```
/* Code for Fault Diagnosis Using Back Propagation of ANN*/
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <unistd.h>

//defining variables

double input[8];
double output [8];
double hidden [11];

double weights1[8][11];
double weights2[11][8];
double prevweights1[8][11];
double prevweights2[11][8];
int loc;
double error[8];
double normal[8] ={ 390,200,100,1490,483,200,150,490};
double tempcopy[8];
double maxi;
double mini;
double* ninput;
double errorb=.1;

double desiredN[8]={1,1,1,1,1,1,1,1};
double desired1[8]={0,1,1,1,1,1,1,1};
double desired2[8]={1,0,1,1,1,1,1,1};
double desired3[8]={1,1,0,1,1,1,1,1};
double desired4[8]={1,1,1,0,1,1,1,1};
double desired5[8]={1,1,1,1,0,1,1,1};
double desired6[8]={1,1,1,1,1,0,1,1};
double desired7[8]={1,1,1,1,1,1,0,1};
double desired8[8]={1,1,1,1,1,1,1,0};
double llim;

//this is for normalization

double max( double a[5])
{
    int i=0;
    double m;
    m=a[i];
    for (i=0;i<5;i++)
    {
        if (a[i]> m)
        {
            m=a[i];
        }
    }
    return m;
}

//this is for normalizaton too

double min( double a[5])
{
    int i=0;
    double m;
    m=a[i];
    for (i=0;i<5;i++)
    {
        if (a[i]< m)
        {
            m=a[i];
        }
    }
    return m;
}


void normalize()
{
    int i;
```

```
    maxi=max(tempcopy);
    mini=min (tempcopy);
    for (i=0; i<8;i++)
    {
        tempcopy[i]=(tempcopy[i]-mini)/ (maxi-mini);
    }
}

void bpa(double desired[8])
{
    int t,mfact,i,j;
    double neta = 0.7;

    for (t=0;t<10;t++)
    {
        mfact=t*5;
        tempcopy[1]=llim;
        for (i=1;i<=8;i++)
        {
            tempcopy [i]=normal[i];
        }
        normalize();
        ninput=tempcopy;
        for ( i=0;i<11;i++)
        {
            for (j=0;j<8;j++)
            {
                hidden[i]=hidden[i] +weights1[j][i]*ninput[j];
            }
        }

        for ( i=0; i<11;i++)
        {
            hidden [i]=1/(1 + exp(-hidden[i]));
        }

        for (i=0;i<8;i++)
        {
            for (j=0;j<11;j++)
            {
                output[i]=output[i]+ weights2[j][i]* hidden[j];
            }
        }

        for (i=0;i<8;i++)
        {
            output[i]=1/ (1+ exp (-output[i]));
        }

        //calculating delta j's (for an output node)

        for (i=0; i<8; i++)
        {
            error[i]=output[i]*(1-output[i])*(desired[i]-output[i]);
        }

        for (i=0;i<8;i++)
        {
            for (j=0;j<11;j++)
            {
                prevweights1[i][j]=weights1[i][j];
            }
        }

        for (i=0;i<11;i++)
        {
            for (j=0;j<8;j++)
            {
                prevweights2[i][j]=weights2[i][j];
            }
        }

        for (i=0;i<8;i++)
        {
            for (j=0;j<11;j++)
            {
```

```
                prevweights2[i][j]=prevweights2[i][j] + neta*error[j]*output[j];
            }
        }

    }

}



void main()
{
    int i,j,k;
    system("cls");
    //step 1 of the bpa : initializing the weight s to small random values
    for (i=0; i<5; i++)
    {
        for (j=0; j<7;j++)
        {
            k = rand();
            weights1[i][j]= (k % 100) / 100;
            prevweights1[i][j]=weights1[i][j];
        }
    }

    for (i=0; i<7; i++)
    {
        for (j=0; j<5;j++)
        {
            k = rand();
            weights2[i][j]= ( k % 100 ) / 100;
            prevweights2[i][j]=weights1[i][j];
        }
    }


    cout << "TRAINING THE NETWORK\n";
    cout << flush;
//  fflush(stdout);
// steps 2,3,4 of the bpa : presenting the inputs and the desired outputs, calculating actual out
puts & adapting weights



    do
    {

        llim=normal[1]-150;
        bpa(desired1);
        llim=normal[1]+50;
        bpa(desired1);
        llim=normal[2]-150;
        bpa(desired2);
        llim=normal[2]+30;
        sleep(5);
        bpa(desired2);
        llim=normal[3]-70;
        bpa(desired3);
        llim=normal[3]+30;
        bpa(desired3);
        llim=normal[4]-490;
        bpa(desired4);
        llim=normal[4]+150;
        bpa(desired4);
        llim=normal[5]-300;
        bpa(desired5);
        llim=normal[5]+50;
        bpa(desired5);
        llim=normal[6]-100;
        bpa(desired6);
        llim=normal[6]+20;
        bpa(desired6);
        llim=normal[7]-100;
        bpa(desired7);
        llim=normal[7]+20;
```

```
        bpa(desired7);
        llim=normal[8]-150;
        bpa(desired8);
        llim=normal[8]+30;
        bpa(desired8);

        //check to see if all errors are under .1

        for (i=0;i<8;i++)
        {
            if (error[i]>.1) errorb=.11;
        }
    }

    while (errorb > .1);


        cout << "Training Complete\n";
    cout << "Enter the values for the Heat Exchanger CSTR system \n";
        cout << "Enter values in the order(Thin,Thout,Tcin,Fh,Fc,Cin,Tj,Fj)\n";
        cout << "The normal values are (390,200,100,1490,483,.200,150,490) \n";

    for (i=0; i<8; i++)
    {
        cin>>input[i];
                input[5]= input[5]* 1000;
        if (input[i] != normal[i])
        {
            desiredN[i]=0;
            loc=i+1;
        }
    }

    cout << "Fault Diagnosed is : F"<<loc<<" \n";
    cout <<"Output Matrix is : [ ";
    for (i=0; i<8; i++)
    {
        cout << desiredN[i]<<" ";
    }
    cout << "]\n";
}
//end of main
```