

LABORATORIO DI PROGRAMMAZIONE 1
CORSO DI LAUREA IN MATEMATICA
UNIVERSITÀ DEGLI STUDI DI MILANO
2019–2020

INDICE

Parte 1. Lettura di caratteri e stringhe dal terminale	3
Esercizio 1	3
<i>Visualizzare caratteri e stringhe con printf</i>	3
Tempo: 5 min.	3
Esercizio 2	3
<i>Leggere caratteri con getchar</i>	3
Tempo: 5 min.	3
Esercizio 3	3
<i>Eco di stringhe</i>	3
Tempo: 10 min.	3
Esercizio 4	4
<i>Lunghezza delle stringhe</i>	4
Tempo: 10 min.	4
Esercizio 5	4
<i>Ultimo carattere</i>	4
Tempo: 15 min.	4
Esercizio 6	4
<i>Genere di un nome</i>	4
Tempo: 10 min.	4
Esercizio 7	5
<i>Eco iterato</i>	5
Tempo: 25 min.	5
 Parte 2. Esercizi con array di numeri	 5
Esercizio 8	5
<i>Media aritmetica</i>	5
Tempo: 10 min.	5
Esercizio 9	5
<i>Media aritmetica con lunghezza specificata dall'utente</i>	5
Tempo: 15 min.	5
Esercizio 10	6
<i>Varianza e scarto quadratico medio</i>	6
Tempo: 15 min.	6
Esercizio 11	6
<i>Ordinamento BubbleSort</i>	6

Tempo: 25 min.	7
Esercizio 12	7
<i>Ordinamento: numero di scambi</i>	7
Tempo: 15 min.	7
Esercizio 13	8
<i>Mediana</i>	8
Tempo: 25 min.	8
Parte 3. Esercizi con gli array di char	8
Esercizio 14	8
<i>Lunghezza media</i>	8
Tempo: 20 min.	8
Esercizio 15	8
<i>Lunghezza massima</i>	8
Tempo: 15 min.	8
Esercizio 16	8
<i>Frequenza di un carattere</i>	8
Tempo: 15 min.	9
Esercizio 17	9
<i>Frequenza di più caratteri</i>	9
Tempo: 20 min.	9
Esercizio 18	9
<i>Conversione in maiuscola</i>	9
Tempo: 15 min.	9
Esercizio 19	9
<i>Cercare una sottostringa</i>	9
Tempo: 30 min.	9

Parte 1. Lettura di caratteri e stringhe dal terminale

La nozione di stringa

Nell'informatica generale, una *stringa*, in inglese *string*, è una sequenza di caratteri. Molti linguaggi di programmazione includono un tipo predefinito i cui valori sono stringhe. Il linguaggio C, invece, non prevede un tipo apposito. In effetti, nel contesto del linguaggio C una stringa è semplicemente un “array (o vettore) di caratteri”. Spesso, nei linguaggi di programmazione, le stringhe sono terminate per convenzione da un carattere speciale che ne indica la fine. In C questo carattere è `\0`, il cui codice ASCII è 0. Più propriamente, quindi, le stringhe in C sono “array di caratteri terminati da `\0`.” In questa prima parte della lezione vi eserciterete con l'uso delle funzioni `getchar`, `scanf` e `fgets` della libreria standard, definite in `stdio.h`, che si possono usare per leggere da terminale caratteri e stringhe.

ESERCIZIO 1

Visualizzare caratteri e stringhe con `printf`.

Tempo: 5 min.

Scrivete un programma che dichiari una variabile `str` di tipo array di `char` e una variabile `c` di tipo `char`, e assegni loro come valori, rispettivamente, la costante stringa "Pippo" e la costante carattere 'Z'. Il programma, prima di terminare, deve visualizzare il contenuto delle variabili `str` e `c`.

La chiamata
`printf("%s%c", "fg", 'H');`
visualizza `fgH`.

ESERCIZIO 2

Leggere caratteri con `getchar`.

Tempo: 5 min.

Scrivete un programma che legga, uno alla volta, due caratteri inseriti dall'utente, li memorizzi in due variabili `c1` e `c2` di tipo `char`, e li visualizzi nuovamente nell'ordine in cui sono stati inseriti.

Nel programma che avete scritto per risolvere l'Esercizio 2, quale carattere restituisce la funzione `getchar` se l'utente preme il tasto INVIO immediatamente? È sempre bene verificare il comportamento dei programmi per valori inattesi dei dati in ingresso.

ESERCIZIO 3

Eco di stringhe.

Tempo: 10 min.

Scrivete un programma che chieda all'utente di inserire una frase, e la visualizzi di nuovo tale e quale.

Usate la funzione `fgets` per leggere la frase inserita dall'utente. Con `scanf`, invece, i risultati sono insoddisfacenti: cosa succede se l'utente include spazi nella frase? Provate!

ESERCIZIO 4

Lunghezza delle stringhe.

Tempo: 10 min.

Scrivete un programma che chieda all'utente di inserire una stringa, e visualizzi la lunghezza della stringa immessa dall'utente.

Quale costrutto della programmazione strutturata è naturale usare per calcolare la lunghezza della stringa?

Quale valore restituisce il programma che avete scritto per risolvere l'Esercizio 4, se l'utente preme il tasto INVIO immediatamente?

ESERCIZIO 5

Ultimo carattere.

Tempo: 15 min.

Scrivete un programma che chieda all'utente di inserire una stringa e un carattere. Il programma memorizza la stringa nella variabile `s` (di tipo array di `char`) e il carattere nella variabile `c` (di tipo `char`). Il programma scrive in uscita `si'` se la stringa contenuta nella variabile `s` termina con il carattere contenuto nella variabile `c`, e no altrimenti.

Quale costrutto della programmazione strutturata è naturale usare per accedere all'ultimo carattere della stringa? E per stabilire se esso coincide col carattere in `c`?

Come si comporta il programma che avete scritto per risolvere l'Esercizio 5 se l'utente preme il tasto INVIO immediatamente alla richiesta di inserire `s`, o alla richiesta di inserire `c`, o a entrambe?

ESERCIZIO 6

Genere di un nome.

Tempo: 10 min.

Questo esercizio si risolve rapidamente riutilizzando il codice che hai scritto per l'Esercizio 5.

Scrivete un programma che chieda all'utente di inserire un nome proprio di persona, e tenti di dedurre automaticamente se trattasi di un nome maschile o femminile. A tale scopo, adottate la seguente strategia: se il nome inserito termina per `a` o per `e`, assumete che il genere sia femminile; se invece termina per `o` o per `i`, assumete che sia maschile; altrimenti, accettate la sconfitta e dichiarate di non saper trarre conclusioni.

Per risolvere l'Esercizio 6, avete dovuto usare il costrutto selezione della programmazione strutturata, nella forma dell'istruzione `if` del C. Quante volte lo avete applicato, nell'intero programma? È possibile ridurre il numero di applicazioni del costrutto, o il vostro programma è già ottimale da questo punto di vista?

ESERCIZIO 7

Eco iterato.

Tempo: 25 min.

(1) Scrivete un programma che legga in ingresso una stringa inserita dall'utente, e la riscriva tale e quale sulla console. L'esecuzione deve continuare in questo modo fino a che l'utente non prema subito INVIO senza scrivere altro.

(2) Modificate poi il programma in modo da produrre in uscita, prima della terminazione, il numero complessivo di caratteri inseriti dall'utente durante l'esecuzione.

Parte 2. Esercizi con array di numeri

Qualche definizione preliminare

Si consideri una n -upla di numeri reali, $S := (r_i)_{i=1}^n \in \mathbb{R}^n$, per un intero $n \geq 1$. La *media aritmetica* di S è data da:

$$\mu_S := \frac{\sum_{i=1}^n r_i}{n}$$

La *varianza* di S è data da:

$$\sigma_S^2 := \frac{\sum_{i=1}^n (r_i - \mu_S)^2}{n}$$

Lo *scarto quadratico medio* di S è dato da:

$$\sigma_S := \sqrt{\sigma_S^2}$$

ESERCIZIO 8

Media aritmetica.

Tempo: 10 min.

Scrivete un programma che chieda all'utente di inserire 5 valori `double`, li memorizzi in un array di `double` della dimensione corrispondente, calcoli la media aritmetica dei valori inseriti, la visualizzi e termini.

La chiamata
`scanf ("%lf", &x);`
memorizza il numero
inserito dall'utente
nella variabile `x` di
tipo `double`.

ESERCIZIO 9

Media aritmetica con lunghezza specificata dall'utente.

Tempo: 15 min.

Modificate la soluzione dell'Esercizio 8 di modo che sia l'utente a decidere, all'inizio del programma, quanti valori `double` inserire in luogo dei 5 prestabiliti in quell'esercizio. L'utente deve dunque specificare la dimensione dell'array di `double` come numero intero all'inizio del programma. Il programma dichiarerà poi l'array di `double` della dimensione appropriata. Aggiungete un controllo dell'errore che costringa l'utente a reinserire la dimensione dell'array se il valore specificato non è un intero positivo.

ESERCIZIO 10

Varianza e scarto quadratico medio.

Tempo: 15 min.

Modificate la soluzione dell'Esercizio 9 di modo che il programma calcoli e visualizzi la varianza e lo scarto quadratico medio dei valori inseriti, oltre alla loro media aritmetica.

Per estrarre la radice quadrata usate la funzione `sqrt()` definita in `math.h`. (Su alcuni sistemi è necessario compilare con l'opzione `-lm` affinché si possano usare le librerie matematiche. Per esempio: `gcc -lm CiaoMondo.c.`)

Algoritmi di ordinamento: BubbleSort

Si consideri un array `v[N]` di N numeri (di tipo `int`, `float` o `double`). Un problema classico dell'informatica è quello di riordinare efficientemente i valori di `v[N]` secondo la loro magnitudine non decrescente. Se, per esempio, $N = 4$ e l'array `v` è:

$$\begin{aligned}v[0] &= 2 \\v[1] &= -1 \\v[2] &= 4 \\v[3] &= 2\end{aligned}$$

il problema consiste nello spostare le posizioni degli elementi di `v` fino ad ottenere:

$$\begin{aligned}v[0] &= -1 \\v[1] &= 2 \\v[2] &= 2 \\v[3] &= 4\end{aligned}$$

Un *algoritmo di ordinamento* (in inglese, *sorting algorithm*) è un algoritmo che risolve tale problema. Una discussione esauriente degli algoritmi di ordinamento esula dagli scopi del nostro corso, perché richiede nozioni di teoria della complessità. Nell'esercizio seguente proverete a implementare uno dei più semplici algoritmi di ordinamento, noto come **BubbleSort**. L'idea è quella di scandire ripetutamente l'array `v` dall'inizio alla fine, scambiando di posto gli elementi adiacenti che non siano già nell'ordine relativo corretto; la scansione è ripetuta fino a che non si è eseguito più alcuno scambio, il che vuol dire che l'array è adesso ordinato. La Figura 1 riporta lo psuedocodice dell'algoritmo **BubbleSort**. L'efficienza di **BubbleSort** è molto modesta rispetto ad algoritmi di ordinamento più raffinati, che noi non discuteremo; nelle applicazioni reali **BubbleSort** non andrebbe mai usato.

ESERCIZIO 11

Ordinamento BubbleSort.

Input. Un array $v[N]$ di `int` o `float` o `double`.

Output. L'array $v[N]$ riordinato (ordine non decrescente).

```
scambio = true;
while (scambio==true)
    scambio=false;
    for ( i=0; i<=N-2; i++ )
        if ( v[i]>v[i+1] )
            scambia v[i] e v[i+1]
            scambio=true;
```

FIGURA 1. Pseudocodice dell'algoritmo `BubbleSort`.

Tempo: 25 min.

Scrivete un programma che implementi l'algoritmo di ordinamento `BubbleSort` secondo lo pseudocodice riportato in Figura 1. Per testare il programma dichiarate e inizializzate nel vostro codice un array v di `int` o `double` di lunghezza modesta, per esempio 5 elementi. Visualizzate il contenuto di v prima e dopo l'esecuzione dell'algoritmo.

ESERCIZIO 12

Ordinamento: numero di scambi.

Tempo: 15 min.

Modificate la soluzione dell'Esercizio 11 di modo che il programma calcoli e visualizzi il numero di scambi eseguiti. A tal fine utilizzate una variabile contatore di tipo `int`, inizializzata a zero, che incrementerete ogni volta che il programma esegue uno scambio. Visualizzate il numero di scambi prima di terminare l'esecuzione. Calcolate poi il valore $N^2/2$, dove N è la lunghezza dell'array da ordinare, e visualizzate anche questo valore.

Nell'Esercizio 12, fissata la lunghezza N dell'array in ingresso, per quale disposizione degli elementi dell'array è massimo il numero di scambi eseguiti dall'algoritmo? E come si rapporta l'ordine di grandezza di questo valore con la quantità $N^2/2$? Sulla base delle vostre risposte, provate a fare un'ipotesi sul numero di scambi richiesti dall'algoritmo `BubbleSort` nel caso pessimo dell'esecuzione. (*Soluzione.* Non è in effetti difficile dimostrare che `BubbleSort` richiede, nel caso pessimo, un numero quadratico di scambi nella lunghezza dell'input, cioè, un numero di scambi dell'ordine di N^2 .)

La mediana

Si consideri nuovamente una n -upla di numeri reali, $S := (r_i)_{i=1}^n \in \mathbb{R}^n$, per un intero $n \geq 1$. La *mediana* di S è definita, operazionalmente, come segue.

- Si ordinino i valori r_i i modo non decrescente, ottenendo una nuova n -upla $(\hat{r}_i)_{i=1}^n$.
- Se n è dispari, la mediana di S è il valore $\hat{r}_{\frac{n+1}{2}}$.
- Se n è pari, la mediana di S è la media aritmetica di $\hat{r}_{\frac{n}{2}}$ e $\hat{r}_{\frac{n}{2}+1}$.

ESERCIZIO 13

Mediana.

Tempo: 25 min.

Modificate la soluzione dell'Esercizio 10 di modo che il programma calcoli e visualizzi la mediana dei valori inseriti, oltre alla loro media, varianza e scarto quadratico medio. Per ordinare i valori usate l'algoritmo `BubbleSort` implementato nell'Esercizio 11.

Parte 3. Esercizi con gli array di char

ESERCIZIO 14

Lunghezza media.

Tempo: 20 min.

Scrivete un programma che legga in ingresso delle stringhe inserite dall'utente, e calcoli la lunghezza media delle stringhe inserite. L'inserimento delle stringhe da parte dell'utente termina quando l'utente preme direttamente INVIO alla richiesta di immissione dati.

È necessario memorizzare tutte le stringhe inserite dall'utente per calcolarne la lunghezza media?

ESERCIZIO 15

Lunghezza massima.

Tempo: 15 min.

Scrivete un programma che legga in ingresso delle stringhe inserite dall'utente, e calcoli la lunghezza massima fra le lunghezze delle stringhe inserite. L'inserimento delle stringhe da parte dell'utente termina quando l'utente preme direttamente INVIO alla richiesta di immissione dati.

ESERCIZIO 16

Frequenza di un carattere.

Tempo: 15 min.

Scrivete un programma che legga in ingresso una stringa inserita dall'utente, legga inoltre un carattere inserito dall'utente, e calcoli la frequenza del carattere all'interno della stringa. La frequenza è definita come il rapporto (tipo `double` o `float`) fra il numero di occorrenze del carattere all'interno della stringa e la lunghezza della stringa.

ESERCIZIO 17

Frequenza di più caratteri.

Tempo: 20 min.

Modificate la vostra soluzione dell'Esercizio 16 per far sì che il programma, dopo aver chiesto all'utente l'inserimento del carattere di cui calcolare la frequenza e aver visualizzato il risultato, iteri in modo da dare la possibilità all'utente di ripetere il calcolo per un nuovo carattere. L'iterazione prosegue fino a quando l'utente digita direttamente INVIO alla richiesta di immissione del carattere, nel qual caso il programma termina.

ESERCIZIO 18

Conversione in maiuscola.

Tempo: 15 min.

Scrivete un programma che legga in ingresso una stringa inserita dall'utente, e modifichi la stringa letta convertendo tutti i suoi caratteri in maiuscola. Il programma visualizza poi la stringa convertita in maiuscola e termina.

ESERCIZIO 19

Cercare una sottostringa.

Tempo: 30 min.

Scrivete un programma che legga in ingresso una stringa inserita dall'utente, ne legga una seconda, e stampi SI se la seconda stringa compare come sottostringa della prima stringa, e NO altrimenti. Si intende qui per *sottostringa* una qualunque sottosequenza di caratteri consecutivi. Per esempio, "Pinco P" è sottostringa di "Il mio amico Pinco Pallino", mentre "inco p", "incoP" e "Il mio amico Pinco Pallino?" non lo sono.

Per convertire un carattere in maiuscola usate la funzione `toupper(c)` del file d'intestazione `ctype.h` della libreria standard, che accetta in ingresso il carattere `c` e lo restituisce convertito in maiuscola, o lo lascia invariato nel caso in cui `c` non abbia un corrispettivo in maiuscola.