```c
/*questo codice definisce una rete (percettrone multistrato) composta da tre layers
 * il primo ha due neuroni, il secondo tre e l ultimo uno; la funzione
 di trasferimento e' la sigmoide*/

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

// Precisione dei valori
typedef double T_Precision;

// Struttura di un neurone
struct Neuron
{
 T_Precision value;// Uscita del percettrone
};
struct Neuron *i_neuron, *h_neuron, *o_neuron;//variabili di tipo Neuron


// Struttura di una connessione tra due neuroni (sinapsi)
struct Synapse
{
 T_Precision weight; // Peso della connessione
};
struct Synapse *h_synapse, *o_synapse;//variabili di tipo Synapse



float sigmoid(float x);//dichiarazione funzione sigmoide

float d_sigmoid(float x);//dichiarazione funzione derivata della sigmoide

void run_network( struct Neuron *i_neuron, size_t input_size,    //dichiarazione funzione che
fa operare la rete di 3 strati,calcola le uscite di
struct Neuron *h_neuron, struct Synapse *h_synapse, size_t hidden_size, //ogni strato, un
percettrone alla volta, fino a determinare le uscite
struct Neuron *o_neuron, struct Synapse *o_synapse, size_t output_size );//della rete



 int main( void )

{
 // Dimensioni della rete
 const size_t input_size = 2;
 const size_t hidden_size = 3;
 const size_t output_size = 1;

 // Creo i neuroni degli strati
struct Neuron i_neuron[ input_size ];
struct Neuron h_neuron[ hidden_size ];
struct Neuron o_neuron[ output_size ];

 // Creo le sinapsi degli strati (+ quelle del bias)
struct Synapse h_synapse[ (input_size + 1) * hidden_size ];
struct Synapse o_synapse[ (hidden_size + 1) * output_size ];

// Imposto i pesi sinaptici
h_synapse[0].weight = -3.9340242026; //sinapsi da i a h
h_synapse[1].weight = 8.2505052824;
h_synapse[2].weight = 0.3043062447;  //peso del bias
```

```
h_synapse[3].weight = 6.8427240303;
h_synapse[4].weight = 6.8347133644;
h_synapse[5].weight = -1.4695877520;

h_synapse[6].weight = 8.2647545506;
h_synapse[7].weight = -4.1937195208;
h_synapse[8].weight = 0.5536933684;

o_synapse[0].weight = -12.2588589492; //sinapsi da h a o
o_synapse[1].weight = 13.6915483842;
o_synapse[2].weight = -12.2575228405;
o_synapse[3].weight = 4.8756901299;

// Iteratore
size_t i, j;

// Stampo l'intestazione della tabella
printf( "[x1]\t[x2]\t[y]\n" );


// Provo tutte le combinazioni degli ingressi
for ( j = 0; j <= 1; j++ )
    {

    for ( i = 0; i <= 1; i++ )
        {

        // Imposto gli ingressi
        i_neuron[0].value = i;
        i_neuron[1].value = j;

        // Eseguo la rete neurale la funzione mi calcola le uscite dei neuroni dei vari strati
        run_network( i_neuron, input_size,
                h_neuron, h_synapse, hidden_size,
                o_neuron, o_synapse, output_size );

        // Stampo gli ingressi e la rispettiva uscita
        printf( "%.0f\t%.0f\t%.0f\n", i_neuron[0].value, i_neuron[1].value,o_neuron[0].value);
             }
     }


    return 0;
}



float sigmoid(float x)//definizione funzione sigmoide
{
     float exp_value;
     float return_value;

     /*** Exponential calculation ***/
     exp_value = exp((double) -x);

     /*** Final sigmoid value ***/
     return_value = 1 / (1 + exp_value);

     return return_value;
}
```

```c
float d_sigmoid(float x)//definizione funzione derivata della sigmoide
{
     float exp_value;
     float return_value;

     /*** Exponential calculation ***/
     exp_value = exp((double) -x);

     /*** Final sigmoid value ***/
     return_value = (1 / (1 + exp_value))*(1-(1 / (1 + exp_value)));


     return return_value;
}



void run_network(struct Neuron *i_neuron, size_t input_size,    //definizione della funzione
che fa operare la rete di 3 strati,calcola le uscite di
struct Neuron *h_neuron,struct Synapse *h_synapse, size_t hidden_size, //ogni strato, un
percettrone alla volta, fino a determinare le uscite
struct Neuron *o_neuron, struct Synapse *o_synapse, size_t output_size )//della rete

{

 // Potenziale di attivazione
 float potential;
 // Iteratori
 size_t j, i;

    // Calcolo le uscite dello strato intermedio hidden
    for ( i = 0; i < hidden_size; i++ )

        {
            // Azzero il potenziale di attivazione
            potential = 0;
                // Calcolo il potenziale di attivazione
                    for ( j = 0; j < input_size; j++ )
                    {
                        potential += i_neuron[j].value * h_synapse[i * ( input_size + 1 ) +
                    j].weight;
                    }
            // Aggiungo il valore del bias
            potential += h_synapse[i * ( input_size + 1 ) + j].weight;

            // Calcolo l'uscita del neurone sulla base del potenziale di attivazione
            h_neuron[i].value = sigmoid( potential ); //uscita del generico neurone nello
                strato hidden
        }



    // Calcolo le uscite dello strato di uscita
    for ( i = 0; i < output_size; i++ )

        {
            // Azzero il potenziale di attivazione
            potential = 0;
                // Calcolo il potenziale di attivazione
                    for ( j = 0; j < hidden_size; j++ )
                    {
```

```
                        potential += h_neuron[j].value * o_synapse[i * ( hidden_size + 1 ) +
                        j].weight;
                    }

            // Aggiungo il valore del bias
            potential += o_synapse[i * ( hidden_size + 1 ) + j].weight;

            // Calcolo l'uscita del neurone sulla base del potenziale di attivazione
            o_neuron[i].value = sigmoid( potential );//uscita del neurone output
        }
}//fine funzione (procedura) run_network
```