Assignment 1

1. Implement a 32-bit LFSR. Use

$$p(X) = X^{32} + X^{22} + X^{2} + X^{1} + 1$$

as the corresponding connecting polynomial. (5)

2. Implement TRIVIUM. Use the test vectors given here. (10)

3. Implement RC4. Your program should work for any l-byte key, where 10 <= l <= 40. Use your implementation to verify Mantin's second output byte bias. (10)

1.LFSR

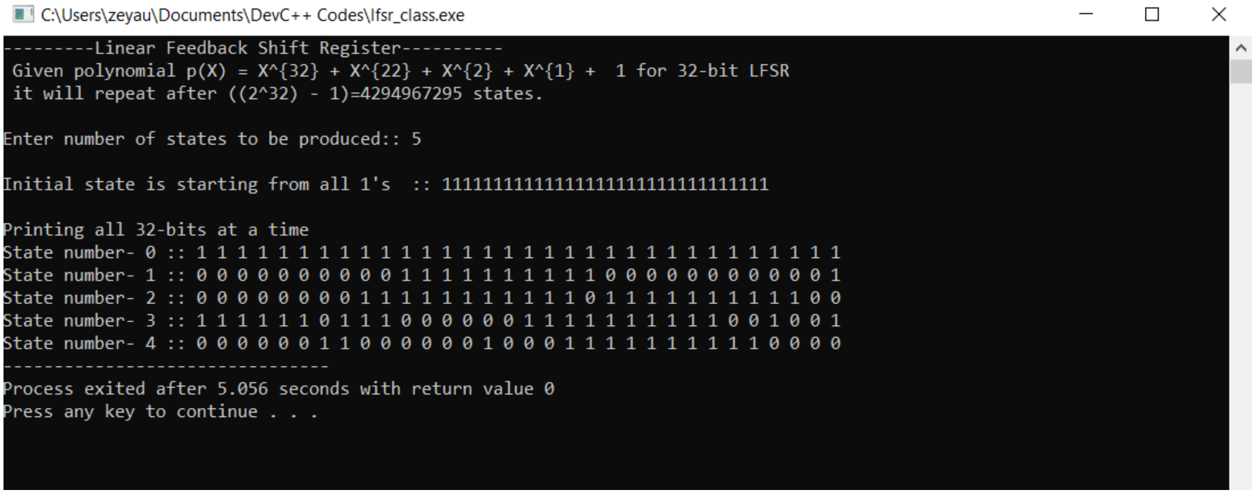
Input:: Number of states to be produced
Polynomial is already given.

Output:: printing all 32 bits of a state

Code: initial array has been taken as all 1’s
lfsr.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void print(int *a,int n){
        int i;
        for (i = 0;i < n;i++) {
                printf("%d", a[i]);
        }
        printf("\t");
}

int main()
{
        int arr[32] = { 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 };
        int out_bit, in_bit;  //array initialised as all 1's
        //int arr[4] = {1,1,1,1};    //testing
        int x = 1,i,j,n=32,bitss;

        printf("---------Linear Feedback Shift Register----------\n Given polynomial p(X) = X^{32} + X^{22} + X^{2} + X^{1}
+  1 for 32-bit LFSR\n it will repeat after ((2^32) - 1)=4294967295 states.\n");

        printf("\nEnter number of states to be produced:: ");
        scanf("%d",&bitss);

        //printf("x xor 1 = %d\n", (x ^ 1)^1); //testing
        //printf("x xor 0 = %d\n", x ^ 0); //testing
        printf("\nInitial state is starting from all 1's  :: ");
        print(arr,n);
        printf("\n");
        printf("\nPrinting all 32-bits at a time");
        for (i = 0;i < 32*bitss;i++) {
                //printf("Iteration Number = %d\t",i);
                if(i%32 == 0) printf("\nState number- %d :: ",i/32);
                in_bit = (((arr[0] ^ arr[1]) ^ arr[2]) ^ arr[22]);
                //in_bit = (arr[3] ^ arr[2]);
                //printf("In_bit = %d\t",in_bit);

                //shifting process
                out_bit = arr[0];
                for (j = 0;j < n-1;j++) {
                        arr[j] = arr[j + 1];
                }
                arr[n-1] = in_bit;
                //print(arr,n);
                printf("%d ", out_bit);
        }
        return 0;
}
```

Sample Output:: for n=4 states



```
---------Linear Feedback Shift Register----------
 Given polynomial p(X) = X^{32} + X^{22} + X^{2} + X^{1} +  1 for 32-bit LFSR
 it will repeat after ((2^32) - 1)=4294967295 states.

Enter number of states to be produced:: 5

Initial state is starting from all 1's  :: 11111111111111111111111111111111

Printing all 32-bits at a time
State number- 0 :: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
State number- 1 :: 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1
State number- 2 :: 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0
State number- 3 :: 1 1 1 1 1 1 0 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0 1
State number- 4 :: 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0
--------------------------------
Process exited after 5.056 seconds with return value 0
Press any key to continue . . .
```

2.Trivium
Input:: Stream length for an input containing all 0’s
Key:: take from user in hex format
IV :: all 0’s

Output:: Code will print the encrypted text(stream) in hex format which can be verified from below test vectors for Trivium
https://github.com/cantora/avr-crypto-lib/blob/master/testvectors/trivium-80.80.test-vectors

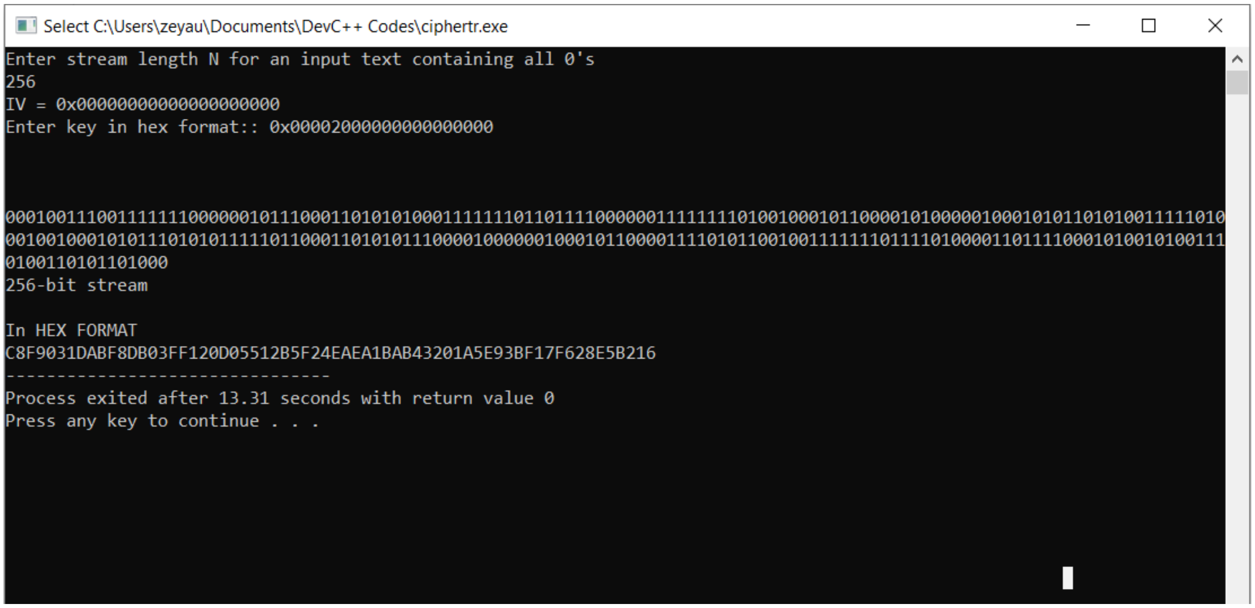
Code: ciphertr.c *file has been compiled and executed on Dev-C++ tool.*

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
char stream[1000];
void print(int *arr){
        int i;
        for(i=0;i<288;i++)
                printf("%d",arr[i]);
        printf("\n");
}
void hex_to_bin(char *st, int *keyy, int m){
        int i,j,num,a,b,c,d;
        int k = 79;
        int flag=0;
        for(i=0;i<strlen(st);i++){
                if(st[i]>='0' && st[i]<='9'){
            num=st[i]-0x30;
            if(num==0)   {a=0;b=0;c=0;d=0;}  if(num==2)   {a=0;b=0;c=1;d=0;}  if(num==4)   {a=0;b=1;c=0;d=0;}  if(num==6)
{a=0;b=1;c=1;d=0;}
            if(num==1)   {a=0;b=0;c=0;d=1;}  if(num==3)   {a=0;b=0;c=1;d=1;}  if(num==5)   {a=0;b=1;c=0;d=1;}  if(num==7)
{a=0;b=1;c=1;d=1;}
            if(num==8)  {a=1;b=0;c=0;d=0;} if(num==9)  {a=1;b=0;c=0;d=1;}
                }
          else{
            switch(st[i]){
              case 'A': num=10; a=1;b=0;c=1;d=0;
                                                  break;
              case 'B': num=11; a=1;b=0;c=1;d=1;
                                                  break;
              case 'C': num=12;  a=1;b=1;c=0;d=0;
                                                  break;
              case 'D': num=13;  a=1;b=1;c=0;d=1;
                                                  break;
              case 'E': num=14;  a=1;b=1;c=1;d=0;
                                                  break;
              case 'F': num=15;  a=1;b=1;c=1;d=1;
                                                  break;
              default: num=0;
            }
          }
          //printf("%d ",num);
          if(flag==0){
                keyy[k-7]= a; keyy[k-6]=b;keyy[k-5]= c;keyy[k-4]= d;
                flag=1;
          }
          else{
                keyy[k-3]= a; keyy[k-2]=b;keyy[k-1]= c;keyy[k]= d;
                flag=0;
                k= k-8;
                }
        }
        /*printf("\nPrinting keyy :: ");
        for(i=0;i<80;i++)
        {
                printf("%d",keyy[i]);
        }*/
        printf("\n");
}
void bin_to_hex(int *buff, int n){
        int i,j,k,val,num,stream_i,flag=0;
        char ch;
        //for(i=0;i<n;i++)        printf("%d",buff[i]);
        printf("\nIn HEX FORMAT\n");
        stream_i=n/4 -1;
        for(i=0;i<n;i+=4){
                val = buff[i]*8 + buff[i+1]*4 + buff[i+2]*2 + buff[i+3];
          if(val>9)
          {
            switch(val)
            {
              case 10: ch = 'A'; break;
              case 11: ch = 'B';break;
              case 12: ch = 'C';break;
              case 13: ch = 'D';break;
              case 14: ch = 'E';break;
              case 15: ch = 'F';break;
              default: ch = 'z';
            }
          }
          else
                ch = 48+val;
          //printf("%c",ch);
          if(flag==0){
                stream[stream_i-1]=ch;
                flag=1;
                }
                else{
                        stream[stream_i]=ch;
                        stream_i -= 2;
                        flag=0;
                }
        }
        for(i=0;i<n/4;i++){
                printf("%c",stream[i]);
        }

}
int main()
{
        int s[288],iv[80]={0},key[80]={0};
        int i, j,t1,t2,t3,z[2048]={0},N,z_rev[2048];
        int buff[8];
        char str[80];
        fflush(stdin);
        printf("Enter stream length N for an input text containing all 0's\n");
        scanf("%d",&N);
        //input iv and key
```

```
printf("IV = 0x00000000000000000000\n");
printf("Enter key in hex format:: 0x");
scanf("%s",str);
//printf("%s",str);
hex_to_bin(str,key,80);

//state initialization
//(st1; : : : ; st93) := (k1; : : : ; k80; 0; : : : ; 0)
//(st94; : : : ; st177) : = (iv1; : : : ; iv80; 0; 0; 0; 0)
//(st178; : : : ; st288) : = (0; : : : ; 0; 0; 1; 1; 1) :
for (i = 0;i < 80;i++)              s[i] = key[i];
for (i = 80;i < 93;i++)                         s[i] = 0;
for (i = 93;i < 173;i++)            s[i] = iv[i-93];
s[173] = 0;
s[174] = 0;
s[175] = 0;
s[176] = 0;
for (i = 177;i < 285;i++)           s[i] = 0;
s[285] = 1;
s[286] = 1;
s[287] = 1;

//print(s);
//testing
//t1 = 0 ^ (0 & 1) ^ 1 ^ 1;
//printf("%d ", t1);

//Init phase
for (i = 0;i < (4*288);i++) {
            t1 = (s[65] ^ (s[90] & s[91]) )^ (s[92] ^ s[170]);
            t2 = (s[161] ^ (s[174] & s[175]) )^ (s[176] ^ s[263]);
            t3 = (s[242] ^ (s[285] & s[286]) )^ (s[287] ^ s[68]) ;

            //printf("t1:%d t2:%d t3:%d\t",t1,t2,t3);

            for(j=92;j>0;j--)
                        s[j] = s[j-1];
            s[0]=t3;
            for(j=176;j>93;j--)
                        s[j] = s[j-1];
            s[93]=t1;
            for(j=287;j>177;j--)
                        s[j] = s[j-1];
            s[177]=t2;

            //print(s);
            //getch();
}
//print(s);

printf("\n\n");
//GetBits
for(i=0;i<N;i++){
            t1 = s[65] ^ s[92];
            t2 = s[161] ^ s[176];
            t3 = s[242] ^ s[287];

            z[i] = ((t1 ^ t2 ) ^ t3);
            printf("%d",z[i]);

            t1 = (t1 ^ (s[90] & s[91]) )^ s[170];
```

```
        t2 = (t2 ^ (s[174] & s[175]) )^ s[263];
        t3 = (t3 ^ (s[285] & s[286]) )^ s[68];

        for(j=92;j>0;j--)       s[j] = s[j-1];
        s[0]=t3;
        for(j=176;j>93;j--)     s[j] = s[j-1];
        s[93]=t1;
        for(j=287;j>177;j--)    s[j] = s[j-1];
        s[177]=t2;
    }
    printf("\n");
    j=0;
    for(i=N-1;i>=0;i--){
        z_rev[j]=z[i];
        //printf("%d",z_rev[j]);
        j++;
    }
    //binary to hex
    printf("%d-bit stream\n",N);
    bin_to_hex(z_rev,N);

    return 0;
}
```

Sample output:



```
Enter stream length N for an input text containing all 0's
256
IV = 0x00000000000000000000
Enter key in hex format:: 0x00002000000000000000



000100111001111111000000101110001101010100011111110110111100000011111111010010001011000010100000100010101101010011111010
001001000101011101010111110110001101010111000010000001000101100001111010110010011111110111101000011011110001010010100111
0100110101101000
256-bit stream

In HEX FORMAT
C8F9031DABF8DB03FF120D05512B5F24EAEA1BAB43201A5E93BF17F628E5B216
--------------------------------
Process exited after 13.31 seconds with return value 0
Press any key to continue . . .
```

3.RC4
Input:: all 0’s
Key:: take from user in hex format(for any byte length)

Output:: Code will print the encrypted text in hex format which can be verified from below test vectors for RC4 provided by IETF
https://tools.ietf.org/html/rfc6229

rc4.c file has been compiled and executed on Dev-C++ tool.

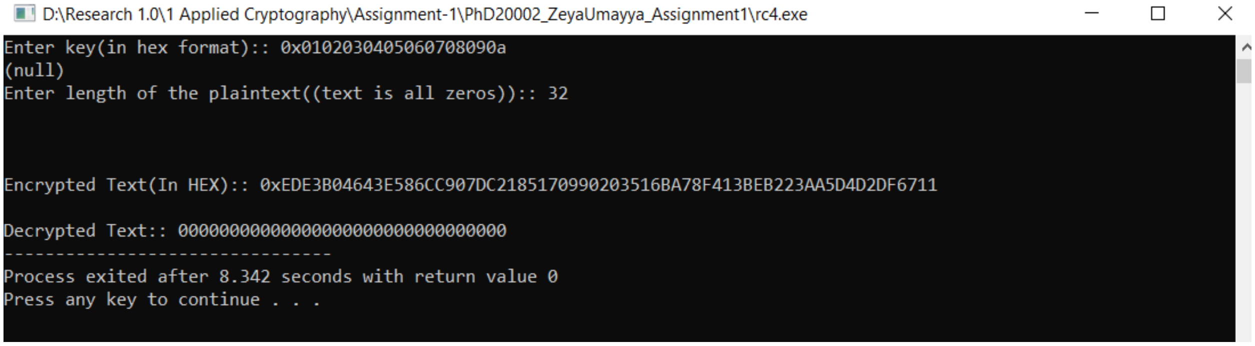
Code rc4.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
void byte_to_hex(int *buf,int n){
                int i;
                const char * hex = "0123456789ABCDEF";
                char str[100], * pout = str;
                str[0]='\0';
    for(i=0; i < n; ++i){
                //printf("%d ",*buf);
        *pout++ = hex[(*buf>>4)&0xF];
        *pout++ = hex[(*buf++)&0xF];
        //*pout++ = ':';
                }
                str[n*2]='\0';
                printf("%s \n",str);
}
int getNum(char ch)
{
    int num=0;
    if(ch>='0' && ch<='9')
    {
        num=ch-0x30;
    }
    else
    {
        switch(ch)
        {
            case 'A': case 'a': num=10; break;
            case 'B': case 'b': num=11; break;
            case 'C': case 'c': num=12; break;
            case 'D': case 'd': num=13; break;
            case 'E': case 'e': num=14; break;
            case 'F': case 'f': num=15; break;
            default: num=0;
        }
    }
    return num;
}
int hex2int(char hex[])
{
    int x=0;
    x=(getNum(hex[0]))+(getNum(hex[1]))*16;

    return x;
}
```

```
int main()
{
        int i,j,l,kk,s[256],k[320],t[320],kstr,rev[200];
        int len,temp,ct[200],pt[200]={0},lent;
        char str[80],ch[2];
        int num[320];
        fflush(stdin);

        printf("Enter key(in hex format):: 0x");
        gets(str);
        //procesing input hex key
        j=(strlen(str)/2) -1;
        //j=0;
        printf("%s",str[strlen(str)]);
        for(i=strlen(str)-1;i>0;i-=2){
                ch[0]=str[i];
                ch[1]=str[i-1];
                num[j]=hex2int(ch);
                j--;
        }

        //done
        printf("\nEnter length of the plaintext((text is all zeros)):: ");
        //gets(pt);
        scanf("%d",&lent);
        printf("\n");

        //len = strlen(str)/2;
        len = strlen(str)/2;
        //initialize stream and key
        for(i=0;i<256;i++){
                s[i] = i;
                t[i] = num[i%len];
                //printf("%d ",t[i]);
        }
        printf("\n");
        //KEYGEN
        j=0;
        for(i=0;i<256;i++){
                j = (j+s[i]+t[i]) % 256;
                temp=s[i];
                s[i]=s[j];
                s[j]=temp;
        }

        //PseudoRandomGeneration
        len = strlen(pt);
        i=0;
        j=0;
        for(l=0;l<lent;l++){
                i = (i+1) % 256;
                j = (j+s[i]) % 256;
                temp=s[i];
                s[i]=s[j];
                s[j]=temp;
                kk = (s[i]+s[j]) % 256;
                ct[l] = pt[l] ^ s[kk];
                //printf("%c",ct[l]);

                //decrypting for verification purpose
```

```
            rev[l] = s[kk];
    }
    //encrypted text
    printf("\nEncrypted Text(In HEX):: 0x");
    byte_to_hex(ct,lent);

    for(i=0;i<l;i++){
            //printf("%d ",ct[i]);
            //byte_to_hex(ct[i]);
            //printf("%c",byte_to_hex(ct[i])) ;
    }

    printf("\nDecrypted Text:: ");
    //decrypted text
    for(i=0;i<l;i++){
            printf("%d",(ct[i]^rev[i]) );
    }



    return 0;
}
```

Sample output for the first test vector – key length 10byte



D:\Research 1.0\1 Applied Cryptography\Assignment-1\PhD20002_ZeyaUmayya_Assignment1\rc4.exe

```
Enter key(in hex format):: 0x0102030405060708090a
(null)
Enter length of the plaintext((text is all zeros)):: 32



Encrypted Text(In HEX):: 0xEDE3B04643E586CC907DC2185170990203516BA78F413BEB223AA5D4D2DF6711

Decrypted Text:: 00000000000000000000000000000000
--------------------------------
Process exited after 8.342 seconds with return value 0
Press any key to continue . . .
```

After analyzing multiple outputs, it can be seen that the second output byte of RC4 is (slightly) biased towards 0.