của thư mục đề án. Sau đó, thêm vào Resources của đề án.



4. Ngay dưới hàm tạo, anh/chị khai báo phương thức riêng tư thứ nhất, có tên HexHopLe. Phương thức này có một tham biến hình thức, có tên c, với kiểu char. Phương thức này trả về kiểu luận lý. Giá trị trả về là biểu thức

```
((c >= '0' && c <= '9') || (c > 'A') || (c <= 'F'))
```

5. Anh/chị khai báo phương thức riêng tư thứ hai, có tên HexStrHopLe. Phương thức này có một tham biến hình thức, có tên s, với kiểu chuỗi. Phương thức này trả về kiểu luận lý. Nhiệm vụ của phương thức này là:

```
if (s.Length != 16) return false;
for (int i = 0; i < s.Length; i++)
   if (! HexHopLe(s[i])) return false;
return true;
```

6. Anh/chị khai báo phương thức riêng tư thứ ba, có tên HexToInt. Phương thức này có một tham biến hình thức, có tên c, với kiểu ký tự. Phương thức này trả về kiểu số nguyên. Giá trị trả về của phương thức này là biểu thức

```
(c < 'A') ? c - 48 : c - 55
```

7. Anh/chị khai báo phương thức riêng tư thứ tư, có tên IntToHex. Phương thức này có một tham biến hình thức, có tên n, với kiểu số nguyên. Phương thức này trả về kiểu ký tự. Giá trị trả về của phương thức này là biểu thức

```
(n <= 9) ? (char) (n + 48) : (char) (n + 55)
```

8. Anh/chị khai báo phương thức riêng tư thứ năm, có tên HexToBinStr. Phương thức này có một tham biến hình thức, có tên c, với kiểu tứ tự. Phương thức này trả về kiểu chuỗi. Nhiệm vụ của phương thức này là:

```
int n = HexToInt(c);
string s = "";
for (int i = 0; i < 4; i++)
{
   string d = (n % 2 == 1) ? "1" : "0";
   s = d + s;
   n /= 2;
}
return s;
```

9. Anh/chị khai báo phương thức riêng tư thứ sáu, có tên HexStrToBinStr. Phương thức này có một tham biến hình thức, có tên s, với kuểu chuỗi. Phương thức này trả về kiểu chuỗi. Nhiệm vụ của phương thức này là:

```
string kq = "";
for (int i = 0; i < s.Length; i++)
   kq += HexToBinStr(s[i]);
return kq;
```

10. Anh/chị khai báo phương thức riêng tư thứ bảy, có tên HexXorHex. Phương thức này có hai tham biến hình thức, lần Lượt có tên c1 và c2, đều có kiểu ký tự. Phương thức này trả về kiểu ký tự. Nhiệm vụ của phương thức này là:

```
int n1 = HexToInt(c1);
int n2 = HexToInt(c2);
return IntToHex(n1 ^ n2); // ^ là Bitwise XOR
```

11. Anh/chị khai báo phương thức riêng tư thứ tám, có tên HexStrXorHexStr. Phương thức này có hai tham biến hình thức, lần lượt có tên s1 và s2, đều có kiểu chuỗi. Phương thức này trả về kiểu chuỗi. Nhiệm vụ của phương thức này là:

```
string s = "";
for (int i = 0; i < s1.Length; i++)
   s += HexXorHex(s1[i], s2[i]);
return s;
```

12. Anh/chị khai báo phương thức riêng tư thứ chín, có tên BinStrToTrongSoHamming. Phương thức này có một tham biến hình thức, có tên s, với kiểu chuỗi. Phương thức này trả về kiểu số nguyên. Nhiệm vụ của phương thức này là:

```
// Trọng số của dãy số nhị phân
int count = 0;
for(int i= 0; i<s.Length; i++)
   if (s[i] == '1') count++;
return count;
```

13. Anh/chị khai báo phương thức riêng tư thứ mười, có tên BinStrKhoangCachHamming. Phương thức này có hai tham biến hình thức, lần lượt có tên s1 và s2, đều có kiểu chuỗi. Phương thức này trả về kiểu số nguyên. Nhiệm vụ của phương thức này là:

```
// Khoảng cách Hamming giữa hai dãy nhị phân
int count = 0;
for (int i = 0; i < s1.Length; i++)
   if (s1[i] != s2[i]) count++;
return count;
```

14. Anh/chị nhấp đôi vào txtAHexStr, để có trình xử lý biến cố TextChanged của đối tượng txtAHexStr, có tên txtAHexStr_TextChanged.
15. Anh/chị dùng chuột chỉ vào txtAHexStr_TextChanged, để tìm chỗ xuất hiện thứ hai trong đề án như sau:

```
this.txtAHexStr.TextChanged += new System.EventHandler(this.txtAHexStr_TextChanged);
```

16. Anh/chị sửa lại câu lệnh đặt bẩy này như sau:

```
this.txtAHexStr.TextChanged += new System.EventHandler(this.txtHexStr_TextChanged);
```

17. Anh/chị thêm một câu lệnh đặt bẩy như sau:

```
this.txtBHexStr.TextChanged += new System.EventHandler(this.txtHexStr_TextChanged);
```

18. Anh/chị sửa lại tiêu đề khai báo phương thức dỡ dang như sau:

```
private void txtHexStr_TextChanged(object sender, EventArgs e)
{
}
```

19. Anh/chị đánh nội dung vào thân của phương thức xử lý biến cố này như sau:

```
btnTinhToan.Enabled = HexStrHopLe(txtAHexStr.Text) && HexStrHopLe(txtBHexStr.Text);
```

20. Anh/chị nhấp đôi vào btnTinhToan, để có trình xử lý biến cố Click của btnTinhToan, có tên btnTinhToan_Click. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến hình thức thứ hai, có tên e, với kiểu EventArgs. Nhiệm vụ của phương thức này là:

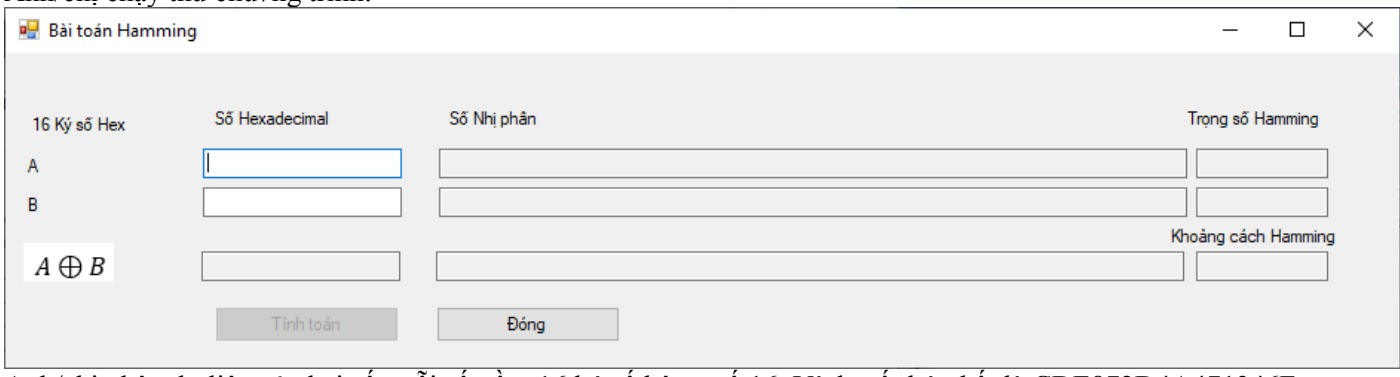
```
txtABinStr.Text = HexStrToBinStr(txtAHexStr.Text);
txtATrongSo.Text = BinStrToTrongSoHamming(txtABinStr.Text).ToString();
```

```
        txtBBinStr.Text = HexStrToBinStr(txtBHexStr.Text);
        txtBTrongSo.Text = BinStrToTrongSoHamming(txtBBinStr.Text).ToString();

        txtAxorBHexStr.Text = HexStrXorHexStr(txtAHexStr.Text, txtBHexStr.Text);
        txtAxorBBinStr.Text = HexStrToBinStr(txtAxorBHexStr.Text);

        int TrongSoAxorB = BinStrToTrongSoHamming(txtAxorBBinStr.Text);
        int KhoangCachHammingAvaB = BinStrKhoangCachHamming(txtABinStr.Text, txtBBinStr.Text);

        txtKhoangCach.Text = KhoangCachHammingAvaB.ToString();
        if (TrongSoAxorB == KhoangCachHammingAvaB)
           MessageBox.Show("Chương trình Hamming tính đúng trọng số và khoảng cách", this.Text);
        else
           MessageBox.Show("Chương trình bị lỗi sai", this.Text);
```

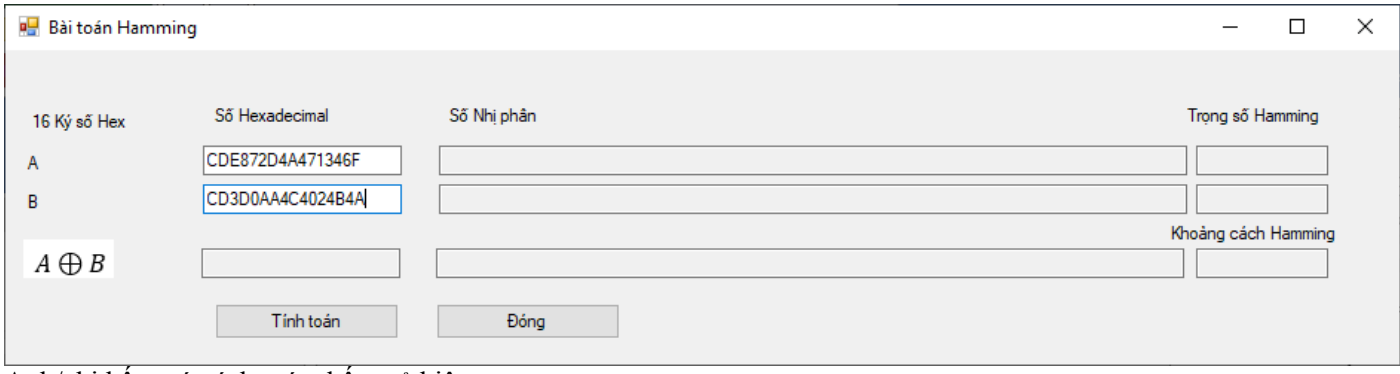
21. Anh/chị nhấp đôi vào btnDong, để có trình xử ltý biến cố Click, của đối tượng btnDong, có tên btnDong_Click. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến hình thức thứ hai, có tên e, với kiểu EventArgs. Nhiệm vụ của phương thức này là:

```
        this.close();
```

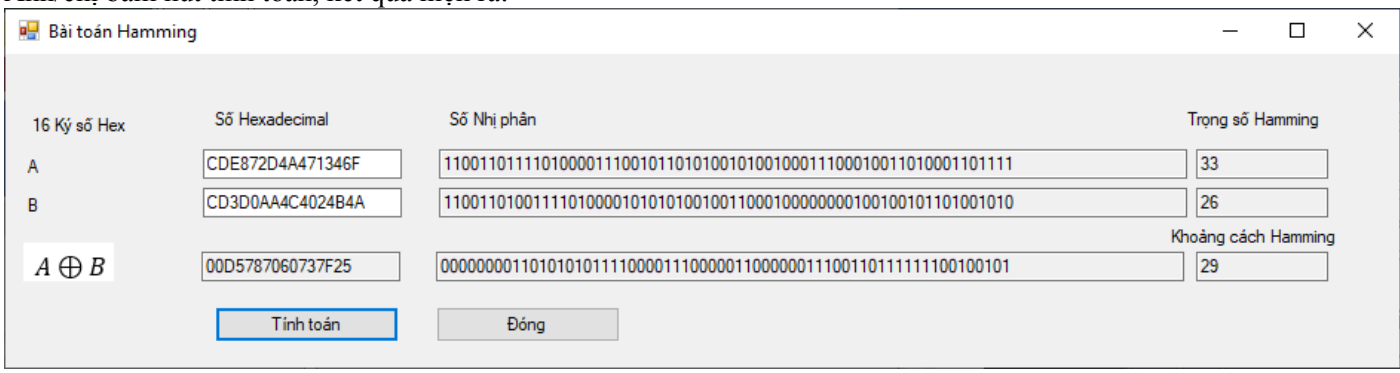
22. Anh/chị chạy thử chu7ng trình:

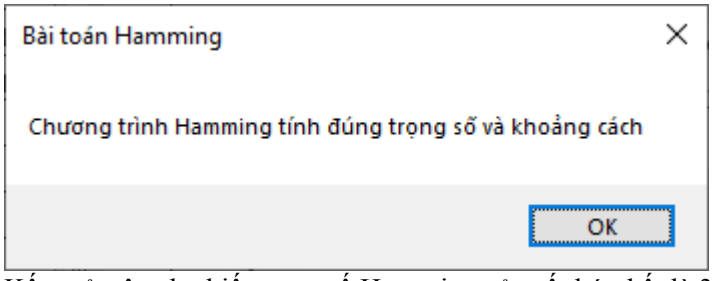


23. Anh/chị nhập dữ liệu của hai số, mỗi số gồm 16 ký số hệ cơ số 16. Ví dụ số thứ nhất là CDE872D4A471346F và số thứ hai là CD3D0AA4C4024B4A.



24. Anh/chị bấm nút tính toán, kết quả hiện ra:





25. Kết quả trên cho biết trọng số Hamming của số thứ nhất là 33, còn trọng số Hamming của số thứ hai là 26. Đồng thời, kết quả phép toán Xor của số thứ nhất và số thứ hai, viết theo hệ 16 là: 00D5787060737F25. Trong số của số này, cũng chính là khoảng cách Hamming của hai số đã cho ban đầu.
26. Nhắc lại:
    $||A \oplus B||$ chính là trọng số Hamming của $A \oplus B$ và cũng là khoảng cách Hamming của A và B, viết là $d(A, B)$

Bài 2: **Trình mã hóa đơn giản V3.141892**

1. Anh/chị tạo một đề án bằng *Visual C#*, có tên *MaHoaDonGian*.
2. Sửa ngay *form1.cs* thành *frmMaHoaGiaiMa.cs*, đồng thời sửa thành phần Text của form này thành "*Trình mã hóa rất đơn giản bởi WannBe V. 3.141592*".
3. Anh/chị tạo thêm ba lớp công cộng mới có tên *MaHoa*, *ProgressEventArgs*, và *CacGiaiThuat*.
4. Anh/chị tạo thêm một form mới có tên *StartUp*.
5. Anh/chị tạo thêm một thư mục mới có tên *GiaiThuat*. Trong thư mục *GiaiThuat* có hai thư mục con
   - Thư mục con thứ nhất có tên *AES*. Trong thư mục con này có bảy lớp, lần lượt có tên:
     - *BaseTransform*
     - *FileIO*
     - *Key*
     - *Matrix*
     - *MultiplicativeInverse*
     - *ProcessAES*
     - *TransformTables*
   - Thư mục con thứ hai có tên *DES*. Trong thư mục con này có ba lớp, lần lượt có tên:
     - *DESData*
     - *FileIO*
     - *ProcessDES*

   Ngoài ra, trong thư mục *GiaiThuat*, còn có lớp công cộng *CommonProcess*.
6. Trong tập tin *ProgressEventArgs.cs* có đến hai lớp công cộng:
   - Lớp công cộng thứ nhất, có tên *ProgressInitArgs*, thừa kế từ *EventArgs*.
     - Khai báo biến: Chỉ có một biến công cộng, có tên *Maximum*, với kiểu *int*.
     - Các hàm tạo: Có một hàm tạo duy nhất một tham biến. Tham biến hình thức có tên *Max*, với kiểu *int*. Nhiệm vụ của hàm tạo này là gán thành phần dữ liệu công cộng *Maximum* theo tham biến hình thức *Max*.
   - Lớp công cộng thứ hai, có tên *ProgressEventArgs*, thừa kế từ *EventArgs*.
     - Khai báo biến: Chỉ có một biến công cộng, có tên *Increment*, với kiểu *int*.
     - Các hàm tạo: Có một hàm tạo duy nhất một tham biến. Tham biến hình thức có tên *Inc*, với kiểu *int*. Nhiệm vụ của hàm tạo này là gán thành phần dữ liệu công cộng *Increment* theo tham biến hình thức *Inc*.
7. Tập tin *DESData.cs* trong thư mục *DES*. Tập tin này có đến hai lớp:
   - Lớp thứ nhất có tên *DESData*.
     - Khai báo biến: Có 16 biến công cộng.
       - Biến công cộng tĩnh thứ nhất, có tên *pc_1*, chỉ đọc, với kiểu *int[]*, và được gán trị đầu
         { 57, 49, 41, 33, 25, 17, 9, 1, 58, 50,
          42, 34, 26, 18, 10, 2, 59, 51, 43, 35,
          27, 19, 11, 3, 60, 52, 44, 36, 63, 55,
          47, 39, 31, 23, 15, 7, 62, 54, 46, 38,
          30, 22, 14, 6, 61, 53, 45, 37, 29, 21,
          13, 5, 28, 20, 12, 4 }; // 56 phần tử
       - Biến công cộng tĩnh thứ hai, có tên *pc_2*, chỉ đọc, với kiểu *int[]*, và được gán trị đầu
         { 14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21,

10, 23, 19, 12, 4, 26, 8, 16, 7, 27,
20, 13, 2, 41, 52, 31, 37, 47, 55, 30,
40, 51, 45, 33, 48, 44, 49, 39, 56, 34,
53, 46, 42, 50, 36, 29, 32 }; // 48 phần tử

- Biến công cộng tĩnh thứ ba, có tên *ip*, chỉ đọc, với kiểu *int[]*, và được gán trị đầu
{ 58, 50, 42, 34, 26, 18, 10, 2, 60, 52,
44, 36, 28, 20, 12, 4, 62, 54, 46, 38,
30, 22, 14, 6, 64, 56, 48, 40, 32, 24,
16, 8, 57, 49, 41, 33, 25, 17, 9, 1, 59,
51, 43, 35, 27, 19, 11, 3, 61, 53, 45,
37, 29, 21, 13, 5, 63, 55, 47, 39, 31,
23, 15, 7 }; // Hoán vị 64 số
- Biến công cộng tĩnh thứ tư, có tên *ip_1*, chỉ đọc, với kiểu *int[]*, và được gán trị đầu
{ 40, 8, 48, 16, 56, 24, 64, 32, 39, 7,
47, 15, 55, 23, 63, 31, 38, 6, 46, 14,
54, 22, 62, 30, 37, 5, 45, 13, 53, 21,
61, 29, 36, 4, 44, 12, 52, 20, 60, 28,
35, 3, 43, 11, 51, 19, 59, 27, 34, 2,
42, 10, 50, 18, 58, 26, 33, 1, 41, 9,
49, 17, 57, 25 }; // Hoán vị ngược 64 số
- Biến công cộng tĩnh thứ năm, có tên *pc_e*, chỉ đọc, với kiểu *int[]*, và được gán trị đầu
{ 32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8,
9, 10, 11, 12, 13, 12, 13, 14, 15, 16,
17, 16, 17, 18, 19, 20, 21, 20, 21, 22,
23, 24, 25, 24, 25, 26, 27, 28, 29, 28,
29, 30, 31, 32, 1 }; // 48 phần tử
- Biến công cộng tĩnh thứ sáu, có tên *pc_p*, chỉ đọc, với kiểu *int[]*, và được gán trị đầu
{ 16, 7, 20, 21, 29, 12, 28, 17, 1, 15, 23,
26, 5, 18, 31, 10, 2, 8, 24, 14, 32,
27, 3, 9, 19, 13, 30, 6, 22, 11, 4, 25 }; // Hoán vị 32 phần tử
- Biến công cộng tĩnh thứ bảy, có tên *nrOfShifts*, chỉ đọc, với kiểu *int[]*, và được gán trị đầu
{ 0, 1, 1, 2, 2, 2, 2, 2, 2, 1,
2, 2, 2, 2, 2, 2, 1 }; // 17 phần tử với tổng số 28=0+1+2+3+4+5+6+7
- Biến công cộng tĩnh thứ tám, có tên *s1*, chỉ đọc, với kiểu *int[,]*, và được gán trị đầu
{ { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },
{ 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
{ 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
{ 15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } };
- Biến công cộng tĩnh thứ chín, có tên *s2*, chỉ đọc, với kiểu *int[,]*, và được gán trị đầu
{ { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },
{ 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
{ 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
{ 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } };
- Biến công cộng tĩnh thứ mười, có tên *s3*, chỉ đọc, với kiểu *int[,]*, và được gán trị đầu
{ { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },
{ 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
{ 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
{ 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } };
- Biến công cộng tĩnh thứ mười một, có tên *s4*, chỉ đọc, với kiểu *int[,]*, và được gán trị đầu
{ { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },
{ 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },
{ 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
{ 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } };
- Biến công cộng tĩnh thứ mười hai, có tên *s5*, chỉ đọc, với kiểu *int[,]*, và được gán trị đầu
{ { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },
{ 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
{ 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
{ 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } };

- Biến công cộng tĩnh thứ mười ba, có tên *s6*, chỉ đọc, với kiểu *int[,]*, và được gán trị đầu
  { { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },
  { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
  { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
  { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } };
- Biến công cộng tĩnh thứ mười bốn, có tên *s7*, chỉ đọc, với kiểu *int[,]*, và được gán trị đầu
  { { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },
  { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
  { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
  { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } };
- Biến công cộng tĩnh thứ mười năm, có tên *s8*, chỉ đọc, với kiểu *int[,]*, và được gán trị đầu
  { { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },
  { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
  { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
  { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };
- Biến công cộng tĩnh thứ mười sáu, có tên *sBoxes*, với kiểu *List<int[,]>*, và được gán trị đầu bằng hàm tạo không tham biến *new List<int[,]>();*

o Các hàm tạo: Chỉ có một hàm tạo tĩnh không tham biến. Nhiệm vụ của hàm tạo này thực hiện 8 lệnh sau đây:

```
sBoxes.Add(s1); // Thêm 4 dòng x 16 cột (hoán vị các số từ 0 đến 15)
sBoxes.Add(s2);
sBoxes.Add(s3);
sBoxes.Add(s4);
sBoxes.Add(s5);
sBoxes.Add(s6);
sBoxes.Add(s7);
sBoxes.Add(s8);
```

- Lớp thứ hai có tên *Keys*.
  - Khai báo biến: Có ba biến công cộng:
    - Biến công cộng thứ nhất, có tên *Cn*, với kiểu *string[]*, và được khởi tạo bằng *new string[17]*.
    - Biến công cộng thứ hai, có tên *Dn*, với kiểu *string[]*, và được khởi tạo bằng *new string[17]*.
    - Biến công cộng thứ ba, có tên *Kn*, với kiểu *string[]*, và được khởi tạo bằng *new string[16]*.

8. Tập tin lớp *FileIO.cs* trong thư mục *DES*.
   - Các phương thức: Có hai phương thức công cộng tĩnh:
     - Phương thức công cộng tĩnh thứ nhất, có tên *FileReadToBinary*. Phương thức này có một tham biến hình thức, có tên *filename*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
FileStream fs = new FileStream("C:\\" + filename, FileMode.Open);

Console.WriteLine("File size : " + fs.Length);

// Read from file 100bytes per 1 time and transform to binary data.

int fileLength = (int)fs.Length;

StringBuilder text = new StringBuilder((int)fs.Length * 8);

byte[] bytes = new byte[100];
int startindex = 0;
int IsEnd = -1;

while (fs.Read(bytes, startindex, bytes.Length) != 0)
{
   if (IsEnd > 0)
   {

   }
```

```
            foreach (byte b in bytes)
            {
                if (text.Length == fileLength * 8)
                {
                    break;
                }

                text.Append(ProcessDES.FromDecimalToBinary(b));
            }
        }

        fs.Close();
        return text.ToString();
```

- Phương thức công cộng tĩnh thứ hai, có tên *WriteBinaryToFile*. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên *filename*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *binaryText*, với kiểu *string*. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
        FileStream fs = new FileStream("C:\\"+ filename, FileMode.Create);
        StringBuilder sub_text = new StringBuilder(800);
        byte[] bytes = new byte[100];
        int length = 800;

        for (int i = 0; i <= binaryText.Length / 800; i++)
        {
            int remain = binaryText.Length - i * 800;
            if (remain < 800)
            {
                length = remain;
            }

            sub_text.Remove(0, sub_text.Length);
            sub_text.Append(binaryText.Substring(i * 800, length));

            for (int j = 0; j < sub_text.Length / 8; j++)
            {
                bytes[j] = ProcessDES.FromBinaryToByte(sub_text.ToString().Substring(j * 8, 8));
            }

            fs.Write(bytes, 0, sub_text.Length / 8);
        }
        fs.Close();
```

9. Tập tin lớp *CommonProcess.cs* trong thư mục GiaiThuat. Lớp này là lớp *trừu tượng*.
   - Các phương thức: Có hai phương thức công cộng *trừu tượng*:
     - Phương thức công cộng *trừu tượng* thứ nhất, có tên *EncryptionStart*. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất, có tên *text*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *key*, với kiểu *string*. Tham biến hình thức thứ ba, có tên *IsTextBinary*, với kiểu luận lý. Phương thức này trả về kiểu *string*.
     - Phương thức công cộng *trừu tượng* thứ hai, có tên *DecryptionStart*. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất, có tên *text*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *key*, với kiểu *string*. Tham biến hình thức thứ ba, có tên *IsTextBinary*, với kiểu luận lý. Phương thức này trả về kiểu *string*.
10. Tập tin lớp *ProcessDES.cs* trong thư mục *DES*. Lớp này được thừa kế từ lớp *CommonProcess*.
    - Các *biến cố*: Có hai biến cố công cộng.
      - *Biến cố* công cộng thứ nhất, có tên *InitProgress*, với kiểu *frmMaHoaGiaiMa.ProgressInitHandler*.
      - *Biến cố* công cộng thứ hai, có tên *IncrementPrgress*, với kiểu *frmMaHoaGiaiMa.ProgressEventHandler*.

- Các hàm tạo: Có một hàm tạo duy nhất hai tham biến. Tham biến hình thứ thứ nhất, có tên *IncProg*, với kiểu *frmMaHoaGiaiMa.ProgressEventArgs*. Tham biến hình thứ thứ hai, có tên *InitProg*, với kiểu *frmMaHoaGiaiMa.ProgressInitHandler*. Nhiệm vụ của hàm tạo này là:

```
this.IncrementProgress = IncProg;
this.InitProgress = InitProg;
```

- Các phương thức: Có 2 phương thức *ảo được bảo vệ*, có 2 phương thức *công cộng tĩnh*, 19 phương thức *công cộng*, và 2 phương thức *công cộng đè lên*.
  - Phương thức ảo được bảo vệ thứ nhất, có tên *OnIncrementProgress*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *e*, với kiểu *ProgressEventArgs*. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
if (IncrementProgress != null)
  IncrementProgress(this, e);
```

  - Phương thức ảo được bảo vệ thứ hai, có tên *OnInitProgress*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên e, với kiểu *ProgressInitArgs*. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
if (InitProgress != null)
  InitProgress(this, e);
```

  - Phương thức công cộng tĩnh thứ nhất, có tên *FromDecimalToBinary*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *binary*, với kiểu int. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
 if (binary < 0)
{
   Console.WriteLine("It requires a integer greater than 0.");
   return null;
}

string binarystring = "";
int factor = 128;

for (int i = 0; i < 8; i++)
{
   if (binary >= factor)
   {
      binary -= factor;
      binarystring += "1";
   }
   else
   {
      binarystring += "0";
   }
   factor /= 2;
}
return binarystring;
```

  - Phương thức công cộng tĩnh thứ hai, có tên *FromBinaryToByte*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *binary*, với kiểu *string*. Phương thức này trả về kiểu *byte*. Nhiệm vụ của phương thức này là:

```
byte value = 0;
int factor = 128;

for (int i = 0; i < 8; i++)
{
   if (binary[i] == '1')
   {
      value += (byte)factor;
   }
```

```
            factor /= 2;
        }
        return value;
```

- Phương thức công cộng thứ nhất, có tên *FromTextToHex*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
        string hexstring = "";

        foreach (char word in text)
        {
            hexstring += String.Format("{0:X}", Convert.ToInt32(word));
        }

        return hexstring;
```

- Phương thức công cộng thứ hai, có tên *FromTextToBinary*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
        StringBuilder binarystring = new StringBuilder(text.Length * 8);

        foreach (char word in text)
        {
            int binary = (int)word;
            int factor = 128;

            for (int i = 0; i < 8; i++)
            {
                if (binary >= factor)
                {
                    binary -= factor;
                    binarystring.Append("1");
                }
                else
                {
                    binarystring.Append("0");
                }
                factor /= 2;
            }
        }
        return binarystring.ToString();
```

- Phương thức công cộng thứ ba, có tên *FromHexToBinary*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *hexstring*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
        string binarystring = "";

        try
        {
            for (int i = 0; i < hexstring.Length; i++)
            {
                int hex = Convert.ToInt32(hexstring[i].ToString(), 16);
                int factor = 8;

                for (int j = 0; j < 4; j++)
                {
                    if (hex >= factor)
                    {
                        hex -= factor;
                        binarystring += "1";
```

```
                }
                else
                {
                    binarystring += "0";
                }
                factor /= 2;
            }
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message + " - wrong hexa integer format.");
    }

    return binarystring;
```

- Phương thức công cộng thứ tư, có tên *DoPermutation*. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất có tên *text* với kiểu *string*. Tham biến hình thức thứ hai có tên *order* với kiểu *int[]*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
    StringBuilder PermutatedText = new StringBuilder(order.Length);
    for (int i = 0; i < order.Length; i++)
    {
        PermutatedText.Append(text[order[i] - 1]);
    }
    return PermutatedText.ToString();
```

- Phương thức công cộng thứ năm, cũng có tên *DoPermutation*. Phương thức này cũng có hai tham biến hình thức. Tham biến hình thứ thứ nhất, có tên *text*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *order*, với kiểu *int[,]*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
    // text chứa chuỗi 6 bit
    // order là mảng 4 dòng 16 cột
    string PermutatedText = "";

    // Bit đầu ghép bit cuối làm số dòng (0-->3)
    int rowIndex = Convert.ToInt32(text[0].ToString() + text[text.Length - 1].ToString(), 2);
    // 4 bit giữa làm số cột (0-->15)
     int colIndex = Convert.ToInt32(text.Substring(1, 4), 2);

     // Từ 6 bit chuyển thành 4 bit
    PermutatedText = ProcessDES.FromDecimalToBinary(order[rowIndex, colIndex]);
    // Lấy phần tử ở mảng order, với giá trị từ 0 đến 15, chuyển sang chuỗi nhị phân

    return PermutatedText;
```

- Phương thức công cộng thứ sáu, có tên *SetHalvesKey*. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên *IsLeft*, với kiểu luận lý. Tham biến hình thức thứ hai, có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
     if ((text.Length % 8) != 0)
    {
        Console.WriteLine("The key is not multiple of 8bit.");
        return null;
    }

    int midindex = (text.Length / 2) - 1;

    string result = "";

    if (IsLeft)
    {
        result = text.Substring(0, midindex + 1);
```

```
        }
        else
        {
            result = text.Substring(midindex + 1);
        }

        return result;
```

- Phương thức công cộng thứ bảy, có tên *SetLeftHalvesKey*. Phương thức này có một tham biến hình thức. Tham biến hình thức, có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
return this.SetHalvesKey(true, text);
```

- Phương thức công cộng thứ tám, có tên *SetRightHalvesKey*. Phương thức này có một tham biến hình thức. Tham biến hình thức, có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
return this.SetHalvesKey(false, text);
```

- Phương thức công cộng thứ chín, có tên *LeftShift*. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên *text*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *count*, với kiểu *int*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
        // Đẩy vòng sang trái
        if (count < 1)
        {
            Console.WriteLine("The count of leftshift is must more than 1 time.");
            return null;
        }
        string temp = text.Substring(0, count);
        StringBuilder shifted = new StringBuilder(text.Length);
        shifted.Append(text.Substring(count) + temp);

        return shifted.ToString();
```

- Phương thức công cộng thứ mười, cũng có tên *LeftShift*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *text*, với kiểu *string*. Phương thức này trả vè kiểu *string*. Nhiệm vụ của phương thức này là:

```
return this.LeftShift(text, 1);
```

- Phương thức công cộng thứ mười một, có tên *SetAllKeys*. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên *C0*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *D0*, với kiểu *string*. Phương thức này trả về kiểu *Keys*. Nhiệm vụ của phương thức này là:

```
        Keys keys = new Keys();
        keys.Cn[0] = C0;
        keys.Dn[0] = D0;

        for (int i = 1; i < keys.Cn.Length; i++)    // Thực hiện 16 vòng với i=1..16
        {
            keys.Cn[i] = this.LeftShift(keys.Cn[i - 1], DESData.nrOfShifts[i]);
            keys.Dn[i] = this.LeftShift(keys.Dn[i - 1], DESData.nrOfShifts[i]);
            keys.Kn[i - 1] = this.DoPermutation(keys.Cn[i] + keys.Dn[i], DESData.pc_2);
        }
        return keys;
```

- Phương thức công cộng thứ mười hai, có tên *setTextMultipleOf64Bits*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
         if ((text.Length % 64) != 0)
        {
            int maxLength = ((text.Length / 64) + 1) * 64;
            text = text.PadRight(maxLength, '0');
        }
```

```
return text;
```

- Phương thức công cộng thứ mười ba, có tên *IsEnough*. Phương thức này có hai tham biến hình thức, Tham biến hình thức thứ nhất, có tên *i*, với kiểu *int*. Tham biến thứ hai, có tên *IsReverse*, với kiểu luận lý. Phương thức này trả về kiểu luận lý. Nhiệm vụ của phương thức này là:

```
return IsReverse ? i >= 0 : i < 16;
```

- Phương thức công cộng thứ mười bốn, có tên *E_Selection*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *Rn_1*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
string ExpandedText = this.DoPermutation(Rn_1, DESData.pc_e);
return ExpandedText;
```

- Phương thức công cộng thứ mười năm, có tên *XOR*. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên *text1*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *text2*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
if (text1.Length != text2.Length)
        {
           Console.WriteLine("Two data blocks for XOR must get same size.");
           return null;
        }

        StringBuilder XORed_Text = new StringBuilder(text1.Length);

        for (int i = 0; i < text1.Length; i++)
        {
           if (text1[i] != text2[i])
           {
              XORed_Text.Append("1");
           }
           else
           {
              XORed_Text.Append("0");
           }
        }

        return XORed_Text.ToString();
```

- Phương thức công cộng thứ mười sáu, có tên *sBox_Transform*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
// Chuyển khóa vòng 48 bit thành kết xuất 32 bit
        StringBuilder TransformedText = new StringBuilder(32);

        for (int i = 0; i < 8; i++)
        {
           string temp = text.Substring(i * 6, 6); // Mỗi lần lấy 6 bit
           // Chuyển thành 4 bit
           TransformedText.Append(this.DoPermutation(temp, DESData.sBoxes[i]));
        }

        return TransformedText.ToString();
```

- Phương thức công cộng thứ mười bảy, có tên *P*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
string PermutatedText = this.DoPermutation(text, DESData.pc_p);
        return PermutatedText;
```

- Phương thức công cộng thứ mười tám, có tên *f*. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên *Rn_1*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *Kn*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
        // Nhận nửa phải 32 bit, và một khóa vòng 48 bit, sinh ra một kết xuất 32 bit
        string E_Rn_1 = this.E_Selection(Rn_1);

        string XOR_Rn_1_Kn = this.XOR(E_Rn_1, Kn);

        string sBoxedText = this.sBox_Transform(XOR_Rn_1_Kn);

        string P_sBoxedText = this.P(sBoxedText);

        return P_sBoxedText;
```

- Phương thức công cộng thứ mười chín, có tên *FinalEncription*. Phương thức này có bốn tham biến hình thức. Tham biến hình thức thứ nhất, có tên *L0*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *R0*, với kiểu *string*. Tham biến hình thức thứ ba, có tên *keys*, với kiểu *Keys*. Tham biến hình thức thứ tư, có tên *InReverse*, với kiểu luận lý. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
        string Ln = "", Rn = "", Ln_1 = L0, Rn_1 = R0;

        int i = 0;

        if (IsReverse)
        {
           i = 15;
        }

        while (this.IsEnough(i, IsReverse))
        {
           Ln = Rn_1;
           Rn = this.XOR(Ln_1, this.f(Rn_1, keys.Kn[i]));

           //Next Step of L1, R1 is L2 = R1, R2 = L1 + f(R1, K2),
          // hence, value of Step1's Ln, Rn is Rn_1, Ln_1 in Step2.
           Ln_1 = Ln;
           Rn_1 = Rn;

           if (IsReverse == false) i++;
           else i--;
        }
        string R16L16 = Rn + Ln;
        string Encripted_Text = this.DoPermutation(R16L16, DESData.ip_1);
        return Encripted_Text;
```

- Phương thức công cộng *đè lên* thứ nhất, có tên *EncryptionStart*. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất, có tên *text*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *key*, với kiểu *string*. Tham biến hình thức thứ ba, có tên *IsTextBinary*, với kiểu luận lý. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
        // Get 16 sub-keys using key

        string hex_key = this.FromTextToHex(key);
        string binary_key = this.FromHexToBinary(hex_key);
        string key_plus = this.DoPermutation(binary_key, DESData.pc_1);

        string C0 = "", D0 = "";

        C0 = this.SetLeftHalvesKey(key_plus);
        D0 = this.SetRightHalvesKey(key_plus);

        Keys keys = this.SetAllKeys(C0, D0);
```

```
// Encrypt process
//string hex_text = this.FromTextToHex(text);
string binaryText = "";

if (IsTextBinary == false)
{
   binaryText = this.FromTextToBinary(text);
}
else
{
   binaryText = text;
}

binaryText = this.setTextMutipleOf64Bits(binaryText);

//Initialize Progress Bar
OnInitProgress(new ProgressInitArgs(binaryText.Length));

StringBuilder EncryptedTextBuilder = new StringBuilder(binaryText.Length);

for (int i = 0; i < (binaryText.Length / 64); i++)
{
   string PermutatedText = this.DoPermutation(binaryText.Substring(i * 64, 64), DESData.ip);

   string L0 = "", R0 = "";

   L0 = this.SetLeftHalvesKey(PermutatedText);
   R0 = this.SetRightHalvesKey(PermutatedText);

   string FinalText = this.FinalEncription(L0, R0, keys, false);

   EncryptedTextBuilder.Append(FinalText);

   // Increase Progress Bar
   OnIncrementProgress(new ProgressEventArgs(FinalText.Length));
}

return EncryptedTextBuilder.ToString();
```

- Phương thức công cộng *đè lên* thứ hai, có tên *DecryptionStart*. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất, có tên *text*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *key*, với kiểu *string*. Tham biến hình thức thứ ba, có tên *IsTextBinary*, với kiểu luận lý. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
// Get 16 sub-keys using key
string hex_key = this.FromTextToHex(key);
string binary_key = this.FromHexToBinary(hex_key);
string key_plus = this.DoPermutation(binary_key, DESData.pc_1);

string C0 = "", D0 = "";

C0 = this.SetLeftHalvesKey(key_plus);
D0 = this.SetRightHalvesKey(key_plus);

Keys keys = this.SetAllKeys(C0, D0);

// Decrypt process
string binaryText = "";
if (IsTextBinary == false)
```

```
        {
            binaryText = this.FromTextToBinary(text);
        }
        else
        {
            binaryText = text;
        }

        binaryText = this.setTextMutipleOf64Bits(binaryText);

        // Initialize Progress Bar
        OnInitProgress(new ProgressInitArgs(binaryText.Length));

        StringBuilder DecryptedTextBuilder = new StringBuilder(binaryText.Length);

        for (int i = 0; i < (binaryText.Length / 64); i++)
        {
            string PermutatedText = this.DoPermutation(binaryText.Substring(i * 64, 64), DESData.ip);

            string L0 = "", R0 = "";

            L0 = this.SetLeftHalvesKey(PermutatedText);
            R0 = this.SetRightHalvesKey(PermutatedText);

            string FinalText = this.FinalEncription(L0, R0, keys, true);

            // It's for correct subtracted '0' that have added for set text multiple of 64bit
            if ((i * 64 + 64) == binaryText.Length)
            {
                StringBuilder last_text = new StringBuilder(FinalText.TrimEnd('0'));

                int count = FinalText.Length - last_text.Length;

                if ((count % 8) != 0)
                {
                    count = 8 - (count % 8);
                }

                string append_text = "";

                for (int k = 0; k < count; k++)
                {
                    append_text += "0";
                }

                DecryptedTextBuilder.Append(last_text.ToString() + append_text);
            }
            else
            {
                DecryptedTextBuilder.Append(FinalText);
            }

            // Increase Progress Bar
            OnIncrementProgress(new ProgressEventArgs(FinalText.Length));
        }

        return DecryptedTextBuilder.ToString();//.TrimEnd('0');
```

11. Tập tin lớp *BaseTransform.cs* trong thư mục *AES*
    - Các phương thức: Có 10 phương thức công cộng tĩnh.
        - Phương thức công cộng tĩnh thứ nhất, có tên *FromTextToHex*. Phương thức này có một tham biến hình thức. Tham biến hình thức có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
StringBuilder hexstring = new StringBuilder(text.Length * 2);
foreach (char word in text)
{
   hexstring.Append(String.Format("{0:X}", Convert.ToInt32(word)));
}
return hexstring.ToString();
```

        - Phương thức công cộng tĩnh thứ hai, có tên *FromHexToText*. Phương thức này có một tham biến hình thức có tên *hexstring*, với kiểu string. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
StringBuilder text = new StringBuilder(hexstring.Length / 2);
for (int i = 0; i < (hexstring.Length / 2); i++)
{
   string word = hexstring.Substring(i * 2, 2);
   text.Append((char)Convert.ToInt32(word, 16));
}
return text.ToString();
```

        - Phương thức công cộng tĩnh thứ ba, có tên *FromBinaryToText*. Phương thức này có một tham biến hình thức có tên *binarystring*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
text = new StringBuilder(binarystring.Length / 8);
for (int i = 0; i < (binarystring.Length / 8); i++)
{
   string word = binarystring.Substring(i * 8, 8);
   text.Append((char)Convert.ToInt32(word, 2));
   //text += (char)Convert.ToInt32(word, 16);
}

return text.ToString();
```

        - Phương thức công cộng tĩnh thứ tư, có tên *setTextMultipleOf64Bits*. Phương thức này có một tham biến hình thức, có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
 int maxLength = 0;

if ((text.Length % 64) != 0)
{
   maxLength = ((text.Length / 64) + 1) * 64;
}

text = text.PadRight(maxLength, '0');

return text
```

        - Phương thức công cộng tĩnh thứ năm, có tên *FromDecimalToBinary*. Phương thức này có một tham biến hình thức, có tên *binary*, với kiểu *int*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
if (binary < 0)
{
   Console.WriteLine("It requires a integer greater than 0.");
   return null;
}
string binarystring = "";
int factor = 128;
```

```
for (int i = 0; i < 8; i++)
{
   if (binary >= factor)
   {
      binary -= factor;
      binarystring += "1";
   }
   else
   {
      binarystring += "0";
   }
   factor /= 2;
}
return binarystring;
```

- Phương thức công cộng tĩnh thứ sáu, có tên *FromBinaryToByte*. Phương thức này có một tham biến hình thức, có tên *binary*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
byte value = 0;
int factor = 128;

for (int i = 0; i < 8; i++)
{
   if (binary[i] == '1')
   {
      value += (byte)factor;
   }

   factor /= 2;
}
return value;
```

- Phương thức công cộng tĩnh thứ bảy, có tên *FromHexToBinary*. Phương thức này có một tham biến hình thức, có tên *hexstring*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
StringBuilder binarystring = new StringBuilder(hexstring.Length * 4);

try
{
   for (int i = 0; i < hexstring.Length; i++)
   {
      int hex = Convert.ToInt32(hexstring[i].ToString(), 16);
      int factor = 8;
      for (int j = 0; j < 4; j++)
      {
         if (hex >= factor)
         {
            hex -= factor;
            binarystring.Append("1");
         }
         else
         {
            binarystring.Append("0");
         }
         factor /= 2;
      }
   }
}
catch (Exception e)
```

```
{
    Console.WriteLine(e.Message + " - wrong hexa integer format.");
}
return binarystring.ToString();
```

- Phương thức công cộng tĩnh thứ tám, có tên *FromBinaryToHex*. Phương thức này có một tham biến hình thức, có tên *binarystring*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
StringBuilder hexstring = new StringBuilder(binarystring.Length / 4);
for (int i = 0; i < binarystring.Length/4; i++)
{
    int word = Convert.ToInt32(binarystring.Substring(i*4, 4), 2);
    hexstring.Append(String.Format("{0:X}", word));
}
return hexstring.ToString();
```

- Phương thức công cộng tĩnh thứ chín, có tên *FromTextToBinary*. Phương thức này có một tham biến hình thức, có tên *text*, với kiểu *string*. Phương thức này trả vè kiểu *string*. Nhiệm vụ của phương thức này là:

```
StringBuilder binarystring = new StringBuilder(text.Length * 8);

foreach (char word in text)
{
    int binary = (int)word;
    int factor = 128;

    for (int i = 0; i < 8; i++)
    {
        if (binary >= factor)
        {
            binary -= factor;
            binarystring.Append("1");
        }
        else
        {
            binarystring.Append("0");
        }
        factor /= 2;
    }
}
return binarystring.ToString();
```

- Phương thức công cộng tĩnh thứ mười, có tên s*etTextMultipleOf128Bits*. Phương thức này có một tham biến hình thức, có tên *text*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
 if ((text.Length % 128) != 0)
{
    int maxLength = ((text.Length / 128) + 1) * 128;
    text = text.PadRight(maxLength, '0');
}
return text;
```

12. Tập tin lớp *FileIO.cs* trong thư mục *AES.*

- Các phương thức: Có hai phương thức công cộng tĩnh.
  - Phương pháp công cộng tĩnh thứ nhất, có tên *FileReadToBinary*. Phương thức này có một tham biến hình thúc, có tên *filename*, với kiểu *string*. Phương thức này trả về kiểu *string*. Nhiệm vụ của phương thức này là:

```
FileStream fs = new FileStream(filename, FileMode.Open);
        Console.WriteLine("File size : " + fs.Length);

        // Read from file 100 bytes per 1 time and transform to binary data.
```

```
int fileLength = (int)fs.Length;

StringBuilder text = new StringBuilder((int)fs.Length * 8);

byte[] bytes = new byte[100];
int startindex = 0;
// int IsEnd = -1;

while (fs.Read(bytes, startindex, bytes.Length) != 0)
{
  // if (IsEnd > 0)
  // {

  // }
   foreach (byte b in bytes)
   {
      if (text.Length == fileLength * 8)
      {
         break;
      }

      text.Append(BaseTransform.FromDecimalToBinary(b));
   }
}
fs.Close();
return text.ToString();
```

- Phương pháp công cộng tĩnh thứ hai, có tên *WriteBinaryToFile*. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên *filename*, với kiểu *string*. Tham biến hình thức thứ hai, có tên *binaryText*, với kiểu *string*. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
 // Write binary encrypted or decrypted data to file.
FileStream fs = new FileStream(filename, FileMode.Create);
StringBuilder sub_text = new StringBuilder(800);
byte[] bytes = new byte[100];
int length = 800;

for (int i = 0; i <= binaryText.Length / 800; i++)
{
   int remain = binaryText.Length - i * 800;
   if (remain < 800)
   {
      length = remain;
   }
   sub_text.Remove(0, sub_text.Length);
   sub_text.Append(binaryText.Substring(i * 800, length));

   for (int j = 0; j < sub_text.Length / 8; j++)
   {
      bytes[j] = BaseTransform.FromBinaryToByte(sub_text.ToString().Substring(j * 8, 8));
      if (remain < 800)
      {
         Console.Write(bytes[j].ToString());
      }
   }
   fs.Write(bytes, 0, sub_text.Length / 8);
}
Console.WriteLine();
```

```
        fs.Close();
```

13. Tập tin *Matrix.cs* trong thư mục *AES*. Tập tin này có hai lớp:
    - Lớp thứ nhất có tên *Matrix*:
        - Khai báo biến: Có ba biến riêng tư.
            - Biến riêng tư thứ nhất, có tên *matrix*, với kiểu *string[,]*.
            - Biến riêng tư thứ hai, có tên *rows*, với kiểu *int*, và giá trị ban đầu 0.
            - Biến riêng tư thứ ba, có tên *columns*, với kiểu *int*, và giá trị ban đầu 0.
        - Các hàm tạo: Có ba hàm tạo.
            - Hàm tạo thứ nhất là hàm tạo ba tham biến. Tham biến thứ nhất có tên *text*, với kiểu *string*. Tham biến thứ hai, có tên *r*, với kiểu *int*. Tham biến thứ ba, có tên *c*, với kiểu *int*. Nhiệm vụ của hàm tạo này là:

```
(text.Length != c * r * 8)
        {
           text = text.PadRight(c * r * 8 - text.Length, '0');
        }

        matrix = new string[r, c];
        int count = 0;
        this.rows = r;
        this.columns = c;

        for (int i = 0; i < c; i++)
        {
           for (int j = 0; j < r; j++)
           {
              matrix[j, i] = text.Substring(count * 8, 8);
              count++;
           }
        }
```

            - Hàm tạo thứ hai là hàm tạo hai tham biến. Tham biến thứ nhất có tên *r*, với kiểu *int*. Tham biến thứ hai, có tên *c*, với kiểu *int*. Hàm tạo này được thừa kế từ *this("",r,c)* và không làm gì thêm.
            - Hàm tạo thứ ba là hàm tạo một tham biến. Tham biến có tên *text*, với kiểu string. Hàm tạo này được thừa kế từ *this(text, 4,4)* và không làm gì thêm.
        - Các đặc trưng và trình chỉ số: Có hai đặc trưng và một trình chỉ số.
            - Đặc trưng thứ nhất, có tên Rows, với kiểu int. Đặc trưng này chỉ đọc và gắn liền với thành phần dữ liệu riêng tư rows.
            - Đặc trưng thứ hai, có tên Columns, với kiểu int. Đặc trưng này chỉ đọc và gắn liền với thành phần dữ liệu riêng tư columns.
            - Trình chỉ số ứng với cách lấy móc vuông hai chỉ số. Tham biến hình thức của chỉ số thứ nhất, có tên *i*, với kiểu *int*. Tham biến hình thức thứ hai của chỉ số thứ hai, có tên *j*, với kiểu *int*. Trình lấy chỉ số này có kiểu string. Nhiệm vụ của đặc trưng này là:

```
        get
        {
           return matrix[i, j];
        }
        set
        {
           //If it gets hexa decimal transform to binary
           if (value.Length == 2)
           {
              matrix[i, j] = BaseTransform.FromHexToBinary(value);
           }
           else if (value.Length == 8)
           {
              matrix[i, j] = value;
           }
        }
```

- Các phương thức: Có 1 phương thức *công cộng đè lên* và 5 phương thức *công cộng*.
  - Phương thức công cộng đè lên có tên ToString. Phương thức này không có tham biến hình thức. Phương thức này trả về kiểu string. Nhiệm vụ của phương thức này là:

```
StringBuilder text = new StringBuilder(128);
if (matrix != null)
{
   for (int j = 0; j < columns; j++)
   {
      for (int i = 0; i < rows; i++)
      {
         text.Append(matrix[i, j]);
      }
   }
}
return text.ToString();
```

  - Phương thức công cộng thứ nhất, có tên SetState. Phương thức này có một tham biến hình thức, có tên text, với kiểu string. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
if (text.Length != columns * rows * 8)
{
   throw new Exception("It's not equal size to state");
}

int count = 0;

for (int i = 0; i < columns; i++)
{
   for (int j = 0; j < rows; j++)
   {
      matrix[j, i] = text.Substring(count * 8, 8);
      count++;
   }
}
```

  - Phương thức công cộng thứ hai, có tên getRow. Phương thức này có một tham biến hình thức, có tên rowindex, với kiểu int. Phương thức này trả về kiểu string[]. Nhiệm vụ của phương thức này là:

```
string[] row = new string[this.columns];

if (rowindex > this.rows)
{
   throw new IndexOutOfRangeException("out of row index error.");
}

for (int i = 0; i < this.columns; i++)
{
   row[i] = matrix[rowindex, i];
}

return row;
```

  - Phương thức công cộng thứ ba, có tên setRow. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất có tên row, với kiểu string[]. Tham biến hình thức thứ hai, có tên rowindex, với kiểu int. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
if (rowindex > this.rows)
{
   throw new IndexOutOfRangeException("out of row index error.");
}

for (int i = 0; i < this.columns; i++)
```

```
        {
            matrix[rowindex, i] = row[i];
        }
```

- Phương thức công cộng thứ tư, có tên getWord. Phương thức này có một tham biến hình thức, có tên wordindex, với kiểu int. Phương thức này trả về kiểu string[]. Nhiệm vụ của phương thức này là:

```
        string[] word = new string[this.rows];
        if (wordindex > this.rows)
        {
            throw new IndexOutOfRangeException("out of column index error.");
        }
        for (int i = 0; i < this.rows; i++)
        {
            word[i] = matrix[i, wordindex];
        }
        return word;
```

- Phương thức công cộng thứ năm, có tên setWord. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên word, với kiểu string[]. Tham biến hình thức thứ hai, có tên wordindex, với kiểu int. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
        if (wordindex > this.rows)
        {
            throw new IndexOutOfRangeException("out of column index error.");
        }

        for (int i = 0; i < this.rows; i++)
        {
            matrix[i, wordindex] = word[i];
        }
```

- Lớp thứ hai có tên MatrixMultiplication
  - Các phương thức: Có 3 phương thức công cộng tĩnh.
    - Phương thức công cộng tĩnh thứ nhất, có tên MixColumnsMultiply. Phương thức này có hai tham biến hình thức, có tên a và b, đều có kiểu Matrix. Phương thức này trả về kiểu Matrix. Nhiệm vụ của phương thức này là:

```
/* If A is an m-by-n matrix and B is an n-by-p matrix, then their matrix product AB is the m-by-p matrix (m rows, p columns) */
        //A - m rows, n columns
        //B - n rows, p columns
        //AB - m rows, p columns
        Matrix c = new Matrix(a.Rows, b.Columns);
        //string temp2 = "";

        for (int j = 0; j < c.Columns; j++)
        {
            //temp.Remove(0, temp.Length);

            for (int i = 0; i < c.Rows; i++)
            {
                StringBuilder temp = new StringBuilder(32);

                temp.Append(AES.MultiplicativeInverse.GetInverse(a[i, 0], b[0, j], "00011011", 8));
                temp.Append(AES.MultiplicativeInverse.GetInverse(a[i, 1], b[1, j], "00011011", 8));
                temp.Append(AES.MultiplicativeInverse.GetInverse(a[i, 2], b[2, j], "00011011", 8));
                temp.Append(AES.MultiplicativeInverse.GetInverse(a[i, 3], b[3, j], "00011011", 8));

                string temp2 = "";
```

```
                temp2 = AES.MultiplicativeInverse.XOR(temp.ToString().Substring(0, 8),
temp.ToString().Substring(8, 8));

                temp2 = AES.MultiplicativeInverse.XOR(temp2, temp.ToString().Substring(16, 8));
                temp2 = AES.MultiplicativeInverse.XOR(temp2, temp.ToString().Substring(24, 8));

                c[i, j] = temp2;
            }
        }
        //Console.WriteLine(c.ToString());
        return c;
```

- Phương thức công cộng tĩnh thứ hai, có tên Multiply. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất và thứ hai, lần lượt có a và b, đều có kiểu Matrix. Tham biến hình thức thứ ba, có tên IsMixColumns, với kiểu luận lý. Phương thức này trả về kiểu Matrix. Nhiệm vụ của phương thức này là:

```
        if (IsMixColumns)
        {
            return MatrixMultiplication.MixColumnsMultiply(a, b);
        }
        else
        {
            return null;
        }
```

- Phương thức công cộng tĩnh thứ ba, có tên XOR. Phương thức này có hai tham biến hình thức, có tên a và b, đều có kiểu Matrix. Phương thức này trả về kiểu Matrix. Nhiệm vụ của phương thức này là:

```
        Matrix c = new Matrix(a.Rows, a.Columns);

        for (int i = 0; i < c.Rows; i++)
        {
            for (int j = 0; j < c.Columns; j++)
            {
                c[i, j] = MultiplicativeInverse.XOR(a[i, j], b[i, j]);
            }
        }

        return c;
```

14. Tập tin lớp Keys.cs trong thư mục AES.
    - Khai báo biến: Có một biến công cộng duy nhất.
        - Biến công cộng có tên RoundKeys, với kiểu List<Matrix>, và được khởi tạo bằng new List<Matrix>(11);
    - Các hàm tạo: Có một hàm tạo không tham biến.
        - Hàm tạo không tham biến: Không làm gì cả.
    - Các phương thức: Có một phương thức công cộng duy nhất.
        - Phương thức công cộng có tên setCipherKey. Phương thức này có một tham biến hình thức, có tên CipherKey, với kiểu Matrix. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
        if (RoundKeys.Count == 0)
        {
            this.RoundKeys.Add(CipherKey);
        }
        else
        {
            RoundKeys.Clear();
            RoundKeys.Add(CipherKey);
        }
```

```
RoundKeys.Add(new Matrix(4, 4));
RoundKeys.Add(new Matrix(4, 4));
RoundKeys.Add(new Matrix(4, 4));
RoundKeys.Add(new Matrix(4, 4));
RoundKeys.Add(new Matrix(4, 4));
RoundKeys.Add(new Matrix(4, 4));
RoundKeys.Add(new Matrix(4, 4));
RoundKeys.Add(new Matrix(4, 4));
RoundKeys.Add(new Matrix(4, 4));
RoundKeys.Add(new Matrix(4, 4));
```

15. Tập tin lớp MultiplicativeInverse.cs trong thư mục AES.
    - Các phương thức: Có 3 phương thức công cộng tĩnh.
        - Phương thức cộng cộng tĩnh thứ nhất, có tên LeftShift2. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên text, với kiểu string. Tham biến hình thức thứ hai, có tên level, với kiểu int. Phương thức này trả về kiểu string. Nhiệm vụ của phương thức này là:

```
// Non-circular left shift
//string temp = text.Substring(0, count);
StringBuilder shifted = new StringBuilder(text.Length);
shifted.Append(text.Substring(1) + "0");

if (!level.Equals(8))
{
   for (int i = 0; i <= text.Length - (1 + level); i++)
   {
      shifted[i] = '0';
   }
}
return shifted.ToString();
```

        - Phương thức cộng cộng tĩnh thứ hai, có tên GetInverse. Phương thức này có bốn tham biến hình thức. Tham biến hình thức thứ nhất và thứ hai, lần lượt có tên text1 và text2, đều có kiểu string. Tham biến hình thức thứ ba, có tên mx, với kiểu int. Tham biến hình thức thứ tư, có tên n, với kiểu int. Phương thức này trả về kiểu string. Nhiệm vụ của phương thức này là:

```
string[] multiplyTable = new string[n];

if (text1.IndexOf('1') > text2.IndexOf('1'))
{
   string temp = text2;
   text2 = text1;
   text1 = temp;
}
multiplyTable[0] = text1;
for (int i = 1; i < n; i++)
{
   multiplyTable[i] = MultiplicativeInverse.LeftShift2(multiplyTable[i - 1], n);

   if (multiplyTable[i - 1][text1.Length - n].Equals('1'))
   {
      multiplyTable[i] = MultiplicativeInverse.XOR(multiplyTable[i], mx);
   }
}
string Mul_Inverse = "";
for (int i = 0; i < text2.Length; i++)
{
   if (text2[i].Equals('1'))
   {
      if (Mul_Inverse.Equals(""))
      {
```

```
                Mul_Inverse = multiplyTable[(text2.Length - 1) - i];
            }
            else
            {
                Mul_Inverse = MultiplicativeInverse.XOR(Mul_Inverse,
                                                        multiplyTable[(text2.Length - 1) - i]);
            }
        }
    }
    if (Mul_Inverse.Equals(""))
    {
        Mul_Inverse = "00000000";
    }
    return Mul_Inverse;
```

- Phương thức cộng cộng tĩnh thứ ba, có tên XOR. Phương thức này có hai tham biến hình thức, có tên text1 và text2, đều có kiểu string. Phương thức này trả về kiểu string. Nhiệm vụ của phương thức này là:

```
// XOR Operation
    if (text1.Length != text2.Length)
    {
        int count = Math.Abs(text1.Length - text2.Length);
        string temp = "";

        for (int i = 0; i < count; i++)
        {
            temp += "0";
        }

        if (text1.Length > text2.Length)
        {
            text2 = temp + text2;
        }
        else
        {
            text1 = temp + text1;
        }
    }
    StringBuilder XORed_Text = new StringBuilder(text1.Length);
    //string XORed_Text = "";

    for (int i = 0; i < text1.Length; i++)
    {
        if (text1[i] != text2[i])
        {
            XORed_Text.Append("1");
        }
        else
        {
            XORed_Text.Append("0");
            //XORed_Text += "0";
        }
    }
    return XORed_Text.ToString();
```

16. Tập tin lớp ProcessAES.cs trong thư mục AES. Lớp này được thùa kế từ CommonProcess.
    - Các biến cố: Có hai biến cố công cộng
        - Biến cố thứ nhất, có tên IncrementProgress, với kiểu frmMaHoaGiaiMa.ProgressEventArgs.
        - Biến cố thứ hai, có tên InitProgress, với kiểu frmMaHoaGiaiMa.ProgressInitHandler.

- Các hàm tạo: Có một hàm tạo duy nhất hai tham biến.
  - Tham biến hình thức thứ nhất có tên IncProg, với kiểu frmMaHoaGiaiMa.ProgressEventArgs. Tham biến hình thức thứ hai, có tên InitProg, với kiểu frmMaHoaGiaiMa.ProgressInitHandler. Nhiệm vụ của hàm tạo này là:

```
this.IncrementProgress = IncProg;
this.InitProgress = InitProg;
```

- Các phương thức: Có 2 phương thức ảo được bảo vệ, có 3 phương thức riêng tư, có 5 phương thức công cộng, và 2 phương thức công cộng đè lên.
  - Phương thức ảo được bảo vệ thứ nhất, có tên OnIncrementProgress. Phương thức này có một tham biến hình thức, có tên e, với kiểu ProgressEventArgs. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
if (IncrementProgress != null)
        IncrementProgress(this, e);
```

  - Phương thức ảo được bảo vệ thứ hai, có tên OnInitProgress. Phương thức này có một tham biến hình thức, có tên e, với kiểu ProgressInitArgs. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
if (InitProgress != null)
        InitProgress(this, e);
```

  - Phương thức riêng tư thứ nhất, có tên CircularLeftShift. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên row, với kiểu string[]. Tham biến hình thức thứ hai, có tên count, với kiểu int. Phương thức này trả về kiểu string[]. Nhiệm vụ của phương thức này là:

```
for (int i = 0; i < count; i++)
{
   string temp = row[0];
   row[0] = row[1];
   row[1] = row[2];
   row[2] = row[3];
   row[3] = temp;
}
return row;
```

  - Phương thức riêng tư thứ hai, có tên CircularRightShift. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên row, với kiểu string[]. Tham biến hình thức thứ hai, có tên count, với kiểu int. Phương thức này trả về kiểu string[]. Nhiệm vụ của phương thức này là:

```
for (int i = 0; i < count; i++)
{
   string temp = row[3];
   row[3] = row[2];
   row[2] = row[1];
   row[1] = row[0];
   row[0] = temp;
}
return row;
```

  - Phương thức riêng tư thứ ba, có tên RotWord. Phương thức này có một tham biến hình thức, có tên state, với kiểu string[]. Phương thức này trả về kiểu string[]. Nhiệm vụ của phương thức này là:

```
return this.CircularLeftShift(state, 1);
```

  - Phương thức công cộng thứ nhất, có tên KeyExpansion. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên key, với kiểu Keys. Tham biến hình thức thứ hai, có tên IsReverse, với kiểu luận lý. Phương thức này trả về kiểu Keys. Nhiệm vụ của phương thức này là:

```
for (int i = 4; i < key.RoundKeys.Count * 4; i++)
{
   string[] Wi_1 = key.RoundKeys[(i - 1) / 4].getWord((i - 1) % 4);

   Matrix mat_Wi_1 = new Matrix(4, 1);
   mat_Wi_1.setWord(Wi_1, 0);

   if (i % 4 == 0)
```

```
            {
                Wi_1 = this.RotWord(Wi_1);
                mat_Wi_1.setWord(Wi_1, 0);
                mat_Wi_1 = this.SubBytes(mat_Wi_1, false);
                mat_Wi_1 = MatrixMultiplication.XOR(mat_Wi_1, TransformTables.Rcon[(i - 1) / 4]);
            }

            Matrix Wi_4 = new Matrix(4, 1);
            Wi_4.setWord(key.RoundKeys[(i - 4) / 4].getWord((i - 4) % 4), 0);

            Matrix temp = MatrixMultiplication.XOR(mat_Wi_1, Wi_4);

            string[] Wi = temp.getWord(0);

            key.RoundKeys[i / 4].setWord(Wi, i % 4);
        }

        return key;
```

- Phương thức công cộng thứ hai, có tên AddRoundKey. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất có tên state, với kiểu Matrix. Tham biến hình thức thứ hai, có tên key, với kiểu Keys. Tham biến hình thức thứ ba, có tên Round, với kiểu int. Phương thức này trả về kiểu Matrix. Nhiệm vụ của phương thức này là:

```
        // AddRoundKey
        if (Round > key.RoundKeys.Count - 1)
        {
            throw new IndexOutOfRangeException(
                                "The round key is must between 0 and 10 in 128 bit AES.");
        }
        return MatrixMultiplication.XOR(state, key.RoundKeys[Round]);
```

- Phương thức công cộng thứ ba, có tên SubBytes. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất có tên state, với kiểu Matrix. Tham biến hình thức thứ hai, có tên IsReverse, với kiểu luận lý. Phương thức này trả về kiểu Matrix. Nhiệm vụ của phương thức này là:

```
        // SubBytes
        for (int i = 0; i < state.Rows; i++)
        {
            for (int j = 0; j < state.Columns; j++)
            {
                int row = Convert.ToInt32(state[i, j].Substring(0, 4), 2);
                int column = Convert.ToInt32(state[i, j].Substring(4, 4), 2);

                if (IsReverse == false)
                {
                    state[i, j] = TransformTables.sbox[row, column];
                }
                else
                {
                    state[i, j] = TransformTables.inverse_sbox[row, column];
                }
            }
        }

        return state;
```

- Phương thức công cộng thứ tư, có tên ShiftRows. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên state, với kiểu Matrix. Tham biến hình thức thứ hai, có tên IsReverse, với kiểu luận lý. Phương thức này trả về kiểu Matrix. Nhiệm vụ của phương thức này là:

```
        // ShiftRows
        for (int i = 1; i < state.Rows; i++)
```

```
        {
            if (IsReverse == false)
            {
                state.setRow(this.CircularLeftShift(state.getRow(i), i), i);
            }
            else
            {
                state.setRow(this.CircularRightShift(state.getRow(i), i), i);
            }
        }
        return state;
```

- Phương thức công cộng thứ năm, có tên MixColumns. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên state, với kiểu Matrix. Tham biến hình thức thứ hai, có tên IsReverse, với kiểu luận lý. Phương thức này trả về kiểu Matrix. Nhiệm vụ của phương thức này là:

```
// MixColumns
        if (IsReverse == false)
        {
            state = MatrixMultiplication.Multiply(TransformTables.MixColumnFactor, state, true);
        }
        else
        {
            state = MatrixMultiplication.Multiply(TransformTables.Inverse_MixColumnFactor, state,
                                                  true);
        }
        return state;
```

- Phương thức công cộng đè lên thứ nhất, có tên EncryptionStart. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất, có tên PlainText, với kiểu string. Tham biến hình thức thứ hai, có tên CipherKey, với kiểu string. Tham biến hình thức thứ ba, có tên IsTextBinary, với kiểu luận lý. Phương thức này trả về kiểu string. Nhiệm vụ của phương thức này là:

```
        // Encryption Process
        StringBuilder binaryText = null;
        if (IsTextBinary == false)
        {
            PlainText = BaseTransform.FromTextToBinary(PlainText);
        }
        else
        {
            //binaryText = PlainText;
        }
        binaryText = new StringBuilder(BaseTransform.setTextMutipleOf128Bits(PlainText));

        StringBuilder EncryptedTextBuilder = new StringBuilder(binaryText.Length);

        // Make All-round keys
        Matrix Matrix_CipherKey = new Matrix(BaseTransform.FromHexToBinary(CipherKey));
        Keys key = new Keys();
        key.setCipherKey(Matrix_CipherKey);
        key = this.KeyExpansion(key, false);

        // Initialize Progress Bar
        OnInitProgress(new ProgressInitArgs(binaryText.Length));

        //Matrix state = new Matrix(4, 4);

        for (int j = 0; j < (binaryText.Length / 128); j++)
        {
            //state.setState(binaryText.ToString().Substring(j * 128, 128));
```

```
            Matrix state = new Matrix(binaryText.ToString().Substring(j * 128, 128));
            state = this.AddRoundKey(state, key, 0);

            for (int i = 1; i < 11; i++)
            {
                if (i == 10)
                {
                    state = this.SubBytes(state, false);
                    state = this.ShiftRows(state, false);
                    state = this.AddRoundKey(state, key, i);
                }
                else
                {
                    state = this.SubBytes(state, false);
                    state = this.ShiftRows(state, false);
                    state = this.MixColumns(state, false);
                    state = this.AddRoundKey(state, key, i);
                }
            }

            EncryptedTextBuilder.Append(state.ToString());

            // Increase Progress Bar
            OnIncrementProgress(new ProgressEventArgs(state.ToString().Length));
        }
        return EncryptedTextBuilder.ToString();
```

- Phương thức công cộng đè lên thứ hai, có tên DecryptionStart. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất, có tên PlainText, với kiểu string. Tham biến hình thức thứ hai, có tên CipherKey, với kiểu string. Tham biến hình thức thứ ba, có tên IsTextBinary, với kiểu luận lý. Phương thức này trả về kiểu string. Nhiệm vụ của phương thức này là:

```
        // Decryption Process
        string binaryText = "";

        if (IsTextBinary == false)
        {
            binaryText = BaseTransform.FromTextToBinary(PlainText);
        }
        else
        {
            binaryText = PlainText;
        }

        StringBuilder DecryptedTextBuilder = new StringBuilder(binaryText.Length);

        // Make All-round keys
        Matrix Matrix_CipherKey = new Matrix(BaseTransform.FromHexToBinary(CipherKey));
        Keys key = new Keys();
        key.setCipherKey(Matrix_CipherKey);
        key = this.KeyExpansion(key, false);

        // Initialize Progress Bar
        OnInitProgress(new ProgressInitArgs(binaryText.Length));

        //Matrix state = new Matrix(4, 4);

        for (int j = 0; j < (binaryText.Length / 128); j++)
```

```
        {
            //state.setState(binaryText.Substring(j * 128, 128));
            Matrix state = new Matrix(binaryText.Substring(j * 128, 128));
            state = this.AddRoundKey(state, key, 10);
            for (int i = 9; i >= 0; i--)
            {
                if (i == 0)
                {
                    state = this.ShiftRows(state, true);
                    state = this.SubBytes(state, true);
                    state = this.AddRoundKey(state, key, i);
                }
                else
                {
                    state = this.ShiftRows(state, true);
                    state = this.SubBytes(state, true);
                    state = this.AddRoundKey(state, key, i);
                    state = this.MixColumns(state, true);

                    //DecryptedTextBuilder.Append(state.ToString());
                }
            }
            // It's for correct subtracted '0' that have added for set text multiple of 128bit
            if ((j * 128 + 128) == binaryText.Length)
            {
                StringBuilder last_text = new StringBuilder(state.ToString().TrimEnd('0'));
                int count = state.ToString().Length - last_text.Length;
                if ((count % 8) != 0)
                {
                    count = 8 - (count % 8);
                }
                string append_text = "";
                for (int k = 0; k < count; k++)
                {
                    append_text += "0";
                }
                DecryptedTextBuilder.Append(last_text.ToString() + append_text);
            }
            else
            {
                DecryptedTextBuilder.Append(state.ToString());
            }
            // Increase Progress Bar
            OnIncrementProgress(new ProgressEventArgs(state.ToString().Length));
        }
        //return DecryptedTextBuilder.ToString().TrimEnd('0');
        return DecryptedTextBuilder.ToString();
```

17. Tập tin lớp TransformTables.cs trong thư mục AES.
    - Khai báo biến: Có 4 biến công cộng tĩnh chỉ đọc và 1 biến công cộng tĩnh.
        - Biến công cộng tĩnh chỉ đọc thứ nhất, có tên sbox, với kiểu string[,], và được gán trị đầu:

```
new string[,] {
        {"63","7c","77","7b","f2","6b","6f","c5","30","01","67","2b","fe","d7","ab","76"},
        {"ca","82","c9","7d","fa","59","47","f0","ad","d4","a2","af","9c","a4","72","c0"},
        {"b7","fd","93","26","36","3f","f7","cc","34","a5","e5","f1","71","d8","31","15"},
        {"04","c7","23","c3","18","96","05","9a","07","12","80","e2","eb","27","b2","75"},
        {"09","83","2c","1a","1b","6e","5a","a0","52","3b","d6","b3","29","e3","2f","84"},
        {"53","d1","00","ed","20","fc","b1","5b","6a","cb","be","39","4a","4c","58","cf"},
```

```
        {"d0","ef","aa","fb","43","4d","33","85","45","f9","02","7f","50","3c","9f","a8"},
        {"51","a3","40","8f","92","9d","38","f5","bc","b6","da","21","10","ff","f3","d2"},
        {"cd","0c","13","ec","5f","97","44","17","c4","a7","7e","3d","64","5d","19","73"},
        {"60","81","4f","dc","22","2a","90","88","46","ee","b8","14","de","5e","0b","db"},
        {"e0","32","3a","0a","49","06","24","5c","c2","d3","ac","62","91","95","e4","79"},
        {"e7","c8","37","6d","8d","d5","4e","a9","6c","56","f4","ea","65","7a","ae","08"},
        {"ba","78","25","2e","1c","a6","b4","c6","e8","dd","74","1f","4b","bd","8b","8a"},
        {"70","3e","b5","66","48","03","f6","0e","61","35","57","b9","86","c1","1d","9e"},
        {"e1","f8","98","11","69","d9","8e","94","9b","1e","87","e9","ce","55","28","df"},
        {"8c","a1","89","0d","bf","e6","42","68","41","99","2d","0f","b0","54","bb","16"}};
```

- Biến công cộng tĩnh chỉ đọc thứ hai, có tên inverse_sbox, với kiểu string[,], và được gán trị đầu:

```
new string[,] {
        {"52","09","6a","d5","30","36","a5","38","bf","40","a3","9e","81","f3","d7","fb"},
        {"7c","e3","39","82","9b","2f","ff","87","34","8e","43","44","c4","de","e9","cb"},
        {"54","7b","94","32","a6","c2","23","3d","ee","4c","95","0b","42","fa","c3","4e"},
        {"08","2e","a1","66","28","d9","24","b2","76","5b","a2","49","6d","8b","d1","25"},
        {"72","f8","f6","64","86","68","98","16","d4","a4","5c","cc","5d","65","b6","92"},
        {"6c","70","48","50","fd","ed","b9","da","5e","15","46","57","a7","8d","9d","84"},
        {"90","d8","ab","00","8c","bc","d3","0a","f7","e4","58","05","b8","b3","45","06"},
        {"d0","2c","1e","8f","ca","3f","0f","02","c1","af","bd","03","01","13","8a","6b"},
        {"3a","91","11","41","4f","67","dc","ea","97","f2","cf","ce","f0","b4","e6","73"},
        {"96","ac","74","22","e7","ad","35","85","e2","f9","37","e8","1c","75","df","6e"},
        {"47","f1","1a","71","1d","29","c5","89","6f","b7","62","0e","aa","18","be","1b"},
        {"fc","56","3e","4b","c6","d2","79","20","9a","db","c0","fe","78","cd","5a","f4"},
        {"1f","dd","a8","33","88","07","c7","31","b1","12","10","59","27","80","ec","5f"},
        {"60","51","7f","a9","19","b5","4a","0d","2d","e5","7a","9f","93","c9","9c","ef"},
        {"a0","e0","3b","4d","ae","2a","f5","b0","c8","eb","bb","3c","83","53","99","61"},
        {"17","2b","04","7e","ba","77","d6","26","e1","69","14","63","55","21","0c","7d"}};
```

- Biến công cộng tĩnh chỉ đọc thứ ba, có tên MixColumnFactor, với kiểu Matrix, và được gán trị đầu new Matrix(BaseTransform.FromHexToBinary("02010103030201010103020101010302"));
- Biến công cộng tĩnh chỉ đọc thứ tư, có tên Inverse_MixColumFactor, với kiểu Matrix, và được gán trị đầu new Matrix(BaseTransform.FromHexToBinary("0e090d0b0b0e090d0d0b0e09090d0b0e"));
- Biến công cộng tĩnh có tên Rcon, với kiểu List<Matrix>, và được gán trĩ đầu new List<Matrix>(10);

- Các hàm tạo: Có một hàm tạo tĩnh duy nhất không tham biến.
  - Nhiệm vụ của hàm tạo tĩnh này là thực hiện 10 lệnh sau:

```
TransformTables.Rcon.Add(new Matrix(BaseTransform.FromHexToBinary("01000000"), 4, 1));
TransformTables.Rcon.Add(new Matrix(BaseTransform.FromHexToBinary("02000000"), 4, 1));
TransformTables.Rcon.Add(new Matrix(BaseTransform.FromHexToBinary("04000000"), 4, 1));
TransformTables.Rcon.Add(new Matrix(BaseTransform.FromHexToBinary("08000000"), 4, 1));
TransformTables.Rcon.Add(new Matrix(BaseTransform.FromHexToBinary("10000000"), 4, 1));
TransformTables.Rcon.Add(new Matrix(BaseTransform.FromHexToBinary("20000000"), 4, 1));
TransformTables.Rcon.Add(new Matrix(BaseTransform.FromHexToBinary("40000000"), 4, 1));
TransformTables.Rcon.Add(new Matrix(BaseTransform.FromHexToBinary("80000000"), 4, 1));
TransformTables.Rcon.Add(new Matrix(BaseTransform.FromHexToBinary("1b000000"), 4, 1));
TransformTables.Rcon.Add(new Matrix(BaseTransform.FromHexToBinary("36000000"), 4, 1));
```

18. Tập tin lớp CacGiaiThuat.cs.
    - Khai báo biến: Có 4 biến công cộng tĩnh chỉ đọc.
      - Biến công cộng tĩnh chỉ đọc thứ nhất, có tên DES, với kiểu int, và giá trị đầu 0.
      - Biến công cộng tĩnh chỉ đọc thứ hai, có tên AES, với kiểu int, và giá trị ban đầu 1.
      - Biến công cộng tĩnh chỉ đọc thứ ba, có tên DES_File, với kiểu int, và giá trị ban đầu 2
      - Biến công cộng tĩnh chỉ đọc thứ tư, có tên AES_File, với kiểu int, và giá trị ban đầu 3.
19. Tập tin lớp MaHoa.cs
    - Khai báo biến: Có một biến riêng tư duy nhất.
      - Biến riêng tư có tên cProcess, với kiểu CommonProcess, và giá trị ban đầu null.
    - Các hàm tạo: Có một hàm tạo duy nhất ba tham biến.
      - Tham biến hình thức thứ nhất có tên algorithm_number, với kiểu int. Tham biến hình thức thứ hai, có tên IncrementProgress, với kiểu frmMaHoaGiaiMa.ProgressEventHandler. Tham biến hình thức thứ ba,

có tên InitProgress, với kiểu frmMaHoaGiaiMa.ProgressInitEventHandler. Nhiệm vụ của hàm tạo này là:

```
if (CacGiaiThuat.DES == algorithm_number || CacGiaiThuat.DES_File == algorithm_number)
{
   cProcess = new DES.ProcessDES(IncrementProgress, InitProgress);
}
else
{
   cProcess = new AES.ProcessAES(IncrementProgress, InitProgress);
}
```

- Các phương thức: Có 4 phương thức công cộng.
  - Phương thức công cộng thứ nhất, có tên EncryptionStart. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất, có tên filename, với kiểu string. Tham biến hình thức thứ hai, có tên EncryptedFilename, với kiểu string. Tham biến hình thức thứ ba, có tên key, với kiểu string. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
string binarytext = AES.FileIO.FileReadToBinary(filename);
binarytext =  this.EncryptionStart(binarytext, key, true);
AES.FileIO.WriteBinaryToFile(EncryptedFilename, binarytext);
```

  - Phương thức công cộng thứ hai, có tên EncryptionStart. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất, có tên text, với kiểu string. Tham biến hình thức thứ hai, có tên key, với kiểu string. Tham biến hình thứ thứ ba, có tên IsBinary, với kiểu luận lý. Phương thức này trả về kiểu string. Nhiệm vụ của phương thức này là:

```
return cProcess.EncryptionStart(text, key, IsBinary);
```

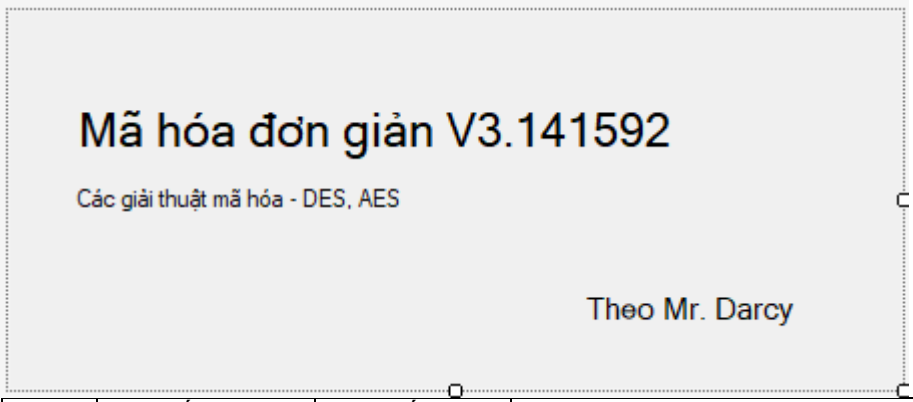
  - Phương thức công cộng thứ ba, có tên DecryptionStart. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất, có tên filename, với kiểu string. Tham biến hình thức thứ hai, có tên DecryptedFilename, với kiểu string. Tham biến hình thức thứ ba, có tên key, với kiểu string. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
string binarytext = AES.FileIO.FileReadToBinary(filename);
binarytext = this.DecryptionStart(binarytext, key, true);
AES.FileIO.WriteBinaryToFile(DecryptedFilename, binarytext);
```

  - Phương thức công cộng thứ tư, có tên DecryptionStart. Phương thức này có ba tham biến hình thức. Tham biến hình thức thứ nhất, có tên text, với kiểu string. Tham biến hình thức thứ hai, có tên key, với kiểu string. Tham biến hình thứ thứ ba, có tên IsBinary, với kiểu luận lý. Phương thức này trả về kiểu string. Nhiệm vụ của phương thức này là:

```
return cProcess.DecryptionStart(text, key, IsBinary);
```

20. Thiết kế giao diện Form StartUp.cs như sau:



STT	Tên đối tượng	Loại đối tượng	Text	Font.Size	Vị trí
1	lblDeAn	Label	Mã hóa đơn giản V3.141592	18	Dòng đầu tiên
2	lblDienGiai	Label	Các giải thuật mã hóa - DES, AES		Ngay dưới lblDeAn
3	lblTacGia	Label	Theo Mr. Darcy		Ngay dưới lblDienGia

Ghi chú: Cần sửa lại thành phần Text của lblTacGia thành *họ và tên của sinh viên*.

21. Các hàm tạo của StartUp.cs: Chỉ có một hàm tạo, trong đó, chỉ có một lệnh có sẵn InitializeComponent();
22. Các phương thức trong StartUp.cs: Có một phương thức riêng tư.
    - Phương thức riêng tư có tên CloseForm. Phương thức này không có tham biến hình thức, cũng không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
this.Close();
```

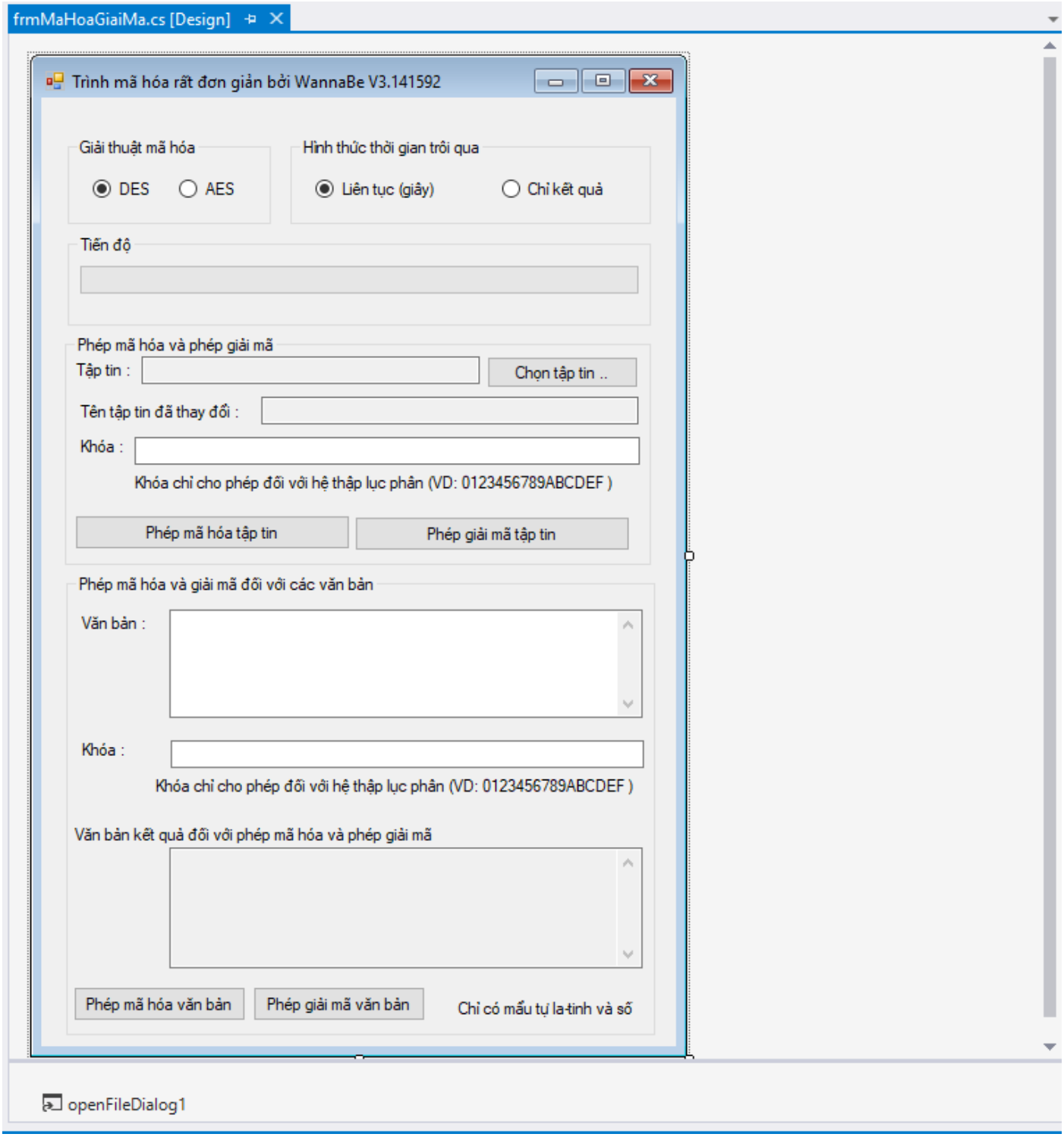
23. Các xử lý biến cố trong StartUp.cs: Có hai phương thức xử lý:
    - Phương thức xử lý biến cố Elapsed của timer1, có tên timer1_Elapsed. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên sender, với kiểu object. Tham biến hình thứ thứ hai, có tên e, với kiểu System.Timers.ElapsedEventArgs. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
 if (this.InvokeRequired) // This shouldn't happen since we are on the same thread
{
   this.Invoke(new MethodInvoker(CloseForm));
}
else
{
   CloseForm();
}
```

    - Phương thức xứ lý biến cố Load của StartUp, có tên StartUp_Load. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên sender, với kiểu object. Tham biến hình thứ thứ hai, có tên e, với kiểu EventArgs. Phương thức này kho6nhg có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
System.Timers.Timer timer1 = new System.Timers.Timer();

timer1.Elapsed += new System.Timers.ElapsedEventHandler(timer1_Elapsed);
timer1.Interval = 2000;
timer1.Start();
```

24. Thiết kế giao diện của frmMaHoaGiaiMa như sau:



STT	Tên đối tượng	Loại đối tượng	Text	Vị trí
1	BoxAlogrithm	GroupBox	Giải thuật mã hóa	Dòng đầu tiên
2	BoxElapsedTime	GroupBox	Hình thức thời gian trôi qua	Bên phải BoxAlgorithm
3	BoxProgress	GroupBox	Tiến độ	Ngay dưới BoxAlgorithm và BoxElapsedTime
4	BoxFileCrypt	GroupBox	Phép mã hóa và phép giải mã	Ngay dưới BoxProgress
5	BoxTextCrypt	GroupBox	Phép mã hóa và giải mã đối với các văn bản	Ngay dưới BoxFileCrypt
6	openFileDialog1	OpenFileDialog		Góc dưới màn hình

Bên trong BoxAlogorithm:

STT	Tên đối tượng	Loại đối tượng	Text	Checked	Vị trí
1	rbDES	RadioButton	DES	True	Cột bên trái
2	rbAES	RadioButton	AES	False	Cột bên phải

Bên trong BoxElapsedTime:

STT	Tên đối tượng	Loại đối tượng	Text	Checked	Vị trí
1	rbContinuous	RadioButton	Liên tục (giây)	True	Cột bên trái
2	rbResult	RadioButton	Chỉ kết quả	False	Cột bên phải

Bên trong BoxProgress:

STT	Tên đối tượng	Loại đối tượng	Text	Value	Vị trí
1	prbProgress	ProgressBar		0	Toàn bộ thanh ngang dài

Bên trong BoxFileCrypt:

STT	Tên đối tượng	Loại đối tượng	Text	ReadOnly	Vị trí
1	lblFile	Label	Tập tin		Dòng thứ nhất
2	txtFile	TextBox	(để trống)	True	Bên phải lblFile
3	btnFileSelect	Button	Chọn tập tin		Bên phải txtFile
4	lblAltered	Label	Tên tập tin đã thay đổi		Ngay dưới lblFile
5	txtAlteredFile	TextBox	(để trống)	True	Bên phải lblAltered
6	lblKey1	Label	Khóa		Ngay dưới lblAltered
7	txtKey_File	TextBox	(để trống)		Bên phải lblKey1
8	lblDesc1	Label	Khóa chỉ cho phép đối với hệ thập lục phân (VD: 0123456789ABCDEF)		Ngay dưới txtKey_File
9	btnFileEncrypt	Button	Phép mã hóa tập tin		Ngay dưới lblDesc1
10	btnFileDEscrypt	Button	Phép giải mã tập tin		Bên phải btnFileEncrypt

Bên trong BoxTextCrypted:

STT	Tên đối tượng	Loại đối tượng	Text	ScrollBars	Mulitple Line	Vị trí
1	lblText	Label	Văn bản			Dòng thứ nhất
2	txtPlainText	TextBox		Vertical	True	Bên phải lblText
3	lblKey2	Label	Khóa			Ngay dưới lblText
4	txtKey	TextBox			False	Bên phải lblKey2
5	lblDesc2	Label	Khóa chỉ cho phép đối với hệ thập lục phân (VD: 0123456789ABCDEF)			Ngay dưới txtKey
6	lblEncrypted	Label	Văn bản kết quả đối với phép mã hóa và phép giải mã			Ngay dưới lblDesc2
7	txtEncrypted	TextBox	(để trống)	Vertical	True	Ngay dưới lblEncrypted
8	btnTextEncrypt	Button	Phép mã hóa văn bản			Ngay dưới txtEncrypted
9	btnTextDecrypt	Button	Phép giải mã văn bản			Bên phải btnTextEncrypt
10	lblDesc3	Label	Chỉ có mẩu tự la-tinh và số			Bên phải btnTextDecrypt

25. Khai báo biến trong frmMaHoaGiaiMa:
    - Các ủy quyền: Có hai ủy quyền:

- Ủy quyền công cộng thứ nhất, có tên ProgressInitHandler, với hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên sender, với kiểu object. Tham biến hình thứ thứ hai, có tên e, với kiểu ProgressInitArgs.
- Ủy quyền công cộng thứ hai, có tên ProgressEventHandler, với hai tham biến hình thức. Tham biến hình thức thứ nhất, có tên sender, với kiểu object. Tham biến hình thứ thứ hai, có tên e, với kiểu ProgressEventArgs.

- Các biến cố: Có hai biến cố công cộng tĩnh:
  - Biến cố công cộng tĩnh thứ nhất, có tên IncrementProgress, với kiểu ProgressEventHandler.
  - Biến cố công cộng tĩnh thứ hai, có tên InitProgress, với kiểu ProgressInitHandler.
- Các biến riêng tư: 11 biến riêng tư.
  - Biến riêng tư thứ nhất, có tên crypthrapher, với kiểu MaHoa, và trị ban đầu null.
  - Biến riêng tư thứ hai, có tên elapsed, với kiểu nguyên, và trị ban đầu 0.
  - Biến riêng tư thứ ba, có tên IsFile, với kiểu luận lý, và giá trị ban đầu false.
  - Biến riêng tư thứ tư, có tên IsEncryption, với kiểu luận lý, và trị ban đầu false.
  - Biến riêng tư thứ năm, có tên start, với kiểu DateTime.
  - Biến riêng tư thứ sáu, có tên end, với kiểu DateTime.
  - Biến riêng tư thứ bảy, có tên result, với kiểu TimeSpan.
  - Biến riêng tư thứ tám, có tên timerDelegate, với kiểu TimerCallback.
  - Biến riêng tư thứ chín, có tên autoEvent, với kiểu AutoResetEvent.
  - Biến riêng tư thứ mười, có tên timer1, với kiểu System.Threading,Timer.
  - Biến riêng tư thứ mười một, có tên wkr, với kiểu BackgroundWorker.

26. Các hàm tạo trong frmMaHoaGiaiMa: Có một hàm tạo duy nhất không tham biến, trong đó có sẵn lệnh duy nhất InitializeComponent();
27. Các phương thức trong frmMaHoaGiaiMa: Có 2 phương thức công cộng và 10 phương thức riêng tư.
    - Phương thức công cộng thứ nhất, có tên DisableButtons. Phương thức này không có tham buến hình thức, cũng không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
CheckForIllegalCrossThreadCalls = false;

btnFileDecrypt.Enabled = false;
btnFileEncrypt.Enabled = false;
btnFileSelect.Enabled = false;
btnTextDecrypt.Enabled = false;
btnTextEncrypt.Enabled = false;
```

    - Phương thức công cộng thứ hai, có tên EnableButtons. Phương thức này không có tham biến hình thức, cũng không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
btnFileDecrypt.Enabled = true;
btnFileEncrypt.Enabled = true;
btnFileSelect.Enabled = true;
btnTextDecrypt.Enabled = true;
btnTextEncrypt.Enabled = true;
```

    - Phương thức riêng tư thứ nhất, có tên StartProcess. Phương thức này không có tham biến hình thức, cũng không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
if (rbContinuous.Checked)
{
   timer1 = new System.Threading.Timer(timerDelegate, autoEvent, 0, 1000);
   wkr.RunWorkerAsync();
}
else if (rbResult.Checked)
{
   DisableButtons();
   start = DateTime.Now;
   this.StartSelectedProcess();
   end = DateTime.Now;

   result = end - start;

   lblProgress.Text = "Elapsed Time : " + result.ToString();
```

```
        if (IsFile == true)
        {
            MessageBox.Show(lblEncryptedFilename.Text + " File Encryption/Decryption is complete.");
        }
        EnableButtons();
    }
```

- Phương thức riêng tư thứ hai, có tên StartSelectedProcess. Phương thức này không có tham biến hình thức, cũng không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
elapsed = 0;
    ClearProgressBar();

    if (IsEncryption)   // Mã hóa
    {
        if (IsFile)     // Mã hóa tập tin
        {
            cryptographer = new MaHoa(rbDES.Checked ? CacGiaiThuat.DES_File :
                                CacGiaiThuat.AES_File, IncrementProgress, InitProgress);
            cryptographer.EncryptionStart(txtFile.Text.Replace("\\", "\\\\"),
                                txtAlteredFile.Text.Replace("\\", "\\\\"), txtKey_File.Text.ToUpper());
        }
        else
        {                   // Mã hóa văn bản
            txtEncrypted.Clear();
            cryptographer = new MaHoa(rbDES.Checked ? CacGiaiThuat.DES : CacGiaiThuat.AES,
                                                IncrementProgress, InitProgress);
            txtEncrypted.Text = cryptographer.EncryptionStart(txtPlainText.Text, txtKey.Text.ToUpper(),
                                                                    false);
        }
    }
    else
    {   // Giải mã
        if (IsFile)
        {   // Giải mã tập tin
            cryptographer = new MaHoa(rbDES.Checked ? CacGiaiThuat.DES_File :
                                                CacGiaiThuat.AES_File, IncrementProgress, InitProgress);
            cryptographer.DecryptionStart(txtFile.Text.Replace("\\", "\\\\"),
                                        txtAlteredFile.Text.Replace("\\", "\\\\"), txtKey_File.Text.ToUpper());
        }
        else
        {   // Giải mã văn bản
            txtEncrypted.Clear();
            cryptographer = new MaHoa(rbDES.Checked ? CacGiaiThuat.DES : CacGiaiThuat.AES,
                                                IncrementProgress, InitProgress);
            txtEncrypted.Text = AES.BaseTransform.FromBinaryToText(
                                cryptographer.DecryptionStart(txtPlainText.Text, txtKey.Text.ToUpper(), true));
        }
    }
```

- Phương thức riêng tư thứ ba, có tên FileCheck. Phương thức này không có tham biến hình thức. Phương thức này trả về kiểu luận lý. Nhiệm vụ của phương thức này là:

```
    if (txtFile.Text == "")
    {
        MessageBox.Show("Please select a file for encryption/decryption");
        return false;
    }
    return true;
```

- Phương thức riêng tư thứ tư, có tên KeyCheck. Phương thức này không có tham biến hình thức. Phương thức này trả về kiểu luận lý. Nhiệm vụ của phương thức này là:

```
if (rbDES.Checked)
{
   if (key.Length == 16)
   {
      return true;
   }
   else
   {
      MessageBox.Show("The key must be 16-HexDecimal Length for DES algorithm");
      return false;
   }
}
else
{
   if (key.Length == 32)
   {
      return true;
   }
   else
   {
      MessageBox.Show("The key must be 32-HexDecimal Length for AES algorithm");
      return false;
   }
}
```

- Phương thức riêng tư thứ năm, có tên ClearProgressBar. Phương thức này không có tham biến hình thức, cũng không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
prbProgress.Value = 0;
```

- Phương thức riêng tư thứ sáu, có tên IsHexaDecimal. Phương thức này có một tham biến hình thức có tên value, với kiểu char. Phương thức này trả về kiểu luận lý. Nhiệm vụ của phương thức này là:

```
if ((('0' <= value) && (value <= '9')) ||
   (('A' <= value) && (value <= 'F')) ||
   (('a' <= value) && (value <= 'f')) ||
   (8 == (int)value))
{
   return true;
}
return false;
```

- Phương thức riêng tư thứ bảy, có tên increase. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến hình thức thứ hai, có tên e, với kiểu ProgressEventArgs. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
prbProgress.Increment(e.Increment);
```

- Phương thức riêng tư thứ tám, có tên Initialize. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến hình thức thứ hai, có tên e, với kiểu ProgressInitArgs. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
prbProgress.Maximum = e.Maximum;
```

- Phương thức riêng tư thứ chín, có tên wkr_RunWorkerCompleted. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến hình thức thứ hai, có tên e, với kiểu RunWorkerCompletedEventArgs. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
timer1.Dispose();
if (IsFile)
{
   MessageBox.Show(lblEncryptedFilename.Text + " File Encryption/Decryption is complete.");
}
```

```
EnableButtons();
```

- Phương thức riêng tư thứ mười, có tên wkr_DoWork. Phương thức này có hai tham biến hình thức. Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến hình thức thứ hai, có tên e, với kiểu DoWorkEventArgs. Phương thức này không có kiểu dữ liệu trả về. Nhiệm vụ của phương thức này là:

```
DisableButtons();
this.StartSelectedProcess();
```

28. Các xử lý biến cố trong frmMaHoaGiaiMa: Có 10 phương thức riêng tư xử lý biến cố và một phương thức công cộng xử lý biến cố.
    - Phương thức riêng tư thứ nhất xử lý biến cố Click của btnFileEncrypt có tên btnFileEncrypt _Click. Phương thức này có hai tham biến hình thức.Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến thứ hai, có tên e, với kiểu EventArgs. Nhiệm vụ của phương thức này là:

```
 if (!KeyCheck(txtKey_File.Text) || !FileCheck())
{
   return;
}
IsFile = true;
IsEncryption = true;
StartProcess();
```

    - Phương thức riêng tư thứ hai xử lý biến cố Click của btnFileDecrypt có tên btnFileDecrypt _Click. Phương thức này có hai tham biến hình thức.Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến thứ hai, có tên e, với kiểu EventArgs. Nhiệm vụ của phương thức này là:

```
if (!KeyCheck(txtKey_File.Text) || !FileCheck())
{
   return;
}
IsFile = true;
IsEncryption = false;

StartProcess();
```

    - Phương thức riêng tư thứ ba xử lý biến cố Click của btnTextEncrypt có tên btnTextEncrypt _Click. Phương thức này có hai tham biến hình thức.Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến thứ hai, có tên e, với kiểu EventArgs. Nhiệm vụ của phương thức này là:

```
if (!KeyCheck(txtKey.Text))
{
   return;
}
IsFile = false;
IsEncryption = true;
StartProcess();
```

    - Phương thức riêng tư thứ tư xử lý biến cố Click của btnTextDecrypt có tên btnTextDecrypt _Click. Phương thức này có hai tham biến hình thức.Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến thứ hai, có tên e, với kiểu EventArgs. Nhiệm vụ của phương thức này là:

```
if (!KeyCheck(txtKey.Text))
{
   return;
}
IsFile = false;
IsEncryption = false;
StartProcess();
```

    - Phương thức riêng tư thứ năm xử lý biến cố Click của btnFileSelect có tên btnFileSelect_Click. Phương thức này có hai tham biến hình thức.Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến thứ hai, có tên e, với kiểu EventArgs. Nhiệm vụ của phương thức này là:

```
txtFile.Clear();
txtAlteredFile.Clear();

if (openFileDialog1.ShowDialog() == DialogResult.OK)
```

```
{
   txtFile.Text = openFileDialog1.FileName;
   txtAlteredFile.Text = openFileDialog1.FileName.Replace(".", "_2.");
}
```

- Phương thức riêng tư thứ sáu xử lý biến cố Load của frmMaHoaGiaiMa có tên frmMaHoaGiaiMa_Load. Phương thức này có hai tham biến hình thức.Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến thứ hai, có tên e, với kiểu EventArgs. Nhiệm vụ của phương thức này là:

```
StartUp startup = new StartUp();
startup.ShowDialog();

// Initialize progressbar
prbProgress.Maximum = 100;
prbProgress.Step = 1;

InitProgress += new ProgressInitHandler(Initialize);
IncrementProgress += new ProgressEventHandler(increase);

// Initialize timer
timerDelegate = new TimerCallback(this.timer1_Tick);
autoEvent = new AutoResetEvent(false);

// Initialize BackgroundWorker for ecryption and decryption process
wkr = new BackgroundWorker();
wkr.DoWork += new DoWorkEventHandler(wkr_DoWork);
wkr.RunWorkerCompleted += new
                          RunWorkerCompletedEventHandler(wkr_RunWorkerCompleted);
```

- Phương thức riêng tư thứ bảy xử lý biến cố CheckedChanged của rbDES có tên rbDES_ CheckedChanged. Phương thức này có hai tham biến hình thức.Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến thứ hai, có tên e, với kiểu EventArgs. Nhiệm vụ của phương thức này là:

```
txtKey_File.MaxLength = 16;
txtKey.MaxLength = 16;
```

- Phương thức riêng tư thứ tám xử lý biến cố CheckedChanged của rbAES có tên rbAES_ CheckedChanged. Phương thức này có hai tham biến hình thức.Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến thứ hai, có tên e, với kiểu EventArgs. Nhiệm vụ của phương thức này là:

```
txtKey_File.MaxLength = 32;
txtKey.MaxLength = 32;
```

- Phương thức riêng tư thứ chín xử lý biến cố KeyPress của txtKey_File có tên txtKey_File_KeyPress. Phương thức này có hai tham biến hình thức.Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến thứ hai, có tên e, với kiểu KeyPressEventArgs. Nhiệm vụ của phương thức này là:

```
if (!IsHexaDecimal(e.KeyChar))
   e.Handled = true;
else
   e.Handled = false;
```

- Phương thức riêng tư thứ mười xử lý biến cố KeyPress của txtKey có tên txtKey_KeyPress. Phương thức này có hai tham biến hình thức.Tham biến hình thức thứ nhất có tên sender, với kiểu object. Tham biến thứ hai, có tên e, với kiểu KeyPressEventArgs. Nhiệm vụ của phương thức này là:

```
if (!IsHexaDecimal(e.KeyChar))
   e.Handled = true;
else
  e.Handled = false;
```

- Phương thức công cộng xử lý biến cố Tick của timer1, có tên Timer1_Tick. Phương thức này có một tham biến hình thức, có tên stateInfo, với kiểu object. Phương thức này không có kiểu dữ kiệu trả về. Nhiệm vụ của phương thức này là:

```
this.Invoke(new MethodInvoker(delegate ()
{
   elapsed++;
```

```
            lblProgress.Text = "Elapsed Time : " + elapsed.ToString() + " sec.";
        }));
```

29. Thử lại toàn bộ ứng dụng có thể mã hóa và giải mã tập tin, cũng như văn bản với khóa dài 64 bit và 128 bit.

---oOo---