# LABORATORIO DI PROGRAMMAZIONE 1
## CORSO DI LAUREA IN MATEMATICA
## UNIVERSITÀ DEGLI STUDI DI MILANO

**2018–2019**

**3.IV.2019**

VINCENZO MARRA

INDICE



Ultima revisione: 3 aprile 2019.

## Parte 1. Caratteri e stringhe con la libreria `Prog1`

**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".[a] Per facilitare i primi esercizi, la libreria didattica `Prog1` mette a disposizione un tipo definito (non primitivo) `String`, introdotto nel file di intestazione `Stringhe.h`, assieme a un piccolo insieme di funzioni di utilità per elaborare le stringhe, nei file `Stringhe.h` e `IO.h`. Il tipo definito `String` non è altro che un sinonimo di `char[n]` con $n$ una costante intera positiva molto grande. Fate riferimento alla documentazione della libreria `Prog1` per una descrizione completa delle funzioni disponibili. Gli esercizi di questa prima parte riguardano l'uso di queste funzioni.

[a] 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`."

### Esercizio 1

*Visualizzare caratteri e stringhe con* `printf`.

Tempo: 10 min.

Il tipo `String` è definito nel file di intestazione `Stringhe.h` della libreria didattica `Prog1`. Il tipo `char`, invece, è primitivo. Scrivete un programma che dichiari una variabile `str` di tipo `String` e una variabile `c` di tipo `char`, e assegni loro come valori, rispettivamente, la costante stringa `"Pippo"`[1] e la costante carattere `'Z'`. Il programma, prima di terminare, deve visualizzare il contenuto delle variabili `str` e `c`. A tal fine usate la funzione `printf` della libreria standard assieme ai caratteri di conversione `%s` e `%c` — la chiamata `printf("%s%c","Pluto",'Z')` visualizza `PlutoZ`.

### Esercizio 2

*Leggere caratteri con* `leggi_car`.

Tempo: 10 min.

Il file di intestazione `IO.h` della libreria didattica `Prog1` definisce la funzione `char leggi_car(void)` che permette di acquisire un carattere inserito dall'utente. Scrivete un programma che legga, uno alla volta, due caratteri inseriti dall'utente, il memorizzi in due variabili `c1` e `c2` di tipo `char`, e li visualizzi nuovamente nell'ordine in cui sono stati inseriti. (*Suggerimenti.* L'assegnazione `c1=leggi_car();` memorizza in `c1` un carattere letto da terminale. Volendo inserire un controllo d'errore,

[1] Per motivi che chiariremo nelle lezioni frontali, è necessario dichiarare e inizializzare le variabili di tipo `String` con un'unica istruzione. In questo esempio: `String str = "Pippo";` .

l'espressione `c1==0` risulta vera dopo questa chiamata se la lettura non è andata a buon fine per un problema di I/O del sistema. Per visualizzare il contenuto di una variabile di tipo carattere con la funzione `printf` della libreria standard, si usi il carattere di conversione `%c` — la chiamata `printf("%c",'A')` visualizza `A`.)

**Domanda**

Quale valore restituisce il programma che avete scritto per risolvere l'Esercizio 2, se l'utente preme il tasto INVIO immediatamente? Ricordate sempre di verificare il comportamento dei vostri programmi per valori estremi o non previsti dei dati in ingresso.

## Esercizio 3

*Eco di stringhe*.

Tempo: 15 min.

Scrivete un programma che chieda all'utente di inserire una frase, e la visualizzi di nuovo tale e quale. (*Suggerimenti.* Usate il tipo `String` definito nel file di intestazione `Stringhe.h` della libreria `Prog1`. Usate anche la funzione `int leggi_str(String, String)` del file di intestazione `IO.h`. Poiché il file `IO.h` include già il file `Stringhe.h`, basterà includere nel vostro programma `IO.h`. Usate infine il carattere di conversione `%s` per stampare con `printf` la stringa letta.)

## 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. (*Suggerimenti.* Usate il tipo `String` definito nel file di intestazione `Stringhe.h` della libreria `Prog1`. Usate la funzione `int lung_str(String)` del file di intestazione `String.h` per ottenere la lunghezza della stringa.)

**Domanda**

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 non primitivo `String`) e il carattere nella variabile `c` (di tipo primitivo `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. (*Suggerimenti.* Per leggere il carattere usate la funzione `char leggi_car(void)` del file di intestazione `IO.h`. Per appurare se `s` termina per `c`, usate la funzione `int finisce_con(String,char)` di `Stringhe.h`.)

**Domanda**

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.

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. (*Suggerimento.* Per analizzare il carattere finale della stringa in ingresso, usate la funzione `int finisce_con(String, char)` della libreria `Prog1`. Occorre includere `String.h`.)

**Domanda**

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.

Per calcolare il numero di caratteri contenuti in una stringa, usate inizialmente la funzione `int lung_str(String)` della libreria didattica `Prog1`. Scrivete poi una seconda versione del programma in cui implementate il calcolo della lunghezza, senza invocare `int lung_str(String)`, usando un ciclo `for`. Si ricordi che `String` è solo un sinonimo di `char[n]` con $n$ molto grande. Vi serve qui conoscere il valore di $n$? Qual è la condizione di terminazione del ciclo `for`?

## 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 $n \geqslant 1$. 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: 15 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.

### 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. (*Suggerimento.* Per estrarre la radice quadrata usate la funzione `double sqrt(double)` del file di intestazione `math.h` della libreria standard. Su alcuni sistemi, come abbiamo già visto, 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, $\texttt{N} = 4$ e l'array `v` è:

$$\texttt{v}[0] = 2$$
$$\texttt{v}[1] = -1$$
$$\texttt{v}[2] = 4$$
$$\texttt{v}[3] = 2$$

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

$$\texttt{v}[0] = -1$$
$$\texttt{v}[1] = 2$$
$$\texttt{v}[2] = 2$$
$$\texttt{v}[3] = 4$$

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 semplicemente quella di scandire l'array `v` dall'inizio alla fine, scambiando di posto gli elementi adiacenti che non siano già nell'ordine relativo corretto. 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**.**

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

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.

inizializzate nel vostro codice un array `v` di `int` o `double` di 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 $\mathtt{N}^2/2$, dove `N` è la lunghezza dell'array da ordinare, e visualizzate anche questo valore.

**Domanda**

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 $\mathtt{N}^2/2$? 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, dell'ordine cioè di $\mathtt{N}^2$.)

**La mediana**

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

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

### 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.

### 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 inserite 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. (*Suggerimento.* Per convertire un carattere in maiuscola usate la funzione `int toupper(int 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.)

### Esercizio 19

*Cercare una sottostringa.*

Tempo: 30 min.

Scrivete un programma che legga in ingresso una stringa inserite 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.

(V. Marra) Dipartimento di Matematica *Federigo Enriques*, Università degli Studi di Milano, via Cesare Saldini, 50, I-20133 Milano
*Email address*: vincenzo.marra@unimi.it