```c
/*questo programma descrive il comportamento del percettrone a due ingressi
e una uscita  e calcola la funzione OR come uscita, la funzione di attivazione
e la sommatoria del prodotto dei pesi e degli ingressi e la funzione di trasferimento e la
funzione a gradino, quindi il percettrone ha solo due uscite o zero o uno,
in ingresso abbiamo 4 coppie che combinate ci danno i valori OR, e' inoltre definita la
funzione
di addestramento LA RETE E' FORMATA DA TRE NEURONI SU DUE STRATI, non posso fare lo XOR*/

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

typedef double T_Precision;



T_Precision transfer_function(const T_Precision *x,const T_Precision *w, size_t n);//
dichiarazione funzione di trasferimento

void training_function( T_Precision *x,T_Precision *w, size_t n,
const T_Precision *dx, const T_Precision *dy, size_t sn,
const T_Precision eta, const T_Precision desired_error);//dichiarazione funzione di
addestramento


 int main( void )
 {
        // Inizializzo il generatore di numeri pseudocasuali
        srand( (size_t) time( NULL ) );

        // Numero di ingressi
        const size_t n = 2;

        // Ingressi del percettrone (più il bias)
        T_Precision x[n + 1];

        // Pesi sinaptici (più il peso del bias)
        T_Precision w[n + 1];

        // Uscita del percettrone
        T_Precision y;

        // Iteratori
        size_t i, j;

        // Imposto il valore negativo unitario del bias
        x[n] = -1;

        // Numero di esempi dell'insieme di addestramento
        const size_t sn = 4;

        // Preparo l'insieme di addestramento per l'operatore OR, se voglio fare AND cambio i
valori
        const T_Precision dx[] = {0,0,0,1,1,0,1,1};

        //valori desiderati
        const T_Precision dy[] = { 0, 1, 1, 1 }; // AND 0,0,0,1  OR 0,1,1,1  XOR 0,1,1,0 non
funziona

        // Tasso di apprendimento
        const T_Precision eta = 0.5;
```

```
        // Errore desiderato
        const T_Precision desired_error = 0.00001;









        // Addestro il percettrone
        training_function( x, w, n, dx, dy, sn, eta, desired_error );//chiamata funzione di
addestramento

        // 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
                            x[0] = i;
                            x[1] = j;

                            // Calcolo l'uscita del percettrone
                            y = transfer_function( x, w, n );

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

                        }
        }


        return 0;

}




void training_function( T_Precision *x,T_Precision *w, size_t n,
const T_Precision *dx, const T_Precision *dy, size_t sn,
const T_Precision eta, const T_Precision desired_error)//definizione funzione di addestramento


{
        // Uscita calcolata
        T_Precision y;
        // Errore della rete
        T_Precision error;
        // Errore dell'uscita
        T_Precision dEdy;
        // Errore dei pesi sinaptici
        T_Precision dEdw[n];
        // Modifica del peso sinaptico
        T_Precision deltaw;
        // Contatore delle epoche
        size_t epochs = 0;
        // Iteratore
        size_t i, t;
```

```
        // Log di lavoro
        printf( "Inizio l'addestramento (eta = %.2f, errore desiderato = %f).\n", desired_error
        );


                // Inizializzo i pesi sinaptici con dei valori casuali compreso il peso bias
                for ( i = 0; i <= n; i++ )
                {
                    w[i] = rand() % 10;
                }


        // Continuo l'addestramento finché non raggiungo
        // l'errore desiderato (max 100 epoche)

        do

        {
                // Stampo il numero dell'epoca corrente
                printf( "Epoca #%zu: ", epochs );

                // Azzero l'errore della rete
                error = 0.0;

                // Azzero gli errori dei pesi sinaptici (compreso il bias)
                for ( i = 0; i <= n; i++ )
                {
                    dEdw[i] = 0;
                }

                // Ripeto per tutti gli esempi nell'insieme di addestramento, sn e' il numero
                delle coppie di ingressi
                for ( t = 0; t < sn; t++ )
                {

                            // Prendo gli ingressi dall'esempio
                            for ( i = 0; i < n; i++ )
                                {
                                    x[i] = dx[t * n + i];
                                }

                            // Calcolo l'uscita del percettrone
                            // con gli ingressi dell'esempio
                            y = transfer_function( x, w, n );


                            // Calcolo l'errore sull'uscita
                            dEdy = -(dy[t] - y);


                            // Calcolo l'errore dei pesi sinaptici
                            for ( i = 0; i <= n; i++ )
                                {
                                    dEdw[i] += dEdy * x[i];
                                }


                            // Calcolo il quadrato dell'errore necessario
                            // per trovare l'errore quadratico medio
                            // E(x) = SUM( e^2 ) / n_samples
                            error += ( dEdy * dEdy ) / 2.0;
```

```
                }

                // Correggo i pesi usando la regola delta
                for ( i = 0; i <= n; i++ )
                    {
                            // Calcolo la modifica del peso
                            deltaw = - eta * dEdw[i];

                            // Applico la modifica al peso sinaptico
                            w[i] = w[i] + deltaw;

                            // Stampa il nuovo valore del peso
                            printf( "w[%zu]=%.1f (E=%.1f), ", i, w[i], dEdw[i] );
                    }

                // Termino la riga di log aggiungendo l'errore corrente
                printf( "MSE=%f\n", error );

                // Incremento il numero delle epoche
                epochs++;
        }

        while ( error > desired_error && epochs < 100 );

        // Log di lavoro
        printf( "Addestramento terminato dopo %zu epoche.\n\n", epochs ); //fine definizione
        della funzione di addestramento
}




T_Precision transfer_function(const T_Precision *x,const T_Precision *w, size_t n) //
definizione funzione di trasferimento
    {

                                T_Precision activation_function(const T_Precision *x,const
                                T_Precision *w, size_t n);//dichiarazione funzione di
                                attivazione

                                T_Precision potential = activation_function(x,w,n);
                                return (T_Precision) (potential>0);
    }//fine definizione funzione di trasferimento




T_Precision activation_function(const T_Precision *x,const T_Precision *w, size_t n) //
definizione funzione di attivazione
{

T_Precision potential = 0;
size_t i;

for (i=0;i<=n;i++)
    {
    potential += x[i]*w[i];
    }
    return potential;
}//fine definizione funzione di attivazione
```