DRONACHARYA GROUP OF INSTITUTIONS, GREATER NOIDA

Affiliated to Mahamaya Technical University, Noida

Approved by AICTE

Cryptography and Network Security Lab (EIT - 751)

```
1.  /* Program for RSA Algorithm */
2.  #include<conio.h>
3.  #include<math.h>
4.  int funct(char p)
5.  {
6.     int x;
7.     if(p>='a'&&p<='z')
8.       x=p-96;
9.     if(p>='A'&&p<='Z')
10.      x=p-64;
11.    return x;
12. }
13. void main()
14. {
15.  int i,p,q,n,z,e,s,c[30],p2[30],flag[30],tra;
16.  float l,d;
17.  long double x,y;
18.  unsigned long int div,div1;
19.  char pra[30];
20.  clrscr();
21.  printf("Enter the Values of P & Q");
22.  scanf("%d%d",&p,&q);
23.  n=p*q;
24.  z=(p-1)*(q-1);
25.  printf("Enter the Encryption key");
26.  scanf("%d",&e);
27.  l=1*(abs(z));
28.  d=ceil(l/e);
29.  printf("Enter The Plain Text");
30.  scanf("%s",pra);
31. for(i=0;i<strlen(pra);i++)
32. {
33. flag[i]=0;
34. if(pra[i]>='a'&&pra[i]<='z')
35.  flag[i]=1;
36. else
37.  continue;
38. }
39.  printf("Encryption\n");
40.  for(i=0;i<strlen(pra);i++)
41.  {
42.    s=funct(pra[i]);
43.    x=pow(((long double)s),((long double)e));
44.    div=(unsigned long int)x;
45.    c[i]=div%n;
46.    printf("%d  ",c[i]);
47.  }
48.  printf("\nDecryption\n");
49.  for(i=0;i<strlen(pra);i++)
50.  {
51.    tra=c[i];
```

```
52.         y=pow(((long double)tra),((long double)d));
53.          div1=(unsigned long int)y;
54.          p2[i]=div1%n;
55.          if(flag[i]==0)
56.            printf("%c   ",(p2[i]+64));
57.          else
58.                    printf("%c  ",(p2[i]+96));
59.       }
60.       getch();
61.     }
```

```
/*************************************************************
Diffie–Hellman key exchange algorithm
-------------------------------------
This program demonstrates the Diffie-Hellman exchange key
algorithm

Tips to improve  security
-------------------------
[1] Pseudo random number generator[Prime Number] should be
    changed to a true random source
    Ref::http://www.random.org/

[2] An Arbitrary-precision arithmetic library should be used
    to increase the [Prime number] key strength to 1024 bits
    Ref::http://en.wikipedia.org/wiki/Arbitrary-precision_arithmetic

Diffie–Hellman key exchange process,this is what the program is suppose to do
-----------------------------------------------------------------------------
[1.]Alice and Bob agree to use a prime number p=23 and base g=5.

[2.]Alice chooses a secret integer a=6, then sends Bob A = ga mod p
    A = 56 mod 23
    A = 8

[3.]Bob chooses a secret integer b=15, then sends Alice B = gb mod p
    B = 515 mod 23
    B = 19

[4.]Alice computes s = B a mod p
    s = 196 mod 23
    s = 2

[5]Bob computes s = A b mod p
    s = 815 mod 23
    s = 2

Ref::http://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

*************************************************************/
#include <iostream>
#include <math.h>
#include <cstdlib>
#include <ctime>
using namespace std;
/*************************************************************
Check to see if a valid prime number
*************************************************************/
int64_t is_prime(int64_t _iVal)
```

```
{
   char bPrime;
   bPrime = true;
   for (int ii(2); ii < _iVal / 2; ii++)
   {
      if (!(_iVal % ii)){
         bPrime = false;
         break;
      }
   }
   return bPrime;
}
/*************************************************************
Get random prime number
*************************************************************/
int64_t get_prime_number(int64_t MAX)
{
   int64_t rPrime;
   char result;
   srand ( time(NULL) );

   while (1)/** Until we get valid prime number */
   {
      /**
      Randomize a number
      */
      rPrime = rand() % MAX + 1;
      /**
      Deal with the Even Number returned
      */
      if ( rPrime % 2 == 0 ){
         /**
         Let's not waste this number change
         it to a ODD value by adding 1 & check
         if prime number
         */
         rPrime = rPrime + 1;
         /**
         Check if prime number
         */
         result = is_prime(rPrime);
         if(result == true){
            break;
         }
      /**
      Deal with the Odd Number returned
      */
      }else{
         /**
         Check if prime number
         */
```

```
            result = is_prime(rPrime);
            if(result == true){
                break;
            }
        }
    }
    return rPrime;
}
/*************************************************************
Diffie–Hellman ~ Create public key
*************************************************************/
int create_public_key(int64_t pn, int64_t gn, int64_t s_key)
{
    int64_t exp = ceil(pow(gn,s_key));
    int64_t pub_key = abs(exp % pn);
    return pub_key;
}
/*************************************************************
Diffie–Hellman ~ Compute shared secret key
*************************************************************/
int shared_secret_key(int64_t pn, int64_t s_key, int64_t pub_key)
{
    int64_t exp = ceil(pow(pub_key,s_key));
    int64_t shared_secret_key = abs(exp % pn);
    return shared_secret_key;
}
/*************************************************************
Main Program
*************************************************************/
int main()
{
    /**
    Get a random prime number & set g_base to 2 or 5
    */
    //int p_number = get_prime_number(99999999);
    int64_t p_number = 1039;//Static prime number
    cout << "Random Prime Number is:: " << p_number << endl;
    int64_t g_base = 5;//public [base number should set to 2 or 5]

    /**
    Create alice a public key
    */
    int64_t alice_pub_key = create_public_key(p_number, g_base, 6/*secret_key*/);
    cout << "Alice's public key = " << alice_pub_key << endl;

    /**
    Alice sends bob Public values p = p_number, g = g_base, A = alice_pub_key
    */

    /**
    Create bob a public key
```

```
151.      */
152.      int64_t bob_pub_key = create_public_key(p_number, g_base, 15/*secret_key*/);
153.      cout << "Bobs's public key = " << bob_pub_key << endl;
154.
155.      /**
156.      Bob sends Alice his Public value B = bob_pub_key
157.      */
158.
159.      /**
160.      Compute Alice's shared secret key
161.      */
162.      int64_t alice_secret_key = shared_secret_key(p_number, 6/*secret_key*/,
      bob_pub_key);
163.      cout << "Alice's shared secret key = " << alice_secret_key << endl;
164.
165.      /**
166.      Compute Bob's shared secret key
167.      */
168.      int64_t bob_secret_key = shared_secret_key(p_number, 15/*secret_key*/,
      alice_pub_key);
169.      cout << "Bob's shared secret key = " << bob_secret_key << endl;
170.
171.      /**
172.      NOTE
173.      ----
174.      Both Alice & Bob shared secret keys should match if the program
175.      has done it's job properly
176.      */
177.      return 0;
178.  }
```

```
/*************************************************************************
* *
* Implement elliptic curve point addition for polynomial basis form. *
* This follows R. Schroeppel, H. Orman, S. O'Mally, "Fast Key Exchange with*
* Elliptic Curve Systems", CRYPTO '95, TR-95-03, Univ. of Arizona, Comp. *
* Science Dept. *
*************************************************************************/
void poly_esum (p1, p2, p3, curv)
POINT *p1, *p2, *p3;
CURVE *curv;
{
INDEX i;
FIELD2N x1, y1, theta, onex, theta2;
ELEMENT check;
/* check if p1 or p2 is point at infinity */
check = 0;
SUMLOOP(i) check |= p1->x.e[i] | p1->y.e[i];
if (!check)
{
copy_point( p2, p3);
return;
}
check = 0;
SUMLOOP(i) check |= p2->x.e[i] | p2->y.e[i];
if (!check)
{
copy_point( p1, p3);
return;
}
```

The lines that compute also check to see if the sum is 0. The error check only tells us if the input x field elements are identical. Since we are not doubling, this error check assumes we are adding P to $\tilde{P}$. The result is the point at infinity. If we should be doubling, it is an error at a higher level.

```
/* compute theta = (y_1 + y_2)/(x_1 + x_2) */
null(&x1);
null(&y1);
check = 0;
SUMLOOP(i)
{
x1.e[i] = p1->x.e[i] ^ p2->x.e[i];
```

O□

□̣=*y*1 + *y*2̲ □*x*1 + *x*2̲*x*1 + *x*2


```
y1.e[i] = p1->y.e[i] ^ p2->y.e[i];
check |= x1.e[i];
}
if (!check) /* return point at infinity */
{
null(&p3->x);
null(&p3->y);
return;
```

```
}
```
The error check does prevent us from attempting to invert 0. The value is computed first. The check for the value of curv->form is performed once, and the loop contains only the terms required to create the sum. The result is the value for *x*3 in equation

```
poly_inv( &x1, &onex);
poly_mul( &onex, &y1, &theta); /* compute y_1/x_1 = theta */
poly_mul(&theta, &theta, &theta2); /* then theta^2 */
/* with theta and theta^2, compute x_3 */
if (curv->form)
SUMLOOP (i)
p3->x.e[i] = theta.e[i] ^ theta2.e[i] ^ p1->x.e[i] ^ p2->x.e[i]
^ curv->a2.e[i];
else
SUMLOOP (i)
p3->x.e[i] = theta.e[i] ^ theta2.e[i] ^ p1->x.e[i]
^ p2->x.e[i];
/* next find y_3 */
SUMLOOP (i) x1.e[i] = p1->x.e[i] ^ p3->x.e[i];
poly_mul( &x1, &theta, &theta2);
SUMLOOP (i) p3->y.e[i] = theta2.e[i] ^ p3->x.e[i] ^ p1->y.e[i];
}
```
The last three lines of code implement equation . All field elements are handled as members of the field modulo poly_prime, the irreducible polynomial. The (*x*,*y*) point value is returned in the designated location. OK, now let's code up . This time there is one point of input, one curve input, and one point as output for *P*3 = 2*P*1.

```
/* elliptic curve doubling routine for Schroeppel's algorithm over polymomial
basis. Enter with p1, p3 as source and destination as well as curv
to operate on. Returns p3 = 2*p1.
*/
void poly_edbl (p1, p3, curv)
POINT *p1, *p3;
CURVE *curv;
```

```
{
FIELD2N x1, y1, theta, theta2, t1;
INDEX i;
ELEMENT check;
check = 0;
SUMLOOP (i) check |= p1->x.e[i];
if (!check)
{
null(&p3->x);
null(&p3->y);
return;
}
/* first compute theta = x + y/x */
poly_inv( &p1->x, &x1);
poly_mul( &x1, &p1->y, &y1);
SUMLOOP (i) theta.e[i] = p1->x.e[i] ^ y1.e[i];
```

```
/* next compute x_3 */
poly_mul( &theta, &theta, &theta2);
if(curv->form)
SUMLOOP (i) p3->x.e[i] = theta.e[i] ^ theta2.e[i] ^ curv->a2.e[i];
else
SUMLOOP (i) p3->x.e[i] = theta.e[i] ^ theta2.e[i];
/* and lastly y_3 */
theta.e[NUMWORD] ^= 1; /* theta + 1 */
poly_mul( &theta, &p3->x, &t1);
poly_mul( &p1->x, &p1->x, &x1);
SUMLOOP (i) p3->y.e[i] = x1.e[i] ^ t1.e[i];
}
```

This routine is a straightforward copy of the equations. Adding 1 to the theta value is simple; we just flip the last bit. Again, we check to see if the input attempts to double a zero x value. The routine returns the point at infinity if so. That this is the correct value is obvious from equation (5.18). The point (0,) is on the curve, and, since $\tilde{P} = (x, y + x)$, we have a point that is its own negative. Doubling this point is the same as adding the negative of the same point, which gives us the point at infinity. Finally, we need to be able to subtract two points. Just as we can negate and add with integers, this routine negates the second point, as in equation (5.20), and adds it to the first point. Here's the code.

```
/* subtract two points on a curve. just negates p2 and does a sum.
Returns p3 = p1 - p2 over curv.
*/
void poly_esub (p1, p2, p3, curv)
POINT *p1, *p2, *p3;
CURVE *curv;
{
POINT negp;
INDEX i;
copy ( &p2->x, &negp.x);
null (&negp.y);
SUMLOOP(i) negp.y.e[i] = p2->x.e[i] ^ p2->y.e[i];
poly_esum (p1, &negp, p3, curv);
}
```

Optimal normal basis elliptic curve subroutines

```
void esum (p1, p2, p3, curv)
POINT *p1, *p2, *p3;
CURVE *curv;
{
INDEX i;
FIELD2N x1, y1, theta, onex, theta2;
/* compute theta = (y_1 + y_2)/(x_1 + x_2) */
null(&x1);
null(&y1);
```

```
SUMLOOP(i)
{
x1.e[i] = p1->x.e[i] ^ p2->x.e[i];
y1.e[i] = p1->y.e[i] ^ p2->y.e[i];
}
opt_inv( &x1, &onex);
opt_mul( &onex, &y1, &theta);
copy( &theta, &theta2);
rot_left(&theta2);
/* with theta and theta^2, compute x_3 */
if (curv->form)
SUMLOOP (i)
p3->x.e[i] = theta.e[i] ^ theta2.e[i] ^ p1->x.e[i] ^ p2->x.e[i]
```

```
^ curv->a2.e[i];
else
SUMLOOP (i)
p3->x.e[i] = theta.e[i] ^ theta2.e[i] ^ p1->x.e[i] ^ p2->x.e[i];
/* next find y_3 */
SUMLOOP (i) x1.e[i] = p1->x.e[i] ^ p3->x.e[i];
opt_mul( &x1, &theta, &theta2);
SUMLOOP (i) p3->y.e[i] = theta2.e[i] ^ p3->x.e[i] ^ p1->y.e[i];
}
```
The only real difference here is the squaring of theta. This is a simple rotation. The opt_inv routine given in chapter 4 is a bit slow, but the call to it won't change if we replace it with something faster. Other than the replacement calls to opt_* routines instead of poly_* routines, the code is almost the same as previously described. Here is the code to double a point using optimal normal basis mathematics.
```
/* elliptic curve doubling routine for Schroeppel's algorithm over normal basis. Enter with p1, p3
as source and destination as well as curve to operate on. Returns p3 = 2*p1.
*/
void edbl (p1, p3, curv)
POINT *p1, *p3;
CURVE *curv;
{
FIELD2N x1, y1, theta, theta2, t1;
INDEX i;
/* first compute theta = x + y/x */
opt_inv( &p1->x, &x1);
opt_mul( &x1, &p1->y, &y1);
SUMLOOP (i) theta.e[i] = p1->x.e[i] ^ y1.e[i];
/* next compute x_3 */
copy( &theta, &theta2);
rot_left(&theta2);
if(curv->form)
SUMLOOP (i) p3->x.e[i] = theta.e[i] ^ theta2.e[i] ^ curv-
>a2.e[i];
else
SUMLOOP (i) p3->x.e[i] = theta.e[i] ^ theta2.e[i];
/* and lastly y_3 */
one( &y1);
```

```
SUMLOOP (i) y1.e[i] ^= theta.e[i];
opt_mul( &y1, &p3->x, &t1);
copy( &p1->x, &x1);
rot_left( &x1);
```

```
SUMLOOP (i) p3->y.e[i] = x1.e[i] ^ t1.e[i];
}
```
The major differences here are, again, that the squaring operation is only a rotation and the way we add 1. The subroutine one is called to create the constant first and then it is added to theta. This takes a few more steps, but the squaring operation makes up for it.

Finally, we subtract two points. This is the same as the polynomial math routine, but we have to call the optimal normal basis algorithms to get the right answers.

```
void esub (p1, p2, p3, curv)
POINT *p1, *p2, *p3;
CURVE *curv;
{
POINT negp;
INDEX i;
copy ( &p2->x, &negp.x);
null (&negp.y);
SUMLOOP(i) negp.y.e[i] = p2->x.e[i] ^ p2->y.e[i];
esum (p1, &negp, p3, curv);
}
```

```
/* Reversing a String */
#include<stdio.h>
#include<string.h>
main()
{
 char str[50],revstr[50];
 int i=0,j=0;
 printf("Enter the string to be reversed : ");
 scanf("%s",str);
 for(i=strlen(str)-1;i>=0;i--)
 {
 revstr[j]=str[i];
 j++;
 }
 revstr[j]='\0';
 printf("Input String : %s",str);
 printf("\nOutput String : %s",revstr);
 getch();
}
```

```
/*CAESER CIPHER*/
#include <stdio.h>
#include <string.h> //for strlen();
#include <ctype.h> //for toupper();

char *abc = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; //Alphabet defined as global variable

int subst(char x); //Function defined

int main(void)
{
	char key;
	int key2;

	puts("Enter a 1 character long text");
	scanf("%c", &key);

	key = toupper(key); //converts all letters to uppercase
	key2 = subst(key); //calling substitution function

	printf("%d\n", key2);

	return 0;
}

int subst(char x)
{
	int out, i, len;
	len = strlen(abc); //do len be abc string's length

	for((i = 0); (i = len); i++)
	{
		if(abc[i] == x)
		{
			out = i;
		}
	/*This should search each abc's characters and compare it with x (which is the key variable) and, if they're the same, the function will return the corresponding character's place in the array as result (so, A = 0, B = 1, etc.)*/
	}

	return out;
}
```

001 //This program will decipher encrypted .txt files by counting the amount of letters and calculating the shift from there.

```
002
003
004 #include <iostream>
005 #include <fstream>
006 #include <string>
007 #include <iomanip>
008
009 using namespace std;
010
011 /*This function will help me count the letters from 'a' to 'z'
012 in the file. Hopefully it will help me find the most frequently
013 used letter */
014 void characterCount(char ch, int list[]);
015
016 //This function will help me calculate the shift
017 void calcShift(int& shift, int list[]);
018
019 //This function will write the output to a file.
020 void writeOutput(ifstream &in, ofstream &out, int shift);
021
022 int main ()
023 {
024    //Declaring variables thus far
025    int shift = 0;
026    char ch;
027    string inputFileName;
028    ifstream inputFile;
029
030    //User inputs the file name
031    cout << "Input file name:
";
032    getline(cin, inputFileName);
033
034    // Open the input file.
035    inputFile.open(inputFileName.c_str());      // Need .c_str() to convert a C++ string to a C-
style string
036
037    // Check the file opened successfully.
038    if ( ! inputFile.is_open())
```

```
039     {
040         cout << "Unable to open the file." << endl;
041         exit(1);
042     }
043     //Setting list contents equal to 0
044     int list[25] = {0};
045
046     //Adds up the total number of times each letter appeared in the file.
047     while (inputFile.peek() != EOF)
048     {
049         inputFile.get(ch);
050         characterCount(ch, list);
051     }
052
053     //Rewind it
054     inputFile.clear();
055     inputFile.seekg(0);
056
057     //Calculate the shift based on the most characters counted
058     calcShift (shift, list);
059
060     //Just a test to see if this works...
061     cout << "The shift for this text is " << shift << endl;
062
063     return 0;
064 }
065
066
067
068 /*This function will help me count the letters from 'a' to 'z'
069 in the file. Hopefully it will help me find the most frequently
070 used letter */
071    void characterCount(char ch, int list[])
072 {
073     //Will go through every character and add to the given index if the letter appears
074     switch(ch)
075     {
076         case 'a':
077         case 'A':
```

```
078             list[0]++;
079             break;
080         case 'b':
081         case 'B':
082             list[1]++;
083             break;
084         case 'c':
085         case 'C':
086             list[2]++;
087             break;
088         case 'd':
089         case 'D':
090             list[3]++;
091             break;
092         case 'e':
093         case 'E':
094             list[4]++;
095             break;
096         case 'f':
097         case 'F':
098             list[5]++;
099             break;
100         case 'g':
101         case 'G':
102             list[6]++;
103             break;
104         case 'h':
105         case 'H':
106             list[7]++;
107             break;
108         case 'i':
109         case 'I':
110             list[8]++;
111             break;
112         case 'j':
113         case 'J':
114             list[9]++;
115             break;
116         case 'k':
```

```
117         case 'K':
118             list[10]++;
119             break;
120         case 'l':
121         case 'L':
122             list[11]++;
123             break;
124         case
'm':
125         case 'M':
126             list[12]++;
127             break;
128         case 'n':
129         case 'N':
130             list[13]++;
131             break;
132         case 'o':
133         case 'O':
134             list[14]++;
135             break;
136         case 'p':
137         case 'P':
138             list[15]++;
139             break;
140         case 'q':
141         case 'Q':
142             list[16]++;
143             break;
144         case 'r':
145         case 'R':
146             list[17]++;
147             break;
148         case 's':
149         case 'S':
150             list[18]++;
151             break;
152         case 't':
153         case 'T':
154             list[19]++;
```

```
155             break;
156         case 'u':
157         case 'U':
158             list[20]++;
159             break;
160         case 'v':
161         case 'V':
162             list[21]++;
163             break;
164         case 'w':
165         case 'W':
166             list[22]++;
167             break;
168         case 'x':
169         case 'X':
170             list[23]++;
171             break;
172         case 'y':
173         case 'Y':
174             list[24]++;
175             break;
176         case 'z':
177         case 'Z':
178             list[25]++;
179             break;
180         default:
181             break;
182     }
183 }
184
185 //This function will help me calculate the shift
186 void calcShift(int& shift, int list[])
187 {
188     int e = list[0];   //Starting the list at 0 of course to find 'e' (using e because it's the letter that's supposed to be most frequent
189     char letter = ' '; //Making sure that letter is emtpy
190
191     //Determining Which number appeared the most with a for loop
```

```
192     for (int i = 0; i < 26; i++)
193     {
194        if (list[i] < list[i+1])
195        {
196           e = list[i+1];
197        }
198        else if (list[i] > list[i+1])
199        {
200           e = list[i];
201        }
202     }
203
204     //Switching to e to find out which letter showed up the most
205     switch (e)
206     {
207        case 0:
208           letter = 'a';
209           break;
210        case 1:
211           letter = 'b';
212           break;
213        case 2:
214           letter = 'c';
215           break;
216        case 3:
217           letter = 'd';
218           break;
219        case 4:
220           letter = 'e';
221           break;
222        case 5:
223           letter = 'f';
224           break;
225        case 6:
226           letter = 'g';
227           break;
228        case 7:
229           letter = 'h';
230           break;
```

```
231         case 8:
232             letter = 'i';
233             break;
234         case 9:
235             letter = 'j';
236             break;
237         case 10:
238             letter = 'k';
239             break;
240         case 11:
241             letter = 'l';
242             break;
243         case 12:
244             letter = 'm';
245             break;
246         case 13:
247             letter = 'n';
248             break;
249         case 14:
250             letter = 'o';
251             break;
252         case 15:
253             letter = 'p';
254             break;
255         case 16:
256             letter = 'q';
257             break;
258         case 17:
259             letter = 'r';
260             break;
261         case 18:
262             letter = 's';
263             break;
264         case 19:
265             letter = 't';
```

```
266             break;
267         case 20:
268             letter = 'u';
269             break;
270         case 21:
271             letter = 'v';
272             break;
273         case 22:
274             letter = 'w';
275             break;
276         case 23:
277             letter = 'x';
278             break;
279         case 24:
280             letter = 'y';
281             break;
282         case 25:
283             letter = 'z';
284             break;
285     }
286     //This will show the letter that appeared most as a int, which then will be worked into a loop to determine the offset
287     shift = static_cast<int>(letter);
288 }
```

```
/* PLAYFAIR CIPHER*/
public class playfair
{
  private String KeyWord=new String();
  private String Key=new String();
  private char matrix_arr[][]= new char[5][5];

  public void setKey(String k)
  {
    KeyWord=k;
  }

  public void KeyGen()
  {
    boolean flag=true;
    char current;

    Key=KeyWord;

    for ( int i=0 ; i<26 ; i++)
    {
      current=(char)(i+97);

      if(current=='j')
        continue;

      for(int j=0 ; j< KeyWord.length() ; j++ )
      {
        if (current == KeyWord.charAt(j))
        {
          flag=false;
          break;
        }
      }

      if(flag)
        Key=Key+current;

      flag=true;
    }

    System.out.println(Key);
    matrix ();

  }

  private void matrix ()
  {
    int counter=0;

    for (int i=0 ; i<5 ;i++)
```

```
    {
      for (int j=0 ; j<5 ; j++)
      {
        matrix_arr[i][j]=Key.charAt(counter);
        System.out.printf("%s ",matrix_arr[i][j]);

        counter++;
      }

      System.out.println("\n");
    }
  }

  private String [] Divid2Pairs (String Original)
  {
    int size= Original.length();

    if(size%2!=0)
      size++;

    String x[]= new String[size/2];

    int counter=0;

    for ( int i=0 ; i<size/2 ;i++)
    {
      x[i]=Original.substring(counter, counter+2);
      System.out.println(x[i]);
      counter=counter+2;
    }

    return x;
  }

  public int[]  GetDiminsions(char letter)
  {
    int []key=new int[2];

    if ( letter == 'j')
      letter='i';

    for (int i=0 ; i<5 ;i++)
    {
      for (int j=0 ; j<5 ; j++)
      {
        if(matrix_arr[i][j] == letter)
        {
          key[0]=i;
          key[1]=j;
          break;
        }
```

```
    }

  }

  return key;
}

public String Encript(String Source)
{
  String src_arr[]=Divid2Pairs(Source);

  String Code=new String();

  char one;
  char two;

  int part1[]=new int[2];
  int part2[]=new int[2];

  //start on pair by pair
  for (int i=0 ; i< src_arr.length ;i++ )
  {
    one = src_arr[i].charAt(0);//get first char
    two = src_arr[i].charAt(1);//get second char

    part1 = GetDiminsions(one);//get position of the first char
    part2 = GetDiminsions(two);//get position of the second char

    //check for specail casese
    if(part1[0]==part2[0])//same row
    {
      if (part1[1]<4)
        part1[1]++;

      else
        part1[1]=0;

      if(part2[1]<4)
        part2[1]++;

      else
        part2[1]=0;

    }

    else if (part1[1]==part2[1]) //same column
    {
      if (part1[0]<4)
        part1[0]++;

      else
```

```
            part1[0]=0;

          if(part2[0]<4)
            part2[0]++;

          else
            part2[0]=0;
        }

        else
        {
          int temp=part1[1];
          part1[1]=part2[1];
          part2[1]=temp;
        }


        Code= Code + matrix_arr[part1[0]][part1[1]] + matrix_arr[part2[0]][part2[1]];
      }
      System.out.println(Code);
      return Code;
    }

    public String Decript (String Code)
    {
      String Original=new String();

      String src_arr[]=Divid2Pairs(Code);

      char one;
      char two;

      int part1[]=new int[2];
      int part2[]=new int[2];


      //start on pair by pair
      for (int i=0 ; i<= src_arr.length ;i++ )
      {
        one = src_arr[i].charAt(0);//get first char
        two = src_arr[i].charAt(1);//get second char

        part1 = GetDiminsions(one);//get position of the first char
        part2 = GetDiminsions(two);//get position of the second char

        //check for specail casese
        if(part1[0]==part2[0])//same row
        {
          if (part1[1]>0)
            part1[1]--;
```

```
            else
              part1[1]=4;

            if(part2[1]>0)
              part2[1]--;

            else
              part2[1]=4;

          }

          else if (part1[1]==part2[1]) //same column
          {
            if (part1[0]>0)
              part1[0]--;

            else
              part1[0]=4;

            if(part2[0]>0)
              part2[0]--;

            else
              part2[0]=4;
          }

          else
          {
            int temp=part1[1];
            part1[1]=part2[1];
            part2[1]=temp;
          }


          Original =Original + matrix_arr[part1[0]][part1[1]] + matrix_arr[part2[0]][part2[1]];
        }

        System.out.println(Original);
        return Original;
      }
    }
```

```
1.  /*
2.    This is a program for Encryption and Decryption
3.    This program uses the Simple Data Encryption Standard (SDES) Algorithm.
4.    This Algo takes 8-bits of plaintext at a time and produces 8-bits of ciphertext.
5.    It uses 10-bits of key for Encryption and Decryption.
6.
7.    */
8.
9.  #include<iostream.h>
10. #include<stdio.h>
11. #include<conio.h>
12. #include<string.h>
13. #include<stdlib.h>
14. #include<assert.h>
15. void mainmenu(int *);
16. void menuEn();
17. void menuDe();
18. int DoEnDe(int);
19.
20. class SDES
21. {
22. private:
23.    char KEY[11],K1[9],K2[9],IPOutput[9],InvIPOutput[9];
24.    char F1Output[9],F2Output[9];
25.    char INPUT_BIT[9],OUTPUT_BIT[9];
26.
27. public:
28.    unsigned char INPUT,OUTPUT;
29.
30.    SDES(char *key);
31.    ~SDES();
32.    void GenerateKeys();
33.    char *Left_Shift(char *,int );
34.    void conv_to_bits(unsigned char );
35.    void IP(char *);
36.    void InvIP(char *);
37.    void DES_Encryption(unsigned char );
38.    void DES_Decryption(unsigned char );
39.    void Function_F(char *,char *,int );
40.    char *EX_OR(char *,int );
41.    char *SBOX0(char *);
42.    char *SBOX1(char *);
43.    void SDES::GetChar();
44. };
45. SDES::SDES(char *key)  //Initializes the object with 10-bits key
46. {
47.    int i;
48.    if (strlen(key)!=10)  //Checks for valid length key
49.    {
50.       printf("\nInValid Key-Length %s %d",key,strlen(key));
51.       getch();
```

```
52.         exit(1);
53.      }
54.      for (i=0;i<10;i++)  //Assigning the key privatly
55.      {
56.         KEY[i]=key[i];
57.      }
58.      KEY[10]='\0';
59.      GenerateKeys(); //Key Genaration Starts. Output: (K1/K2)
60.
61.   }
62.
63.   void SDES::GenerateKeys()
64.   {
65.      int P10[10]={3,5,2,7,4,10,1,9,8,6};       //P10 permutation-array
66.      char P10_OP[11];                          //Output of P10 is to be stored here
67.      int P8[8]={6,3,7,4,8,5,10,9};                        //P8 permutation-array
68.      char *P10LEFT,*pl,*pl1,*P10RIGHT,*pr,*pr1,*plpr;
69.      int i;
70.
71.      /*P10 operation is done on main key*/
72.      for (i=0;i<10;i++)
73.         P10_OP[i]=KEY[P10[i]-1];
74.
75.      P10_OP[10]='\0';
76.
77.      /*Dividing 10-bit output of P10 operation into
78.        two parts*/
79.      for (i=0;i<5;i++)
80.      {
81.         P10LEFT[i]=P10_OP[i];
82.         P10RIGHT[i]=P10_OP[i+5];
83.      }
84.      P10LEFT[5]='\0';
85.      P10RIGHT[5]='\0';
86.
87.      pl=new char[6];
88.      pr=new char[6];
89.
90.      /*Perform Left-Circular shift by 1 bit on the
91.        two parts of P10 output*/
92.      pl=Left_Shift(P10LEFT,1);
93.      pr=Left_Shift(P10RIGHT,1);
94.
95.      /*Combine the above two parts after
96.        the left-cicular operation into 'plpr' string*/
97.      for (i=0;i<5;i++)
98.      {
99.         plpr[i]=pl[i];
100.        plpr[i+5]=pr[i];
101.     }
102.     plpr[10]='\0';
```

```
103.
104.     /*Performing P8 Operation on plpr and assigning to K1*/
105.     for (i=0;i<8;i++)
106.        K1[i]=plpr[P8[i]-1];
107.
108.     K1[8]='\0'; //This is our first sub-key K1
109.
110.     /*Again performing Left-Circular-Shift(LCS) by 2 bits on
111.       the output of previous Left-Cicular-Shift(LCS)*/
112.     pl1=Left_Shift(pl,2);
113.     pr1=Left_Shift(pr,2);
114.
115.     /*Combining the output of above LCS2 into 1 string*/
116.     for (i=0;i<5;i++)
117.     {
118.        plpr[i]=pl1[i];
119.        plpr[i+5]=pr1[i];
120.     }
121.     plpr[10]='\0';
122.
123.     /*Again performing P8 operation on the above combined
124.       string*/
125.     for (i=0;i<8;i++)
126.     {
127.        K2[i]=plpr[P8[i]-1];
128.     }
129.     K2[8]='\0'; //This is our second sub-key K2
130.
131.  }
132.
133.  /*Method to perform Left-Circular-Shift on bit-string*/
134.  char *SDES::Left_Shift(char *bs,int n)
135.  {
136.     int length=strlen(bs);
137.     char *char_ptr,firstbit,*str;
138.
139.     char_ptr = new char[length +1];
140.     str=new char[length+1];
141.     char_ptr=bs;
142.
143.     int i,j;
144.     for (j=0;j<n;j++)
145.     {
146.        firstbit=char_ptr[0];
147.
148.        for (i=0;i<length-1;i++)
149.        {
150.           str[i]=char_ptr[i+1];
151.        }
152.        str[length-1]=firstbit;
153.        char_ptr[length]='\0';
```

```
154.          char_ptr=str;
155.      }
156.      char_ptr[length]='\0';
157.      return(str);
158.  }
159.
160.  /*Method to convert unsigned char to bit-string
161.    For Ex. 1="00000001"*/
162.  void SDES::conv_to_bits(unsigned char ch)
163.  {
164.      int i,bit;
165.      INPUT_BIT[8]='\0';
166.      for (i=7;i>=0;i--)
167.      {
168.          bit=ch%2;
169.          ch=ch/2;
170.
171.          if (bit!=0)
172.              INPUT_BIT[i]='1';
173.          else
174.              INPUT_BIT[i]='0';
175.      }
176.
177.  }
178.
179.  /*Method to perform Initial-Permutation*/
180.  void SDES::IP(char *input)
181.  {
182.      int IPArray[8]={2,6,3,1,4,8,5,7};
183.      int i;
184.      IPOutput[8]='\0';
185.      for (i=0;i<8;i++)
186.      {
187.          IPOutput[i]=input[IPArray[i]-1];
188.      }
189.  }
190.
191.  /*Method to perform Inverse of Initial-Permutation*/
192.  void SDES::InvIP(char *input)
193.  {
194.      int InvIPArray[8]={4,1,3,5,7,2,8,6};
195.      int i;
196.      InvIPOutput[8]='\0';
197.      for (i=0;i<8;i++)
198.      {
199.          InvIPOutput[i]=input[InvIPArray[i]-1];
200.      }
201.  }
202.
203.  /*Method to perform SDES-Encryption on 8-bit 'input'*/
204.  void SDES::DES_Encryption(unsigned char input)
```

```
{
   char LIP[5],RIP[5],L1[5],R1[5];
   int i;

   INPUT=input;
   conv_to_bits(INPUT);  //Converts the input to bit-string
   IP(INPUT_BIT);        //Initial-Permutation

   //gotoxy(1,1);
   printf("\nEncrpyting.....");

   /*Dividing the output of IP into 2 parts*/
   for (i=0;i<4;i++)
   {
      LIP[i]=IPOutput[i];
      RIP[i]=IPOutput[i+4];
   }
   LIP[4]='\0';
   RIP[4]='\0';

   /*Sending the above divided parts to Function_F and  sub-key K1*/
   Function_F(LIP,RIP,1);

   /*Dividing the output of the Function_F into 2 parts*/
   for (i=0;i<4;i++)
   {
      L1[i]=F1Output[i];
      R1[i]=F1Output[4+i];
   }
   L1[4]='\0';
   R1[4]='\0';

   /*This time the string-parameters swaped and uses sub-key K2*/
   Function_F(R1,L1,2);

   /*Performing the Inverse IP on the output of the Funtion_F*/
   InvIP(F1Output); //The output of the function will give us
   //Cipher-string

   /*Cipher string is converted back to unsigned char and stored
     in private-variable OUTPUT of this class*/
   GetChar();
}


/*Decryption is just inverse of Encryption
  Here IP, InvIP, E/P, SBOX1 and SBOX2 are same
  But Function_F first operats on sub-key K2 and
  then on sub-key K1*/
void SDES::DES_Decryption(unsigned char input)
{
```

```
256.    char LIP[5],RIP[5],L1[5],R1[5];
257.    int i;
258.
259.    INPUT=input;
260.    conv_to_bits(INPUT);
261.    IP(INPUT_BIT);       //Initial-Permutation
262.
263.    //gotoxy(1,1);
264.    printf("\nDecrpyting.....");
265.
266.    for (i=0;i<4;i++)
267.    {
268.       LIP[i]=IPOutput[i];
269.       RIP[i]=IPOutput[i+4];
270.    }
271.    LIP[4]='\0';
272.    RIP[4]='\0';
273.
274.    Function_F(LIP,RIP,2);
275.
276.    for (i=0;i<4;i++)
277.    {
278.       L1[i]=F1Output[i];
279.       R1[i]=F1Output[4+i];
280.    }
281.    L1[4]='\0';
282.    R1[4]='\0';
283.
284.    Function_F(R1,L1,1);
285.    InvIP(F1Output);
286.    GetChar();
287. }
288.
289. void SDES::Function_F(char *linput,char *rinput,int key)
290. {
291.    int E_P[8]={4,1,2,3,2,3,4,1}; //E/P Operation-Array
292.    int P4[4]={2,4,3,1};         //P4 Operation-Array
293.    int i;
294.    char E_POutput[9],*EXOR_Output,*LEXOR,*REXOR;
295.    char *SBOX0_Output,*SBOX1_Output;
296.    char SBOX_Output[5];
297.    char P4_Output[5];
298.    char fk_Output[5];
299.    char Main_Output[9];
300.
301.    /*E/P Operaion is performed here*/
302.    for (i=0;i<8;i++)
303.    {
304.       E_POutput[i]=rinput[E_P[i]-1];
305.    }
306.    E_POutput[8]='\0';
```

```
307.
308.    /*Bitwise-EXOR is done on E/P Output and sub-key(K1/K2)*/
309.    EXOR_Output=EX_OR(E_POutput,key);
310.
311.    /*Divide the output of Exor in 2 parts*/
312.    LEXOR=new char[strlen(EXOR_Output)/2+1];
313.    REXOR=new char[strlen(EXOR_Output)/2+1];
314.
315.    for (i=0;i<strlen(EXOR_Output)/2;i++)
316.    {
317.       LEXOR[i]=EXOR_Output[i];
318.       REXOR[i]=EXOR_Output[i+4];
319.    }
320.    LEXOR[4]=REXOR[4]='\0';
321.
322.
323.    /*Peforming SBOX0 Operation on left 4 bits*/
324.    SBOX0_Output=SBOX0(LEXOR);
325.
326.    /*Peforming SBOX1 Operation on right 4 bits*/
327.    SBOX1_Output=SBOX1(REXOR);
328.
329.    /*Combining the 2-bits output of both SBOXES in one string*/
330.    for (i=0;i<2;i++)
331.    {
332.       SBOX_Output[i]=SBOX0_Output[i];
333.       SBOX_Output[i+2]=SBOX1_Output[i];
334.    }
335.    SBOX_Output[4]='\0';
336.
337.    /*Performing the P4 operation on SBOX output*/
338.    for (i=0;i<4;i++)
339.    {
340.       P4_Output[i]=SBOX_Output[P4[i]-1];
341.    }
342.    P4_Output[4]='\0';
343.
344.    /*Performing the EXOR operation on 4-bits P4-output
345.      and 4-bits Leftinput of Funtion_F*/
346.    for (i=0;i<4;i++)
347.    {
348.       if (P4_Output[i]==linput[i])
349.          fk_Output[i]='0';
350.       else
351.          fk_Output[i]='1';
352.    }
353.    fk_Output[4]='\0';
354.
355.    /*Cancating the 4-bits output of above EXOR-operation
356.      and 4-bits Right-input of Function_F*/
357.    for (i=0;i<4;i++)
```

```
    {
       Main_Output[i]=fk_Output[i];
       Main_Output[i+4]=rinput[i];
    }
    Main_Output[8]='\0';
    /*Assigning this Cucaneted string to Private variable 'F1Output'*/
    strcpy(F1Output,Main_Output);
}

/*This method EXORS the output ofE/P and sub-keys
  depending on the parameter k.
  k=1:subkey K1  k=2:subkey K2*/
char *SDES::EX_OR(char *ep,int k)
{
    char *output,*key;
    int i,klen;

    output=new char[strlen(ep)+1];
    key=new char[strlen(K1)+1];
    if (k==1)
    {
       strcpy(key,K1);
    } else
    {
       if (k==2)
       {
          strcpy(key,K2);
       } else
       {
          printf("\n\nWrong Choice in the key parameter(1/2)");
          getch();
          exit(1);
       }
    }
    klen=strlen(K1);
    if (strlen(ep)!=klen)
    {
       printf("\ninput=%d is not equal to K=%d",strlen(ep),klen);
       printf("\n\nError in the Output of E/P (Length)..Press any key");
       getch();
       exit(1);
    }
    for (i=0;i<strlen(ep);i++)
    {
       if (ep[i]==key[i])
          output[i]='0';
       else
          output[i]='1';
    }
    output[strlen(ep)]='\0';
    return(output);
```

```
409.  }
410.
411.  /*SBOX0 Operation is defined here*/
412.  char *SDES::SBOX0(char *l)
413.  {
414.     int S0[4][4]={1,0,3,2,  //S0 Matrix
415.        3,2,1,0,
416.        0,2,1,3,
417.        3,1,3,2
418.     };
419.
420.     char *bits[]={"00","01","10","11"};
421.     char lrow[3],lcol[3];
422.     char *SO;
423.     int i,lr,lc,b;
424.
425.     SO=new char[3];
426.
427.     lrow[0]=l[0];
428.     lrow[1]=l[3];
429.     lcol[0]=l[1];
430.     lcol[1]=l[2];
431.
432.     lrow[2]='\0';
433.     lcol[2]='\0';
434.
435.
436.     for (i=0;i<4;i++)
437.     {
438.        if (strcmp(lrow,bits[i])==0)
439.           lr=i;
440.        if (strcmp(lcol,bits[i])==0)
441.           lc=i;
442.     }
443.     b=S0[lr][lc];
444.     for (i=0;i<3;i++)
445.        SO[i]=bits[b][i];
446.     SO[3]='\0';
447.     return(SO);
448.  }
449.  /*SBOX1 Operation is defined here*/
450.  char *SDES::SBOX1(char *l)
451.  {
452.     int S0[4][4]={0,1,2,3,   //S1 Matrix
453.        2,0,1,3,
454.        3,0,1,0,
455.        2,1,0,3
456.     };
457.
458.     char *bits[]={"00","01","10","11"};
459.     char lrow[3],lcol[3];
```

```
460.     char *SO;
461.     int i,lr,lc,b;
462.
463.     SO=new char[3];
464.
465.     lrow[0]=l[0];
466.     lrow[1]=l[3];
467.     lcol[0]=l[1];
468.     lcol[1]=l[2];
469.
470.     lrow[2]='\0';
471.     lcol[2]='\0';
472.
473.
474.     for (i=0;i<4;i++)
475.     {
476.        if (strcmp(lrow,bits[i])==0)
477.           lr=i;
478.        if (strcmp(lcol,bits[i])==0)
479.           lc=i;
480.     }
481.     b=S0[lr][lc];
482.     for (i=0;i<3;i++)
483.        SO[i]=bits[b][i];
484.
485.     SO[3]='\0';
486.     return(SO);
487.  }
488.
489.  /*Method to get back unsigned char from bit-string*/
490.  void SDES::GetChar()
491.  {
492.     int i,j,in;
493.     unsigned char ch=0;
494.     char *bs;
495.     bs=new char[9];
496.     bs=InvIPOutput;
497.     if (strlen(bs)>8)
498.     {
499.        printf("\nWRONG LENGTH STRING");
500.        exit(0);
501.     }
502.     for (i=0;i<8;i++)
503.     {
504.        if (bs[i]=='1')
505.        {
506.           in=1;
507.           for (j=1;j<8-i;j++)
508.           {
509.              in=in*2;
510.           }
```

```
            ch=ch+in;
        }
    }
    OUTPUT=ch;
}

/*Destructor*/
SDES::~SDES()
{
}


char *sfname,*tfname;
char *key;//="1010000010";
void main(void)
{
    //clrscr();
    unsigned char ch,ch1;
    int i,n=10,choice;

    while (1)
    {
        key = new char[11];
        sfname = new char[20];
        tfname = new char[20];
        mainmenu(&choice);
        fflush(stdin);
        switch (choice)
        {
        case 1:
            menuEn();
            DoEnDe(choice);
            break;
        case 2:
            menuDe();
            DoEnDe(choice);
            break;
        case 3:
            exit(0);
        default:
            printf("\nWrong Choice Enter again\nPress any key to return to Main Menu..");
            getch();
            break;
        }
    }
}

void mainmenu(int *c)
{
    //clrscr();
    printf("\nWhat do you want to do..");
```

```
562.     printf("\n1. Encryption");
563.     printf("\n2. Decryption");
564.     printf("\n3. Exit");
565.     printf("\n\nEnter the choice? ");
566.     scanf("%d",c);
567.  }
568.  void menuEn()
569.  {
570.     //clrscr();
571.     sfname=new char[20];
572.     tfname=new char[20];
573.     key=new char[11];
574.     printf("\nEncryption Menu\n\n");
575.     printf("\nEnter the filename to be Encrypted: ");
576.     gets(sfname);
577.     printf("\nEnter the Target file name: ");
578.     gets(tfname);
579.     printf("\nEnter the 10-bits KEY: ");
580.     gets(key);
581.     printf("\n\nNotedown this key, as same key is used for Decryption");
582.     //getch();
583.  }
584.
585.  void menuDe()
586.  {
587.     //clrscr();
588.     sfname=new char[20];
589.     tfname=new char[20];
590.     key=new char[11];
591.     printf("\nDecryption Menu\n\n");
592.     printf("\nEnter the filename to be Decrypted: ");
593.     gets(sfname);
594.     printf("\nEnter the Target file name: ");
595.     gets(tfname);
596.     printf("\nEnter the 10-bits KEY: ");
597.     gets(key);
598.  }
599.
600.  int DoEnDe(int c)
601.  {
602.     SDES S(key);
603.     int i,n;
604.     n=10; //Number of Rounds
605.     unsigned char ch;
606.     FILE *fp;
607.     FILE *ft;
608.     fp=fopen(tfname,"w");
609.     ft=fopen(sfname,"r");
610.     if (fp==NULL)
611.     {
612.        printf("\nTarget File not opened SORRY");
```

```
613.          getch();
614.          fclose(fp);
615.          return(0);
616.       }
617.       if (ft==NULL)
618.       {
619.          printf("\nSource File not opened SORRY");
620.          getch();
621.          fclose(ft);
622.          return(0);
623.       }
624.       while (fread(&ch,1,1,ft)==1)
625.       {
626.          S.OUTPUT=ch;
627.
628.          for (i=0;i<n;i++)
629.          {
630.             if (c==1)
631.                S.DES_Encryption(S.OUTPUT);
632.             if (c==2)
633.                S.DES_Decryption(S.OUTPUT);
634.          }
635.          fwrite(&S.OUTPUT,1,1,fp);
636.       }
637.       printf("\nCompleted!!!!!");
638.       getch();
639.       fclose(fp);
640.       fclose(ft);
641.       return(1);
642.    }
```

```
//a program for railfence algorithem
#include<stdio.h>
#include<conio.h>
#define row 15
#define column 15
FILE *f1,*f2,*f3;
void main()
{
int n,cnt=0,col;
char ch;
void railenc(int,int);
clrscr();
printf("Enter the value of n:");
scanf("%d",&n);
f1=fopen("plain.txt","r");
while((ch=getc(f1))!=EOF)
{
cnt++;
}
printf("%d",cnt);
fclose(f1);
col=cnt/n;
cnt=cnt%n;
if(cnt==0)
{
col=col+n-1;
}
else
{
col++;
col=col+n-1;
}
railenc(n,col);
getch();
}
void railenc(int r,int c)
{
char arr[row][column];
char ch;
int i=0,j=0,k=0,col;
col=c;
//encryption
f1=fopen("plain.txt","r");
while((ch=getc(f1))!=EOF)
{
if(k<c)
{
if(i<r)
{
arr[i][j]=ch;
i++;
```

```
j++;
}
else
{
k++;
j=k;
i=0;
arr[i][j]=ch;
i++;
j++;
}
}
}
ch='$';
while(i<r)
{
arr[i][j]=ch;
i++;
j++;
}
f2=fopen("enc.txt","w");
c=c-r+1;
for(i=0;i<r;i++)
{
for(j=i;j<c;j++)
{
putc(arr[i][j],f2);
}
c++;
}
fclose(f2);
//decryption
i=j=0;
c=col;
c=c-r+1;
f2=fopen("enc.txt","r");
while((ch=getc(f2))!=EOF)
{
if(i<r)
{
if(j<c)
{
arr[i][j]=ch;
j++;
}
else
{
c++;
i++;
j=i;
```

```
arr[i][j]=ch;
j++;

}
}
}
fclose(f2);

//write a plain text into dec.txt file
i=j=k=0;
printf("\ncolumn=%d",c);
f3=fopen("dec.txt","w");
do
{
if(k<c)
{
if(i<r)
{
if(arr[i][j]=='$')
{
goto stop;
}
else
{
putc(arr[i][j],f3);
i++;
j++;
}
}
else
{
k++;
j=k;
i=0;
if(arr[i][j]=='$')
{
goto stop;
}
else
{
putc(arr[i][j],f3);
i++;
j++;
}
}
}}while(arr[i][j]!='$');
stop:
fclose(f3);

}
```