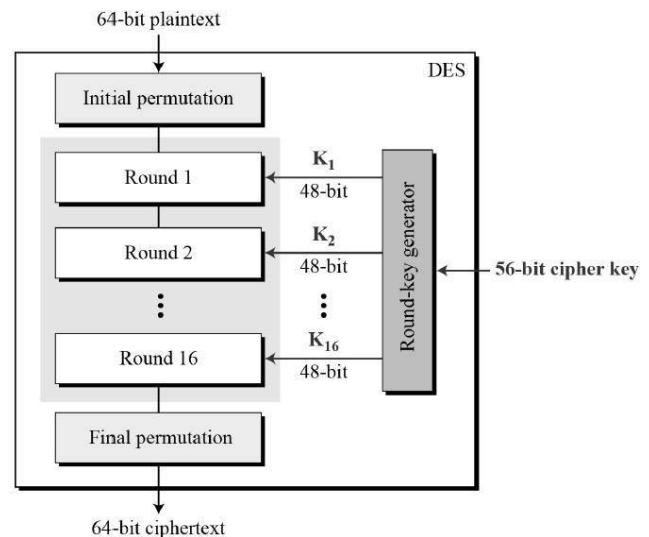


Program – 5

AIM: To implement a program to show encryption and decryption using DES.

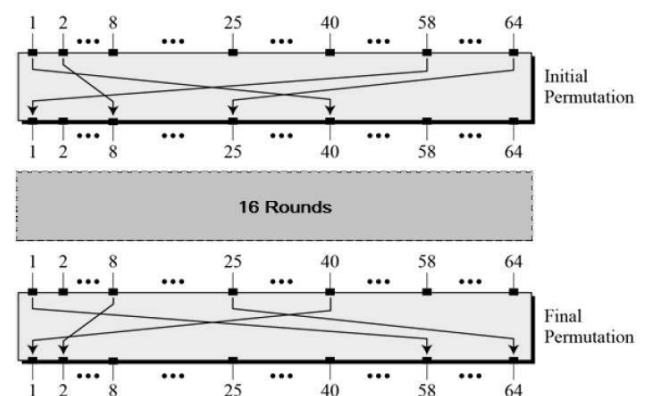
Introduction and Theory

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST). DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only).



Initial and Final Permutation

The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES.



Round Function

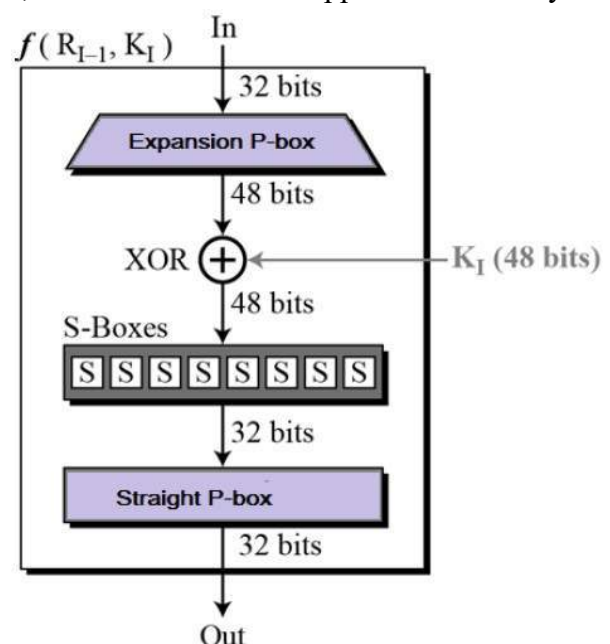
The heart of this cipher is the DES function, f . The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.

Expansion Permutation Box

- Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits.

XOR (Whitener)

- After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.



Program – 5

Substitution Boxes.

- The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output.
- There is a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32-bit section.

Straight Permutation

- The 32-bit output of S-boxes is then subjected to the straight permutation rule.

Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key.

Shifting	
Rounds	Shift
1, 2, 9, 16	one bit
Others	two bits

DES Analysis

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

Avalanche effect

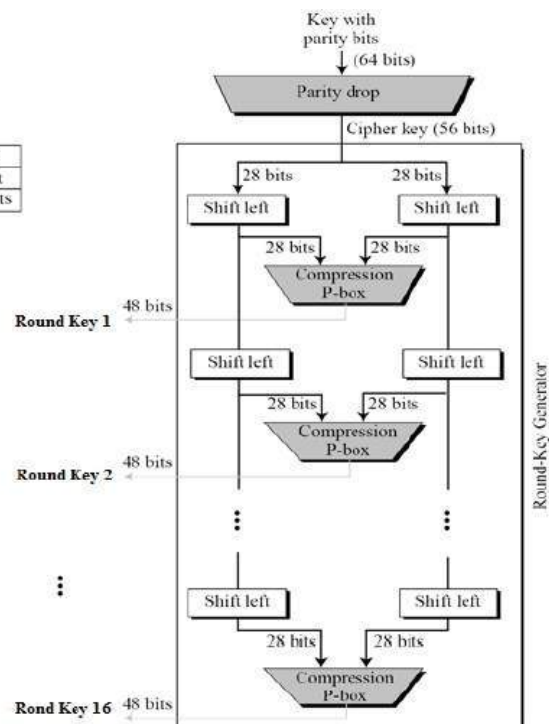
A small change in plaintext results in the very great change in the ciphertext.

Completeness

Each bit of ciphertext depends on many bits of plaintext.

During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.

DES has proved to be a very well-designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.



Program – 5

Code

```
1  #include <stdio.h>
2  #include <fstream>
3  #include <string.h>
4  #include <iostream>
5  #include <stdlib.h>
6  using namespace std;
7
8  int key[64] =
9  {
10     0, 1, 1, 0, 1, 0, 0, 1,
11     1, 1, 1, 1, 1, 0, 1, 0,
12     1, 1, 0, 0, 0, 1, 1, 0,
13     1, 0, 0, 1, 0, 1, 1, 0,
14     1, 0, 0, 1, 0, 1, 1, 0,
15     1, 0, 0, 1, 1, 1, 1, 1,
16     1, 0, 1, 0, 1, 1, 0, 0,
17     0, 1, 1, 0, 1, 0, 0, 1};
18
19  class Des
20  {
21  public:
22     int round_key[16][48], total[64], left[32], right[32], ck[28];
23     int dk[28], E_box[48], z[48], xor1[48], sub[32];
24     int p[32], xor2[32], temp[64], pc1[56], ip[64];
25     int inv[8][8];
26
27     char final[1000];
28     void Init_perm();
29     void PermChoice1();
30     void PermChoice2();
31     void E_Function();
32     void inverse();
33     void xor_two();
34     void xor_oneE(int);
35     void xor_oneD(int);
36     void substitution();
37     void permutation();
38     void keygen();
39     char *Encrypt(char *);
40     char *Decrypt(char *);
41 };
42
43 void Des::Init_perm()
44 {
45     /*
46     Does the initial permutation
47     No real significance
48     */
49     int k = 58, i;
50     for (i = 0; i < 32; i++)
51     {
52         ip[i] = total[k - 1];
53         if (k - 8 > 0)
54             k = k - 8;
```

Program – 5

```
55         else
56             k = k + 58;
57     }
58     k = 57;
59     for (i = 32; i < 64; i++)
60     {
61         ip[i] = total[k - 1];
62         if (k - 8 > 0)
63             k = k - 8;
64         else
65             k = k + 58;
66     }
67 }
68
69 void Des::PermChoice1() //Permutation Choice-1
70 {
71     int k = 57, i;
72     for (i = 0; i < 28; i++)
73     {
74         pc1[i] = key[k - 1];
75         if (k - 8 > 0)
76             k = k - 8;
77         else
78             k = k + 57;
79     }
80     k = 63;
81     for (i = 28; i < 52; i++)
82     {
83         pc1[i] = key[k - 1];
84         if (k - 8 > 0)
85             k = k - 8;
86         else
87             k = k + 55;
88     }
89     k = 28;
90     for (i = 52; i < 56; i++)
91     {
92         pc1[i] = key[k - 1];
93         k = k - 8;
94     }
95 }
96
97 void Des::E_Function()
98 {
99     /* Expansion function from 32 to 48 bits */
100    int exp[8][6], i, j, k;
101    for (i = 0; i < 8; i++)
102    {
103        for (j = 0; j < 6; j++)
104        {
105            if ((j != 0) || (j != 5))
106            {
107                k = 4 * i + j;
108                exp[i][j] = right[k - 1];
109            }
110            if (j == 0)
111            {
```

Program – 5

```
112         k = 4 * i;
113         exp[i][j] = right[k - 1];
114     }
115     if (j == 5)
116     {
117         k = 4 * i + j;
118         exp[i][j] = right[k - 1];
119     }
120 }
121 }
122 exp[0][0] = right[31];
123 exp[7][5] = right[0];
124
125 k = 0;
126 for (i = 0; i < 8; i++)
127     for (j = 0; j < 6; j++)
128         E_box[k++] = exp[i][j];
129 }
130
131 void Des::PermChoice2()
132 {
133     int per[56], i, k;
134     for (i = 0; i < 28; i++)
135         per[i] = ck[i];
136
137     for (k = 0, i = 28; i < 56; i++)
138         per[i] = dk[k++];
139
140     z[0] = per[13]; z[1] = per[16]; z[2] = per[10]; z[3] = per[23];
141     z[4] = per[0]; z[5] = per[4]; z[6] = per[2]; z[7] = per[27];
142     z[8] = per[14]; z[9] = per[5]; z[10] = per[20]; z[11] = per[9];
143     z[12] = per[22]; z[13] = per[18]; z[14] = per[11]; z[15] = per[3];
144     z[16] = per[25]; z[17] = per[7]; z[18] = per[15]; z[19] = per[6];
145     z[20] = per[26]; z[21] = per[19]; z[22] = per[12]; z[23] = per[1];
146     z[24] = per[40]; z[25] = per[51]; z[26] = per[30]; z[27] = per[36];
147     z[28] = per[46]; z[29] = per[54]; z[30] = per[29]; z[31] = per[39];
148     z[32] = per[50]; z[33] = per[46]; z[34] = per[32]; z[35] = per[47];
149     z[36] = per[43]; z[37] = per[48]; z[38] = per[38]; z[39] = per[55];
150     z[40] = per[33]; z[41] = per[52]; z[42] = per[45]; z[43] = per[41];
151     z[44] = per[49]; z[45] = per[35]; z[46] = per[28]; z[47] = per[31];
152 }
153
154 void Des::xor_oneE(int round) //for Encrypt
155 {
156     int i;
157     for (i = 0; i < 48; i++)
158         xor1[i] = E_box[i] ^ round_key[round - 1][i];
159 }
160
161 void Des::xor_oneD(int round) //for Decrypt
162 {
163     int i;
164     for (i = 0; i < 48; i++)
165         xor1[i] = E_box[i] ^ round_key[16 - round][i];
166 }
167
168 void Des::substitution()
```

Program – 5

```
169 {
170     int s1[4][16] =
171     {
172         14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
173         0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
174         4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
175         15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13};
176
177     int s2[4][16] =
178     {
179         15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
180         3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
181         0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
182         13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9};
183
184     int s3[4][16] =
185     {
186         10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
187         13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
188         13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
189         1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12};
190
191     int s4[4][16] =
192     {
193         7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
194         13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
195         10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
196         3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14};
197
198     int s5[4][16] =
199     {
200         2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
201         14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
202         4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
203         11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3};
204
205     int s6[4][16] =
206     {
207         12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
208         10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
209         9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
210         4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13};
211
212     int s7[4][16] =
213     {
214         4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
215         13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
216         1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
217         6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12};
218
219     int s8[4][16] =
220     {
221         13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
222         1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
223         7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
224         2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11};
225     int a[8][6], k = 0, i, j, p, q, count = 0, g = 0, v;
```

Program – 5

```
226
227     for (i = 0; i < 8; i++)
228     {
229         for (j = 0; j < 6; j++)
230         {
231             a[i][j] = xor1[k++];
232         }
233     }
234
235     for (i = 0; i < 8; i++)
236     {
237         p = 1;
238         q = 0;
239         k = (a[i][0] * 2) + (a[i][5] * 1);
240         j = 4;
241         while (j > 0)
242         {
243             q = q + (a[i][j] * p);
244             p = p * 2;
245             j--;
246         }
247         count = i + 1;
248         switch (count)
249         {
250             case 1:
251                 v = s1[k][q];
252                 break;
253             case 2:
254                 v = s2[k][q];
255                 break;
256             case 3:
257                 v = s3[k][q];
258                 break;
259             case 4:
260                 v = s4[k][q];
261                 break;
262             case 5:
263                 v = s5[k][q];
264                 break;
265             case 6:
266                 v = s6[k][q];
267                 break;
268             case 7:
269                 v = s7[k][q];
270                 break;
271             case 8:
272                 v = s8[k][q];
273                 break;
274         }
275
276         int d, i = 3, a[4];
277         while (v > 0)
278         {
279             d = v % 2;
280             a[i--] = d;
281             v = v / 2;
282         }
```

Program – 5

```
283     while (i >= 0)
284     {
285         a[i--] = 0;
286     }
287
288     for (i = 0; i < 4; i++)
289         sub[g++] = a[i];
290 }
291 }
292
293
294 void Des::permutation()
295 {
296     p[0] = sub[15]; p[1] = sub[6]; p[2] = sub[19]; p[3] = sub[20];
297     p[4] = sub[28]; p[5] = sub[11]; p[6] = sub[27]; p[7] = sub[16];
298     p[8] = sub[0]; p[9] = sub[14]; p[10] = sub[22]; p[11] = sub[25];
299     p[12] = sub[4]; p[13] = sub[17]; p[14] = sub[30]; p[15] = sub[9];
300     p[16] = sub[1]; p[17] = sub[7]; p[18] = sub[23]; p[19] = sub[13];
301     p[20] = sub[31]; p[21] = sub[26]; p[22] = sub[2]; p[23] = sub[8];
302     p[24] = sub[18]; p[25] = sub[12]; p[26] = sub[29]; p[27] = sub[5];
303     p[28] = sub[21]; p[29] = sub[10]; p[30] = sub[3]; p[31] = sub[24];
304 }
305
306
307 void Des::xor_two()
308 {
309     int i;
310     for (i = 0; i < 32; i++)
311     {
312         xor2[i] = left[i] ^ p[i];
313     }
314 }
315
316
317 void Des::inverse()
318 {
319     int p = 40, q = 8, k1, k2, i, j;
320     for (i = 0; i < 8; i++)
321     {
322         k1 = p;
323         k2 = q;
324         for (j = 0; j < 8; j++)
325         {
326             if (j % 2 == 0)
327             {
328                 inv[i][j] = temp[k1 - 1];
329                 k1 = k1 + 8;
330             }
331             else if (j % 2 != 0)
332             {
333                 inv[i][j] = temp[k2 - 1];
334                 k2 = k2 + 8;
335             }
336         }
337         p = p - 1;
338         q = q - 1;
339     }
```


Program – 5

```
340 }
341
342
343 char *Des::Encrypt(char *Text1)
344 {
345     int i, a1, j, nB, m, iB, k, K, B[8], n, t, d, round;
346     char *Text = new char[1000];
347     strcpy(Text, Text1);
348     i = strlen(Text);
349     int mc = 0;
350     a1 = i % 8;
351     if (a1 != 0)
352         for (j = 0; j < 8 - a1; j++, i++)
353             Text[i] = ' ';
354     Text[i] = '\0';
355     keygen();
356     for (iB = 0, nB = 0, m = 0; m < (strlen(Text) / 8); m++)
357     {
358         for (iB = 0, i = 0; i < 8; i++, nB++)
359         {
360             n = (int)Text[nB];
361             for (K = 7; n >= 1; K--)
362             {
363                 B[K] = n % 2;
364                 n /= 2;
365             }
366             for (; K >= 0; K--)
367                 B[K] = 0;
368             for (K = 0; K < 8; K++, iB++)
369                 total[iB] = B[K];
370         }
371         Init_perm();
372         for (i = 0; i < 64; i++)
373             total[i] = ip[i];
374
375         for (i = 0; i < 32; i++)
376             left[i] = total[i];
377
378         for (; i < 64; i++)
379             right[i - 32] = total[i];
380         for (round = 1; round <= 16; round++)
381         {
382             E_Function();
383             xor_oneE(round);
384             substitution();
385             permutation();
386             xor_two();
387             for (i = 0; i < 32; i++)
388                 left[i] = right[i];
389             for (i = 0; i < 32; i++)
390                 right[i] = xor2[i];
391         }
392         for (i = 0; i < 32; i++)
393             temp[i] = right[i];
394         for (; i < 64; i++)
395             temp[i] = left[i - 32];
396         inverse();

```

Program – 5

```
397     k = 128;
398     d = 0;
399     for (i = 0; i < 8; i++)
400     {
401         for (j = 0; j < 8; j++)
402         {
403             d = d + inv[i][j] * k;
404             k = k / 2;
405         }
406         final[mc++] = (char)d;
407         k = 128;
408         d = 0;
409     }
410 } //for loop ends here
411 final[mc] = '\0';
412 return (final);
413 }
414
415
416 char *Des::Decrypt(char *Text1)
417 {
418     int i, a1, j, nB, m, iB, k, K, B[8], n, t, d, round;
419     char *Text = new char[1000];
420     unsigned char ch;
421     strcpy(Text, Text1);
422     i = strlen(Text);
423     keygen();
424     int mc = 0;
425     for (iB = 0, nB = 0, m = 0; m < (strlen(Text) / 8); m++)
426     {
427         for (iB = 0, i = 0; i < 8; i++, nB++)
428         {
429             ch = Text[nB];
430             n = (int)ch;
431             for (K = 7; n >= 1; K--)
432             {
433                 B[K] = n % 2;
434                 n /= 2;
435             }
436             for (; K >= 0; K--)
437                 B[K] = 0;
438             for (K = 0; K < 8; K++, iB++)
439                 total[iB] = B[K];
440         }
441         Init_perm();
442         for (i = 0; i < 64; i++)
443             total[i] = ip[i];
444         for (i = 0; i < 32; i++)
445             left[i] = total[i];
446         for (; i < 64; i++)
447             right[i - 32] = total[i];
448         for (round = 1; round <= 16; round++)
449         {
450             E_Function();
451             xor_oneD(round);
452             substitution();
453             permutation();
```

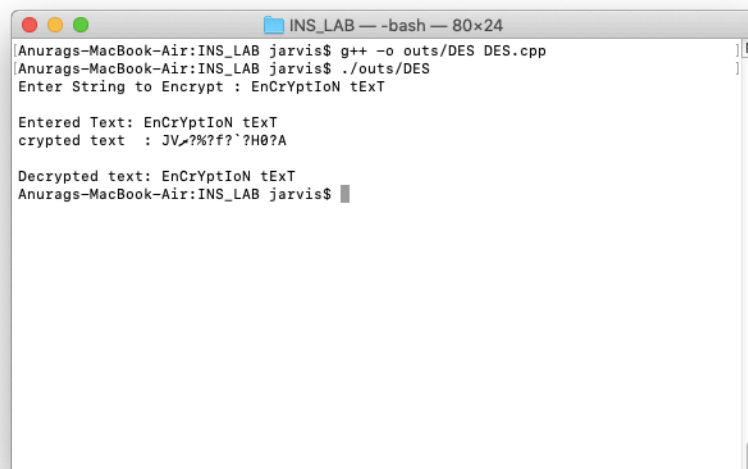
Program – 5

```
454         xor_two();
455         for (i = 0; i < 32; i++)
456             left[i] = right[i];
457         for (i = 0; i < 32; i++)
458             right[i] = xor2[i];
459     }
460     for (i = 0; i < 32; i++)
461         temp[i] = right[i];
462     for (; i < 64; i++)
463         temp[i] = left[i - 32];
464     inverse();
465     k = 128;
466     d = 0;
467     for (i = 0; i < 8; i++)
468     {
469         for (j = 0; j < 8; j++)
470         {
471             d = d + inv[i][j] * k;
472             k = k / 2;
473         }
474         final[mc++] = (char)d;
475         k = 128;
476         d = 0;
477     }
478 } //for loop ends here
479 final[mc] = '\0';
480 char *final1 = new char[1000];
481 for (i = 0, j = strlen(Text); i < strlen(Text); i++, j++)
482     final1[i] = final[j];
483 final1[i] = '\0';
484 return (final);
485 }
486
487
488 void Des::keygen()
489 {
490     PermChoice1();
491
492     int i, j, k = 0;
493     for (i = 0; i < 28; i++)
494     {
495         ck[i] = pc1[i];
496     }
497     for (i = 28; i < 56; i++)
498     {
499         dk[k] = pc1[i];
500         k++;
501     }
502     int noshift = 0, round;
503     for (round = 1; round <= 16; round++)
504     {
505         if (round == 1 || round == 2 || round == 9 || round == 16)
506             noshift = 1;
507         else
508             noshift = 2;
509         while (noshift > 0)
510         {
```

Program – 5

```
511         int t;
512         t = ck[0];
513         for (i = 0; i < 28; i++)
514             ck[i] = ck[i + 1];
515         ck[27] = t;
516         t = dk[0];
517         for (i = 0; i < 28; i++)
518             dk[i] = dk[i + 1];
519         dk[27] = t;
520         noshift--;
521     }
522     PermChoice2();
523     for (i = 0; i < 48; i++)
524         round_key[round - 1][i] = z[i];
525 }
526 }
527
528
529 int main()
530 {
531     Des d1, d2;
532     char *str = new char[1000];
533     char *str1 = new char[1000];
534     cout << "Enter String to Encrypt : ";
535     cin.getline(str, 1000);
536     str1 = d1.Encrypt(str);
537     cout << "\nEntered Text: " << str << endl;
538     cout << "\nEncrypted text : " << str1 << endl;
539     cout << "\nDecrypted text: " << d2.Decrypt(str1) << endl;
540 }
541
542
543
```

Results and Outputs:



```
INS_LAB -- -bash -- 80x24
Anurags-MacBook-Air:INS_LAB jarvis$ g++ -o outs/DES DES.cpp
Anurags-MacBook-Air:INS_LAB jarvis$ ./outs/DES
Enter String to Encrypt : EnCrYptIoN tExT

Entered Text: EnCrYptIoN tExT
rypted text : JVx?%?f?'?H0?A

Decrypted text: EnCrYptIoN tExT
Anurags-MacBook-Air:INS_LAB jarvis$
```

Program – 5

Findings and Learnings:

1. We have implemented DES for Encryption and Decryption