Esercizi fatti a lezione

Vittorio Zaccaria

16 ottobre 2018

Indice

Capitolo 1

Esercizi Linguaggio C

1.1 Conversione tempo

Si scriva un programma in linguaggio C con la seguente firma:

$$(\texttt{tempoSec}) \rightarrow (\texttt{h,m,s})$$

Si assuma che i dati in ingresso, se ve ne sono, siano introdotti da tastiera e che i valori elaborati siano stampati a video.

DATI IN INGRESSO

- `tempoSec`: intervallo di tempo espresso in secondi

DATI DA ELABORARE

- `h`: il numero intero di ore corrispondente all'intervallo introdotto
- `m`: il rimanente numero intero di minuti
- `s`: il rimanente numero di secondi

ESEMPIO

```
Inserisci il numero di secondi: 3661
Equivalgono a 1 ore, 1 minuti e 1 secondi
```

Soluzione

```c
#include <stdio.h>
int main()
{
    int sec, min, h;
    printf("Inserisci il numero di secondi:\n");
    scanf("%d", &sec);

    h = sec/3600;
    min = (sec - h*3600)/60;
    sec = sec - h*3600 - min*60;

    printf("Equivalgono a %d ore, %d minuti e %d secondi\n", h, min, sec);
    return 0;
}
```

1.2 Tracing di programma

In questo esercizio, cosidetto *di tracing*, viene richiesto di simulare ”mentalmente” il programma seguente e predire che cosa stamperà a terminale durante la sua esecuzione. In pratica, fate finta di essere voi il calcolatore ed eseguite le istruzioni partendo dalla prima fino a che non raggiungete l'ultima. Utilizzate un foglio di carta per annotare il valore corrente delle variabili e abbiate cura di tenerlo aggiornato ogni volta che eseguite ”mentalmente” una istruzione.

PROGRAMMA DA STUDIARE:

```c
main()
{
    int i1 = 3, i2 = 4;
    float f1 = 15.45, f2 = 3.1415;
    char c1 = 'a', c2 = 'b';

    /*quanto valgono le seguenti operazioni eseguite in sequenza?*/

    i2 = i1 + 5;
    printf("i2 = %d\n", i2);
    f1 = i1 + 1.1;      /* i1 convertito in float */
    printf("f1 = %f\n", f1);
    f2 = f2 * f2;
    printf("f2 = %f\n", f2);
    i1=f2+8;                /* f2 convertito in intero */
    printf("i1 = %d\n", i1);
    i2 = i2 + c1;       /* c1 = 97, i2 = 105 */
    printf("i2 = %d\n", i2);
    c2 = c2 + 3;        /* 98 + 3 */
    printf("c2 = %c (corrisponde al codice ASCII %d)\n", c2, c2);
    system("pause");
}
```

Soluzione

```
i2 = 8
f1 = 4.100000
f2 = 9.869022
i1 = 17
i2 = 105
c2 = e (corrisponde al codice ASCII 101)
```

1.3 Terna pitagorica

In questo esercizio viene richiesta la scrittura di alcuni frammenti di programma in linguaggio C. A meno che non sia richiesto, non è necessario includere file headers di altre librerie o dichiarare un main. Inoltre, il testo non dichiara esplicitamente la firma dell'eventuale algoritmo da scrivere, il tipo dei dati in ingresso e di quelli da elaborare; tali informazioni sono infatti da dedurre dal testo stesso.

TESTO ESERCIZIO:
Scrivere un programma che verifica se una terna di numeri introdotti dall'utente rispetta il teorema di Pitagora:

$$x^2 + y^2 = z^2$$

Soluzione

```c
#include <stdio.h>
int main() {
        int cat1, cat2, ip;
        printf("Scrivi il valore del primo cateto:\n");
        scanf("%d", & cat1);
        printf("Scrivi il valore del secondo cateto:\n");
        scanf("%d", & cat2);
        printf("Scrivi il valore dell'ipotenusa:\n");
        scanf("%d", & ip);
        if(cat1*cat1 + cat2*cat2 == ip*ip) {
                printf("La terna e' pitagorica\n");
        }
        else {
                printf("La terna non e' pitagorica\n");
        }
}
```

1.4 Tracing di programma

In questo esercizio, cosidetto *di tracing*, viene richiesto di simulare ”mentalmente” il programma seguente e predire che cosa stamperà a terminale durante la sua esecuzione. In pratica, fate finta di essere voi il calcolatore ed eseguite le istruzioni partendo dalla prima fino a che non raggiungete l'ultima. Utilizzate un foglio di carta per annotare il valore corrente delle variabili e abbiate cura di tenerlo aggiornato ogni volta che eseguite ”mentalmente” una istruzione.

PROGRAMMA DA STUDIARE:

```c
int main() {
  int a = 0;
  if (a = 1) {
    printf("A è uguale a 1");
  } else {
    printf("A è uguale a 0");
  }
}
```

Soluzione

L'uso dell'operatore di assegnamento = porta ad eseguire il ramo *then* dell'istruzione di controllo `if` (il valore di un assegnamento è il valore assegnato) quindi viene stampato `A è uguale a 1`.

1.5 Tracing di programma

In questo esercizio, cosidetto *di tracing*, viene richiesto di simulare ”mentalmente” il programma seguente e predire che cosa stamperà a terminale durante la sua esecuzione. In pratica, fate finta di essere voi il calcolatore ed eseguite le istruzioni partendo dalla prima fino a che non raggiungete l'ultima. Utilizzate un foglio di carta per annotare il valore corrente delle variabili e abbiate cura di tenerlo aggiornato ogni volta che eseguite ”mentalmente” una istruzione.

PROGRAMMA DA STUDIARE:

```
1  int main() {
2    int s, i, j;
3    s = 0;
4    for (i = 1; i <= 10; i++) {
5      j = i * 2;
6      /* Misura I e J */
7      while (j > 0) {
8        s = s + 1;
9        j = j - 1;
10     }
11     /* Misura S */
12     if (s % 2 == 0)
13       printf("%d", s);
14   }
15 }
```

Soluzione

Per stabilire cosa (e quando) viene stampato alla linea 13, dobbiamo seguire l'andamento di `s` (la condizione alla riga 12 infatti dipende da `s`). A sua volta, `s` dipende da `i` e `j`; bisogna quindi ricavare l'andamento di `i` e `j` come prima cosa.
Per comodità, fissiamo alla riga 6 ed alla riga 11 i punti in cui ”virtualmente” misuriamo il valore di `i, j` ed `s.` Ogni qualvolta che, eseguendo una istruzione alla volta, passeremo attraverso quelle righe, aggiungeremo i valori correnti delle variabili alla seguente tabella:

```
1  i alla riga 6  = 1 2 3  4  5  6  7  8  9  10
2  j alla riga 6  = 2 4 6  8  10 12 14 16 18 20
3  s alla riga 11 = 2 6 12 20 30 42 56 72 90 110
```

Dato l'andamento di `s`, l'istruzione `printf` verrà sempre eseguita (poiché `s` è sempre pari). Ciò significa che al terminale verrà stampato semplicemente il suo valore:

```
1  2 6 12 20 30 42 56 72 90 110
```

Capitolo 2

Esercizi Linguaggio C - Array semplici e stringhe

2.1 Calcolo del valore massimo all'interno di un vettore

Si scriva un programma in linguaggio C con la seguente firma:

$$(N, \texttt{n}_1, \ldots, \texttt{n}_N) \rightarrow (R)$$

Si assuma che i dati in ingresso, se ve ne sono, siano introdotti da tastiera e che i valori elaborati siano stampati a video.

DATI IN INGRESSO

- N: Rappresenta il numero di valori della sequenza inserita successivamente dall'utente
- n_i: Un valore intero inserito dall'utente come i -simo elemento

DATI DA ELABORARE

- `R`: Il massimo dei valori n_i

ESEMPIO

```
Di quanti valori vuoi calcolare il massimo? 3
Inserisci il valore 1: 7
Inserisci il valore 2: 3
Inserisci il valore 3: -1

Il valore massimo e': 7
```

ULTERIORI VINCOLI E SPIEGAZIONI: Si crei un array che riesca a contenere 50 elementi e si memorizzino i valori inseriti in tale array. Si controlli che il valore di N sia maggiore di zero e inferiore a 50 prima di richiedere i numeri. Nel caso il valore di N sia maggiore di 50, richiederne il valore un'altra volta.

Soluzione

```c
#include <stdio.h>

#define MAX 50

int main() {
  int N;
  int numeri[N];
  int i;
```

```
  int R;
  do {
    printf("Di quanti valori vuoi calcolare il massimo?");
    scanf("%d", &N);
  } while (N > 50 || N <= 0);
  for (i = 0; i < N; i++) {
    printf("Inserisci il valore %d:", i + 1);
    scanf("%d", &numeri[i]);
  }
  R = numeri[0];
  for (i = 1; i < N; i++) {
    if (numeri[i] > R) {
      R = numeri[i];
    }
  }
  printf("Il valore massimo e': %d", R);
  return 0;
}
```

2.2 Tabella caratteri ASCII

Si scriva un programma in linguaggio C con la seguente firma:

$$() \rightarrow (\texttt{l}_1, \texttt{l}_2, \ldots)$$

Si assuma che i dati in ingresso, se ve ne sono, siano introdotti da tastiera e che i valori elaborati siano stampati a video.

DATI DA ELABORARE

- $\texttt{l}_i$: rappresente il carattere i-esimo dell'alfabeto

ESEMPIO

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

ULTERIORI VINCOLI E SPIEGAZIONI: Non è possibile usare più di due `printf` nel codice. Si suggerisce di usare un ciclo `for` e di non introdurre arrays.

Soluzione

```c
#include <stdio.h>

int main() {
  char c;
  for (c = 'A'; c <= 'Z'; c++) {
    printf("%c", c);
  }
  return 0;
}
```

2.3 Conversione stringa da caratteri minuscoli a maiuscoli

In questo esercizio viene richiesta la scrittura di alcuni frammenti di programma in linguaggio C. A meno che non sia richiesto, non è necessario includere file headers di altre librerie o dichiarare un main. Inoltre, il testo non dichiara esplicitamente la firma dell'eventuale algoritmo da scrivere, il tipo dei dati in ingresso e di quelli da elaborare; tali informazioni sono infatti da dedurre dal testo stesso.

TESTO ESERCIZIO:

Scrivere un programma che converte una stringa di caratteri inseriti dall'utente in maiuscolo.

Soluzione

```c
#include <stdio.h>
#define MAX_LEN 100

int main() {
  int offset = 'A' - 'a';
  char s[MAX_LEN];
  int i = 0;
  printf(
      "Inserisci una stringa di caratteri minuscoli (no spazi, no numeri):\n");
  scanf("%s", s);
  while (s[i] != '\0') {
    if (s[i] >= 'a' && s[i] <= 'z') {
      s[i] += offset;
    }
    i++;
  }
  printf("La stringa ora e' %s\n", s);
  return 0;
}
```

2.4 Stringhe palindrome

Si scriva un programma in linguaggio C con la seguente firma:

$$(\texttt{parola}) \rightarrow (\texttt{messaggio})$$

Si assuma che i dati in ingresso, se ve ne sono, siano introdotti da tastiera e che i valori elaborati siano stampati a video.

DATI IN INGRESSO

- `parola`: una stringa di massimo 50 caratteri

DATI DA ELABORARE

- `messaggio`: Si veda gli esempi sotto

ESEMPIO

```
Inserisci una parola: pippo
'pippo' NON è una parola palindroma
```

ESEMPIO

```
Inserisci una parola: anilina
'anilina' è una parola palindroma
```

ULTERIORI VINCOLI E SPIEGAZIONI: Una stringa è ***palindroma*** se, letta da destra a sinistra, equivale alla stessa letta da sinistra a destra.

Soluzione

```c
#include <stdio.h>
#include <string.h>

#define MAX 50

int main() {
  char parola[MAX];
  int i, palindroma, len;

  printf("Inserisci una parola: ");
  scanf("%s", parola);

  len = strlen(parola);
  palindroma = 1;

  for (i = 0; i < len / 2 && palindroma != 0; i++) {
    if (parola[i] != parola[len - 1 - i])
      palindroma = 0;
  }

  printf("'%s' ", parola);

  if (palindroma == 0)
    printf("NON ");

```

```
26    printf("è una parola palindroma\n");
27
28    return 0;
29  }
```

2.5 Conta i caratteri

In questo esercizio viene richiesta la scrittura di alcuni frammenti di programma in linguaggio C. A meno che non sia richiesto, non è necessario includere file headers di altre librerie o dichiarare un main. Inoltre, il testo non dichiara esplicitamente la firma dell'eventuale algoritmo da scrivere, il tipo dei dati in ingresso e di quelli da elaborare; tali informazioni sono infatti da dedurre dal testo stesso.

TESTO ESERCIZIO:

Si supponga di avere una stringa `str` contenente al massimo 100 caratteri alfabetici, senza spazi, ad esempio:

```c
char str[100] = "aadddfffzzzzdd";
```

Scrivere una porzione di codice che, per ogni carattere `c` ***a partire dall'ultimo fino ad arrivare al primo***, stampi senza lasciare spazi il carattere `c`, seguito dal numero di volte che questo compare consecutivamente in `str`.
Ad esempio, per la stringa di cui sopra, il programma deve stampare:

```
d2z4f3d3a2
```

RISPOSTA/SOLUZIONE:

```c
char c;
int freq = 1, i, n = strlen(str) - 1;
c = str[n];
for (i = n - 1; i >= 0; i--) {
  if (str[i] == c) {
    freq++;
  } else {
    printf("%c%d", c, freq);
    freq = 1;
    c = str[i];
  }
}
printf("%c%d\n", c, freq);
```

Capitolo 3

Esercizi Linguaggio C - Numerica e ordinamento

3.1 Stampa divisori di un numero

Si scriva un programma in linguaggio C con la seguente firma:

$$(\texttt{numero}) \rightarrow (\texttt{d}_1\texttt{, d}_2\texttt{, ..., d}_k)$$

Si assuma che i dati in ingresso, se ve ne sono, siano introdotti da tastiera e che i valori elaborati siano stampati a video.

DATI IN INGRESSO

- `numero`: numero intero, maggiore di 0, di cui bisogna trovare i k divisori

DATI DA ELABORARE

- $\texttt{d}_i$: i-esimo divisore di numero

ESEMPIO

```
Inserisci un numero: 10
I divisori sono:
2
5
```

ESEMPIO

```
Inserisci un numero: 20
I divisori sono:
2
4
5
10
```

ULTERIORI VINCOLI E SPIEGAZIONI: Si ricordi che uno dei modi in cui e' possibile verificare se un numero (positivo) e' divisibile per un altro e' verificare se il resto della divisione tra il primo e il secondo numero sia nullo; ovvero, `n` e' divisibile per `i` se `(n % i)` e' uguale a 0.

Soluzione

```c
#include <stdio.h>
int main() {
  int n, i;
  printf("Inserisci un numero: ");
  scanf("%d", &n);
  printf("I divisori sono: ");
  i = 1;
  while (i <= n) {
    if (n % i == 0)
      printf("%d ", i);
    i++;
  }
  printf("\n");
  return 0;
}
```

3.2 Numeri di Fibonacci

Si scriva un programma in linguaggio C con la seguente firma:

$$(\texttt{n}) \rightarrow (\texttt{f}_0\texttt{, ... f}_n)$$

Si assuma che i dati in ingresso, se ve ne sono, siano introdotti da tastiera e che i valori elaborati siano stampati a video.

DATI DA ELABORARE

- $\texttt{f}_j$: il numero della serie di fibonacci in posizione j

DATI IN INGRESSO

- `n`: l'indice dell'ultimo numero della serie da stampare. Si assuma che l'utente inserisca un numero maggiore o uguale a 1

ESEMPIO

```
Indice dell'ultimo numero della serie da stampare: 5
0
1
1
2
3
5
```

ULTERIORI VINCOLI E SPIEGAZIONI:
Ricordiamo che la successione dei numeri di Fibonacci

$$[f_0, f_1, \ldots f_j \ldots]$$

è definita come segue:

$$f_j = \begin{cases} 0 & \text{se la posizione } j = 0 \\ 1 & \text{se la posizione } j = 1 \\ f_{j-1} + f_{j-2} & \text{se la posizione } j > 1 \end{cases} \tag{3.1}$$

Ulteriori vincoli sono i seguenti:

- La sequenza di valori f_j deve essere prodotta con un ciclo
- Non è possibile utilizzare array.
- Il massimo numero di variabili `int` utilizzabili è 5.

Suggerimento: si consiglia, per ciascuna iterazione j di utilizzare due variabili `f1` ed `f2`, che contengano, rispettivamente, i valori di f_{j-1} ed f_{j-2}, da aggiornare ad ogni iterazione.

Soluzione

```c
#include <stdio.h>
main() {
  int f0, f1, f2, j, n;
  printf("Indice dell'ultimo numero della serie da stampare (>1): ");
  scanf("%d", &n);

  printf("0\n");
  printf("1\n");

  f2 = 0;
  f1 = 1;
  j = 2;
  while (j <= n) {
    f0 = f1 + f2;
    printf("%d\n", f0);
    f2 = f1;
    f1 = f0;
    j = j + 1;
  }
}
```

3.3 Bubble sort

In questo esercizio viene richiesta la scrittura di alcuni frammenti di programma in linguaggio C. A meno che non sia richiesto, non è necessario includere file headers di altre librerie o dichiarare un main. Inoltre, il testo non dichiara esplicitamente la firma dell'eventuale algoritmo da scrivere, il tipo dei dati in ingresso e di quelli da elaborare; tali informazioni sono infatti da dedurre dal testo stesso.

TESTO ESERCIZIO:

Si scriva un programma che richieda una sequenza di numeri interi all'utente e la stampi ordinata in modo crescente. Si usi l'algoritmo del Bubble Sort.

Soluzione

```c
#include <stdio.h>
#define DIMENSIONE_ARRAY 10
int main() {
  int elenco[DIMENSIONE_ARRAY];
  int i, j, temporaneo, n;
  do {
    printf("Inserisci il numero di elementi: ");
    scanf("%d", &n);
  } while (n >= DIMENSIONE_ARRAY);
  for (i = 0; i < n; i++) {
    printf("Inserisci elemento numero %d: ", i);
    scanf("%d", &elenco[i]);
  }
  for (i = 0; i < n; i++) {
    for (j = 0; j < n - 1; j++) {
      if (elenco[j] > elenco[j + 1]) {
        temporaneo = elenco[j + 1];
        elenco[j + 1] = elenco[j];
        elenco[j] = temporaneo;
      }
    }
  }
  printf("Array ordinato: ");
  for (i = 0; i < n; i++) {
    printf("%d ", elenco[i]);
  }
}
```

Capitolo 4

Esercizi Linguaggio C - Strutture dati e array

4.1 Memorizzazione di date

In questo esercizio viene richiesta la scrittura di alcuni frammenti di programma in linguaggio C. A meno che non sia richiesto, non è necessario includere file headers di altre librerie o dichiarare un main. Inoltre, il testo non dichiara esplicitamente la firma dell'eventuale algoritmo da scrivere, il tipo dei dati in ingresso e di quelli da elaborare; tali informazioni sono infatti da dedurre dal testo stesso.

TESTO ESERCIZIO:

Si chiede di estendere i tipi del C per rappresentare, in forma organica, i seguenti tipi:

- un tipo di dato `tipo_orario` atto a rappresentare un classico orario di ore, minuti e secondi.
- un tipo di dato `tipo_data` atto a rappresentare una classica data dell'anno.
- (usando i tipi definiti precedentemente) un tipo di dato `tipo_evento` atto a rappresentare un'evento caratterizzato da data e orario.
- un tipo di dato `tipo_programma` atto a rappresentare l'evento della trasmissione di un programma di cui si conosce il nome.
- due variabili, `palinsesto` e `primaserata`, di tipo array di 30 elementi di `tipo_programma`.

Si chiede inoltre di scrivere una parte di programma C che copi in `primaserata` tutti gli elementi di `palinsesto` che sono trasmessi tra le 20 e le 22.

4.2 Calcolo distanza tra due punti su piano cartesiano

Si scriva un programma in linguaggio C che implementi un algoritmo con la seguente firma:

$$(\texttt{a}_x\texttt{, a}_y\texttt{, b}_x\texttt{, b}_y) \rightarrow (\texttt{dist(a,b)})$$

Si assuma che i dati in ingresso, se ve ne sono, siano introdotti da tastiera e che i valori elaborati siano stampati a video. Il programma deve ***ripetutamente*** eseguire l'algoritmo finchè una ***particolare condizione*** di uscita non è verificata.

DATI IN INGRESSO

- $\texttt{a}_x$: coordinata x del punto cartesiano a
- $\texttt{a}_y$: coordinata y del punto cartesiano a
- $\texttt{b}_x$: coordinata x del punto cartesiano b
- $\texttt{b}_y$: coordinata y del punto cartesiano b

DATI DA ELABORARE

- `dist(a,b)`: distanza euclidea fra a e b

CONDIZIONE DI USCITA: Il programma deve terminare se tutte e 4 le coordinate in ingresso sono pari a 0.
ULTERIORI VINCOLI E SPIEGAZIONI: Deve essere definito ed utilizzato un nuovo tipo `Punto` atto a rappresentare in maniera organica un punto bidimensionale.

Soluzione

```c
#include <math.h>
#include <stdio.h>

typedef struct {
  float x;
  float y;
} Punto;

int main() {
  Punto a, b;
  int uscita = 0;
  float distanza;

  do {
    printf("punto 1, coord x: ");
    scanf("%f", &a.x);

    printf("punto 1, coord y: ");
    scanf("%f", &a.y);

    printf("punto 2, coord x: ");
    scanf("%f", &b.x);

    printf("punto 2, coord y: ");
    scanf("%f", &b.y);

    if (a.x == 0 && a.y == 0 && b.x == 0 && b.y == 0) {
      uscita = 1;
```

```
29        printf("Esco dal programma...\n");
30      } else {
31        distanza = sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
32        printf("distanza: %f\n", distanza);
33      }
34    } while (uscita == 0);
35
36    return 0;
37  }
```

Capitolo 5

Esercizi su linguaggio C consigliati

I seguenti sono esercizi semplici che possono essere verificati direttamente al calcolatore. Fallire i primi esercizi indica problemi gravi con la preparazione per l'esame.

1. Qual'è il valore della variabile `b` al termine di questo frammento di programma C?

```
int a;
int b;
a = 3;
b = 2*a;
a = 6;
```

2. Qual'è il valore della variabile `b` al termine di questo frammento di programma C?

```
int a;
int b;
a = 3;
b = 2*a;
a = 6;
b = b*a;
```

3. Qual'è il valore della variabile `b` al termine di questo frammento di programma C?

```
int a;
int b=0;
a = 3;
while(a--) b++;
```

4. Qual'è il valore della variabile `b` al termine di questo frammento di programma C?

```
int a;
int b=0;
a = 3;
while(--a) b++;
```

5. Qual'è il valore della variabile `b` al termine di questo frammento di programma C?

```
int a;
int b=1;
a = 3;
while(--a && b) b--;
```

6. Qual'è il problema con questo programma C?

```
int a;
int b;
a = 3;
b = 2*b*a;
a = 6;
b = b*a;
```

7. Qual'è il valore della variabile `b` al termine di questo frammento di programma C?

```
int a;
int b=1;
for(a=4; a>0; a--) b++;
```

8. Qual'è il valore della variabile `b` al termine di questo frammento di programma C?

```
int a;
int b=1;
for(a=4; a>=0; a--) b++;
```

9. Scrivere un programma che data una stringa `s` da tastiera, la copia in un'altra stringa `d` senza usare la funzione `strcpy`.
10. Scrivere un programma che data una stringa `s` da tastiera, ne calcola e stampa la lunghezza senza usare la funzione `strlen`.
11. Scrivere un programma che date due stringhe `nome` e `cognome` lette da tastiera, le concatena in un unica stringa `nomeECognome` la stampa a video senza usare la funzione `strcat`.
12. Scrivere un programma che dato un vettore di 5 stringhe lette da tastiera, lo ordina usando bubble sort, `strcmp` e `strcpy` (si ricordi che non è possibile assegnare fra di loro stringhe con l'operatore di assegnamento).
13. Modificare il programma che determina se una stringa è palindroma in modo tale che ignori spazi eventualmente presenti nella stringa stessa. Ad esempio la stringa `occorre portar aratro per rocco` è palindroma se gli spazi vengono
14. Scrivere un programma che data una stringa `s` calcoli la frequenza di tutti i caratteri e la stampi a video.
15. Scrivere un programma che legga una serie di caratteri da tastiera e, solo se questi sono tutti numeri, converta il numero corrispondente in una variabile di tipo `int` e la stampi (non si considerino numeri negativi).
16. Scrivere un programma come quello precedente ma considerando che può essere presente un meno (`-`) prima della prima cifra e che quindi il numero possa essere negativo.

17. Scrivere un programma come quello precedente ma considerando che può essere un separatore di decimali e che quindi scriva il numero corrispondente in una variabile di tipo `float`.

18. Scrivere un programma che lette le dimensioni di una matrice ed il valore di ciascun suo elemento determini se questa sia ”diagonale”.

19. Scrivere un programma che lette le dimensioni di una matrice ed il valore di ciascun suo elemento determini se questa sia ”simmetrica”.

Capitolo 6

Informazioni utili

In questo capitolo sono riportate alcune tabelle e guide di riferimento utili alla soluzione degli esercizi.

REGULAR ASCII CHART (character codes 0 – 127)

000d	00h	␀	(nul)	016d	10h	►	(dle)	032d	20h	␣	048d	30h	0	064d	40h	@	080d	50h	P	096d	60h	‘	112d	70h	p
001d	01h	☺	(soh)	017d	11h	◄	(dc1)	033d	21h	!	049d	31h	1	065d	41h	A	081d	51h	Q	097d	61h	a	113d	71h	q
002d	02h	☻	(stx)	018d	12h	↕	(dc2)	034d	22h	"	050d	32h	2	066d	42h	B	082d	52h	R	098d	62h	b	114d	72h	r
003d	03h	♥	(etx)	019d	13h	‼	(dc3)	035d	23h	#	051d	33h	3	067d	43h	C	083d	53h	S	099d	63h	c	115d	73h	s
004d	04h	♦	(eot)	020d	14h	¶	(dc4)	036d	24h	$	052d	34h	4	068d	44h	D	084d	54h	T	100d	64h	d	116d	74h	t
005d	05h	♣	(enq)	021d	15h	§	(nak)	037d	25h	%	053d	35h	5	069d	45h	E	085d	55h	U	101d	65h	e	117d	75h	u
006d	06h	♠	(ack)	022d	16h	▬	(syn)	038d	26h	&	054d	36h	6	070d	46h	F	086d	56h	V	102d	66h	f	118d	76h	v
007d	07h	•	(bel)	023d	17h	↨	(etb)	039d	27h	'	055d	37h	7	071d	47h	G	087d	57h	W	103d	67h	g	119d	77h	w
008d	08h	◘	(bs)	024d	18h	↑	(can)	040d	28h	(	056d	38h	8	072d	48h	H	088d	58h	X	104d	68h	h	120d	78h	x
009d	09h		(tab)	025d	19h	↓	(em)	041d	29h	)	057d	39h	9	073d	49h	I	089d	59h	Y	105d	69h	i	121d	79h	y
010d	0Ah	◙	(lf)	026d	1Ah	→	(eof)	042d	2Ah	*	058d	3Ah	:	074d	4Ah	J	090d	5Ah	Z	106d	6Ah	j	122d	7Ah	z
011d	0Bh	♂	(vt)	027d	1Bh	←	(esc)	043d	2Bh	+	059d	3Bh	;	075d	4Bh	K	091d	5Bh	[	107d	6Bh	k	123d	7Bh	{
012d	0Ch		(np)	028d	1Ch	∟	(fs)	044d	2Ch	’	060d	3Ch	<	076d	4Ch	L	092d	5Ch	\	108d	6Ch	l	124d	7Ch	|
013d	0Dh	♪	(cr)	029d	1Dh	↔	(gs)	045d	2Dh	-	061d	3Dh	=	077d	4Dh	M	093d	5Dh	]	109d	6Dh	m	125d	7Dh	}
014d	0Eh	♫	(so)	030d	1Eh	▲	(rs)	046d	2Eh	.	062d	3Eh	>	078d	4Eh	N	094d	5Eh	^	110d	6Eh	n	126d	7Eh	~
015d	0Fh	☼	(si)	031d	1Fh	▼	(us)	047d	2Fh	/	063d	3Fh	?	079d	4Fh	O	095d	5Fh	_	111d	6Fh	o	127d	7Fh	⌂

EXTENDED ASCII CHART (character codes 128 – 255) LATIN1/CP1252

128d	80h	€	144d	90h		160d	A0h	␠	176d	B0h	°	192d	C0h	À	208d	D0h	Ð	224d	E0h	à	240d	F0h	ð
129d	81h		145d	91h	‘	161d	A1h	¡	177d	B1h	±	193d	C1h	Á	209d	D1h	Ñ	225d	E1h	á	241d	F1h	ñ
130d	82h	‚	146d	92h	’	162d	A2h	¢	178d	B2h	²	194d	C2h	Â	210d	D2h	Ò	226d	E2h	â	242d	F2h	ò
131d	83h	*f*	147d	93h	“	163d	A3h	£	179d	B3h	³	195d	C3h	Ã	211d	D3h	Ó	227d	E3h	ã	243d	F3h	ó
132d	84h	„	148d	94h	”	164d	A4h	¤	180d	B4h	´	196d	C4h	Ä	212d	D4h	Ô	228d	E4h	ä	244d	F4h	ô
133d	85h	…	149d	95h	•	165d	A5h	¥	181d	B5h	µ	197d	C5h	Å	213d	D5h	Õ	229d	E5h	å	245d	F5h	õ
134d	86h	†	150d	96h	–	166d	A6h	¦	182d	B6h	¶	198d	C6h	Æ	214d	D6h	Ö	230d	E6h	æ	246d	F6h	ö
135d	87h	‡	151d	97h	—	167d	A7h	§	183d	B7h	·	199d	C7h	Ç	215d	D7h	×	231d	E7h	ç	247d	F7h	÷
136d	88h	ˆ	152d	98h	˜	168d	A8h	¨	184d	B8h	¸	200d	C8h	È	216d	D8h	Ø	232d	E8h	è	248d	F8h	ø
137d	89h	‰	153d	99h	™	169d	A9h	©	185d	B9h	¹	201d	C9h	É	217d	D9h	Ù	233d	E9h	é	249d	F9h	ù
138d	8Ah	Š	154d	9Ah	š	170d	AAh	ª	186d	BAh	º	202d	CAh	Ê	218d	DAh	Ú	234d	EAh	ê	250d	FAh	ú
139d	8Bh	‹	155d	9Bh	›	171d	ABh	«	187d	BBh	»	203d	CBh	Ë	219d	DBh	Û	235d	EBh	ë	251d	FBh	û
140d	8Ch	Œ	156d	9Ch	œ	172d	ACh	¬	188d	BCh	¼	204d	CCh	Ì	220d	DCh	Ü	236d	ECh	ì	252d	FCh	ü
141d	8Dh		157d	9Dh		173d	ADh		189d	BDh	½	205d	CDh	Í	221d	DDh	Ý	237d	EDh	í	253d	FDh	ý
142d	8Eh	Ž	158d	9Eh	ž	174d	AEh	®	190d	BEh	¾	206d	CEh	Î	222d	DEh	Þ	238d	EEh	î	254d	FEh	þ
143d	8Fh		159d	9Fh	Ÿ	175d	AFh	¯	191d	BFh	¿	207d	CFh	Ï	223d	DFh	ß	239d	EFh	ï	255d	FFh	ÿ

Hexadecimal to Binary

0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

Groups of ASCII-Code in Binary

Bit 6	Bit 5	Group
0	0	Control Characters
0	1	Digits and Punctuation
1	0	Upper Case and Special
1	1	Lower Case and Special

C Reference Card (ANSI)

Program Structure/Functions

type fnc(*type*$_1$, ...);	function prototype
type name;	variable declaration
`int main(void) {`	main routine
declarations	local variable declarations
statements	
`}`	
type fnc(*arg*$_1$, ...) {	function definition
declarations	local variable declarations
statements	
`return` *value*;	
`}`	
`/* */`	comments
`int main(int argc, char *argv[])`	main with args
`exit(`*arg*`);`	terminate execution

C Preprocessor

include library file	`#include <`*filename*`>`
include user file	`#include "`*filename*`"`
replacement text	`#define` *name text*
replacement macro	`#define` *name*`(`*var*`)` *text*
Example. `#define max(A,B) ((A)>(B) ? (A) : (B))`	
undefine	`#undef` *name*
quoted string in replace	`#`
Example. `#define msg(A) printf("%s = %d", #A, (A))`	
concatenate args and rescan	`##`
conditional execution	`#if, #else, #elif, #endif`
is *name* defined, not defined?	`#ifdef, #ifndef`
name defined?	`defined(`*name*`)`
line continuation char	`\`

Data Types/Declarations

character (1 byte)	`char`
integer	`int`
real number (single, double precision)	`float, double`
short (16 bit integer)	`short`
long (32 bit integer)	`long`
double long (64 bit integer)	`long long`
positive or negative	`signed`
non-negative modulo 2^m	`unsigned`
pointer to `int`, `float`,...	`int*, float*,...`
enumeration constant	`enum` *tag* {*name*$_1$=*value*$_1$,...};
constant (read-only) value	*type* `const` *name*;
declare external variable	`extern`
internal to source file	`static`
local persistent between calls	`static`
no value	`void`
structure	`struct` *tag* {...};
create new name for data type	`typedef` *type name*;
size of an object (type is `size_t`)	`sizeof` *object*
size of a data type (type is `size_t`)	`sizeof(`*type*`)`

Initialization

initialize variable	*type name*=*value*;
initialize array	*type name*`[]`={*value*$_1$,...};
initialize char string	`char` *name*`[]="`*string*`";`

Constants

suffix: long, unsigned, float	`65536L, -1U, 3.0F`
exponential form	`4.2e1`
prefix: octal, hexadecimal	`0, 0x` or `0X`
Example. `031` is 25, `0x31` is 49 decimal	
character constant (char, octal, hex)	`'a', '\`*ooo*`', '\x`*hh*`'`
newline, cr, tab, backspace	`\n, \r, \t, \b`
special characters	`\\, \?, \', \"`
string constant (ends with `'\0'`)	`"abc...de"`

Pointers, Arrays & Structures

declare pointer to *type*	*type* `*`*name*;
declare function returning pointer to *type*	*type* `*f();`
declare pointer to function returning *type*	*type* `(*pf)();`
generic pointer type	`void *`
null pointer constant	`NULL`
object pointed to by *pointer*	`*`*pointer*
address of object *name*	`&`*name*
array	*name*`[`*dim*`]`
multi-dim array	*name*`[`*dim*$_1$`][`*dim*$_2$`]`...

Structures

`struct` *tag* {	structure template
declarations	declaration of members
};	
create structure	`struct` *tag name*
member of structure from template	*name*`.`*member*
member of pointed-to structure	*pointer* `->` *member*
Example. `(*p).x` and `p->x` are the same	
single object, multiple possible types	`union`
bit field with *b* bits	`unsigned` *member*`:` *b*;

Operators (grouped by precedence)

struct member operator	*name*`.`*member*
struct member through pointer	*pointer*`->`*member*
increment, decrement	`++, --`
plus, minus, logical not, bitwise not	`+, -, !, ~`
indirection via pointer, address of object	`*`*pointer*`, &`*name*
cast expression to type	`(`*type*`)` *expr*
size of an object	`sizeof`
multiply, divide, modulus (remainder)	`*, /, %`
add, subtract	`+, -`
left, right shift [bit ops]	`<<, >>`
relational comparisons	`>, >=, <, <=`
equality comparisons	`==, !=`
and [bit op]	`&`
exclusive or [bit op]	`^`
or (inclusive) [bit op]	`|`
logical and	`&&`
logical or	`||`
conditional expression	*expr*$_1$ `?` *expr*$_2$ `:` *expr*$_3$
assignment operators	`+=, -=, *=, ...`
expression evaluation separator	`,`

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

Flow of Control

statement terminator	`;`
block delimiters	`{ }`
exit from `switch`, `while`, `do`, `for`	`break;`
next iteration of `while`, `do`, `for`	`continue;`
go to	`goto` *label*`;`
label	*label*`:` `statement`
return value from function	`return` *expr*

Flow Constructions

`if` statement	`if (`*expr*$_1$`)` *statement*$_1$ `else if (`*expr*$_2$`)` *statement*$_2$ `else` *statement*$_3$
`while` statement	`while (`*expr*`)` *statement*
`for` statement	`for (`*expr*$_1$`;` *expr*$_2$`;` *expr*$_3$`)` *statement*
`do` statement	`do` *statement* `while(`*expr*`);`

`switch` statement	`switch (`*expr*`) {`
	`case` *const*$_1$`:` *statement*$_1$ `break;`
	`case` *const*$_2$`:` *statement*$_2$ `break;`
	`default:` *statement*
	`}`

ANSI Standard Libraries

`<assert.h>` `<ctype.h>` `<errno.h>` `<float.h>` `<limits.h>`
`<locale.h>` `<math.h>` `<setjmp.h>` `<signal.h>` `<stdarg.h>`
`<stddef.h>` `<stdio.h>` `<stdlib.h>` `<string.h>` `<time.h>`

Character Class Tests `<ctype.h>`

alphanumeric?	`isalnum(c)`
alphabetic?	`isalpha(c)`
control character?	`iscntrl(c)`
decimal digit?	`isdigit(c)`
printing character (not incl space)?	`isgraph(c)`
lower case letter?	`islower(c)`
printing character (incl space)?	`isprint(c)`
printing char except space, letter, digit?	`ispunct(c)`
space, formfeed, newline, cr, tab, vtab?	`isspace(c)`
upper case letter?	`isupper(c)`
hexadecimal digit?	`isxdigit(c)`
convert to lower case	`tolower(c)`
convert to upper case	`toupper(c)`

String Operations `<string.h>`

`s` is a string; `cs`, `ct` are constant strings

length of `s`	`strlen(s)`
copy `ct` to `s`	`strcpy(s,ct)`
concatenate `ct` after `s`	`strcat(s,ct)`
compare `cs` to `ct`	`strcmp(cs,ct)`
only first `n` chars	`strncmp(cs,ct,n)`
pointer to first `c` in `cs`	`strchr(cs,c)`
pointer to last `c` in `cs`	`strrchr(cs,c)`
copy `n` chars from `ct` to `s`	`memcpy(s,ct,n)`
copy `n` chars from `ct` to `s` (may overlap)	`memmove(s,ct,n)`
compare `n` chars of `cs` with `ct`	`memcmp(cs,ct,n)`
pointer to first `c` in first `n` chars of `cs`	`memchr(cs,c,n)`
put `c` into first `n` chars of `s`	`memset(s,c,n)`

C Reference Card (ANSI)

Input/Output <stdio.h>

Standard I/O

standard input stream	stdin
standard output stream	stdout
standard error stream	stderr
end of file (type is int)	EOF
get a character	getchar()
print a character	putchar(*chr*)
print formatted data	printf("*format*",arg_1,...)
print to string s	sprintf(s,"*format*",arg_1,...)
read formatted data	scanf("*format*",&$name_1$,...)
read from string s	sscanf(s,"*format*",&$name_1$,...)
print string s	puts(s)

File I/O

declare file pointer	FILE *fp;
pointer to named file	fopen("*name*","*mode*")
modes: r (read), w (write), a (append), b (binary)	
get a character	getc(*fp*)
write a character	putc(*chr*,*fp*)
write to file	fprintf(*fp*,"*format*",arg_1,...)
read from file	fscanf(*fp*,"*format*",arg_1,...)
read and store n elts to *ptr	fread(*ptr,eltsize,n,*fp*)
write n elts from *ptr to file	fwrite(*ptr,eltsize,n,*fp*)
close file	fclose(*fp*)
non-zero if error	ferror(*fp*)
non-zero if already reached EOF	feof(*fp*)
read line to string s (< max chars)	fgets(s,max,*fp*)
write string s	fputs(s,*fp*)

Codes for Formatted I/O: "%-+ 0*w.pmc*"

-	left justify
+	print with sign
space	print space if no sign
0	pad with leading zeros
w	min field width
p	precision
m	conversion character: h short, l long, L long double
c	conversion character:
d,i	integer
c	single char
f	double (printf)
f	float (scanf)
o	octal
p	pointer
u	unsigned
s	char string
e,E	exponential
lf	double (scanf)
x,X	hexadecimal
n	number of chars written
g,G	same as f or e,E depending on exponent

Variable Argument Lists <stdarg.h>

declaration of pointer to arguments	va_list *ap*;
initialization of argument pointer	va_start(*ap*,*lastarg*);
lastarg is last named parameter of the function	
access next unnamed arg, update pointer	va_arg(*ap*,*type*)
call before exiting function	va_end(*ap*);

Standard Utility Functions <stdlib.h>

absolute value of int n	abs(n)
absolute value of long n	labs(n)
quotient and remainder of ints n,d	div(n,d)
returns structure with div_t.quot and div_t.rem	
quotient and remainder of longs n,d	ldiv(n,d)
returns structure with ldiv_t.quot and ldiv_t.rem	
pseudo-random integer [0,RAND_MAX]	rand()
set random seed to n	srand(n)
terminate program execution	exit(status)
pass string s to system for execution	system(s)

Conversions

convert string s to double	atof(s)
convert string s to integer	atoi(s)
convert string s to long	atol(s)
convert prefix of s to double	strtod(s,&endp)
convert prefix of s (base b) to long	strtol(s,&endp,b)
same, but unsigned long	strtoul(s,&endp,b)

Storage Allocation

allocate storage	malloc(size), calloc(nobj,size)
change size of storage	newptr = realloc(ptr,size);
deallocate storage	free(ptr);

Array Functions

search array for key	bsearch(key,array,n,size,cmpf)
sort array ascending order	qsort(array,n,size,cmpf)

Time and Date Functions <time.h>

processor time used by program	clock()
Example. clock()/CLOCKS_PER_SEC is time in seconds	
current calendar time	time()
$time_2$-$time_1$ in seconds (double)	difftime($time_2$,$time_1$)
arithmetic types representing times	clock_t,time_t
structure type for calendar time comps	struct tm

tm_sec	seconds after minute
tm_min	minutes after hour
tm_hour	hours since midnight
tm_mday	day of month
tm_mon	months since January
tm_year	years since 1900
tm_wday	days since Sunday
tm_yday	days since January 1
tm_isdst	Daylight Savings Time flag

convert local time to calendar time	mktime(tp)
convert time in tp to string	asctime(tp)
convert calendar time in tp to local time	ctime(tp)
convert calendar time to GMT	gmtime(tp)
convert calendar time to local time	localtime(tp)
format date and time info	strftime(s,smax,"*format*",tp)
tp is a pointer to a structure of type tm	

Mathematical Functions <math.h>

Arguments and returned values are double

trig functions	sin(x), cos(x), tan(x)
inverse trig functions	asin(x), acos(x), atan(x)
arctan(y/x)	atan2(y,x)
hyperbolic trig functions	sinh(x), cosh(x), tanh(x)
exponentials & logs	exp(x), log(x), log10(x)
exponentials & logs (2 power)	ldexp(x,n), frexp(x,&e)
division & remainder	modf(x,ip), fmod(x,y)
powers	pow(x,y), sqrt(x)
rounding	ceil(x), floor(x), fabs(x)

Integer Type Limits <limits.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system, followed by minimum required values (if significantly different).

CHAR_BIT	bits in char		(8)
CHAR_MAX	max value of char	(SCHAR_MAX or UCHAR_MAX)	
CHAR_MIN	min value of char	(SCHAR_MIN or 0)	
SCHAR_MAX	max signed char		(+127)
SCHAR_MIN	min signed char		(−128)
SHRT_MAX	max value of short		(+32,767)
SHRT_MIN	min value of short		(−32,768)
INT_MAX	max value of int	(+2,147,483,647)	(+32,767)
INT_MIN	min value of int	(−2,147,483,648)	(−32,767)
LONG_MAX	max value of long		(+2,147,483,647)
LONG_MIN	min value of long		(−2,147,483,648)
UCHAR_MAX	max unsigned char		(255)
USHRT_MAX	max unsigned short		(65,535)
UINT_MAX	max unsigned int	(4,294,967,295)	(65,535)
ULONG_MAX	max unsigned long		(4,294,967,295)

Float Type Limits <float.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system.

FLT_RADIX	radix of exponent rep	(2)
FLT_ROUNDS	floating point rounding mode	
FLT_DIG	decimal digits of precision	(6)
FLT_EPSILON	smallest x so $1.0f + x \neq 1.0f$	($1.1E-7$)
FLT_MANT_DIG	number of digits in mantissa	
FLT_MAX	maximum float number	($3.4E38$)
FLT_MAX_EXP	maximum exponent	
FLT_MIN	minimum float number	($1.2E-38$)
FLT_MIN_EXP	minimum exponent	
DBL_DIG	decimal digits of precision	(15)
DBL_EPSILON	smallest x so $1.0 + x \neq 1.0$	($2.2E-16$)
DBL_MANT_DIG	number of digits in mantissa	
DBL_MAX	max double number	($1.8E308$)
DBL_MAX_EXP	maximum exponent	
DBL_MIN	min double number	($2.2E-308$)
DBL_MIN_EXP	minimum exponent	

Send comments and corrections to J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. ⟨jhs@math.brown.edu⟩

Aiuto!

`help x`	mostra la documentazione su x
`doc`	apre la documentazione di matlab
`docsearch x`	cerca x nella documentazione

Comandi generali di matlab

Informative

`whos`	mostra tutte le variabili nel workspace
`ans`	mostra l'ultimo risultato

Pulizia

`clc`	pulisci il contenuto della finestra comandi
`clear`	cancella tutte le variabili dal workspace
`clear x`	cancella solo `x` dal workspace
`close all`	chiude le figure
`close(H)`	chiude la figura H

Caricamento e salvataggio

`save filename`	salva le variabili nel file `filename`
`save filename x,y`	salva solo le variabili x,y nel file `filename`
`save -append filename x`	salva x in un file già esistente
`load filename`	carica le variabili da file

Sistema

`addpath(string)`	aggiunge una directory dove cercare gli script
`pwd`	directory corrente
`mkdir`	crea una directory
`tempdir`	crea una directory temporanea
`exit`	esci da matlab
`dir`	stampa contenuto directory corrente

Funzioni e variabili già presenti in matlab

Generiche

`sum(x)`	somma elementi del vettore x
`prod(x)`	prodotto degli elementi del vettore x
`diff(x)`	differenze fra elementi adiacenti di x
`abs(x)`	valore assoluto; abs(-3) = 3

Arrotondamento

`floor(x)`	tronca x (floor(0.7) = 0)
`ceil(x)`	tronca per eccesso x (ceil(0.1) = 1)
`round(x)`	arrotonda x
`round(x,n)`	arrotonda x alla n-esima cifra decimale

Variabili

`pi`	3.1415...
`inf`	∞
`eps`	floating point accuracy
`1e6`	10^6

Vettori e matrici

Creazione

`j:k`	vettore riga [j, j + 1, ..., k]
`j:i:k`	vettore riga [j, j + **i**, ...]
`ones(a,b)`	matrice a×b di 1
`zeros(a,b)`	matrice a×b di 0
`x[1, 2, 3]=`	vettore riga 1x3
`x[1; 2; 3]=`	vettore colonna 3x1
`x[1, 2; 3, 4]=`	matrice 2x2

Accesso e modifica

`x(2)=4`	scrivi 4 nel secondo elemento di x
`x(:)`	tutti gli elementi di x
`x(j: end )`	gli elementi di x da j fino alla fine
`x(2:5)`	dal secondo al quinto elemento di x
`x([3,2,5])`	sottovettore di x (3^o, poi 2^o poi 5^o elemento di x)
`x(j,:)`	tutti gli elementi della riga j
`x(:,j)`	tutti gli elementi della colonna j

Operatori

`x.*y`	moltiplicazione elemento per elemento
`x./y`	divisione elemento per elemento
`x+y`	somma elemento per elemento
`x-y`	sottrazione elemento per elemento
`A'`	trasposta
`size(x)`	[righe, colonne] di x

Ricerca

`x(x>5)`	gli elementi di x maggiori di 5
`x(x>5)=0`	cambia gli elementi di x maggiori di 5 in 0
`find(A>5)`	trova gli indici degli elementi di A maggiori di 5

Layout

`[A,B]`	concatena orizzontalmente A e B
`[A;B]`	concatena verticalmente A e B

Operatori logici

Semplici valori logici

`&&`	0 && 1 == 0 etc..
`||`	0 || 1 == 1 etc..
`~`	NOT

Vettori di valori logici

`&`	AND elemento per elemnto
`|`	OR elemento per elemento
`~`	NOT elemento per elemento

Operatori relazionali

`==`	Uguaglianza
`~=`	Vero se sono differenti
`>=`	Maggiore uguale
`<=`	Maggiore uguale

Stampa

`format short`	Usa 4 cifre dopo la virgola
`format long`	Usa 16 cifre dopo la virgola
`disp(x)`	Mostra la stringa x
`num2str(x)`	Converte il numero x in una stringa
`mat2str(x)`	Converte una matrice in una stringa
`int2str(x)`	Converte un intero in una stringa
`sprintf(x)`	Converte un oggetto generico in stringa

Grafici

Creazione/Manipolazione Grafici

`fig1 = plot(x,y)`	crea plot 2d e assegna handle a fig1
`fig1 = gcf()`	assegna handle figura corrente a fig1
`fig1 = figure`	crea una nuova figura vuota
`hold on`	abilita sovrascrittura immagini
`hold off`	chiude la figura corrente

Modifica stili grafici

`set(fig1, 'LineWidth', 2)`	cambia dimensione linea
`set(fig1, 'LineStyle', '-')`	cambia stile linea
`-, --, :, -.`	possibili stili di linea
`set(fig1, 'Marker', '.')`	cambia il marker per i punti
`', +, *, x, o, square`	possibili markers
`set(fig1, 'color', 'red')`	cambia colore della linea
`red, blue, green, yellow, black`	possibili colori
`set(fig1, 'MarkerSize', 10)`	cambia la dimensione dei markers
`set(fig1, 'FontSize', 14)`	cambia la dimensione del font

Assi, griglie e leggende

`xlabel('\mu line','FontSize',14)`	assegna un nome all'asse X
`ylim([a b])`	assegna dei limiti all'asse y
`title('name','fontsize',22)`	assegna un titolo al grafico
`grid on/off;`	aggiunge/toglie una griglia
`legend('y1','y2')`	aggiunge una legenda per i plot y1 e y2

Programmazione

if/elseif/else

Esegue `bodyTrue1` se `cond1!=0`, altrimenti se `cond2!=0` esegue `bodyTrue2`, altrimenti esegue `bodyFalse12`. `elseif` così come `else` è opzionale.

```
if(cond1)
   bodyTrue1
elseif cond2
   bodyTrue2
else
   bodyFalse12
end
```

for

Esegue `body` `n` volte; ad ogni iterazione la variabile `i` viene incrementata di 1 fino ad arrivare ad `n`:

```
for i=1:n
   body
end
```

while

Esegue `body` ripetutamente finche' l'espressione `cond` non vale 0:

```
while(cond)
   body
end
```

switch/case

Esegue `bodyA` se `exp` è uguale ad `a`; oppure esegue `bodyB` se `exp` è uguale ad `b`. Se nessun caso è verificato esegue `bodyDefault`.

```
switch exp
case a
   bodyA
case b
   bodyB
...
otherwise
   bodyDefault
end
```

Data import/export

`xlsread/xlswrite`	Spreadsheets (.xls,.xlsm)
`readtable/writetable`	Spreadsheets (.xls,.xlsm)
`dlmread/dlmwrite`	text files (txt,csv)
`load/save -ascii`	text files (txt,csv)
`load/save`	matlab files (.m)
`imread/imwrite`	Image files

Revision: 0.7 - November 20, 2017