0.1 Main program

```
//**THIS PORGRAM DOES THE MONTE-CARLO SIMULATION OF ISING MODEL ON 128*128 LATTICE
//*USING MATROPOLIS ALGORITHM

// row,col =  THE POSITION OF THE SITE WITH RESPECT TO THE TOP LEFT
// N = ORDER OF SPIN MATRIX IS N*N
// s[N][N] = N*N SPIN MATRIX
// H  = MAGNETIC FIELD STRENGTH
// T = TEMPRATURE
// beta = INVERSE TEMPRATURE




#include<iostream>
#include<iomanip>
#include<math.h>
#include"functions.cpp"
#include<fstream>
#include<cstdlib>
using namespace std;
int main()
{
    int N=128,J=1,row,col;
    int s[N][N],no_ensambles=500,mag;
    double T=0,cv,var;
    double sumE=0,sumsqE=0;
    double E,sqE,H=0.01,norm;
    norm = 1.0*N*N*no_ensambles;

    //TO write the data into text files
    ofstream data,data1,data2;
data.open("magnetization_data.txt"); data1.open("energy_data.txt"); data2.open("cv_data.txt"

    initial(*s,N,1);        //INITIALIZATION OF SPIN MATRIX

    for(T =0.05;T<5;T=T+0.1)  //Temprature loop
    {
        mag=0;sumE=0;sumsqE=0;
        for(int en_count=0;en_count<no_ensambles;en_count++)  //ENSAMBLE LOOP
        {
    for(int rep=0;rep<5000;rep++)  // 5000 metroplis flip steps for randomly chosen cites
            {
                row = (float(rand())/float(RAND_MAX))*N ;
                col =(float(rand())/float(RAND_MAX))*N ;
                flip(row,col,N,1.0/T,*s,J,H);
            }
            mag = mag + abs(magnetization(*s,N)); //magnetisation for the ensamble
      energy (*s,N,J,E,H);   //energy for the ensamble is stored in variable "energy"
            sumE   = E+sumE;
            sumsqE = E*E+sumsqE;
        }

        cout<<"  loading........"<<T<<endl;
        sumsqE = sumsqE/double (no_ensambles);
        sumE = sumE/ double (no_ensambles);
        var  = sumsqE - sumE*sumE;
        cv = var/(T*T);
        data2 <<T<<"   "<<cv/double(N*N)<<endl;
        data1 <<T<<"   "<<sumE/double(N*N)<<endl;
        data<< T <<"   "<< double(mag)/norm<<endl;
    }
        data.close();
        data1.close();
        data2.close();
    return 0;
}
```

0.2 Functions

```
#include<iomanip>
#include<math.h>
#include<time.h>

using namespace std;


void initial ( int *s , int N ,int n)    //INTILIAZES THE SPIN MATRIX
{
    if(n==0)
    {
        for(int i=0;i<N;i++)
            for(int j=0;j<N;j++)
                s[i*N+j] = (rand()%2)*2-1;
    }

    else if(n==1)
    {
        for(int i=0;i<N;i++)
            for(int j=0;j<N;j++)
                s[i*(N)+j]=1;
    }
    else cout<<"ERROR : "<<endl;
}

void print( int *s , int N )   //TO PRINT THE MATRIX IF NEEDED
{
    for(int i=0;i<N;i++)
    {
        for(int j=0;j<N;j++)
        {
            cout<<setw(20)<<s[i*(N)+j];
        }   cout<<endl;
    }   cout<<endl;
}

//********THIS FUNCTION DOES ONE FLIP OF SPIN ACCORDING THE MATROPOLIS ALGORITHM*******
// row,col =  THE POSITION OF THE SITE WITH RESPECT TO THE TOP LEFT
// N = ORDER OF SPIN MATRIX IS N*N
// *s = POINTER TO THE SPIN MATRIX
// H  = MAGNETIX FIELD STRENGTH
// beta = INVERSE TEMPRATURE

void flip(  int row , int col ,int N,float beta , int *s, int j, double H)
{
    //dir matrix holds coordinates for 4 nearest neigbours in order {up,down,left,right}
    int dE=0,dir[] = {row-1,col,row+1,col,row,col-1,row,col+1};
    float r;
    for(int i=0;i<8;i++) //circular boundary condition
    {
        if( dir[i]>N-1)
            dir[i]=0;
        if(  dir[i]<0 )
            dir[i]=N-1;
    }

    for(int i=0;i<8;i = i+2 ) //change in energy of one site in changing the spin
    {
        dE = dE+ s[dir[i]*(N) +dir[i+1]] ;
    }   dE = 2*j*s[row*(N)+col]*dE;
        dE = dE + 2*H*s[row*(N)+col];

    if(dE<=0)
        s[row*(N)+col] = -s[row*(N)+col];
    else
    {
        r  = float(rand())/float(RAND_MAX);
       if( r <= exp(-beta*dE))
            s[row*(N)+col] = -s[row*(N)+col];
    }
}

//******THIS PROGRAM RETURNS THE TOTAL MAGNETIZATION OF THE SPIN MATRIX**************
int magnetization( int *s, int N)
{
    int m=0;
    for(int i=0;i<N;i++)
        for(int j=0;j<N;j++)
            m = m +s[i*(N)+j];
    return m;
}

//******THIS FUNCTION RETURNS THE TOTAL ENERGY OF THE CONFIGURATION*********************
// N = ORDER OF SPIN MATRIX IS N*N
// *s = POINTER TO THE SPIN MATRIX
// H  = MAGNETIX FIELD STRENGTH
void energy(int *s , int N ,int j,double &E,double H)
{
    E=0;

    for(int row=0;row<N;row++)
    {
        for(int col=0;col<N;col++)
        {
            int dir[] = {row-1,col,row+1,col,row,col-1,row,col+1};
            for(int k=0;k<8;k=k+2)
            {
                for(int i=0;i<8;i++) //ENERGY OF ONE SITE WITH 4 NEIGBOURS
                {
                    if( dir[i]>N-1)
                        dir[i]=0;
                    if(  dir[i]<0 )
                        dir[i]=N-1;
                }
               E = E- j*s[row*(N)+col]*s[dir[k]*(N) +dir[k+1]];
             }
        }

    }
    E= E/2 ;

    for(int row=0;row<N;row++)
    {
        for(int col=0;col<N;col++)
        {
            E = E - H*s[row*(N)+col];
        }

    }
}

void copy (int *s , int *scopy,int N)   //TO GET THE COPY OF MATRIX
{
    for(int row=0;row<N;row++)
    {
        for(int col=0;col<N;col++)
        {
           scopy[row*(N)+col] = s[row*(N)+col];
        }
    }
}

int matmul( int *A , int *B,int N )   //RETURNS trace( A.transpose(B))
{
    int trace=0;

    for(int i=0;i<N;i++)
    {
        for(int k=0;k<N;k++)
        {
            trace = trace + A[i*N+k]*B[i*N+k];
        }
    }

return trace;
}
```

0.3 Autocorrelation

```
#include<iostream>
#include<iomanip>
#include<math.h>
#include"functions.cpp"
#include<fstream>
#include<cstdlib>
using namespace std;

int main()
{
    int N=128,J=1,row,col;
    int s[N][N],no_ensambles=500,A[N][N],niter;
    double T=2.5;
    double H=0.01,acr;


    ofstream data;
    data.open("autocorrelation.txt");

   for(long int i = 1;i<1e6;i = i*2)
```

```
    {
            initial(*s,N,0);
            copy (*s,*A,N);
            for(int rep=0;rep<i;rep++)
             {
                 row = (float(rand())/float(RAND_MAX))*N ;
                 col =(float(rand())/float(RAND_MAX))*N ;
                 flip(row,col,N,1.0/T,*s,J,H);
             }
 acr = double(matmul(*A,*s,N))/sqrt((double(matmul(*s,*s,N))*double(matmul(*A,*A,N)) ));
             data<<i<<" "<<acr<<endl;
    }
    data.close();
    return 0;
}
```