G#20 implementation of Wiener's attack on RSA

Step-1 START

Step-2 Declare all variables

Step-3 Read value of **RSA** public parameters **[N,e]**

Step-4 check inputs are valid or not

If valid then

Goto next step

Else

Goto previous step

Step-5 call to **continued_fractions(e,N)** calculation function

Step-6 Display continued fractions of **[e / N]**;

Step-7 call to **n = convergent_function(e,N)** calculation function

Step-8 copy all elements into of K and d

While **jj != n** then

{

K[jj]=h[jj];

d[jj]=k[jj];

Display convergents of (K / d);

}

Step-9 **again:** //label used for jumping purpose

Call to final_phi = **check_phi_int(e)** function

If **(final_phi==-1 || final_phi == 2)** **then**

{ Display **"**integer phi is not available for real integer root calculation**";**

exit(1);}

Else{

Display **"**integer phi**";**

a=1;

b=-((N-final_phi)+1);

c=N;

Call to value = **calculate_root(a,b,c)** function

If **(value == -1)** then

goto **again;**

Else{

If **(N == root1 * root2)** then

Display finally we are able to find the correct factors of N;

Display root1 and root2;

RHS = (pow(N,0.25))/3;

If **(LHS < RHS)** then

Display **"ATTACK IS SUCCESSFUL";**

Else

Display **"Attack is not possible";**

Step-10 END

```
//Continued_fractions algo.
                {
                while r != 0 then
                        {
        .               Counter =Counter+1 ;
                        q = e/N;
                        r = e%N;
                        i = i + 1;
                        e=N;
                        N=r;
                        continued_fractions(e,N);   //again call to same function till r !=0
                        }
                }

//convergent_function algo.
                        {
                        h[0]=a[0];
                        h[1]=a[1]*a[0] + 1;
                        k[0]=1;
                        k[1]=a[1];
                        While (h[n-1] != e && k[n-1] != N && a[n] != '\0')  then
                                {
                                h[n]= a[n]*h[n-1] + h[n-2];
                                 k[n]= a[n]*k[n-1] + k[n-2];
                                 n= n + 1;
                                 }return n;
                        }

//check_phi_int function algo.
{
        static  x=0;
        temper=-1;
        While (x<99) then
        {
         Phi[x] = ((e*d[x+1] - 1)/(float)K[x+1]);
         If (floor(phi[x])==phi[x])  then
                {
                 temper=(int)phi[x];
                 LHS = d[x+1];
                 x= x + 1;
                 break;
                }
        else
         x= x + 1;
         }
         If ((d[x] == '\0' && K[x] == '\0'))  then
                return 2;
         else
                return temper;}
```

```
//calculate_root function
{
   D=b*b-4*a*c;
        If (D>=0)  then
        {
                x1=(-b+sqrt(D))/(2*a);
                x2=(-b-sqrt(D))/(2*a);
                If  ((x1-(int)x1)==0 && (x2-(int)x2)==0) then
                        {
                        root1=(int)x1;
                        root2=(int)x2;
                        return 1;
                        }
                else
                        return -1;
        }
}
```