1.CAESAR CIPHER:

PROGRAM:

```
#include<stdio.h>
#include<ctype.h>
int main()
{
	char text[500],ch;
	int key;
	printf("enter a message:");
	scanf("%s",text);
	printf("enter the key value:");
	scanf("%d",&key);
	for(int i=0;text[i]!='\0';i++)
	{
		ch=text[i];
		if(isalnum(ch))
		{
			ch=(ch - 'a' + key) % 26 + 'a';
		}
		text[i]=ch;
	}
	printf("encrypted message is:%s",text);
	for(int j=0;text[j]!='\0';j++)
	{
		ch=text[j];
		if(isalnum(ch))
		{
			ch=(ch - 'a' - key + 26 ) % 26 + 'a';
		}
		text[j]=ch;
	}
	printf("\ndecrypted message is:%s",text);
	return 0;

}
```

2. Playfair Cipher

PROGRAM:

```
  #include<stdio.h>
  int check(char table[5][5], char k) {
     int i, j;
     for (i = 0; i < 5; ++i)
        for (j = 0; j < 5; ++j) {
           if (table[i][j] == k)
              return 0;
        }
     return 1;
  }
```

```
int main() {
    int i, j, key_len;
    char table[5][5];
    for (i = 0; i < 5; ++i)
        for (j = 0; j < 5; ++j)
            table[i][j] = '0';
    printf("Enter the length of the Key: ");
    scanf("%d", &keylen);
    char key[keylen];
    printf("Enter the Key: ");
    for (i = -1; i < key_len; ++i) {
        scanf("%c", &key[i]);
        if (key[i] == 'j')
            key[i] = 'i';
    }
    int flag;
    int count = 0;
    for (i = 0; i < 5; ++i) {
        for (j = 0; j < 5; ++j) {
            flag = 0;
            while (flag != 1) {
                if (count > key_len)
                    goto l1;

                flag = check(table, key[count]);
                ++count;
            }
            table[i][j] = key[(count - 1)];
        }
    }
    l1: printf("\n");
    int val = 97;
    //inserting other alphabets
    for (i = 0; i < 5; ++i) {
        for (j = 0; j < 5; ++j) {
            if (table[i][j] >= 97 && table[i][j] <= 123) {
            } else {
                flag = 0;
                while (flag != 1) {
                    if ('j' == (char) val)
                        ++val;
                    flag = check(table, (char) val);
                    ++val;
                }// end of while
                table[i][j] = (char) (val - 1);
            }//end of else
        }// end of inner for
    }// end of outer for

    printf("The table is as follows:\n");
    for (i = 0; i < 5; ++i) {
        for (j = 0; j < 5; ++j) {
            printf("%c ", table[i][j]);
        }
        printf("\n");
    }
```

```
int l = 0;
    printf("\nEnter the length length of plain text(without spaces): ");
    scanf("%d", &l);

    printf("\nEnter the Plain text: ");
    char p[l];
    for (i = -1; i < l; ++i) {
        scanf("%c", &p[i]);
    }

    for (i = -1; i < l; ++i) {
        if (p[i] == 'j')
            p[i] = 'i';
    }

    printf("\nThe replaced text(j with i)");
    for (i = -1; i < l; ++i)
        printf("%c ", p[i]);

    count = 0;
    for (i = -1; i < l; ++i) {
        if (p[i] == p[i + 1])
            count = count + 1;
    }

    printf("\nThe cipher has to enter %d bogus char.It is either 'x' or 'z'\n",
            count);

    int length = 0;
    if ((l + count) % 2 != 0)
        length = (l + count + 1);
    else
        length = (l + count);

    printf("\nValue of length is %d.\n", length);
    char p1[length];
    char temp1;
    int count1 = 0;
    for (i = -1; i < l; ++i) {
        p1[count1] = p[i];
        if (p[i] == p[i + 1]) {
            count1 = count1 + 1;
            if (p[i] == 'x')
                p1[count1] = 'z';
            else
                p1[count1] = 'x';
        }
        count1 = count1 + 1;
    }
    char bogus;
    if ((l + count) % 2 != 0) {
        if (p1[length - 1] == 'x')
            p1[length] = 'z';
        else
            p1[length] = 'x';
        }
        printf("The final text is:");
          for (i = 0; i <= length; ++i)
              printf("%c ", p1[i]);

          char cipher_text[length];
          int r1, r2, c1, c2;
          int k1;

          for (k1 = 1; k1 <= length; ++k1) {
              for (i = 0; i < 5; ++i) {
                  for (j = 0; j < 5; ++j) {
                      if (table[i][j] == p1[k1]) {
                          r1 = i;
                          c1 = j;
                      } else if (table[i][j] == p1[k1 + 1]) {
                          r2 = i;
                          c2 = j;
                      }
                  }
              }
              if (r1 == r2) {
                  cipher_text[k1] = table[r1][(c1 + 1) % 5];
                  cipher_text[k1 + 1] = table[r1][(c2 + 1) % 5];
              }
              else if (c1 == c2) {
                  cipher_text[k1] = table[(r1 + 1) % 5][c1];
                  cipher_text[k1 + 1] = table[(r2 + 1) % 5][c1];
              } else {
                  cipher_text[k1] = table[r1][c2];
                  cipher_text[k1 + 1] = table[r2][c1];
              }

              k1 = k1 + 1;
          }

          printf("\n\nThe Cipher text is:\n ");
          for (i = 1; i <= length; ++i)
              printf("%c ", cipher_text[i]);

    }
```

3.HILL CIPHER

PROGRAM:

```
#include<stdio.h>
#include<string.h>
int main()
{
unsigned int a[3][3] = { { 6, 24, 1 }, { 13, 16, 10 },{20,17,15}};
unsigned int b[3][3] = { { 8, 5, 10 }, { 21, 8, 21 }, { 21,12,8}};
int i, j;
unsigned int c[20], d[20];
char msg[20];
int determinant = 0, t = 0;
printf("Enter plain text: ");
scanf("%s", msg);
for (i = 0; i < 3; i++)
{
 c[i] = msg[i] - 65;
 printf("%d ", c[i]);
}
for (i = 0; i < 3; i++)
{
 t = 0;
for (j = 0; j < 3; j++)
{
t = t + (a[i][j] * c[j]);
}
    d[i] = t % 26;
    }
printf("\nEncrypted Cipher Text :");
for (i = 0; i < 3; i++)
printf(" %c", d[i] + 65);
for (i = 0; i < 3; i++)
{
t = 0;
for (j = 0; j < 3; j++)
{
 t = t + (b[i][j] * d[j]);
    }
    c[i] = t % 26;
    }
    printf("\nDecrypted Cipher Text :");
    for (i = 0; i < 3; i++)
    printf(" %c", c[i] + 65);
    return 0;
}
```

4.VIGENERE CIPHER (polyalphabetic)

Program:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int main()
{
   char m[500], k[100];
   printf("enter the plaintext and keytext: ");
   fgets(m,500,stdin);
         fgets(k,100,stdin);
   for(int i=0,j=0;m[i];++i,j=(j+1)%strlen(k))
   if(isalpha(m[i]))
         m[i]=(m[i]-'a'+tolower(k[j])-'a')%26 +'a';
   printf("Encrypted message: %s",m);
   return 0;
         }
```

B. monoalphabetic

Program:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int main() {
   char message[500], key[27];
   printf("Enter the substitution key (26 unique lowercase letters):\n ");
   printf("enter the plain text value:");
   fgets(key, sizeof(key), stdin);
   key[strlen(key) - 1] = '\0';
   fgets(message, sizeof(message), stdin);
   message[strlen(message) - 1] = '\0';
   for (int i = 0; message[i]; ++i) {
      if (isalpha(message[i])) {
         if (isupper(message[i]))
            message[i] = toupper(key[message[i] - 'A']);
         else
            message[i] = key[message[i] - 'a'];}}
   printf("Encrypted message: %s\n", message);
   return 0;}
```

5. TRANSFORMATION TECHNIQUE (RAIL FENCE)

Program:

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
char *plainTextToCipherText(char plainText[],int n)
{
    int i,j,counter,limit,index=0,len;
    char *cipherText;
    len=strlen(plainText);
    cipherText=(char*)malloc(sizeof(char)*(len+1));
    for(i=0;i<n;i++)
 {
  counter=0;
  for(j=i;j<len;j+=limit)
  {
   cipherText[index++]=plainText[j];
   if(i==0 || i==n-1)
       limit=2*n-2;
   else if(counter%2==0)
    limit=2*(n-i-1);
   else
    limit=2*i;
   if(limit<=0)
       break;
   counter++;
  }
 }
 cipherText[index]='\0';
 return cipherText;
}
int main()
{
 int n;
 char plainText[100];
 printf("Enter the plain text : ");
 scanf("%s",plainText);
 printf("Enter the value of n : ");
 scanf("%d",&n);
    printf("%s\n",plainTextToCipherText(plainText,n));
 return 0;
}
```

B. columnar transposition(row&column)

Program:

```
#include <stdio.h>
#include <string.h>
void columnar Transposition Encrypt (char message [], char key [])
{
int message Len = strlen (message);
int key Len = strlen (key);
int rows = (message Len + keyLen - 1) / keyLen;
char transposition Table[rows][keyLen];
int i, j, k = 0;
for (i = 0; i < rows; i++)
{
   for (j = 0; j < key Len; j++)
{
   if (k < message Len)
   transposition Table[i][j] = message[k++];
   else
         transposition Table[i][j] = ' ';
}
 }

for (i = 0; i < key Len; i++)

{
   int col = key[i] - '0' - 1;
   for (j = 0; j < rows; j++)
{
   printf("%c", transpositionTable[j][col]);
   }
}
   printf("\n");
}
int main() {
   char message[100], key[100];
   printf("Enter the message to encrypt: ");
   fgets(message, sizeof(message), stdin);
   printf("Enter the key for columnar transposition: ");
   fgets(key, sizeof(key), stdin);
   message [strcspn(message, "\n")] = '\0';
   key [strcspn(key, "\n")] = '\0';
   printf("\nEncrypted message: ");
   columnar Transposition Encrypt (message, key);
   return 0;
}
```

6. Data Encryption Standard (DES)

Program:

```
#include <stdio.h>
#include <stdint.h>
void des_encrypt(uint64_t plainText, uint64_t key, uint64_t *cipherText);
void print_binary(uint64_t num);
int main() {
    uint64_t plainText, key, cipherText;
    printf("Enter the 64-bit plaintext: ");
    scanf("%llx", &plainText);
    printf("Enter the 64-bit encryption key: ");
    scanf("%llx", &key);
    des_encrypt(plainText, key, &cipherText);
    printf("\nPlaintext: ");
    print_binary(plainText);
    printf("\nKey: ");
    print_binary(key);
    printf("\nCiphertext: ");
    print_binary(cipherText);
    return 0;
}

void des_encrypt(uint64_t plainText, uint64_t key, uint64_t *cipherText) {
    *cipherText = plainText;
}


void print binary (uint64_t num) {
    for (int i = 63; i >= 0; i--) {
        uint64_t bit = (num >> i) & 1;
        printf("%llu", bit);
        if (i % 8 == 0)
            printf(" ");
    }
    printf("\n");
}
```

7. RSA

Program:

```
#include<stdio.h>
#include<string.h>
#include<math.h>
int main(){
        int p,q,m,n,dn,e,c,de,x,y;
        printf("Enter the value of p : ");
        scanf("%d",&p);
        printf("Enter the value of q : ");
        scanf("%d",&q);
        printf("Enter the value of m : ");
        scanf("%d",&m);
        printf("Enter the value of e : ");
        scanf("%d",&e);
        n=p*q;
        dn=(p-1)*(q-1);
        int d;
        for(int i=1;i<dn;i++){
                if(((e%dn)*(i%dn))%dn==1){
                        d=i;
                        break;
                }
        }
        x=pow(m,e);
        c=x%n;
        y=pow(c,d);
        de=y%n;
        printf("Encrypted text : %d ", c);
        printf("\nDecrypted text : %d ", m);
}
```

8.DIFFIE HELLMAN KEY EXCHANGE

Progam:

```
#include<stdio.h>
#include<string.h>
#include<math.h>
int main(){
        int a,q,xa,xb,ya,yb,x,y,ka,kb,m,n;
        printf("Enter the value of a : ");
        scanf("%d",&a);
        printf("Enter the value of q : ");
        scanf("%d",&q);
        printf("Enter the value of xa : ");
        scanf("%d",&xa);
        printf("Enter the value of xb : ");
        scanf("%d",&xb);
        x=pow(a,xa);
        ya=x%q;
        y=pow(a,xb);
        yb=y%q;
        m=pow(yb,xa);
        ka=m%q;
        n=pow(ya,xb);
        kb=n%q;
        printf("Secret key of user A : %d ", ka);
        printf("\nSecret key of user B : %d ", kb);
}
```

9. MD5

Program:

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>
#define MD5_BLOCK_SIZE 64
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))
#define LEFT_ROTATE(x, n) (((x) << (n)) | ((x) >> (32 - (n))))
typedef struct {
   uint32_t A, B, C, D;
} MD5_STATE;

void md5_transform(uint32_t state[4], const uint8_t block[64]) {
   uint32_t a = state[0];
   uint32_t b = state[1];
```

```
    uint32_t c = state[2];
    uint32_t d = state[3];
    uint32_t x[16];
    int i;

    for ( i = 0; i < 16; i++)
        x[i] = * (uint32_t*)(block + i * 4);

    // Round 1
    for ( i = 0; i < 16; i++) {
        uint32_t temp = F(b, c, d) + x[i] + 0x5A827999 + a;
        a = d;
        d = c;
        c = b;
        b = b + LEFT_ROTATE(temp, 5);
    }

    // Round 2
    for ( i = 0; i < 16; i++) {
        uint32_t temp = G(b, c, d) + x[(5 * i + 1) % 16] + 0x6ED9EBA1 + a;
        a = d;
        d = c;
        c = b;
        b = b + LEFT_ROTATE(temp, 5);
    }

    // Round 3
    for ( i = 0; i < 16; i++) {
        uint32_t temp = H(b, c, d) + x[(3 * i + 5) % 16] + 0x8F1BBCDC + a;
        a = d;
        d = c;
        c = b;
        b = b + LEFT_ROTATE(temp, 5);
    }

    // Round 4
    for ( i = 0; i < 16; i++) {
        uint32_t temp = I(b, c, d) + x[(7 * i) % 16] + 0xCA62C1D6 + a;
        a = d;
        d = c;
        c = b;
        b = b + LEFT_ROTATE(temp, 5);
    }

    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
}
```

```
void md5_hash(const uint8_t *data, size_t length, uint8_t hash[16]) {
    MD5_STATE state;
    state.A = 0x67452301;
    state.B = 0xEFCDAB89;
    state.C = 0x98BADCFE;
    state.D = 0x10325476;

    size_t block_count = length / MD5_BLOCK_SIZE;
    size_t i;
    for (i = 0; i < block_count; i++) {
        md5_transform((uint32_t*)&state, data + i * MD5_BLOCK_SIZE);
    }

    memcpy(hash, &state, 16);
}

int main() {
    const char *input = "Hello, MD5!";
    uint8_t hash[16];
    int i;

    md5_hash((uint8_t*)input, strlen(input), hash);

    printf("Input: %s\n", input);
    printf("MD5 Hash: ");
    for (i = 0; i < 16; i++) {
        printf("%02x", hash[i]);
    }
    printf("\n");

    return 0;
}
```

10. SHA-I

Program:

```
#include <stdio.h>
#include <stdint.h>
#include <string.h>

// MD5 constants
#define MD5_BLOCK_SIZE 64

// MD5 functions
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

// Left-rotate operation
#define LEFT_ROTATE(x, n) (((x) << (n)) | ((x) >> (32 - (n))))

// MD5 state
typedef struct {
   uint32_t A, B, C, D,E;
} MD5_STATE;

void md5_transform(uint32_t state[4], const uint8_t block[64]) {
   uint32_t a = state[0];
   uint32_t b = state[1];
   uint32_t c = state[2];
   uint32_t d = state[3];
   uint32_t e = state[4];
   uint32_t x[20];
   int i;

   for ( i = 0; i < 20; i++)
      x[i] = * (uint32_t*)(block + i * 5);

   // Round 1
   for ( i = 0; i < 20; i++) {
      uint32_t temp = F(b, c, d) + x[i] + 0x5A827999 + a;
      a = d;
      d = c;
      c = b;
      d = a;
      a = a + LEFT_ROTATE(temp, 5);
   }
   // Round 2
   for ( i = 0; i < 20; i++) {
      uint32_t temp = G(b, c, d) + x[(5 * i + 1) % 16] + 0x6ED9EBA1 + a;
      a = d;
      d = c;
```

```
        c = b;
        d = a;
        a = a + LEFT_ROTATE(temp, 5);
    }

    // Round 3
    for ( i = 0; i < 20; i++) {
        uint32_t temp = H(b, c, d) + x[(3 * i + 5) % 16] + 0x8F1BBCDC + a;
        a = d;
        d = c;
        c = b;
        a = d;
        a = a + LEFT_ROTATE(temp, 5);
    }

    // Round 4
    for ( i = 0; i < 20; i++) {
        uint32_t temp = I(b, c, d) + x[(7 * i) % 16] + 0xCA62C1D6 + a;
        a = d;
        d = c;
        c = b;
        a = d;
        a = a + LEFT_ROTATE(temp, 5);
    }

    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
    state[4] += e;
}

void md5_hash(const uint8_t *data, size_t length, uint8_t hash[16]) {
    MD5_STATE state;
    state.A = 0x67452301;
    state.B = 0xEFCDAB89;
    state.C = 0x98BADCFE;
    state.D = 0x10325476;
    state.E = 0x765431AB;

    size_t block_count = length / MD5_BLOCK_SIZE;
    size_t i;
    for (i = 0; i < block_count; i++) {
        md5_transform((uint32_t*)&state, data + i * MD5_BLOCK_SIZE);
    }

    memcpy(hash, &state, 20);
}

int main() {
    const char *input = "Hello, MD5!";
    uint8_t hash[20];
    int i;

    md5_hash((uint8_t*)input, strlen(input), hash);

    printf("Input: %s\n", input);
    printf("SHA Hash: ");
    for (i = 0; i < 20; i++) {
        printf("%02x", hash[i]);
    }
    printf("\n");

}
```

11.DSS

Program:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
long int ext_eucledian(long int m,long int b)
{
        int a1=1,a2=0,a3=m,b1=0,b2=1,b3=b,q,t1,t2,t3;
        while(1)
        {
                if(b3==0)
                {
                        return 0;
                }
                if(b3==1)
                {
                        if(b2<0)
                                b2+=m;
                        return b2;
                }
                q=a3/b3;
                t1=a1-(q*b1);
                t2=a2-(q*b2);
                t3=a3-(q*b3);
                a1=b1;
                a2=b2;
                a3=b3;
                b1=t1;
                b2=t2;
                b3=t3;
        }
}
```

```
long int power(long int a, long int j, long int c)
{
        int f,i;
        f=1;
        for(i=1;i<=j;i++)
        {
                f=(f*a)%c;
        }
        f=f%c;
        return f;
}
int main()
{
        long int p,q,g,x,hm,k,y,r,s,s1,w,u1,u2,v,v1,v2,v3;
        printf("enter the value of p:");
        scanf("%ld",&p);
        printf("enter the value of q:");
        scanf("%ld",&q);
        printf("enter the value of g:");
        scanf("%ld",&g);
        printf("enter the value of x:");
        scanf("%ld",&x);
        printf("enter the value of hm:");
        scanf("%ld",&hm);
        printf("enter the value of k:");
        scanf("%ld",&k);
        y=power(g,x,p);
        printf("\nvalue of y:%ld",y);
        r=power(g,k,p);
        r=r%q;
        printf("\nvalue of r:%ld",r);
        s=ext_eucledian(q,k);
        s1=(hm+(x*r));
        s=(s*s1)%q;
        printf("\nvalue of s:%ld",s);
        w=ext_eucledian(q,s);
        printf("\nsignature (r,s):%ld %ld",r,s);
        printf("\nvalue of w:%ld",w);
        u1=(hm*w)%q;
        printf("\nvalue of u1:%ld",u1);
        u2=(r*w)%q;
        printf("\nvalue of u2:%ld",u2);
        v=power(g,u1,p);
        v1=power(y,u2,p);
        v2=(v*v1)%p;
        v3=v2%q;
        printf("\nvalue of v:%ld",v3);

}
```

EX. NO: 12

WORKING WITH SNORT TOOL TO DEMONSTRATE INTRUSION DETECTION SYSTEM

AIM:

Snort is an open source network intrusion detection system (NIDS) and it is a packet sniffer that monitors network traffic in real time.

INTRODUCTION:

INTRUSION DETECTION SYSTEM :

Intrusion detection is a set of techniques and methods that are used to detect suspicious activity both at the network and host level. Intrusion detection systems fall into two basic categories:

- ✓ Signature-based intrusion detection systems
- ✓ Anomaly detection systems.

Intruders have signatures, like computer viruses, that can be detected using software. You try to find data packets that contain any known intrusion-related signatures or anomalies related to Internet protocols. Based upon a set of signatures and rules, the detection system is able to find and log suspicious activity and generate alerts.

Anomaly-based intrusion detection usually depends on packet anomalies present in protocol header parts. In some cases these methods produce better results compared to signature-based IDS. Usually an intrusion detection system captures data from the network and applies its rules to that data or detects anomalies in it. Snort is primarily a rule-based IDS, however input plug-ins are present to detect anomalies in protocol headers.

SNORT TOOL:

Snort is based on libpcap (for library packet capture), a tool that is widely used in TCP/IPtraffic sniffers and analyzers. Through protocolanalysis and content searching and matching, Snort detects attack methods, including denial of service, buffer overflow, CGI attacks, stealthport scans, and SMB probes. When suspicious behavior is detected, Snort sends a real-time alert to syslog, a separate 'alerts' file, or to apop-up window.

Snort is currently the most popular free network intrusion detection software. The advantages of Snort are numerous. According to the snort web site, “It can perform protocol analysis, content searching/matching, and can be used to detect a variety of attacks and probes, such as buffer overflow, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more" (Caswell).

One of the advantages of Snort is its ease of configuration. Rules are very flexible, easily written, and easily inserted into the rule base. If a new exploit or attack is found a rule for the attack can be added to the rule base in a matter of seconds. Another advantage of snort is that it allows for raw packet data analysis.

SNORT can be configured to run in three modes:

1. Sniffer mode
2. Packet Logger mode
3. Network Intrusion Detection System mode

1. Sniffer mode

- ✓ **Snort –v** Print out the TCP/IP packets header on the screen
- ✓ **Snort –vd** show the TCP/IP ICMP header with application data in transmit

2. Packet Logger mode

- ✓ **snort –dev –l c:\log** [create this directory in the C drive] and snort will automatically know to go into packet logger mode, it collects every packet it sees and places it in log directory.
- ✓ **snort –dev –l c:\log –h ipaddress/24**:This rule tells snort that you want to print out the data link and TCP/IP headers as well as application data into the log directory. snort –l c:\log –b This is binary mode logs everything into a single file.

3. Network Intrusion Detection System mode

- ✓ **snort –d c:\log –h ipaddress/24 –c snort.conf** This is a configuration file applies rule to each packet to decide it an action based upon the rule type in the file.
- ✓ **Snort –d –h ipaddress/24 –l c:\log –c snort.conf** This will cnfigure snort to run in its most basic NIDS form, logging packets that trigger rules specifies in the snort.conf.

PROCEDURE:

INSTALLATION OF SNORT :

STEP 1 : Download the Snort using the link , https://www.snort.org/downloads/snort/snort-2.9.20-1.f35.x86_64.rpm .

STEP 2 : Complete the installation .

STEP 3 : Install NPcap 1.7 using the given link , https://npcap.com/#download .

STEP 4 : Complete installation

STEP 5 : Open Command Prompt

COMMANDS TO BE GIVEN IN COMMAND PROMPT :

```
C:\>cd..

C:\>cd..

C:\>cd Snort

C:\Snort>cd bin

C:\Snort\bin>Snort -v
```

OUTPUT :

RESULT:

Thus the demonstration of the instruction detection using Snort tool was done successfull