IOT NETWORK TECHNOLOGY LAB

# Develop a program to compute

* 1. **The CRC at the transmitter for the given message and generator polynomial**

# The Syndrome at the receiver to detect the possible error during the transmission

**// Algorithm //**

**TO FIND CRC CODE**

* Enter the generator & message polynomial and their orders.
* Append ‘n’ zeroes at the end of the message (n=order of generator polynomial)
* Generate CRC code for the message by simulating shift registers-
* Simulation of shift registers-
  + The shift register size is equal to the order of the generator of the polynomial.
  + Each bit has two values (current and previous).
  + Two arrays are initialized to zero, one for storing current values and other for previous values.
* Generation of CRC code
  + The input to the shift register is modified by Ex-ORing the LSB of the generator with MSB of the message.
  + If the generator bit is ‘0’, the corresponding bits in shift register are just shifted (the previous value is

assigned to the current value).

* + If the generator bit is ‘1’ the corresponding bit in the shift register is modified by Ex-OR ing with the previous

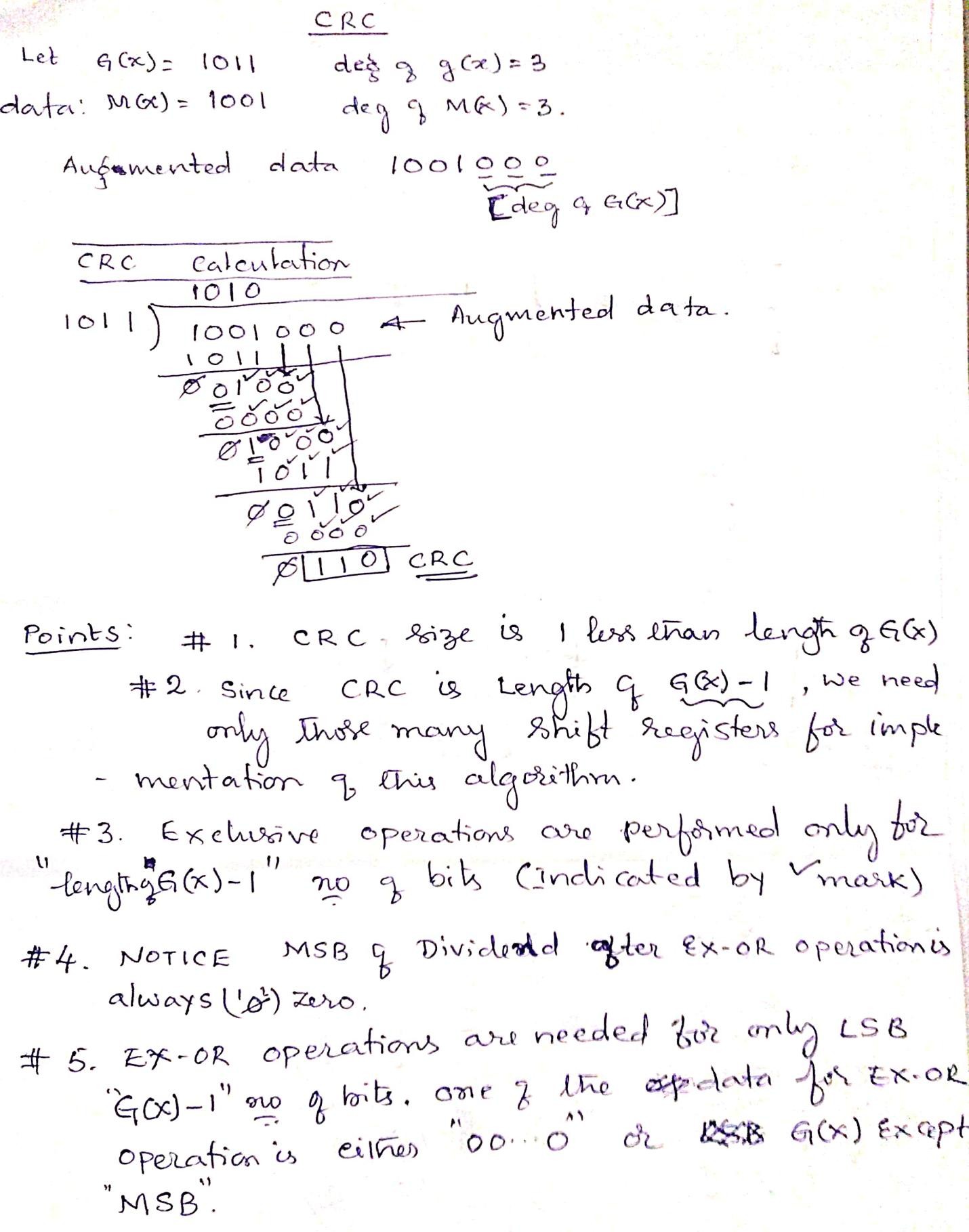
MSB of the generator with the previous bit.

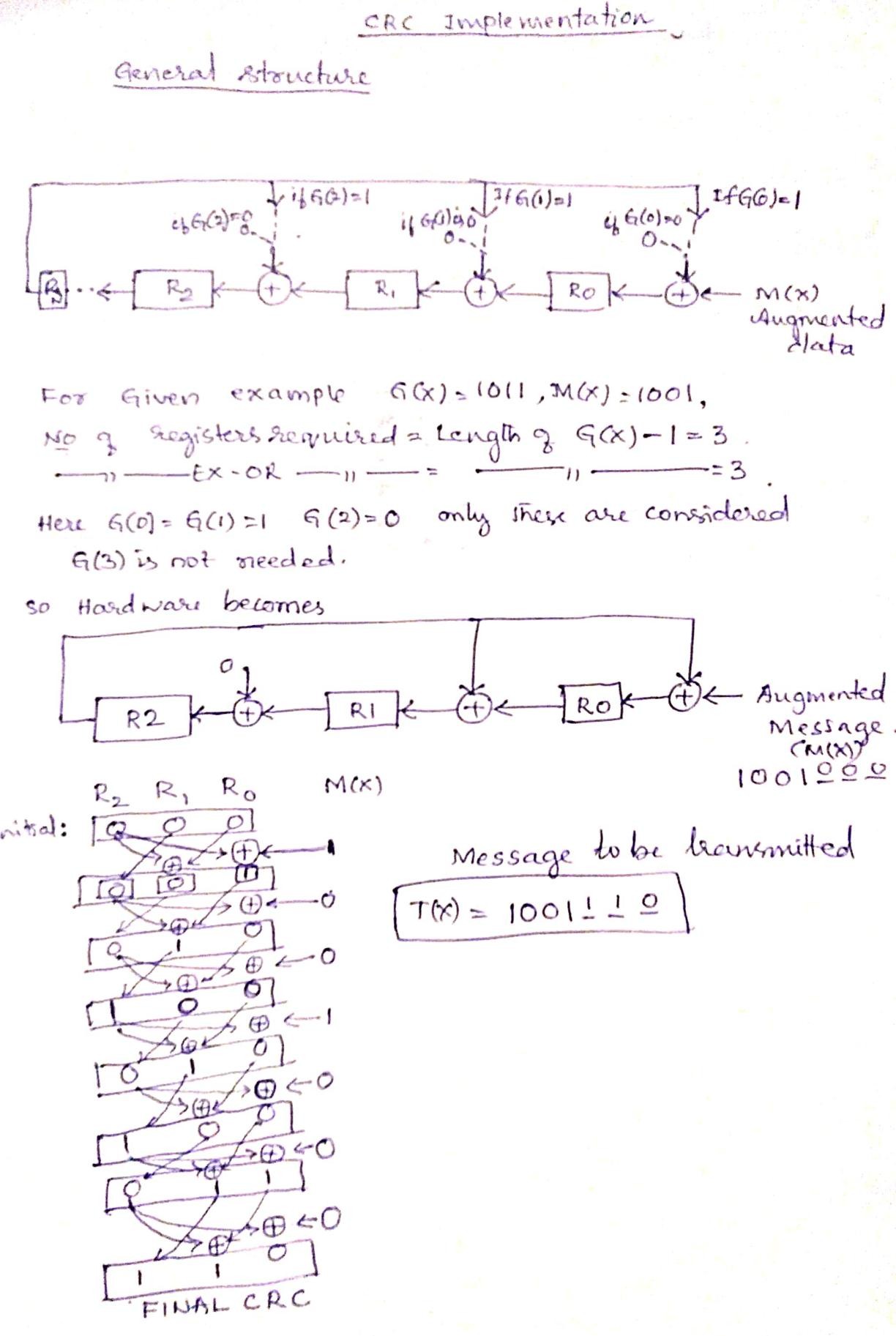
* + The final value of the bits in the current array gives the CRC code.
* Replace appended zeroes with the CRC code and transmit (display the message to be transmitted).

### Syndrome Computation at the Receiver

* Enter the received message.
* Find CRC code as explained above.
* If the CRC code obtained is non-zero, display that the received message is erroneous.
* If the CRC code is zero, display that the received message is without errors.

Illustration with Example





### // CODE //

#include<stdio.h> #include<conio.h>

int dg=16,dm,dt,data[50],gen[17]={1,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1};

int si[16],so[16],tx; void main() {

void crc(int msg[]); int i,j,k;

printf("\n Enter your choice (1/2)\n"); printf("1 for CRC-CCITT \n");

printf("2 for generalised generator polynomial\n"); printf("\n Choice:");

if(getchar()=='2') {

printf("\n Enter the degree of generator polynomial\n"); scanf("%d",&dg);

printf("\n Enter the generator polynomial\n"); for(i=dg;i>=0;i--) scanf("%d",&gen[i]);

}

printf("\n The generator polynomial is \n"); for(i=dg;i>=0;i--) printf("%d",gen[i]);

printf("\n Enter the degree of message\n"); scanf("%d",&dm);

printf("\n Enter the message\n"); for(i=dm;i>=0;i--) scanf("%d",&data[i]);

dt=dm+dg;

for(i=0;i<=dm;i++) data[dt-i]=data[dm-i]; for(i=1;i<=dg;i++) data[dg-i]=0;

tx=1; crc(data);

printf("\n Enter the received message\n"); for(i=dt;i>=0;i--)scanf("%d",&data[i]);

tx=0; crc(data);

}

void crc(int msg[]) { int i,j,k,flag;

for(i=0;i<dg;i++) {

so[i]=0;

si[i]=0;

}

for(i=dt;i>=0;i--) {

if(gen[0]==1) si[0]=so[dg-1]^msg[i]; else si[0]=msg[i];

for(j=1;j<=dg-1;j++)

if(gen[j]==1) si[j]=so[dg-1]^so[j-1]; else si[j]=so[j-1];

printf("\n");

for(k=dg-1;k>=0;k--) { so[k]=si[k]; printf("%d",so[k]);

}

}

if(tx) {

printf("\n CRC code is \n"); for(k=dg-1;k>=0;k--) {

printf("%d",so[k]); msg[k]=so[k];

}

printf("\n Message to be transmitted\n"); for(i=dt;i>=0;i--)printf("%d",msg[i]);

}

if(!tx) { flag=0;

for(i=0;i<=dg-1;i++)

if(so[i]==1) {

flag=1; break;

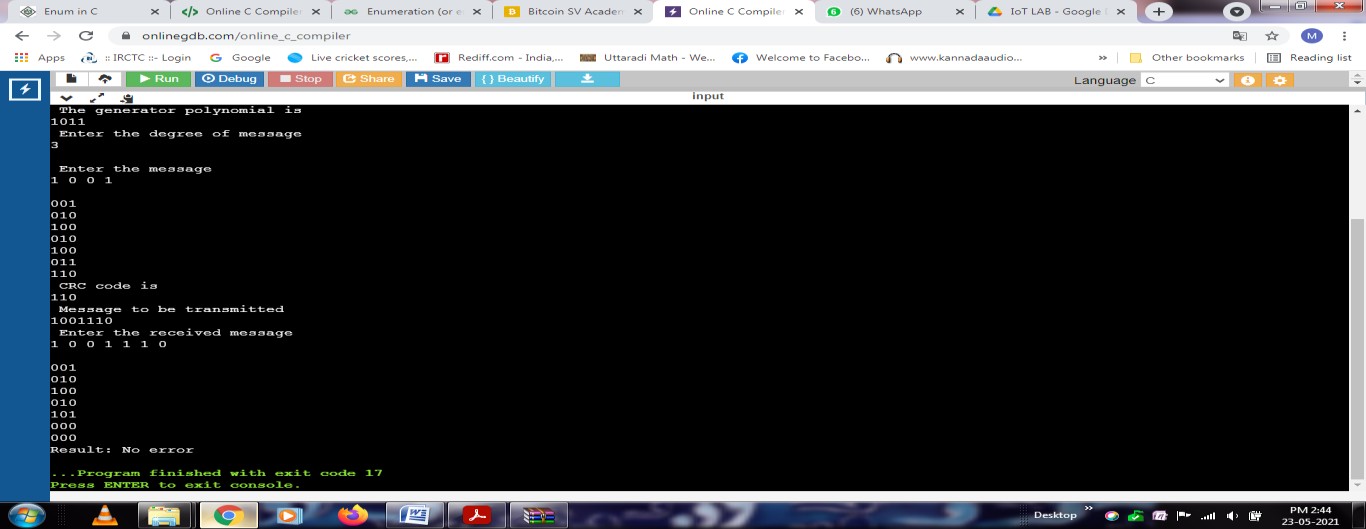
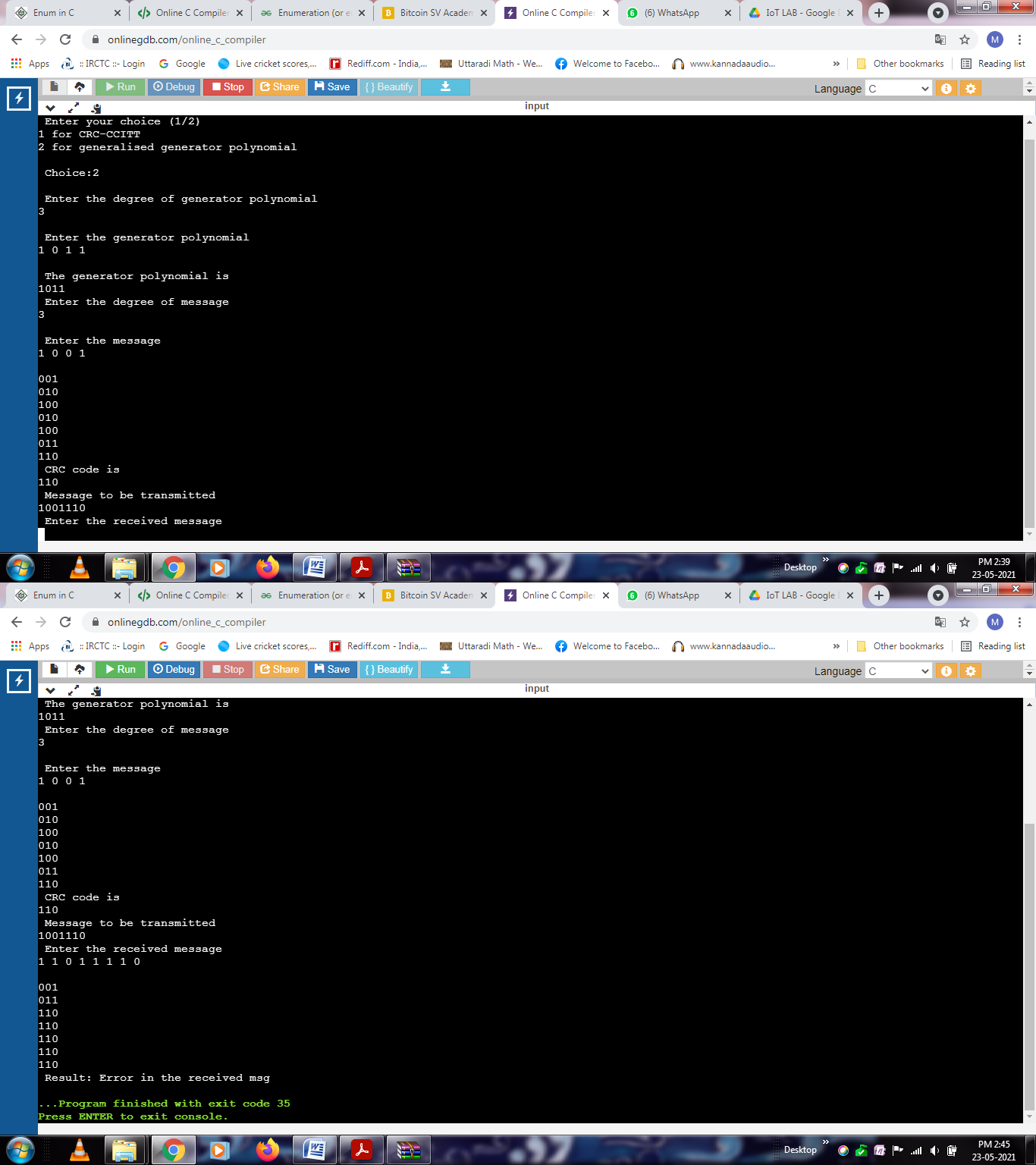
}

if(flag==0) printf("\nResult: No error");

else printf("\n Result: Error in the received msg");

}

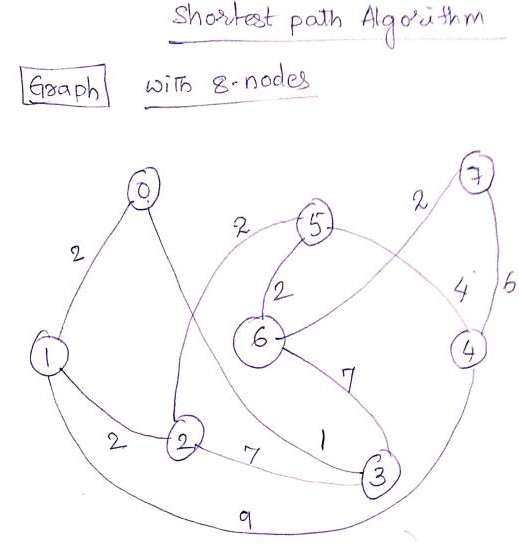
}



# Develop a program to implement a Shortest path routing algorithm for a given network graph and build a routing table for the given node.

### SHORTEST PATH ALGORITHM

1. Enter the distance matrix, which gives the distance of any given node from other nodes. [If the nodes are not directly linked, the distance between them is infinity.]
2. Enter the number of nodes, the source and destination nodes.
3. Create a structure for each node containing details of previous node, length from destination to this node and state label.
4. Start from destination node and find the shortest path to the source node. [The reason for this backward search is that, each node is labeled with its predecessor, while copying the output variable, the path is thus reversed. By reversing the search the two effects cancel and the answer is produced in the correct order.]
   1. Mark the destination node as permanent node, X.
   2. Examine each of the adjacent nodes of the destination node X and the node with the smallest distance becomes the next working node Y.
   3. All the adjacent nodes of Y are checked for least distance and the predecessor is changed for the one with shorter path.
   4. After all the adjacent nodes have been inspected and modified (if needed), entire graph is searched for the tentatively labeled node with the smallest value.
   5. The node is made permanent and becomes the working node for the next search.

#include<stdio.h> #include<stdlib.h> #define maxnode 10

#define infinity 100 int n;

int dist[8][8]=

{0,2,100,1,100,100,100,100,

2,0,2,100,9,100,100,100,

100,2,0,7,100,2,100,100,

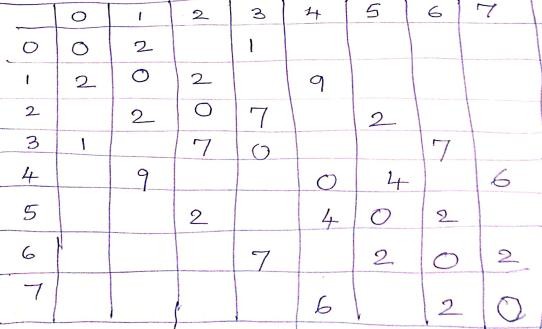
1,100,7,0,100,100,7,100,

100,9,100,100,0,4,100,6,

100,100,2,100,4,0,2,100,

100,100,100,7,100,2,0,2,

100,100,100,100,6,100,2,0};

void main(){ int i,s,j,t; void shrt();

printf("\n Enter the no of nodes"); scanf("%d",&n);

printf("\n enter the source & dest nodes"); scanf("%d%d",&s,&t);

if(s==t){

printf("\n Source is same as the destination \t Hence the Distance is 0");

exit(0);

}

shrt(s,t);

}

void shrt(int s,int t){ struct state {

int predecessor; int length;

enum{per,tent}label;

}state[maxnode];

int i,k,min; struct state \*p;

for(p=&state[0];p<&state[n];p++){ p->predecessor=-1;

p->length=infinity; p->label=tent;

}

state[t].length=0;state[t].label=per; k=t;

do {

for(i=0;i<=n;i++)

if(dist[k][i]!=0 && state[i].label==tent) { if(state[k].length+dist[k][i]<state[i].length) {

state[i].predecessor=k; state[i].length=state[k].length+dist[k][i];

}

} k=0;

min=infinity; for(i=0;i<n;i++)

if(state[i].label==tent && state[i].length<min) { min=state[i].length;

k=i;

}

state[k].label=per;

}while(k!=s); k=s;

do {

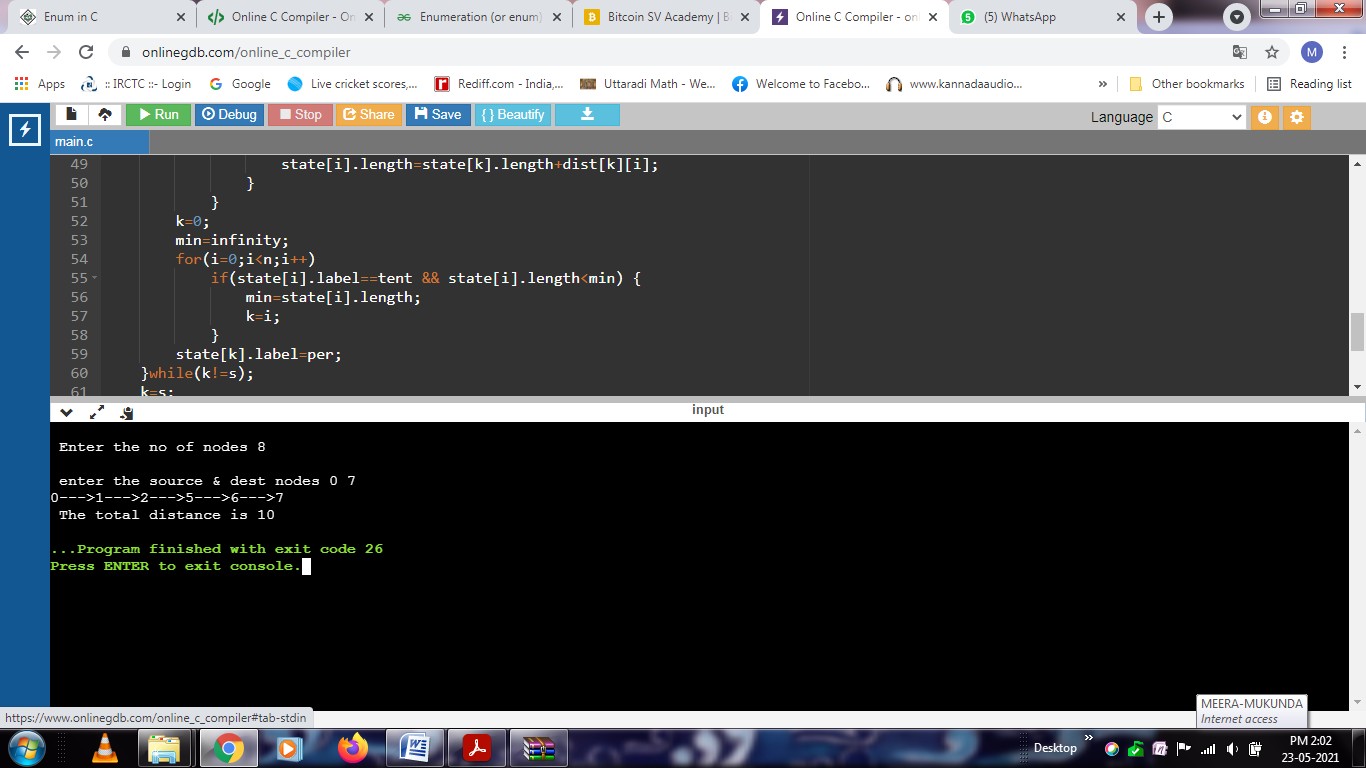
printf("%d",k);

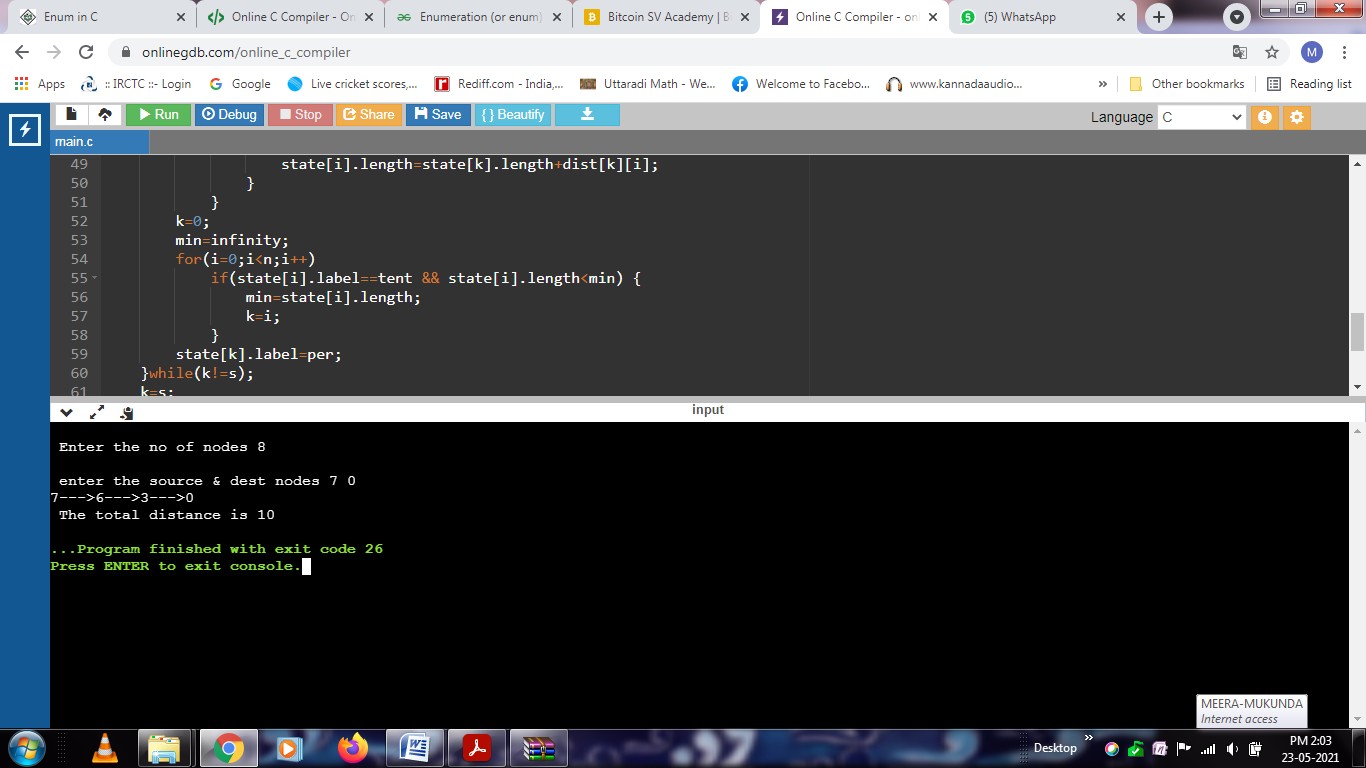
if (k!=t) printf("--->"); k=state[k].predecessor;

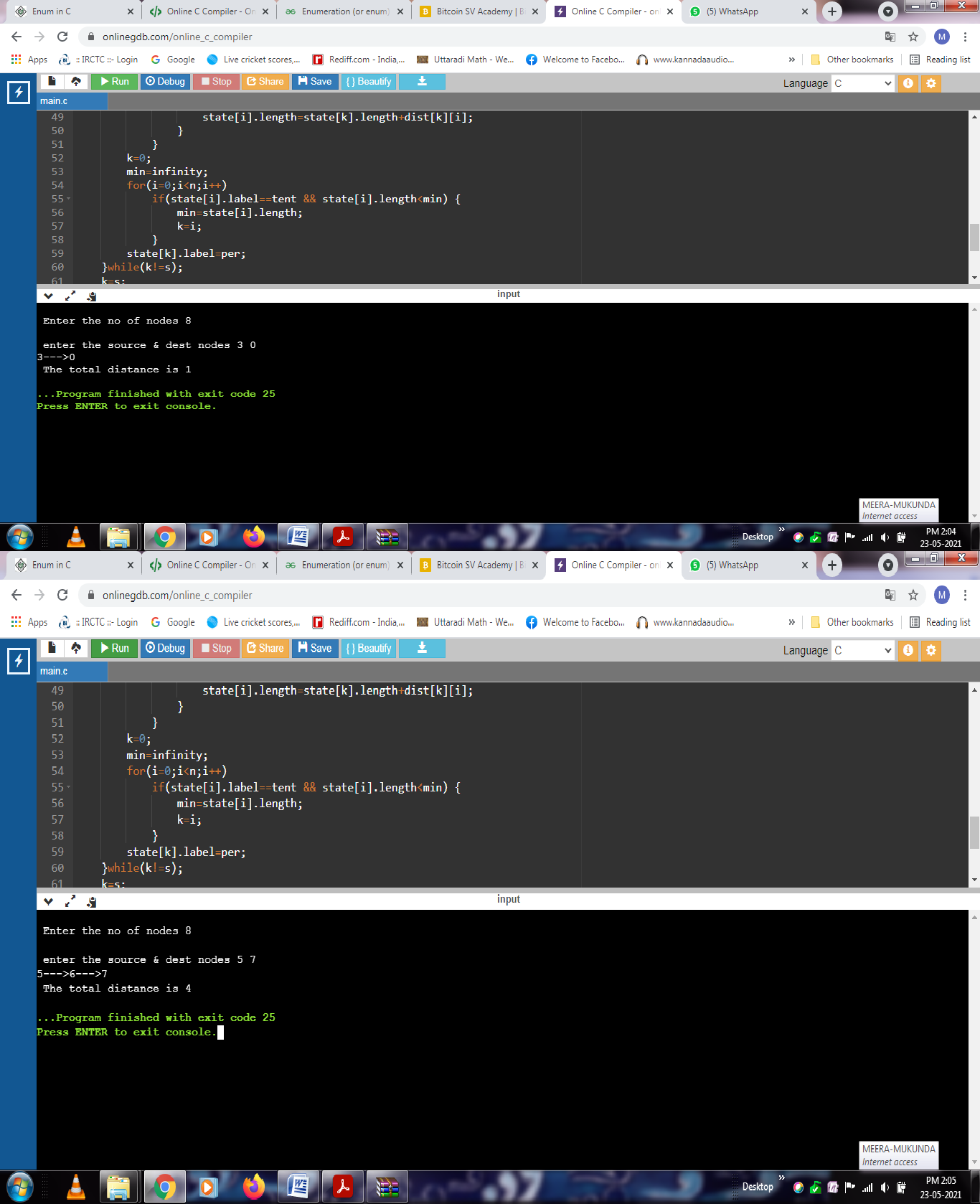
}while(k>=0);

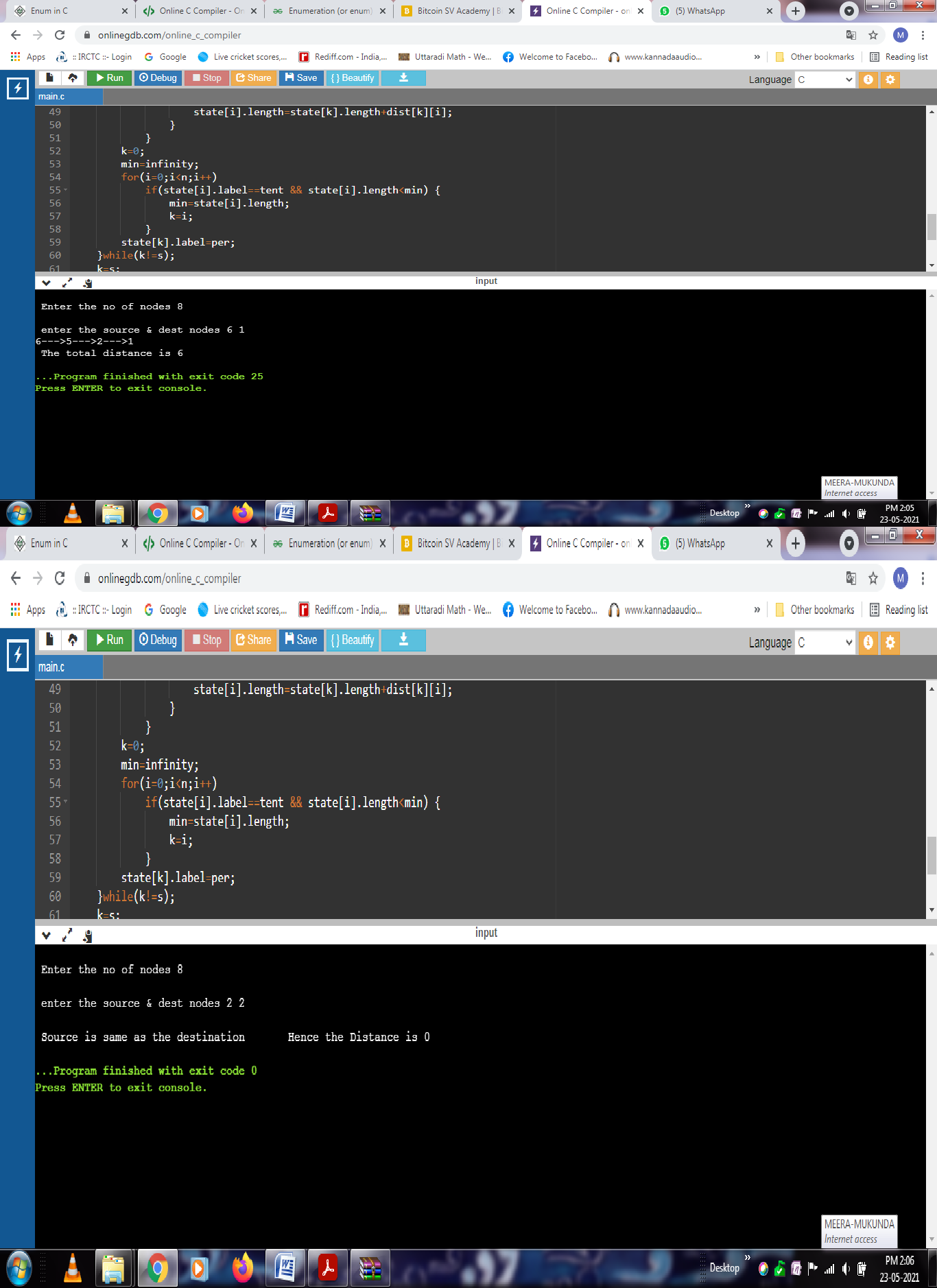
printf("\n The total distance is %d",state[s].length);

}









# Develop a C program to implement Bit stuffing and De-stuffing using HDLC standard

### Algorithm:

* Enter the message.
* Check for 5 consecutive ones, if so, append a ‘0’.
* Append header and trailer flags (both are“01111110”) and display the entire frame.
* Accept the frame.
* Sliding through the entered bit stream identify header and trailer flags.
* If flags are not found display suitable message and do not de-stuff.
* If flags are found retrieve the message.
* De-stuff as follows-
  + Check for 5 consecutive ones.
  + If so skip a bit (zero appended before) after 5 consecutive ones.
* Display the de-stuffed message.

**#include<stdio.h> #include<string.h> #include<stdlib.h>**

int main()

{

int i,j=0,count=0,temp=0,beg\_msg=0,end\_msg=0;

char msg[100],smsg[100],bsmsg[100],rmsg[100],omsg[100]; printf("\n Enter the message:\n");

scanf("%s",msg);

for(i=0,j=0;msg[i]!='\0';i++,j++){ smsg[j]=msg[i];

if(msg[i]=='1') **/\* count the no of successive ones \*/**

count++; else

count=0;

if(count==5){ **/\* stuff a zero after 5 ones \*/**

j++;

smsg[j]='0'; count=0;

}

}

smsg[j]='\0';

printf("\n The stuffed message :\n %s",smsg); strcpy(bsmsg,"01111110"); /\* add header bits \*/ strcat(bsmsg,smsg);

strcat(bsmsg,"01111110"); /\* add the tailor bits\*/ printf("\n The frame to be transmitted :\n %s",bsmsg);

**/\* destuffing \*/**

printf("\n Enter the received message :\n"); scanf("%s",rmsg);

temp=0; i=j=0;

for(i=0;rmsg[i+7]!='\0';i++){

if(rmsg[i]=='0' && rmsg[i+7]=='0'){ **/\* check for header & tailor\*/**

for(j=1;j<7;j++){

if(rmsg[i+j]=='1') temp++;

else {

temp=0; break;

}

}

if(temp==6){ **/\* if 01111110 bit stream found \*/**

if(beg\_msg==0) **/\* for the first time,msg begins \*/**

beg\_msg=i+8;

}

}

else

else{

}

end\_msg=i; **/\* for the second time msg ends \*/**

break;

temp=0;

}

if((beg\_msg==0)||(end\_msg==0)){ printf("\n framing error"); exit(0);

}

count=0;

for(i=beg\_msg,j=0;i<end\_msg;i++,j++){ omsg[j]=rmsg[i];

if(rmsg[i]=='1')

count++; **/\* check for consecutive ones \*/**

else

count=0; **/\* if 0 is found after 5 ones \*/**

if(count==5){

i++; **/\* ignore it \*/**

count=0;

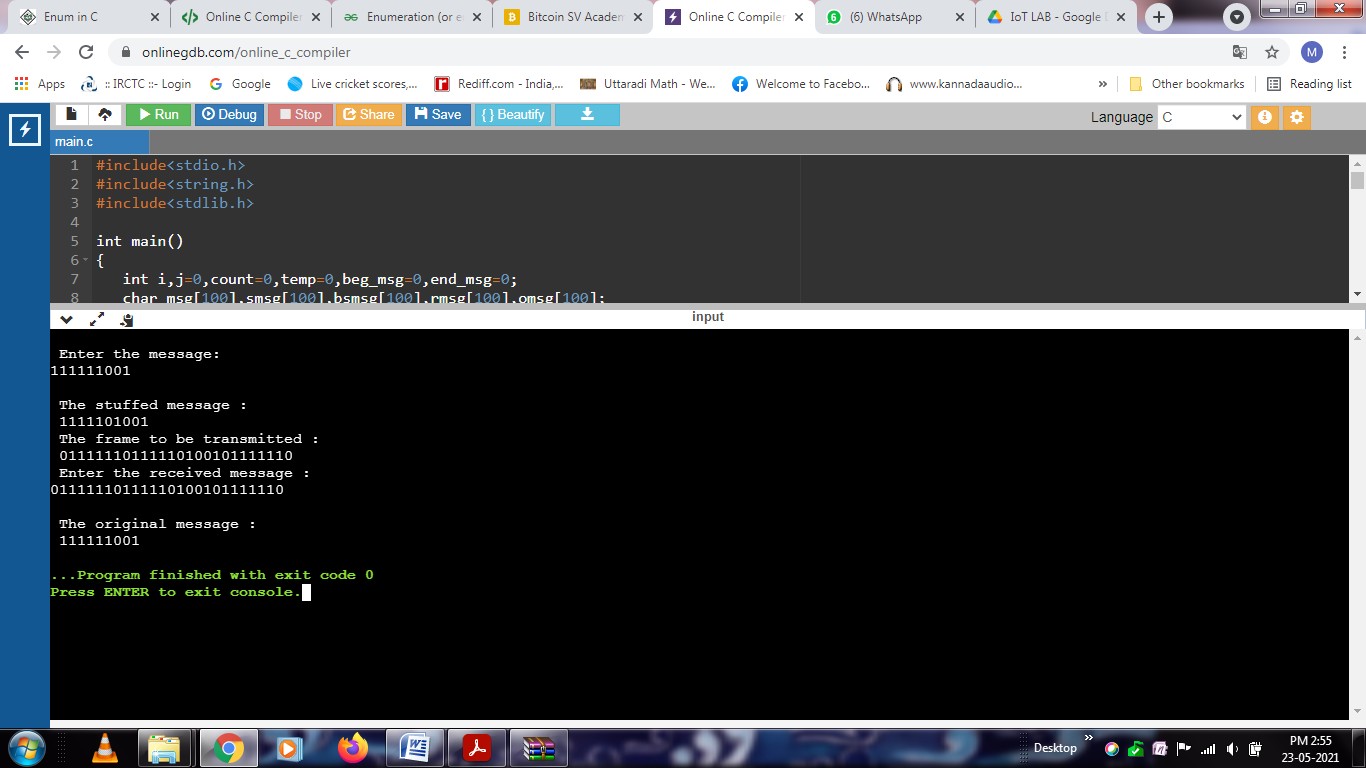
}

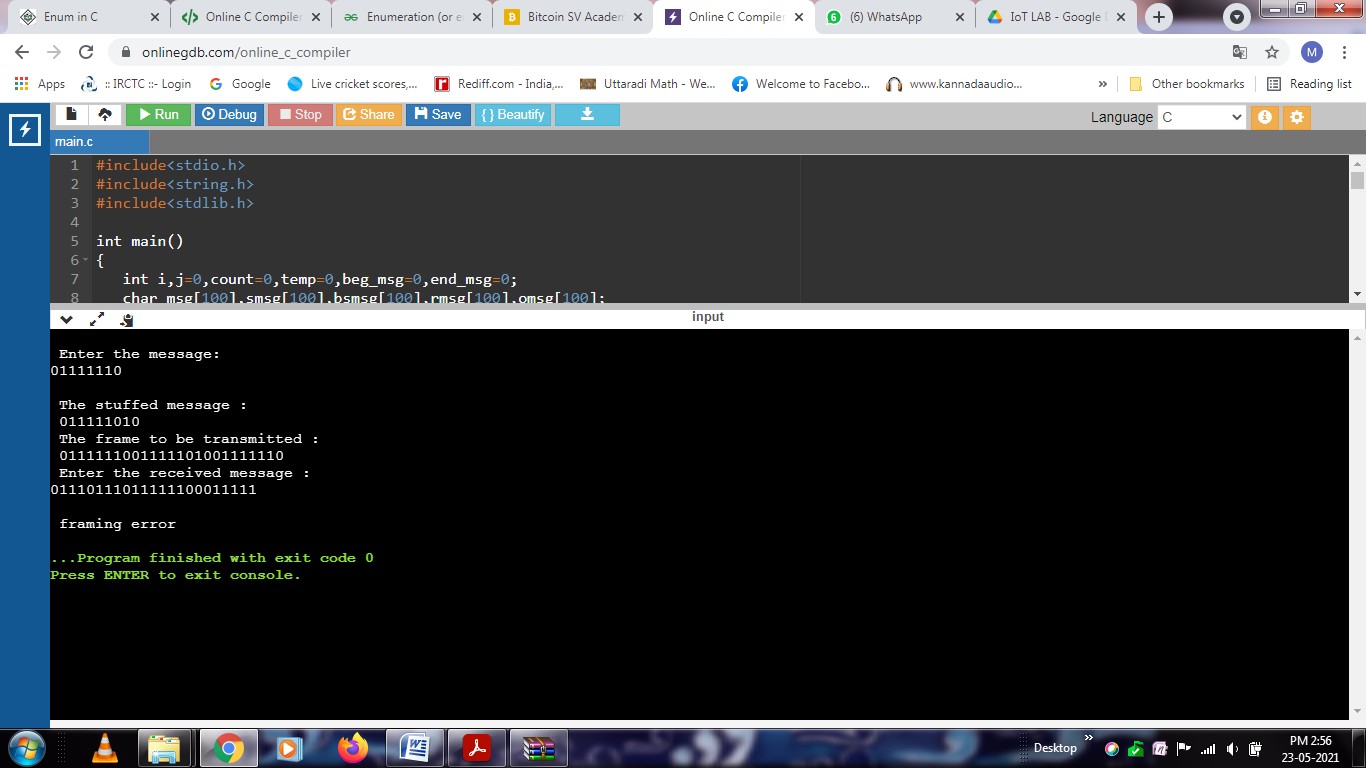
}

omsg[j]='\0';

printf("\n The original message :\n %s",omsg); return 0;

}





# Develop a C program to implement Character stuffing and De- stuffing using HDLC standard

* Enter the message.
* Check for “DLE” in the entire message.
* If found, append another “DLE”.
* Append header (DLE STX) and footer (DLEETX) and display the frame to be transmitted.
* Enter the received the message.
* Sliding through the characters entered identify the header and footer.
* If found destuff as follows.
* Remove the header and footer.
* In the message check for two consecutive “DLE”.
* If so skip one “DLE”(one appended before).
* If not found display suitable message and do not destuff.
* Display the destuffed message.

## Character Stuffing Program

**#include<stdio.h> #include<string.h> #include<stdlib.h>**

**void main()**

**{**

**int i,j=0,hfound=0,tfound=0,error=0;**

**char msg[100],smsg[100],tmsg[100],rmsg[100],dmsg[100];**

**printf("\nEnter the message:\n"); scanf("%s",msg);**

**\*(smsg)='\0';**

**\*(dmsg)='\0';**

**for(i=0,j=0;msg[i]!='\0';i++,j++){**

**if(!strncmp(msg+i,"DLE",3))**

**{**

**smsg[j]='\0'; strcat(smsg,"DLEDLE");**

**i+=2; j+=5;**

**}**

**else smsg[j]=msg[i];**

**}**

**smsg[j]='\0';**

**strcpy(tmsg,"DLESTX"); strcat(tmsg,smsg); strcat(tmsg,"DLEETX");**

**printf("\nThe stuffed message is :\n%s",tmsg);**

**/\* Destuffing \*/**

**printf("\nEnter the received message :\n"); scanf("%s",rmsg);**

**for(i=0,j=0;rmsg[i]!='\0';i++)**

**{**

**if(!strncmp(rmsg+i,"DLE",3))**

**{**

**i+=3;**

**if(!strncmp(rmsg+i,"STX",3))**

**{**

**if(!hfound) hfound=1;**

**else error=1;**

**i+=2;**

**continue;**

**}**

**else if(!strncmp(rmsg+i,"ETX",3))**

**{**

**tfound=1; break;**

**}**

**else if(!strncmp(rmsg+i,"DLE",3)); else error=1;**

**}**

**if(hfound==1)**

**{**

**dmsg[j]=rmsg[i];**

**dmsg[j+1]='\0'; j++;**

**}**

**}**

**dmsg[j]='\0';**

**if(error||!hfound||!tfound)**

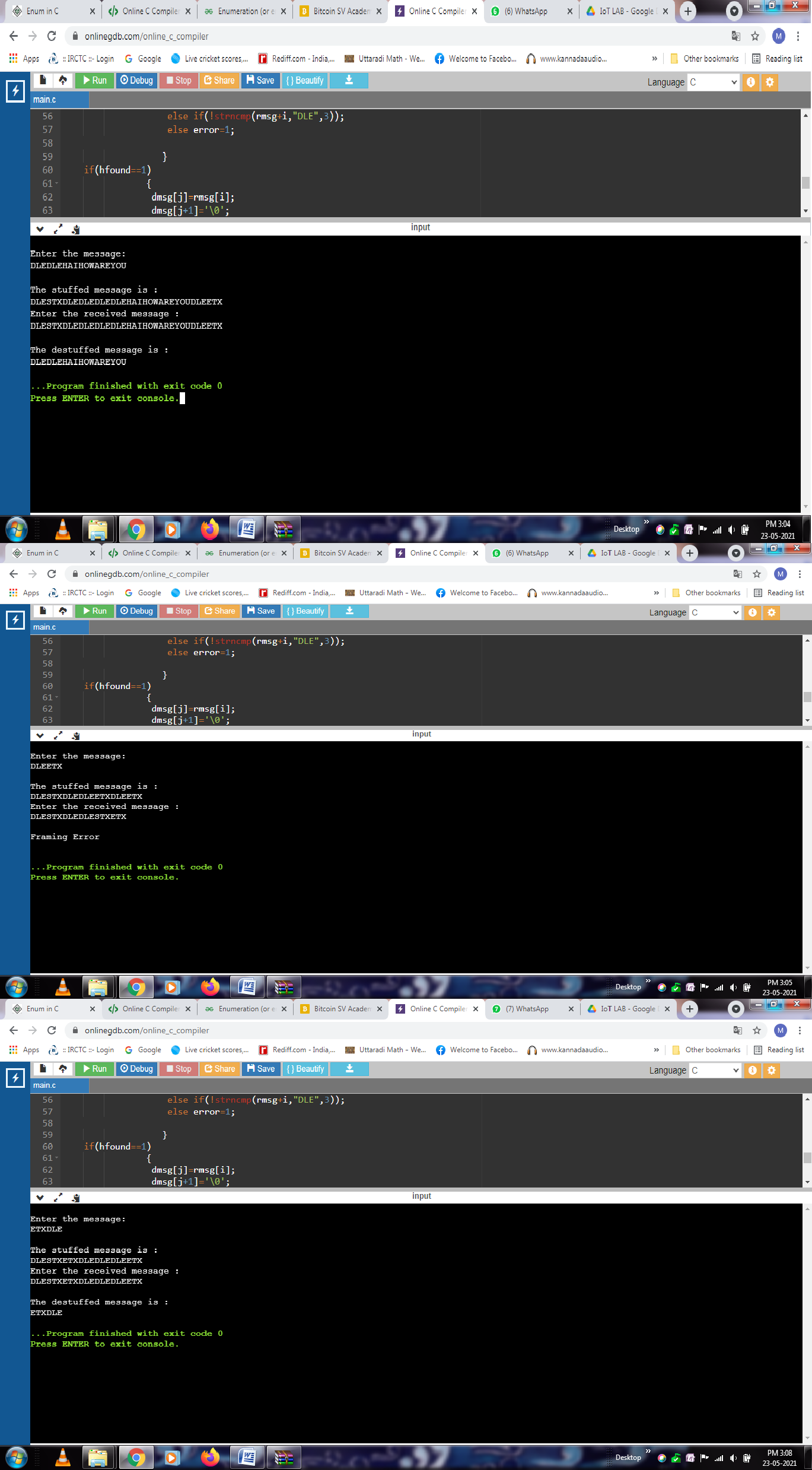
**{**

**printf("\nFraming Error\n "); exit(0);**

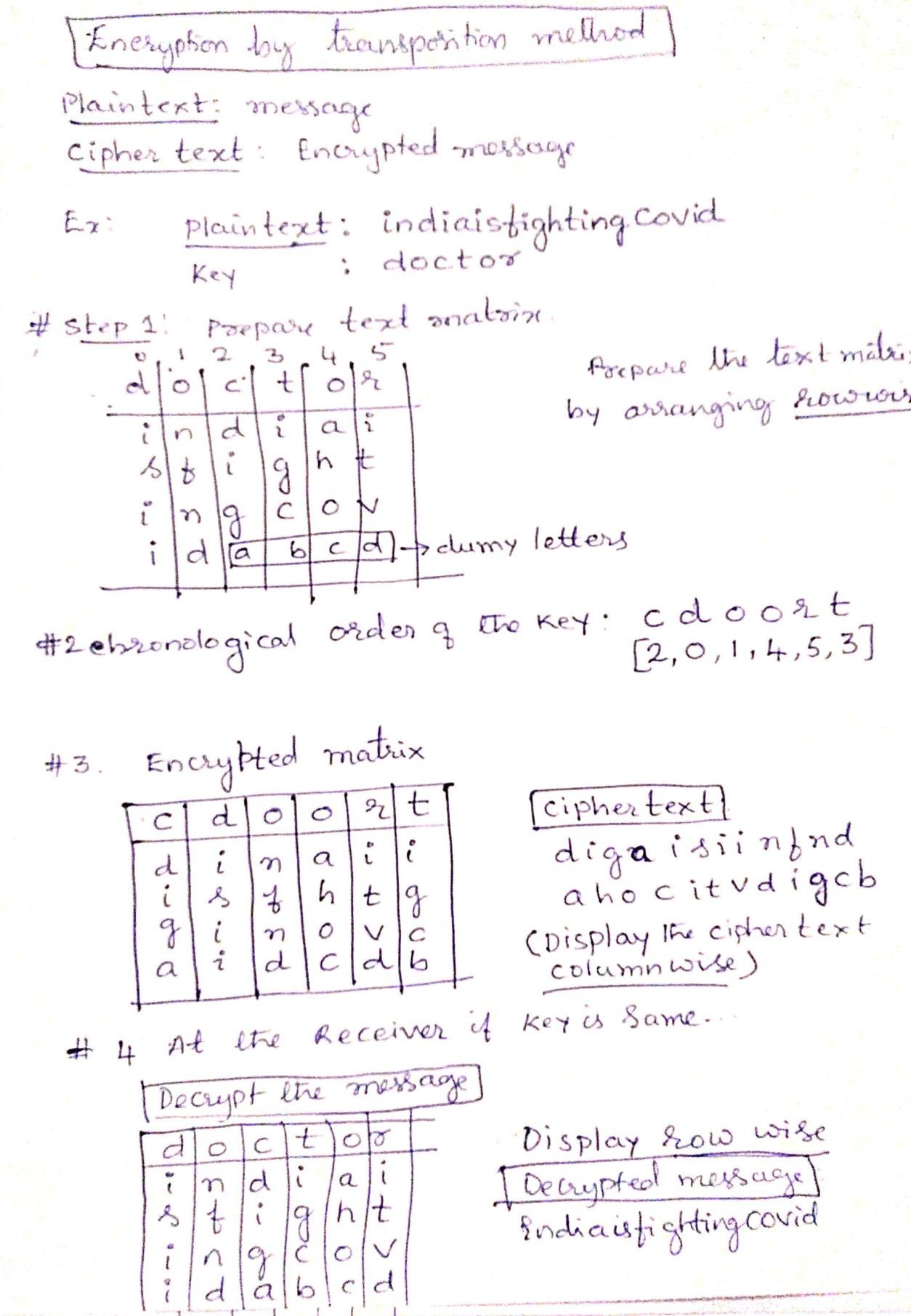
**}**

**printf("\nThe destuffed message is :\n%s",dmsg);**

**}**



# Develop a C program to implement Encryption by Transposition algorithm

⮚ 

* Get the data and keyword from user.
* Calculate the length of data and keyword.
* Obtain order matrix in which the position of each letter in the key word is stored, by alphabetical arrangement
* Arrange the plain text in the form of a matrix (encryption matrix) as follows-
* Number of columns is equal to the keyword length.
* Divide the message length by keyword length.
* If remainder is non-zero, number of rows is one more than the quotient.
* Copy the plain text into the encryption matrix row wise.
* Append redundant characters in unfilled columns (a, b...)
* The encrypted message is obtained by reading text column wise from the encrypted matrix, whose order is specified in the order matrix
* Decryption
* Get keyword and length of message from user.
* Compare with the original keyword, if not matched, display a message and continue if requested.
* If matched decrypt the received message by reading it row wise, neglecting the redundant characters.

**#include<stdio.h> #include<string.h> #include<conio.h> #include<stdlib.h> void main()**

**{**

**char txt[50],kw[10],kw1[10],temp,txt1[10][10],encr[10][10],decr[10][10]; int lenm,lenk,order[10],i,j,k,temp1,c=0,r,b,count;**

**printf("\nEnter the message to be enrypted: "); scanf("%s",txt);**

**lenm=strlen(txt);**

**printf("\nEnter the keyword: "); scanf("%s",kw); lenk=strlen(kw);**

**strcpy(kw1,kw); for(i=0;i<lenk;i++)**

**order[i]=i;**

**//sort the keyword**

**for(i=0;i<lenk;i++)**

**for(j=0;j<lenk;j++) { if(kw1[j]>kw1[i]) {**

**temp=kw1[j]; kw1[j]=kw1[i]; kw1[i]=temp;**

**temp1=order[j]; order[j]=order[i]; order[i]=temp1;**

**}**

**}**

**printf("\n Order of letters after sorting: "); for(i=0;i<lenk;i++)**

**printf("%d ",order[i]); r=lenm/lenk; //no. of rows**

**b=lenm%lenk; if(b!=0) r++;**

**//convert message to matrix count=0;**

**for(i=0;i<r;i++) {**

**for(j=0;j<lenk && count<=lenm;j++) txt1[i][j]=txt[count++];**

**if(count>lenm) {**

**while(b!=lenk) {**

**txt1[i][b]='a'+c++; b++;**

**}**

**}**

**}**

**//encryption**

**printf("\nThe encrypted message :\n"); for(k=0;k<lenk;k++) {**

**for(i=0;i<r;i++) {**

**j=order[k]; encr[i][k]=txt1[i][j];**

**printf("%c",encr[i][k]);**

**}**

**}**

**//at the receiver printf("\nEnter the keyword: "); scanf("%s",kw1);**

**if(strcmp(kw,kw1)) {**

**printf("Wrong Key!!");**

**exit(0);**

**}**

**//decryption**

**for(i=0;i<lenk;i++) {**

**j=order[i]; for(k=0;k<r;k++) decr[k][j]=encr[k][i];**

**}**

**printf("\n The original message is: "); for(i=0;i<r;i++)**

**for(j=0;j<lenk;j++) {**

**if(((i\*lenk)+j)==lenm){**

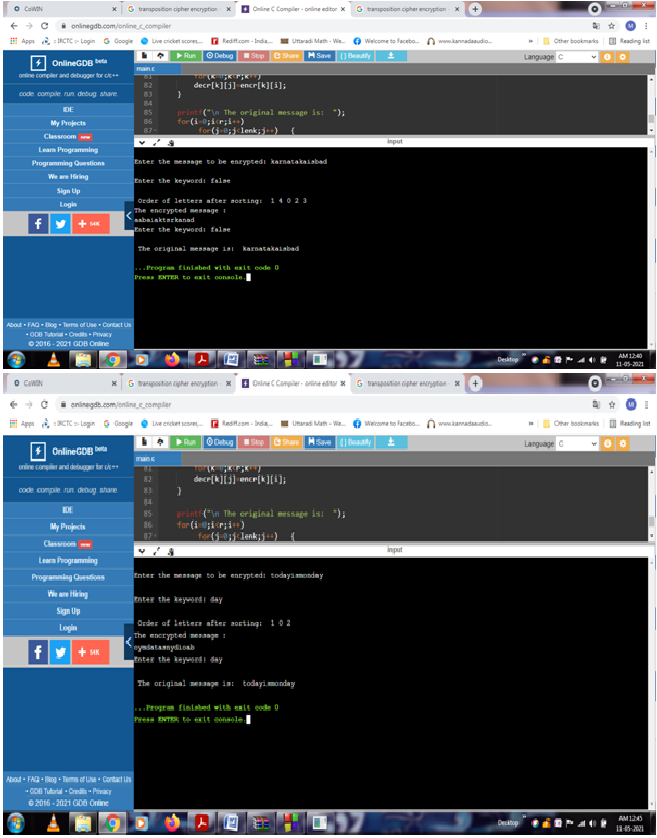
**break;**

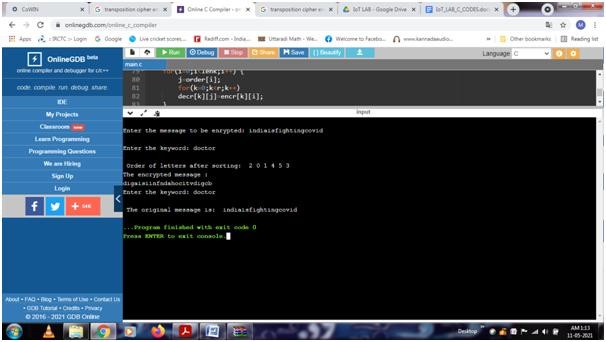
**}**

**printf("%c",decr[i][j]);**

**}**

**}**





# Develop a C program to implement Encryption by Substitution algorithm

**Algorithm:**

* Get the message.
* Replace each character in the plain text by another character at an offset ‘KEY\_SHIFT(3)’ such that the encrypted message has characters in the specified range (English alphabets both upper and lower case in circular method).
* Transmit the message (display the encrypted message).
* Receive the encrypted message.
* Replace each character in the received text by removing the offset ‘KEY\_SHIFT(3)’.
* Display the decrypted message.

**/\* CODE \*/**

**#include<stdio.h> #include<string.h> void main()**

**{**

**char msg[100],encr[100],decr[100],rec[100];**

**int i;**

**printf("\n Enter the message for encryption:\n "); scanf("%s",msg);**

**for(i=0;\*(msg+i)!='\0';i++) {**

**if((\*(msg+i)>='a' && \*(msg+i)<='w')||(\*(msg+i)>='A' && \*(msg+i)<='W'))**

**\*(encr+i)=\*(msg+i)+3; else**

**if((\*(msg+i)>='x' && \*(msg+i)<='z')||(\*(msg+i)>='X' && \*(msg+i)<='Z'))**

**\*(encr+i)=\*(msg+i)+3-26; else**

**\*(encr+i)=\*(msg+i)+3;**

**}**

**\*(encr+i)='\0';**

**printf("\n Encrypted message: %s\n",encr);**

**/\* Decryption \*/**

**printf("\n Enter the message for decryption:\n "); scanf("%s",rec);**

**for(i=0;\*(rec+i)!='\0';i++) {**

**if((\*(rec+i)>='d' && \*(rec+i)<='z')||(\*(rec+i)>='D' && \*(rec+i)<='Z'))**

**\*(decr+i)=\*(rec+i)-3;**

**else**

**if((\*(rec+i)>='a' && \*(rec+i)<='c')||(\*(rec+i)>='A' && \*(rec+i)<='C'))**

**\*(decr+i)=\*(rec+i)-3+26;**

**else**

**}**

**\*(decr+i)=\*(rec+i)-3;**

**\*(decr+i)='\0';**

**printf("\n The decrypted message:\n %s",decr);**

**}**

