7. Esercizi in linguaggio C

Negli esercizi che seguono è richiesto di implementare una emulazione dei circuiti quantistici visti nel testo.

Gli esercizi devono essere svolti usando i tipi di dato e le funzioni definite nella libreria `libSSQ`.
La libreria permette di emulare la creazione di stati di un singolo qubit, due e tre qubits.
Allo stesso modo, si creano gli operatori tensoriali che agiscono sugli stati di qubits.

Per i primi tre esercizi è proposta una soluzione, mentre la soluzione del quarto e del quinto è lasciata al lettore che potrà postarla sul gruppo facebook **Reti Neurali in C** all'indirizzo `https://www.facebook.com/groups/1229389267230026/`

Attenzione: per quanto si possa conoscere a fondo la programmazione in linguaggio C, questi esercizi possono risultare molto *sfidanti*, si consiglia di prendersi il tempo dovuto per analizzare bene la libreria proposta per capirne i principi ispiratori e il modello sottostante.

Il codice riportato è presente anche su GitHub all'indirizzo `https://github.com/francescosisini/LIBRO-Informatica-Quantistica`

7.1 Libreria libSSQ

L'organizzazione in tipi e funzioni proposta nel codice che segua ha lo scopo di ricalcare il più fedelmente possibile i concetti matematici visti nel calcolo tensoriale.
Il codice non è realizzato per massimizzare né l'efficienza né la semplicità d'uso, ma proprio per aderire quanto più fedelmente possibile alla realtà matematica e fisica fin qui introdotta.

Kets

Per rappresentare uno stato fisico di qubits si usano i tipi `ket1` per i singoli qubit e `ket2` e `ket3` per gli stati di due e tre qubits rispettivamente.
Per creare un ket di un singolo qubit si usa la funzione `crea_ket1` a cui si passano per argomento i coefficienti delle ampiezze degli stati della base `base1` che è un tipo `struct` che ha due campi `e1` ed `e2` di tipo `elemento1`.

In prima istanza è possibile che si faccia fatica a recepire il modello C presentato nella libreria, ma con un po' di pazienza, ci si accorgerà che esso è più intuitivo di quanto appaia all'inizio. Per rompere il ghiaccio vediamo un primo esempio in cui si crea lo stato di un qubit di valore zero espresso rispetto alla base standard.

```
double complex a = 1;
double complex b = 1;
double complex norm = csqrt(1/(a*a+b*b));
ket1 psi = crea_ket1(a*nomr,b*norm,base_std1);
```

Si vede dalla prima riga del codice che la libreria usa le macro definite in `complex.h` per le operazioni sui numeri complessi.
La variabile `base_std1` è predefinita nella libreria e rappresenta un'istanza del tipo `base1`. La chiamata alla funzione `crea_ket1` ritorna une `ket1` di ampiezze `a` e `b` riferite agli elementi `e1` ed `e2` della base standard che è una variabile predefinita nella libreria e può essere usata nel codice utente dichiarandola come `extern`

Kets per due e tre qubits

Se si prova a stampare a video i membri `x` e `y` del ket ψ si vede che essi corrispondono esattamente ai due parametri `a` e `b` passati per argomento alla funzione:

```
printf("%g+i%g,%g+i%g -> ",psi.x, psi.y);
```

Quindi esiste una relazione uno a uno tra i parametri passati e i

coefficienti dei ket di base usati per definire lo stato.
Analogamente, per creare un ket di stato di due qubits si devono passare quattro parametri `a`, `b`, `c` e `d` che corrispondono alle due ampiezze di ognuno dei due ket:

```
double complex a = 1;
double complex b = 0;
double complex c = 1;
double complex d = 0;

double complex norm = csqrt(1/(a*a+b*b+c*c+d*d));

a = a*norm;
b = b*norm;
c = c*norm;
d = d*norm;

ket2 psi2 = crea_ket2(a,b,c,d,base_std2);

printf("(%g+i%g,%g+i%g)x(%g+i%g,%g+i%g) = ",a, b,c,d);

printf("%g+i%g,%g+i%g,%g+i%g,%g+i%g\n\n",
psi2.x11, psi2.x12,psi2.x21, psi2.x22);
```

Si noti che, diversamente dal caso di un solo qubit, i coefficienti `x11`, `x12`, `x21` e `x22` dei quattro ket di base usati per definire lo stato dei due qubits sono diversi dei coefficienti dei due qubit considerati singolarmente (vedi paragrafo 5.4).

Per creare il ket di stato di tre qubits si usa la funzione

```
ket3
crea_ket3(double complex x1, double complex y1,
      double complex x2, double complex y2,
      double complex x3, double complex y3,
      base3 b);
```

il cui prototipo è analogo a quelli visti per creare stati di due o un qubit. Si rammenti che in questo caso i parametri che definiscono gli stati dei singoli qubits sono sei mentre i coefficienti di base sono otto $\left(2^3\right)$.

Operatori

Gli operatori sono definiti in modo simile ai ket di stato.
Per quelli che devono operare sui ket di tipo `ket1` viene definito un tipo di dato per rappresentare gli elementi dello spazio

tensoriale $V \otimes V^*$. Il nuovo tipo è chiamato `elemento1_1` , dove i due indici *1* e *_1* indicano che l'operatore ha un elemento dello spazio V e uno dello spazio V^*:

```
   typedef struct
  {
    double complex c[2][2];
  } elemento1_1;
```

Essi sono rappresentati dai tipi o*n_n* dove l'indice *n* può essere sostituito con il numero 1,2 o 3, quindi gli operatori che agiscono su stati di un singolo qubit sono rappresentati dal tipo o*1_1* e similmente i tipi o*2_2* e o*3_3* rappresentano gli operatori che agiscono su stati di due e tre qubits rispettivamente.
L'indice *n* si riferisce ai ket mentre *_n* si riferisce ai bra.
Ogni operatore *o1_1* è definito rispetto alla base `base1_1` costituita da quattro elementi di tipo `elemento1_1` come segue:

```
   typedef struct
{
  elemento1_1 e1e_1; //00
  elemento1_1 e1e_2; //01
  elemento1_1 e2e_1; //10
  elemento1_1 e2e_2; //11


} base1_1;
```

La libreria libSSQ provvede una istanza già predefinita del tipo `base1_1` attraverso la variabile `base_std1_1`

```
const base1_1 base_std1_1 =
  {
    {
      {1,0,
      0,0}
    }
    ,
    {
      {0,1,
      0,0}
    }
    ,
    {
      {0,0,
      1,0}
    }
    ,
    {
      {0,0,
      0,1}
    }
  };
```

La variabile può essere usata dichiarandola `extern` nel codice.
Le base standard `base_std1_1` è definita in modo che il prodotto dei suoi elementi con quelli della base standard dei `ket1`, `base_std1`, soddisfi le relazioni definite nel paragrafo 3.3 (Prodotto fra duali e vettori).
Per creare un operatore si usa la funzione

```
op1_1 crea_tensore1_1
(double complex a11,double complex
a12,double complex a21,
double complex a22, base1_1 b)
```

Per esempio, l'operatore identità viene creato come segue:

```
 op1_1 o = crea_tensore1_1(1,0,0,1,base_std1_1);
```

Una volta creato un operatore si può usare per trasformare uno ket usando la funzione `ket1 trasforma_ket1(op1_1 o,ket1 ket)` come nell'esempio che segue:

```
  ket1 psip = trasforma_ket1(o,psi);
```

Dopo la breve descrizione della libreria e del suo uso si raccomanda ora di studiare il codice nel dettaglio e poi passare senza indugio agli esercizi del paragrafo seguente.

libSSQ.h

```
/*************************************************************************
 *
 * libSSQ @2020 Scuola Sisini
 * Licenza GPL3
 *
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see .
 */

#pragma once
#ifndef LIBSSQ_H
#define LIBSSQ_H

#include

/*____________________SINGOLO QUBIT_________________*/

/*_____________________________________________________
 *
```

```
 * Elemento e in  V
 */
typedef struct
{
  double complex c[2];
} elemento1;

/*_____________________________________________________
 *
 * Base qubit singolo
 */
typedef struct
{
  elemento1 e1;
  elemento1 e2;

} base1;



/*_____________________________________________________
 *
 * Un ket di ampiezze complesse x e y sulla base b
 */
typedef struct
{
  double complex x,y;
  base1 b;
} ket1;


/*___________DOPERATORE TENSORIALE_1X1_____________*/

/*_____________________________________________________
 *
 * Elemento  ee* in V(x)V*
 */
typedef struct
{
  double complex c[2][2];
} elemento1_1;

/*_____________________________________________________
 *
 * Base tensoori V@V*
 */
typedef struct
{
  elemento1_1 e1e_1; //00
  elemento1_1 e1e_2; //01
  elemento1_1 e2e_1; //10
  elemento1_1 e2e_2; //11


} base1_1;

/*_____________________________________________________
 *
 * Un operatore sullo spazio 11
 * di componenti a11, a12, a21, a22 sull base b
 *
 */
typedef struct
{
  double complex a11,a12,a21,a22;
  base1_1 b;
} op1_1;


/*_____________________DOPPIO QUBIT_________________*/
```

```
/*_____________________________________________________
 *
 * Elemento  ee in V(x)V
 */
typedef struct
{
  double complex c[4];
} elemento2;

/*_____________________________________________________
 *
 * Base 2 qubits
 */
typedef struct
{
  elemento2 e1f1; //00
  elemento2 e1f2; //01
  elemento2 e2f1; //10
  elemento2 e2f2; //11


} base2;

/*_____________________________________________________
 *
 * Un ket di 2 qubit di ampiezze complesse x1 e y1
 * e x2, y2  sulla base b
 *
 */
typedef struct
{
  double complex x11,x12,x21,x22;
  base2 b;
} ket2;

/*___________DOPERATORE TENSORIALE_2X2_____________*/

/*_____________________________________________________
 *
 * Elemento  efe*f* in V(x)V(x)V*(x)V*
 */
typedef struct
{
  double complex c[4][4];
} elemento2_2;

/*_____________________________________________________
 *
 * Base tensoori V@V*
 */
typedef struct
{
  //-riga 1
  elemento2_2 e1f1e_1f_1;
  elemento2_2 e1f1e_1f_2;
  elemento2_2 e1f1e_2f_1;
  elemento2_2 e1f1e_2f_2;
  //-riga 2
  elemento2_2 e1f2e_1f_1;
  elemento2_2 e1f2e_1f_2;
  elemento2_2 e1f2e_2f_1;
  elemento2_2 e1f2e_2f_2;
  //-riga 3
  elemento2_2 e2f1e_1f_1;
  elemento2_2 e2f1e_1f_2;
  elemento2_2 e2f1e_2f_1;
  elemento2_2 e2f1e_2f_2;
  //-riga 4
  elemento2_2 e2f2e_1f_1;
```

```
  elemento2_2 e2f2e_1f_2;
  elemento2_2 e2f2e_2f_1;
  elemento2_2 e2f2e_2f_2;

} base2_2;

/*_____________________________________________________
 *
 * Un operatore sullo spazio 11
 * di componenti a11, a12, a21, a22 sull base b
 *
 */
typedef struct
{
  double complex a11,a12,a13,a14,
    a21,a22,a23,a24,
    a31,a32,a33,a34,
    a41,a42,a43,a44;
  base2_2 b;
} op2_2;


/*_____________________TRIPLO QUBIT_____________________*/

/*_____________________________________________________
 *
 * Elemento  eee in V(x)V(x)V
 */
typedef struct
{
  double complex c[8];
} elemento3;

/*_____________________________________________________
 *
 * Base 3 qubits
 */
typedef struct
{
  elemento3 e1f1g1; //000
  elemento3 e1f1g2; //001
  elemento3 e1f2g1; //010
  elemento3 e1f2g2; //011
  elemento3 e2f1g1; //100
  elemento3 e2f1g2; //101
  elemento3 e2f2g1; //110
  elemento3 e2f2g2; //111

} base3;

/*_____________________________________________________
 *
 * Un ket di 3 qubit di ampiezze complesse x1 e y1
 * , x2, y2 e x3, y3  sulla base b
 *
 */
typedef struct
{
  double complex x111,x112,x121,x122,x211,x212,x221,x222;
  base3 b;
} ket3;

/*___________DOPERATORE TENSORIALE_3X3_____________*/

/*_____________________________________________________
 *
 * Elemento  efe*f* in V(x)V(x)V*(x)V*
 */
typedef struct
{
```

```
  double complex c[8][8];
} elemento3_3;

/*_____________________________________________________
 *
 * Base tensoori V@V@V@V*@V*@V*
 */
typedef struct
{
  //-riga 1
  elemento3_3 e1f1g1e_1f_1g_1;
  elemento3_3 e1f1g1e_1f_1g_2;
  elemento3_3 e1f1g1e_1f_2g_1;
  elemento3_3 e1f1g1e_1f_2g_2;
  elemento3_3 e1f1g1e_2f_1g_1;
  elemento3_3 e1f1g1e_2f_1g_2;
  elemento3_3 e1f1g1e_2f_2g_1;
  elemento3_3 e1f1g1e_2f_2g_2;
   //-riga 2
  elemento3_3 e1f1g2e_1f_1g_1;
  elemento3_3 e1f1g2e_1f_1g_2;
  elemento3_3 e1f1g2e_1f_2g_1;
  elemento3_3 e1f1g2e_1f_2g_2;
  elemento3_3 e1f1g2e_2f_1g_1;
  elemento3_3 e1f1g2e_2f_1g_2;
  elemento3_3 e1f1g2e_2f_2g_1;
  elemento3_3 e1f1g2e_2f_2g_2;
   //-riga 3
  elemento3_3 e1f2g1e_1f_1g_1;
  elemento3_3 e1f2g1e_1f_1g_2;
  elemento3_3 e1f2g1e_1f_2g_1;
  elemento3_3 e1f2g1e_1f_2g_2;
  elemento3_3 e1f2g1e_2f_1g_1;
  elemento3_3 e1f2g1e_2f_1g_2;
  elemento3_3 e1f2g1e_2f_2g_1;
  elemento3_3 e1f2g1e_2f_2g_2;
   //-riga 4
  elemento3_3 e1f2g2e_1f_1g_1;
  elemento3_3 e1f2g2e_1f_1g_2;
  elemento3_3 e1f2g2e_1f_2g_1;
  elemento3_3 e1f2g2e_1f_2g_2;
  elemento3_3 e1f2g2e_2f_1g_1;
  elemento3_3 e1f2g2e_2f_1g_2;
  elemento3_3 e1f2g2e_2f_2g_1;
  elemento3_3 e1f2g2e_2f_2g_2;

  //-riga 1
  elemento3_3 e2f1g1e_1f_1g_1;
  elemento3_3 e2f1g1e_1f_1g_2;
  elemento3_3 e2f1g1e_1f_2g_1;
  elemento3_3 e2f1g1e_1f_2g_2;
  elemento3_3 e2f1g1e_2f_1g_1;
  elemento3_3 e2f1g1e_2f_1g_2;
  elemento3_3 e2f1g1e_2f_2g_1;
  elemento3_3 e2f1g1e_2f_2g_2;
   //-riga 2
  elemento3_3 e2f1g2e_1f_1g_1;
  elemento3_3 e2f1g2e_1f_1g_2;
  elemento3_3 e2f1g2e_1f_2g_1;
  elemento3_3 e2f1g2e_1f_2g_2;
  elemento3_3 e2f1g2e_2f_1g_1;
  elemento3_3 e2f1g2e_2f_1g_2;
  elemento3_3 e2f1g2e_2f_2g_1;
  elemento3_3 e2f1g2e_2f_2g_2;
   //-riga 3
  elemento3_3 e2f2g1e_1f_1g_1;
  elemento3_3 e2f2g1e_1f_1g_2;
  elemento3_3 e2f2g1e_1f_2g_1;
  elemento3_3 e2f2g1e_1f_2g_2;
  elemento3_3 e2f2g1e_2f_1g_1;
```

```
  elemento3_3 e2f2g1e_2f_1g_2;
  elemento3_3 e2f2g1e_2f_2g_1;
  elemento3_3 e2f2g1e_2f_2g_2;
   //-riga 4
  elemento3_3 e2f2g2e_1f_1g_1;
  elemento3_3 e2f2g2e_1f_1g_2;
  elemento3_3 e2f2g2e_1f_2g_1;
  elemento3_3 e2f2g2e_1f_2g_2;
  elemento3_3 e2f2g2e_2f_1g_1;
  elemento3_3 e2f2g2e_2f_1g_2;
  elemento3_3 e2f2g2e_2f_2g_1;
  elemento3_3 e2f2g2e_2f_2g_2;

} base3_3;

/*_____________________________________________________
 *
 * Un operatore sullo spazio 111
 * di componenti a11, a12,...,a88 sull base b
 *
 */
typedef struct
{
  double complex
  a11,a12,a13,a14,a15,a16,a17,a18,
    a21,a22,a23,a24,a25,a26,a27,a28,
    a31,a32,a33,a34,a35,a36,a37,a38,
    a41,a42,a43,a44,a45,a46,a47,a48,
    a51,a52,a53,a54,a55,a56,a57,a58,
    a61,a62,a63,a64,a65,a66,a67,a68,
    a71,a72,a73,a74,a75,a76,a77,a78,
    a81,a82,a83,a84,a85,a86,a87,a88;
  base3_3 b;
} op3_3;



/*_____________Funzioni_____________________________*/

/*_____________________________________________________
 *
 * crea un ket di stato di ampiezze complesse
 * x e y sulla base b
 */
ket1
crea_ket1(double complex x, double complex y, base1 b);

/*_____________________________________________________
 *
 * crea un ket di stato di due qubits ampiezze complesse
 * x1, y1, x2 e y2 sulla base b
 * come prodotto dei due key
 * non è possibile creare direttamente ket entangled
 */
ket2
crea_ket2(double complex x1, double complex y1,
          double complex x2, double complex y2,base2 b);

/*_____________________________________________________
 *
 * crea un ket di stato di tre qubits ampiezze complesse
 */
ket3
crea_ket3(double complex x1, double complex y1,
          double complex x2, double complex y2,
          double complex x3, double complex y3,
          base3 b);
```

```
/*_____________________________________________________
 *
 * crea un operatore (tensore 1_1) di coefficienti
 * complessi a11, a12, a21 e a22 sulla base b
 */
op1_1 crea_tensore1_1
(double complex a11,double complex a12,double complex a21,double complex a22,
 base1_1 b);

/*_____________________________________________________
 *
 * crea un operatore (tensore 2_2) di coefficienti
 * complessi a11, a12,...,a44 sulla base b
 */
op2_2 crea_tensore2_2
(double complex a11,double complex a12,double complex a13,double complex a14,
 double complex a21,double complex a22,double complex a23,double complex a24,
 double complex a31,double complex a32,double complex a33,double complex a34,
 double complex a41,double complex a42,double complex a43,double complex a44,
 base2_2 b
 );

/*_____________________________________________________
 *
 * crea un operatore (tensore 3_3) di coefficienti
 * complessi a11, a12,...,a44 sulla base b
 */
op3_3 crea_tensore3_3
(double complex a11,double complex a12,double complex a13,double complex a14,
 double complex a15,double complex a16,double complex a17,double complex a18,

 double complex a21,double complex a22,double complex a23,double complex a24,
 double complex a25,double complex a26,double complex a27,double complex a28,

 double complex a31,double complex a32,double complex a33,double complex a34,
 double complex a35,double complex a36,double complex a37,double complex a38,

 double complex a41,double complex a42,double complex a43,double complex a44,
 double complex a45,double complex a46,double complex a47,double complex a48,

 double complex a51,double complex a52,double complex a53,double complex a54,
 double complex a55,double complex a56,double complex a57,double complex a58,

 double complex a61,double complex a62,double complex a63,double complex a64,
 double complex a65,double complex a66,double complex a67,double complex a68,

 double complex a71,double complex a72,double complex a73,double complex a74,
 double complex a75,double complex a76,double complex a77,double complex a78,

 double complex a81,double complex a82,double complex a83,double complex a84,
 double complex a85,double complex a86,double complex a87,double complex a88,

 base3_3 b
 );



/*_____________________________________________________
 *
 * Trasforma un ket tramite un operatore
 *
 */
ket1 trasforma_ket1(op1_1 o,ket1 ket);

ket2 trasforma_ket2(op2_2 o,ket2 ket);

ket3 trasforma_ket3(op3_3 o,ket3 ket);

#endif
```

libSSQ.c

```
/*****************************************************************************
 *
 * libSSQ @2020 Scuola Sisini
 * Licenza GPL3
 *
    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see .
 */

#include
#include
#include "libSSQ.h"

/*_____________________________________________________
 *
 * Base STANDARD qubit singolo
 */
const base1 base_std1 =
  {
    {{1,0}},//0
    {{0,1}} //1
  };

/*_____________________________________________________
 *
 * Base STANDARD 2qubits
 */
const base2 base_std2=
  {
    {{1,0,0,0}},//00
    {{0,1,0,0}},//01
    {{0,0,1,0}},//10
    {{0,0,0,1}} //11
  };

/*_____________________________________________________
 *
 * Base STANDARD 3qubits
 */
const base3 base_std3=
  {
    {{1,0,0,0,0,0,0,0}},//00
    {{0,1,0,0,0,0,0,0}},//00
    {{0,0,1,0,0,0,0,0}},//00
    {{0,0,0,1,0,0,0,0}},//00

    {{0,0,0,0,1,0,0,0}},//00
    {{0,0,0,0,0,1,0,0}},//00
    {{0,0,0,0,0,0,1,0}},//00
    {{0,0,0,0,0,0,0,1}},//00

  };
```

```
/*_____________________________________________________
 *
 * Base STANDARD opertori 1_1
 */
const base1_1 base_std1_1 =
  {
    {
      {1,0,
      0,0}
    }
    ,
    {
      {0,1,
      0,0}
    }
    ,
    {
      {0,0,
      1,0}
    }
    ,
    {
      {0,0,
      0,1}
    }
  };

/*_____________________________________________________
 *
 * Base STANDARD opertori 2_2
 */
const base2_2 base_std2_2 =
  {
    {
      {1,0,0,0,
       0,0,0,0,
       0,0,0,0,
       0,0,0,0}
    }
    ,
    {
      {0,1,0,0,
       0,0,0,0,
       0,0,0,0,
       0,0,0,0}
    }
    ,
    {
      {0,0,1,0,
       0,0,0,0,
       0,0,0,0,
       0,0,0,0}
    }
    ,
    {
      {0,0,0,1,
       0,0,0,0,
       0,0,0,0,
       0,0,0,0}
    }
    ,
    {
      {0,0,0,0,
       1,0,0,0,
       0,0,0,0,
       0,0,0,0}
    }
    ,
```

```
{
  {0,0,0,0,
   0,1,0,0,
   0,0,0,0,
   0,0,0,0}
}
,
{
  {0,0,0,0,
   0,0,1,0,
   0,0,0,0,
   0,0,0,0}
}
,
{
  {0,0,0,0,
   0,0,0,1,
   0,0,0,0,
   0,0,0,0}
}
,
{
  {0,0,0,0,
   0,0,0,0,
   1,0,0,0,
   0,0,0,0}
}
,
{
  {0,0,0,0,
   0,0,0,0,
   0,1,0,0,
   0,0,0,0}
}
,
{
  {0,0,0,0,
   0,0,0,0,
   0,0,1,0,
   0,0,0,0}
}
,
{
  {0,0,0,0,
   0,0,0,0,
   0,0,0,1,
   0,0,0,0}
}
,
{
  {0,0,0,0,
   0,0,0,0,
   0,0,0,0,
   1,0,0,0}
}
,
{
  {0,0,0,0,
   0,0,0,0,
   0,0,0,0,
   0,1,0,0}
}
,
{
  {0,0,0,0,
   0,0,0,0,
   0,0,0,0,
   0,0,1,0}
}
,
```

```
    {
      {0,0,0,0,
       0,0,0,0,
       0,0,0,0,
       0,0,0,1}
    }
  };

/*_____________________________________________________
 *
 * Base STANDARD opertori 3_3
 */
const base3_3 base_std3_3 =
  {
    1,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,

    0,1,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,

    0,0,1,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,

    0,0,0,1,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,

    0,0,0,0,1,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,

    0,0,0,0,0,1,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,

    0,0,0,0,0,0,1,0,
    0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,
0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,1,0,0,0,
0,0,0,0,0,0,0,0,
```

```
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,1,0,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,
0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
1,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,1,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,1,0,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,1,0,0,0,0,

0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,
```

```
    0,0,0,0,0,0,0,0,
    0,0,0,0,1,0,0,0,

    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,1,0,0,

    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,1,0,

    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,1
  };


//--Servizio
void matrice_x_vettore2
(double complex v_out[2],elemento1_1 m,elemento1 v_in)
{
  for(int i=0;i<2;i++) v_out[i] = 0;

  for(int i=0;i<2;i++)
    {
      for(int j=0;j<2;j++)
        {
          v_out[i]+=m.c[i][j]*v_in.c[j];
        }
    }
}
void matrice_x_vettore4
(double complex v_out[4],elemento2_2 m,elemento2 v_in)
{
  for(int i=0;i<4;i++) v_out[i] = 0;

  for(int i=0;i<4;i++)
    {
      for(int j=0;j<4;j++)
        {
          v_out[i]+=m.c[i][j]*v_in.c[j];
        }
    }
}

void matrice_x_vettore8
(double complex v_out[8],elemento3_3 m,elemento3 v_in)
{
  for(int i=0;i<8;i++) v_out[i] = 0;

  for(int i=0;i<8;i++)
    {
      for(int j=0;j<8;j++)
        {
          v_out[i]+=m.c[i][j]*v_in.c[j];
```

```
        }
    }
}


/*_____________________________________________________
 *
 * crea un ket di stato di ampiezze complesse
 * x e y sulla base b
 */
ket1
crea_ket1(double complex x, double complex y, base1 b)
{
  ket1 ket = {x,y,b};
  return ket;
}

/*_____________________________________________________
 *
 * crea un ket di stato di due qubits ampiezze complesse
 * x1, y1, x2 e y2 sulla base b
 * come prodotto dei due key
 * non è possibile creare direttamente ket entangled
 */
ket2
crea_ket2(double complex x1, double complex y1,
          double complex x2, double complex y2,base2 b)
{
  ket2 ket ={x1*x2,x1*y2,y1*x2,y1*y2,b};
  return ket;

}

/*_____________________________________________________
 *
 * crea un ket di stato di tre qubits ampiezze complesse
 */
ket3
crea_ket3(double complex x1, double complex y1,
          double complex x2, double complex y2,
          double complex x3, double complex y3,
          base3 b)
{
  ket3 ket = {
    x1*x2*x3,
    x1*x2*y3,
    x1*y2*x3,
    x1*y2*y3,

    y1*x2*x3,
    y1*x2*y3,
    y1*y2*x3,
    y1*y2*y3,b
  };
    return ket;

}

/*_____________________________________________________
 *
 * crea un operatore (tensore 1_1) di coefficienti
 * complessi a11, a12, a21 e a22 sulla base b
 */
op1_1 crea_tensore1_1
(double complex a11,double complex a12,double complex a21,double complex a22,
 base1_1 b)
{
  op1_1 o = {a11,a12,a21,a22,b};
  return o;
}
```

```
/*_____________________________________________________
 *
 * crea un operatore (tensore 2_2) di coefficienti
 * complessi a11, a12,...,a44 sulla base b
 */
op2_2 crea_tensore2_2
(double complex a11,double complex a12,double complex a13,double complex a14,
 double complex a21,double complex a22,double complex a23,double complex a24,
 double complex a31,double complex a32,double complex a33,double complex a34,
 double complex a41,double complex a42,double complex a43,double complex a44,
 base2_2 b
 )
{
  op2_2 o={a11,a12,a13,a14,a21,a22,a23,a24,a31,a32,a33,a34,a41,a42,a43,a44,b};
  return o;
}

/*_____________________________________________________
 *
 * crea un operatore (tensore 3_3) di coefficienti
 * complessi a11, a12,...,a88 sulla base b
 */
op3_3 crea_tensore3_3
(double complex a11,double complex a12,double complex a13,double complex a14,
 double complex a15,double complex a16,double complex a17,double complex a18,

 double complex a21,double complex a22,double complex a23,double complex a24,
 double complex a25,double complex a26,double complex a27,double complex a28,

 double complex a31,double complex a32,double complex a33,double complex a34,
 double complex a35,double complex a36,double complex a37,double complex a38,

 double complex a41,double complex a42,double complex a43,double complex a44,
 double complex a45,double complex a46,double complex a47,double complex a48,

 double complex a51,double complex a52,double complex a53,double complex a54,
 double complex a55,double complex a56,double complex a57,double complex a58,

 double complex a61,double complex a62,double complex a63,double complex a64,
 double complex a65,double complex a66,double complex a67,double complex a68,

 double complex a71,double complex a72,double complex a73,double complex a74,
 double complex a75,double complex a76,double complex a77,double complex a78,

 double complex a81,double complex a82,double complex a83,double complex a84,
 double complex a85,double complex a86,double complex a87,double complex a88,

 base3_3 b
 )
{
  op3_3 o ={
    a11, a12, a13, a14,
    a15, a16, a17, a18,

    a21, a22, a23, a24,
    a25, a26, a27, a28,

    a31, a32, a33, a34,
    a35, a36, a37, a38,

    a41, a42, a43, a44,
    a45, a46, a47, a48,

    a51, a52, a53, a54,
    a55, a56, a57, a58,

    a61, a62, a63, a64,
    a65, a66, a67, a68,
```

```
    a71, a72, a73, a74,
    a75, a76, a77, a78,

    a81, a82, a83, a84,
    a85, a86, a87, a88,
    b
  };

  return o;
}

/*_____________________________________________________
 *
 * Trasforma un ket tramite un operatore
 *
 */
ket1 trasforma_ket1(op1_1 o,ket1 ket)
{
  ket1 ketp;

  //Preparazione array di servizio
  double complex v[2];
  elemento1 e[2] = {ket.b.e1,ket.b.e2};
  elemento1_1 ee_[2][2] = {o.b.e1e_1,o.b.e1e_2,o.b.e2e_1,o.b.e2e_2};
  double complex x[2] =  {ket.x,ket.y};
  double complex a[2][2] = {o.a11,o.a12,o.a21,o.a22};

  //Inizializzazione
  ketp.x = 0;
  ketp.y = 0;
  for(int k=0;k<2;k++)
    for(int i=0;i<2;i++)
      {
        //Il tensore o è svliluppato sulla base eie_j come:
        //o = a11*e1e_1+...+a22*e2e_2
        //Il ket ket è sviluppato sulla base ei come:
        //ket = x*e1+y*e2
        //Il prodottotto o*ket si sviluppa linearmente come:
        //o*ket=(a11*e1e_1+...+a22*e2e_2)*(x*e1+y*e2) quindi:
        //o*ket=e1e_1(e1)*a11*x+... dove si ricorda che
        //e1e_1(e1) è un prodotto matrice x colonna
        for(int j=0;j<2;j++)
          {
            matrice_x_vettore2(v,ee_[i][j],e[k]);
            ketp.x += v[0]*a[i][j]*x[k];
            ketp.y += v[1]*a[i][j]*x[k];
          }
      }
  ketp.b = ket.b;
  return ketp;
}

ket2 trasforma_ket2(op2_2 o,ket2 ket)
{
  ket2 ketp;
  ketp.x11 = ketp.x12 = ketp.x21 = ketp.x22 = 0;
  //Preparazione array di servizio
  double complex v[4];
  elemento2 e[4] = {ket.b.e1f1,ket.b.e1f2,ket.b.e2f1,ket.b.e2f2};
  elemento2_2 ee_[4][4] = {
    o.b.e1f1e_1f_1,o.b.e1f1e_1f_2,o.b.e1f1e_2f_1,o.b.e1f1e_2f_2,
    o.b.e1f2e_1f_1,o.b.e1f2e_1f_2,o.b.e1f2e_2f_1,o.b.e1f2e_2f_2,
    o.b.e2f1e_1f_1,o.b.e2f1e_1f_2,o.b.e2f1e_2f_1,o.b.e2f1e_2f_2,
    o.b.e2f2e_1f_1,o.b.e2f2e_1f_2,o.b.e2f2e_2f_1,o.b.e2f2e_2f_2
  };
  double complex x[4] =  {ket.x11,ket.x12,ket.x21,ket.x22};
  double complex a[4][4] = {
    o.a11,o.a12,o.a13,o.a14,
    o.a21,o.a22,o.a23,o.a24,
    o.a31,o.a32,o.a33,o.a34,
```

```
    o.a41,o.a42,o.a43,o.a44
  };

   for(int k=0;k<4;k++)
    for(int i=0;i<4;i++)
      {
        //Per una spiegazione, vedi i commenti di trasforma_ket1
        for(int j=0;j<4;j++)
          {
            matrice_x_vettore4(v,ee_[i][j],e[k]);

            ketp.x11 += v[0]*a[i][j]*x[k];
            ketp.x12 += v[1]*a[i][j]*x[k];
            ketp.x21 += v[2]*a[i][j]*x[k];
            ketp.x22 += v[3]*a[i][j]*x[k];
          }
      }
   ketp.b = ket.b;
   return ketp;
}

ket3 trasforma_ket3(op3_3 o,ket3 ket)
{
  ket3 ketp;
  ketp.x111 = ketp.x112 = ketp.x121 = ketp.x122 =
    ketp.x211 = ketp.x212 = ketp.x221 = ketp.x222 = 0;

  //Preparazione array di servizio
  double complex v[8];

  elemento3 e[8] = {
    ket.b.e1f1g1,
    ket.b.e1f1g2,
    ket.b.e1f2g1,
    ket.b.e1f2g2,
    ket.b.e2f1g1,
    ket.b.e2f1g2,
    ket.b.e2f2g1,
    ket.b.e2f2g2};

  elemento3_3 ee_[8][8] = {
    o.b.e1f1g1e_1f_1g_1,
    o.b.e1f1g1e_1f_1g_2,
    o.b.e1f1g1e_1f_2g_1,
    o.b.e1f1g1e_1f_2g_2,
    o.b.e1f1g1e_2f_1g_1,
    o.b.e1f1g1e_2f_1g_2,
    o.b.e1f1g1e_2f_2g_1,
    o.b.e1f1g1e_2f_2g_2,

    o.b.e1f1g2e_1f_1g_1,
    o.b.e1f1g2e_1f_1g_2,
    o.b.e1f1g2e_1f_2g_1,
    o.b.e1f1g2e_1f_2g_2,
    o.b.e1f1g2e_2f_1g_1,
    o.b.e1f1g2e_2f_1g_2,
    o.b.e1f1g2e_2f_2g_1,
    o.b.e1f1g2e_2f_2g_2,

    o.b.e1f2g1e_1f_1g_1,
    o.b.e1f2g1e_1f_1g_2,
    o.b.e1f2g1e_1f_2g_1,
    o.b.e1f2g1e_1f_2g_2,
    o.b.e1f2g1e_2f_1g_1,
    o.b.e1f2g1e_2f_1g_2,
    o.b.e1f2g1e_2f_2g_1,
    o.b.e1f2g1e_2f_2g_2,

    o.b.e1f2g2e_1f_1g_1,
    o.b.e1f2g2e_1f_1g_2,
```

```
  o.b.e1f2g2e_1f_2g_1,
  o.b.e1f2g2e_1f_2g_2,
  o.b.e1f2g2e_2f_1g_1,
  o.b.e1f2g2e_2f_1g_2,
  o.b.e1f2g2e_2f_2g_1,
  o.b.e1f2g2e_2f_2g_2,

  //-----------------

  o.b.e2f1g1e_1f_1g_1,
  o.b.e2f1g1e_1f_1g_2,
  o.b.e2f1g1e_1f_2g_1,
  o.b.e2f1g1e_1f_2g_2,
  o.b.e2f1g1e_2f_1g_1,
  o.b.e2f1g1e_2f_1g_2,
  o.b.e2f1g1e_2f_2g_1,
  o.b.e2f1g1e_2f_2g_2,

  o.b.e2f1g2e_1f_1g_1,
  o.b.e2f1g2e_1f_1g_2,
  o.b.e2f1g2e_1f_2g_1,
  o.b.e2f1g2e_1f_2g_2,
  o.b.e2f1g2e_2f_1g_1,
  o.b.e2f1g2e_2f_1g_2,
  o.b.e2f1g2e_2f_2g_1,
  o.b.e2f1g2e_2f_2g_2,

  o.b.e2f2g1e_1f_1g_1,
  o.b.e2f2g1e_1f_1g_2,
  o.b.e2f2g1e_1f_2g_1,
  o.b.e2f2g1e_1f_2g_2,
  o.b.e2f2g1e_2f_1g_1,
  o.b.e2f2g1e_2f_1g_2,
  o.b.e2f2g1e_2f_2g_1,
  o.b.e2f2g1e_2f_2g_2,

  o.b.e2f2g2e_1f_1g_1,
  o.b.e2f2g2e_1f_1g_2,
  o.b.e2f2g2e_1f_2g_1,
  o.b.e2f2g2e_1f_2g_2,
  o.b.e2f2g2e_2f_1g_1,
  o.b.e2f2g2e_2f_1g_2,
  o.b.e2f2g2e_2f_2g_1,
  o.b.e2f2g2e_2f_2g_2,
    };

double complex x[8] =  {
  ket.x111,
  ket.x112,
  ket.x121,
  ket.x122,
  ket.x211,
  ket.x212,
  ket.x221,
  ket.x222
};

double complex a[8][8] = {
  o.a11,o.a12,o.a13,o.a14,o.a15,o.a16,o.a17,o.a18,
  o.a21,o.a22,o.a23,o.a24,o.a25,o.a26,o.a27,o.a28,
  o.a31,o.a32,o.a33,o.a34,o.a35,o.a36,o.a37,o.a38,
  o.a41,o.a42,o.a43,o.a44,o.a45,o.a46,o.a47,o.a48,

  o.a51,o.a52,o.a53,o.a54,o.a55,o.a56,o.a57,o.a58,
  o.a61,o.a62,o.a63,o.a64,o.a65,o.a66,o.a67,o.a68,
  o.a71,o.a72,o.a73,o.a74,o.a75,o.a76,o.a77,o.a78,
  o.a81,o.a82,o.a83,o.a84,o.a85,o.a86,o.a87,o.a88
};

 for(int k=0;k<8;k++)
```

```
    for(int i=0;i<8;i++)
      {
        //Per una spiegazione, vedi i commenti di trasforma_ket1
        for(int j=0;j<8;j++)
          {
            matrice_x_vettore8(v,ee_[i][j],e[k]);

            ketp.x111 += v[0]*a[i][j]*x[k];
            ketp.x112 += v[1]*a[i][j]*x[k];
            ketp.x121 += v[2]*a[i][j]*x[k];
            ketp.x122 += v[3]*a[i][j]*x[k];
            ketp.x211 += v[4]*a[i][j]*x[k];
            ketp.x212 += v[5]*a[i][j]*x[k];
            ketp.x221 += v[6]*a[i][j]*x[k];
            ketp.x222 += v[7]*a[i][j]*x[k];
          }

      }
   ketp.b = ket.b;
   return ketp;
}
```

7.2 Esercizi

Si risolvano i seguenti esercizi usando la libreria definita nel paragrafo precedente. Il codice con le soluzioni degli esercizi è presentato nel paragrafo successivo.

1. Si consideri un qubit definito da un fotone che si trovi nello stato $|0\rangle$. Si agisca con l'operatore σ_x di Pauli. Qual'è ora lo stato in cui si trova il qubit?
2. Si implementi il gate CNOT e se ne verifichi l'azione sullo stato: $|1\rangle \otimes |1\rangle$ e sullo stato $|0\rangle \otimes |1\rangle$.
3. Si faccia riferimento al paragrafo 6.2 e si implementi la trasmissione di un solo qubit per comunicare una informazione che ha quattro possibili valori, come l'esempio dei quattro punti cardinali presente nel testo.
4. Si realizzi un circuito full-adder reversibile.
5. Si implementi una comunicazione basata sul teletrasporto quantistico.

Note di compilazione

Le soluzioni proposte sono scritte in C standard e compilano sotto ogni architettura e sistema operativo. Per la compilazione sotto Linux si ricorda di linkare la libreria matematica:

```
gcc  -o  e3  libSSQ.c esercizio_3.c -lm
```

Usando il compilatore GCC l'uso dei caratteri di formattazione `%g` genera un innocuo messaggio di warning.

7.3 Soluzioni

1. **Soluzione**: Il testo dell'esercizio fa riferimento ad un singolo qubit quindi per svolgere l'esercizio dovremo usare il tipo `ket1`.
   L'operatore σ_x può essere costruito ricordando la sua forma matriciale definita nel paragrafo 5.2. Gli elementi della matrice devono essere passati come argomento alla funzione `crea_tensore1_1` come segue:

```
#include
#include
#include
#include "libSSQ.h"

extern base1 base_std1;
extern base1_1 base_std1_1;


int main()
{
  printf("_____QUBIT SINGOLO____\n");

  double complex a = 1;
  double complex b = 0;
  double complex norm =csqrt( 1/(a*a+b*b));

  ket1 psi = crea_ket1(a*norm,b*norm,base_std1);

  op1_1 o = crea_tensore1_1(0,1,1,0,base_std1_1);

  ket1 psip = trasforma_ket1(o,psi);

  printf("\nOperatore sigma_x:\n",psi.x, psi.y);
  printf("%g+i%g,%g+i%g -> ",psi.x, psi.y);
  printf("%g+i%g,%g+i%g\n",psip.x, psip.y);

}
```

   Compilando ed eseguendo il codice si vede che il ket di stato passa da $|0\rangle$ a $|1\rangle$, quindi il qubit passa da 0 ad 1.
2. **Soluzione**: Il gate CNOT ha due qubit di ingresso e due di uscita, quindi per rappresentare lo stato di ingresso è necessario usare il tipo `ket2`.
   L'operatore CNOT può essere costruito usando la sua forma matriciale definita nel paragrafo 5.4. Gli elementi della

matrice devono essere passati come argomento alla funzione `crea_tensore2_2` come segue:

```
#include
#include
#include
#include "libSSQ.h"

extern base2 base_std2;
extern base2_2 base_std2_2;


int main()
{
  printf("_____2 QUBITs____\n");

  double complex a = 1;
  double complex b = 0;
  double complex c = 1;
  double complex d = 0;

  double complex norm1 =
    csqrt(1/(a*conj(a)+b*conj(b)));
  double complex norm2 =
    csqrt(1/(c*conj(c)+d*conj(d)));

  ket2 psi2 = crea_ket2(a*norm1,b*norm1,c*norm2,d*norm2,base_std2);

  op2_2 o2 = crea_tensore2_2(
                       1,0,0,0,
                       0,1,0,0,
                       0,0,0,1,
                       0,0,1,0,
                       base_std2_2);

 ket2  psip2 = trasforma_ket2(o2,psi2);

  printf("Operatore CNOT:\n");
  printf("%g+i%g,%g+i%g,%g+i%g,%g+i%g -> ",psi2.x11, psi2.x12,psi2.x21,
 psi2.x22);
  printf("%g+i%g,%g+i%g,%g+i%g,%g+i%g\n",psip2.x11, psip2.x12,psip2.x21,
 psip2.x22);

  a = 0;
  b = 1;
  c = 0;
  d = 1;

  norm1 =
    csqrt(1/(a*conj(a)+b*conj(b)));
  norm2 =
    csqrt(1/(c*conj(c)+d*conj(d)));

  psi2 = crea_ket2(a*norm1,b*norm1,c*norm2,d*norm2,base_std2);

  psip2 = trasforma_ket2(o2,psi2);


  printf("Operatore CNOT:\n");
  printf("%g+i%g,%g+i%g,%g+i%g,%g+i%g -> ",psi2.x11, psi2.x12,psi2.x21,
 psi2.x22);
  printf("%g+i%g,%g+i%g,%g+i%g,%g+i%g\n",psip2.x11, psip2.x12,psip2.x21,
 psip2.x22);

}
```

Compilando ed eseguendo il codice si vede che il ket di stato passa da $|0\rangle$ a $|1\rangle$, quindi il qubit passa da 0 ad 1.

3. **Soluzione**: il primo passaggio per risolvere questo esercizio è di creare uno stato entangled. È stato spiegato nella teoria che uno stato entangled è tale perché non può essere scritto come il prodotto di due stati singoli, per tale motivo non è possibile usare la funzione `crea_ket2`, ma il ket deve essere creato specificando manualmente le sue componenti.
   Usando l'operatore σ_x si deve trasformare il qubit del signor Fabbri in modo da codificare il numero 1, cioè il SUD (con riferimento all'esempio della teoria).
   Per decodificare lo stato si deve operare in successione con il gate CNOT e il gate $H \otimes I$, le cui componenti devono essere calcolate sviluppando il prodotto tensoriale dell'operare H e I.

```
#include
#include
#include
#include "libSSQ.h"

extern base2 base_std2;
extern base2_2 base_std2_2;


int main()
{
  printf("_____Dense Coding____\n\n");

  printf("1)Viene creato e condiviso uno stato entangled |00>+|11> tra
 Fabbri e Magri");
  double complex norm = 1/sqrt(2);
  //Non può essere costruito come il prodotto di due ket
  ket2 ent_k = {1*norm,0,0,1*norm,base_std2};

  //Operatore XxI
  op2_2 o2 = crea_tensore2_2(
                       0,1,0,0,
                       1,0,0,0,
                       0,0,0,1,
                       0,0,1,0,
                       base_std2_2);

 ket2  psip2 = trasforma_ket2(o2,ent_k);

 printf("\n\n2)Fabbri codifica il valore 1 (SUD) usando l'operatore
 XxI:\n\n");

 printf("\t%g+i%g,%g+i%g,%g+i%g,%g+i%g -> ",ent_k.x11, ent_k.x12,
 ent_k.x21, ent_k.x22);
 printf("\t%g+i%g,%g+i%g,%g+i%g,%g+i%g\n",psip2.x11, psip2.x12,
 psip2.x21, psip2.x22);

 printf("\n3)Fabbri invia il proprio qubit a Magri\n");
```

```
printf("\n4)Magri applica un operatore CNOT e uno HxI:\n\n");
//Passo1: trasforma con Cnot
o2 = crea_tensore2_2(
                      1,0,0,0,
                      0,1,0,0,
                      0,0,0,1,
                      0,0,1,0,
                      base_std2_2);

ket2 psipp2 = trasforma_ket2(o2,psip2);

printf("\t%g+i%g,%g+i%g,%g+i%g,%g+i%g -> ",psip2.x11, psip2.x12,
psip2.x21, psip2.x22);
printf("\t%g+i%g,%g+i%g,%g+i%g,%g+i%g\n",psipp2.x11, psipp2.x12,
psipp2.x21, psipp2.x22);

double complex h = norm*1;
o2 = crea_tensore2_2(
                      h,0,h,0,
                      0,h,0,h,
                      h,0,-h,0,
                      0,h,0,-h,
                      base_std2_2);

ket2 psippp2 = trasforma_ket2(o2,psipp2);

printf("\t%g+i%g,%g+i%g,%g+i%g,%g+i%g -> ",psipp2.x11, psipp2.x12,
psipp2.x21, psipp2.x22);
printf("\t%g+i%g,%g+i%g,%g+i%g,%g+i%g\n",psippp2.x11, psippp2.x12,
psippp2.x21, psippp2.x22);
printf("\n5)Magri ha ricevuto |01> cioè SUD\n\n");


}
```

Compilando ed eseguendo il codice si vede che il ket entangled passa dallo stato $|00\rangle + |11\rangle$ allo stato $|01\rangle$ come atteso.

Conclusioni

Concludiamo questo testo con una domanda:

La computazione quantistica prenderà mai il posto di quella classica?

Probabilmente questa domanda sta spaventando molti, sia tra i programmatori che tra i grandi della produzione industriale. È ovvio che l'idea che una nuova tecnologia possa prendere il posto di quella a cui si è dedicato energie e risorse non sia piacevole.
Penso però che chi si occupa di informatica classica possa stare relativamente tranquillo.
Come scrisse Landau, la formulazione stessa della Meccanica Quantistica è intrinsecamente impossibile senza l'inclusione della meccanica classica, e penso che lo stesso valga per la computazione quantistica.
L'informatica quantistica sfrutta caratteristiche della fisica che non sono apprezzabili nel mondo classico in cui vive l'essere umano, per questo credo sarà necessario anche in futuro disporre di sistemi informatici classici per *trasportare* i risultati quantistici nel mondo classico.

Tirando le somme, è probabile che la computazione quantistica si svilupperà molto in futuro, ma anche che l'informatica classica rimarrà al suo fianco, allo stesso modo in cui la fisica classica continua a servire da supporto a quella quantistica.

Come approfondire gli argomenti

Gli argomenti che sono stati trattati in questo testo sono tutti piuttosto complessi e meritevoli ognuno di uno studio dedicato.
Chi fosse interessato ad approfondirli per completare la propria conoscenza può iniziare leggendo i libri e gli articoli riportati nel capitolo finale della bibliografia. In base alla propria

preparazione di partenza potrà trovare i riferimenti citati come utili o difficoltosi, in ogni caso costituiscono un punto di partenza.
Buon proseguimento.

Risposte alle domande

In alcuni e-reader, la numerazione delle risposte possibili alle domande compare come 1, 2, 3 anziché a), b), c). In quel caso, si interpretino le risposte: a come 1, b come 2 e c come 3.

- **1** - a.
- **2** - b.
- **3** - b.
- **4** - a.
- **5** - b.
- **6** - a.
- **7** - a.
- **8** - b.
- **9** - c.
- **10** - c.
- **11** - c.
- **12** - c.
- **13** - b.
- **14** - a.
- **15** - a.
- **16** - c.
- **17** - a.
- **18** - a.
- **19** - b.
- **20** - a.

Bibliografia e riferimenti

- Barz, S. et al. A two-qubit photonic quantum processor and its application to solving systems of linear equations. Sci. Rep. 4, 6115 (2014).
- Bennett, C. H. The thermodynamics of computation—a review. International Journal of Theoretical Physics. 21 (12): 905–940, (1982).
- Bohr, N. The Quantum Postulate and the Recent Development of Atomic Theory. Nature (1928)
- Dirac, P. The Principles of Quantum Mechanics. Snowball publishing (1957).
- Fermi,E. Termodinamica. Bollati Boringhieri. (1958).
- Feynman, R. P. Simulating physics with computers. Int. J. Theor. Phys. (1982).
- Jogenfors, J. Breaking the Unbreakable: Exploiting Loopholes in Bell's Theorem to Hack Quantum Cryptography. DOI:10.3384/diss.diva-140912 (2017)
- Karthik S. et al., Efficient quantum algorithm for solving travelling salesman problem: An IBM quantum experience. arXiv:1805.10928 (2018)
- Knill E., Laflamme R., Milburn G. Efficient Linear Optics Quantum Computation. arXiv:quant-ph/0006088 (2000)
- Knill, E., Laflamme, R., Milburn, G. A scheme for efficient quantum computation with linear optics. Nature (2001)
- Kocher, A. and Commins, D., Polarization Correlation of Photons Emitted in an Atomic Cascade, Phys. Rev. Lett. (1967)
- Laundau, L., Lifsits, E. Meccanica quantistica, teoria non relativistica. Editori riuniti university press (2010).
- Meany T. et al. Laser written circuits for quantum photonics. Laser Photon. Rev. 9, 363–384 (2015).
- Reck, M., Zeilinger, A., Bernstein, H. J. & Bertani, P. Experimental realization of an discrete unitary operator.

Phys. Rev. Lett. 73, 58 (1994).
- Rieffel E., Polak W. Quantum Computing. A Gentle Introduction. MIT Press (2011)
- Segrè, E. Personaggi e Scoperte della Fisica Contemporanea. A. M. Editore (1996).
- Shannon E. A Mathematical Theory of Communication, Bell System Technical Journal (1948)
- Tonomura, A., Endo J., Matsuda T., and T. Kawasaki. Demonstration of single-electron buildup of an interference pattern. American Journal of Physics (1989).

Altre pubblicazioni di Scuola Sisini

Introduzione alle reti neurali con esempi in linguaggio C

Reti neurali non supervisionate: il cognitrone di Fukushima