Introduzione all'Algoritmica per i Licei (C++).

1 – Algoritmi e programmazione imperativa.

versione 13 gennaio 2015

Elio Giovannetti
Dipartimento di Informatica
Università di Torino

Quest'opera è distribuita con Licenza Creative Commons
Attribuzione - Non commerciale - Condividi allo stesso modo 3.0 Italia.
http://creativecommons.org/licenses/by-nc-sa/3.0/it/legalcode

Programmazione imperativa: concetti fondamentali.

1)
- variabile;
- assegnazione;
- stringhe;
- vettore o array;
- input-output elementare;

2)
- sequenza di istruzioni;
- controllo del flusso di esecuzione (istruzioni composte):
  - istruzioni if e if-else;
  - istruzione while;
  - istruzione for

La programmazione imperativa

È un modo di concepire e scrivere i programmi che corrisponde abbastanza al modo in cui funziona lo hardware della macchina:

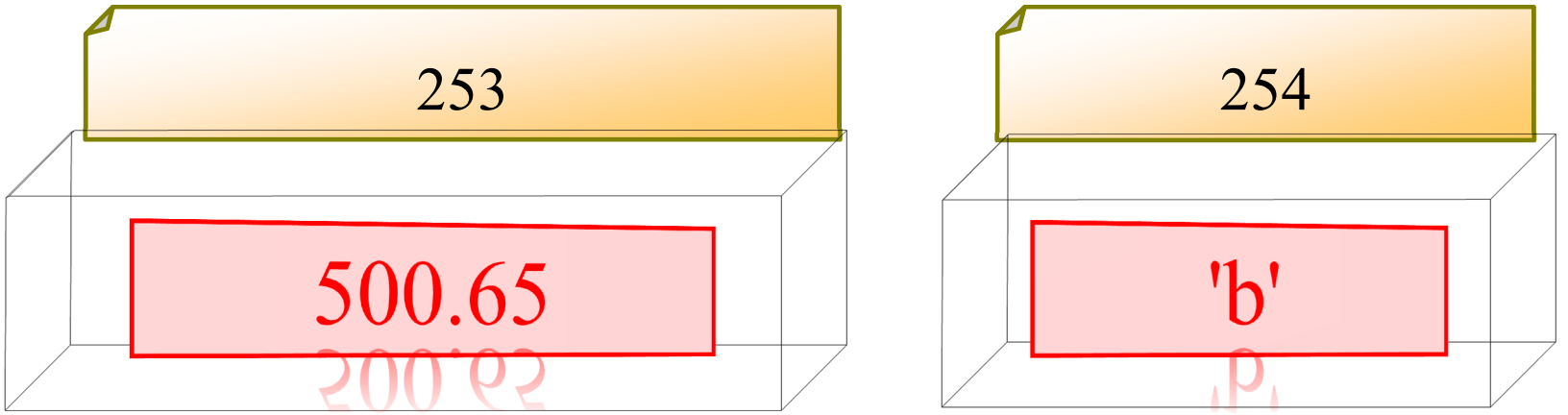
- Un programma è costituto da una sequenza di istruzioni o comandi (per questo si chiama imperativa !).
- La macchina esegue il programma eseguendo un'istruzione dopo l'altra (alcune istruzioni hanno l'effetto di far eseguire un'istruzione successiva piuttosto che un'altra, oppure di far ripetere una sequenza di istruzioni, ecc.).
- La macchina ha uno stato interno, che è costituito dai contenuti delle celle di memoria, o *variabili* (e da qual è la prossima istruzione da eseguire).
- Un' istruzione può modificare tale stato.
- Istruzioni di input/output permettono di comunicare con l'esterno, per ricevere i dati e fornire i risultati.
- Il paradigma imperativo non è l'unico !

Concetti fondamentali: variabili e assegnazione.

variabile (o cella di memoria)

può essere pensata come un contenitore o scatola:

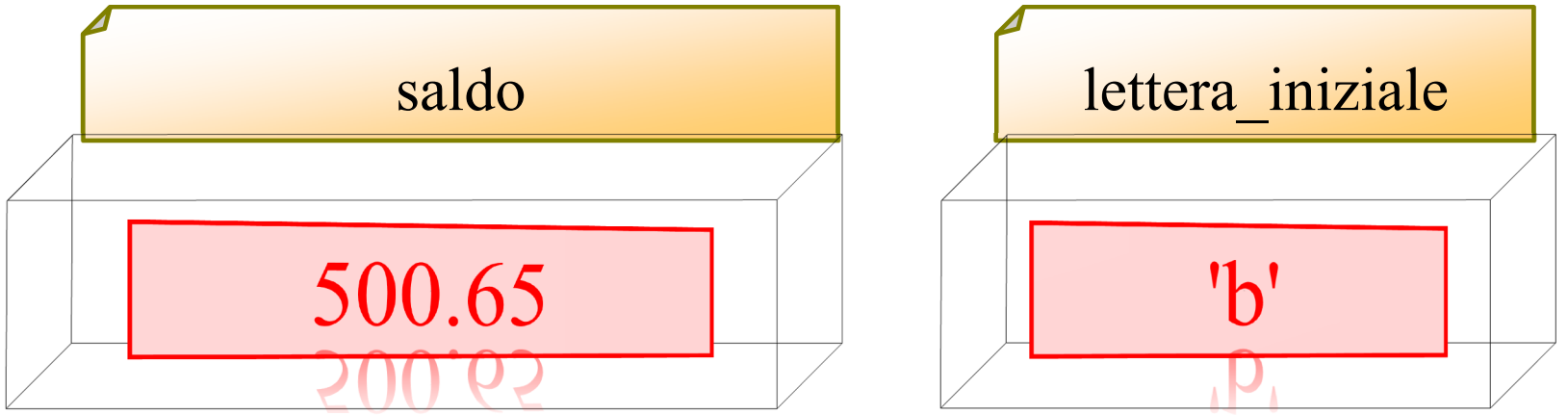
- dotata di un'etichetta esterna che è il suo indirizzo o nome,
- e contenente un valore, che è un ente astratto, come un numero, o un carattere, ecc.



Concetti fondamentali: variabili e assegnazione.

variabile (o cella di memoria)

In un programma Java, C, Python, ecc.,
il nome della variabile non è il suo indirizzo numerico, bensì un nome simbolico dichiarato dal programmatore (o, come si vedrà in seguito, un'espressione simbolica di altro genere).
La traduzione di tali nomi in indirizzi è fatta all'atto dell'esecuzione, senza intervento del programmatore.



Identificatori.

I nomi di variabile nei linguaggi di programmazione devono obbedire a certe regole generali, lievemente diverse da linguaggio a linguaggio.

I nomi (di variabile, o, come vedremo, di funzione, ecc.) in un linguaggio di programmazione si chiamano più correttamente identificatori. Le regole più importanti, comuni a tutti i linguaggi, sono che un identificatore

- non può iniziare con una cifra (ma può contenere cifre);
- non può contenere spazi bianchi, segni -, +, ecc.
- non può coincidere con una delle parole che, come vedremo, hanno un significato predefinito nel linguaggio (*if, while*, ecc.).

Esempio: 44gatti, crea-lista, crea lista non sono identificatori permessi (in realtà crea lista sono due distinti identificatori).

gatti44, creaLista, crea_lista sono invece permessi.

Maiuscole e minuscole

C++, come tutti i linguaggi moderni, distingue le minuscole dalle maiuscole. Pertanto gli identificatori:

`totalespese, Totalespese, totaleSpese, TOTalespeSE`

sono interpretati come identificatori tutti fra loro distinti.

Anche `totale_spese` e `totalespese` sono ovviamente identificatori distinti.

In alcuni linguaggi (ma non nel Dev-C++ come lo useremo noi) sono ammissibili negli identificatori anche le lettere accentate, ma non è una buona pratica scrivere programmi in cui vi siano identificatori contenenti caratteri speciali.

Regole di stile.

Le regole di stile non sono regole "obbligatorie", nel senso che un programma che le viola ma è sintatticamente corretto può essere eseguito. Tuttavia è importante seguire alcune regole:

- gli identificatori tutti maiuscoli sono considerati pessimo stile, eccetto che per le costanti;
- gli identificatori devono suggerire il significato ad essi associato: occorre trovare un compromesso fra significatività e lunghezza del nome;
- gli indici e i contatori si indicano di solito con le lettere i, j, k;
- per avere identificatori composti di più parole si può seguire una delle due convenzioni più diffuse:
  - ogni parola successiva inizia con maiuscola: `totaleSpese`
  - le parole sono separate dal carattere _ : `totale_spese`

oppure scrivere semplicemente `totalespese` !

Concetti fondamentali: variabili e assegnazione.

ASSEGNAZIONE (assignment)

è un'istruzione (cioè un comando all'esecutore) che permette di cambiare il contenuto di una cella di memoria:

Nel vecchio Pascal o nel nuovissimo Grace si scrive così:

```
saldo := saldoIniziale
```

copia il contenuto della cella *saldoIniziale* nella cella *saldo*, cancellando il precedente contenuto di *saldo*;

```
saldo := saldo - prelievo
```

calcola la differenza fra il contenuto della cella *saldo* e quello della cella *prelievo*, e rimette il risultato nella cella *saldo*.

Ma in C, C++, Java, Python, come in quasi tutti i linguaggi di adesso, l'assegnazione è indicata dal simbolo di uguaglianza, pur NON essendo l'uguaglianza !

Concetti fondamentali: variabili e assegnazione.

ASSEGNAZIONE (assignment)

è un'istruzione (cioè un comando all'esecutore) che permette di cambiare il contenuto di una cella di memoria:

In C++ si scrive così:

```
saldo = saldoIniziale
```

copia il contenuto della cella *saldoIniziale* nella cella *saldo*, cancellando il precedente contenuto di *saldo*;

```
saldo = saldo - prelievo
```

calcola la differenza fra il contenuto della cella *saldo* e quello della cella *prelievo*, e rimette il risultato nella cella *saldo*.

Ma in C, C++, Java, Python, come in quasi tutti i linguaggi di adesso, l'assegnazione è indicata dal simbolo di uguaglianza, pur NON essendo l'uguaglianza !

Concetti fondamentali: variabili e assegnazione.

NOTA BENE

il simbolo "=" NON indica l'uguaglianza matematica!

la scrittura

```
a = b;
```

NON è un'espressione affermante che a e b sono uguali;
è un comando o istruzione che copia in a il contenuto di b.
Un simbolo più conveniente per l'assegnazione sarebbe:

a ⟸ b

(l'informazione fluisce da destra verso sinistra)

Subito dopo l'esecuzione dell'istruzione i contenuti di a e di b sono uguali, ma poi possono di nuovo variare indipendentemente l'uno dall'altro.

L'assegnazione non è l'uguaglianza.

La scelta sintattica operata molti anni fa per motivi pratici nel linguaggio C, e poi impostasi quasi universalmente, può creare parecchie difficoltà nei principianti:

il simbolo "**=**", graficamente simmetrico, che in matematica denota la relazione di uguaglianza, che è una relaz. simmetrica, viene usato per indicare un'operazione che non è simmetrica!

È difficile, all'inizio, resistere al "subconscio matematico" in cui il simbolo "**=**" denota qualcosa di simmetrico!

E l'uguaglianza allora come si indica?

Con un doppio carattere "=":

==

Ad es., per stabilire se il contenuto di a è uguale a quello di b al fine di eseguire un'istruzione oppure un'altra, si scrive:

```
if(a == b) cout << "a e b sono uguali" << endl;
else cout << "a e b non sono uguali";
```

In tal caso scrivere

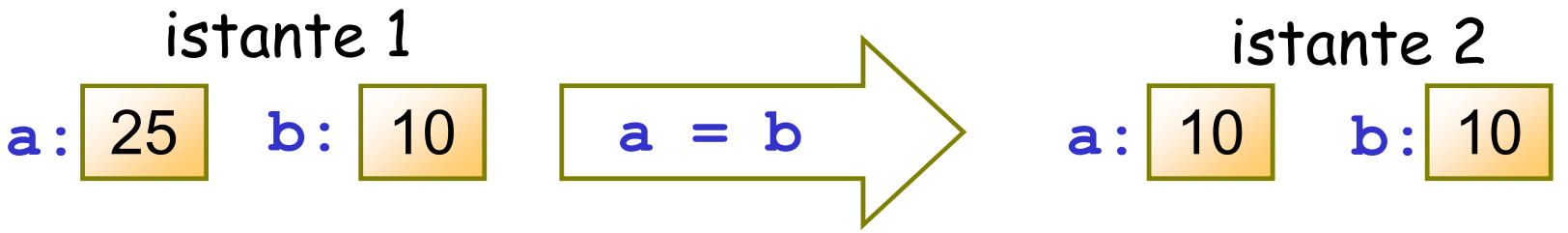
`if(a = b)` è un errore logico, benché (per ragioni che non è il caso di approfondire) non sia un errore sintattico ...

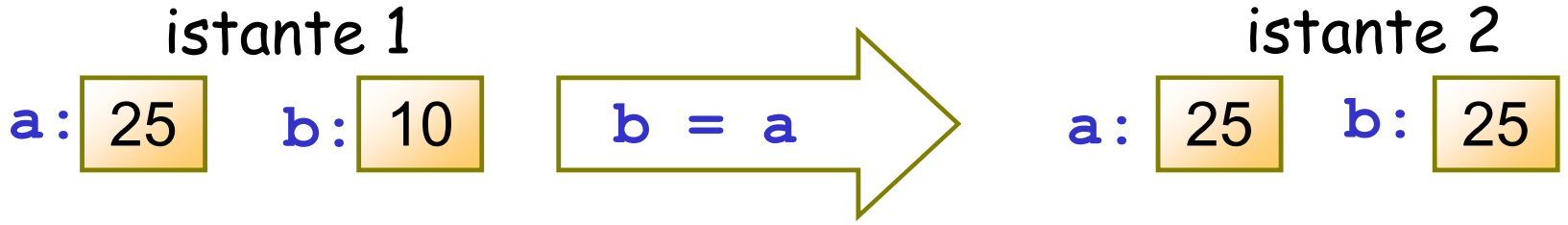
In altri linguaggi esso è invece un errore segnalato dal compilatore.

Variabili e assegnazione.

Dunque l'operazione di assegnazione, a differenza dell'uguaglianza matematica, NON è simmetrica!

`a = b;` NON è la stessa cosa che `b = a;`





Nota Bene

Nell'istante immediatamente successivo all'esecuzione della istruzione `a = b` i contenuti di `a` e di `b` sono uguali, quindi l'uguaglianza `a == b` è vera.

Ma `a` e `b` non sono diventati sinonimi! Se uno dei due viene successivamente modificato, l'altro rimane invariato.

L'assegnazione non è simmetrica (cont.)

`a = 3` è un'operazione perfettamente legale, che mette il valore 3 nel contenitore a, cancellando il valore precedente

`3 = a` è una scrittura priva di senso, perché 3 non è un contenitore! non si può mettere qualcosa nel numero 3!

`a = b+c` è un'operazione perfettamente legale, che mette in a la somma dei valori contenuti in b e in c

`b+c = a` è una scrittura priva di senso, perché l'*espressione* b+c non denota un contenitore, bensì un numero: la somma dei contenuti di b e c (si possono sommare i contenuti, NON i contenitori)

Ancora sull'asimmetria dell'operatore "="

Osserva, dagli esempi precedenti, che i nomi di variabile vengono interpretati in modo diverso a seconda che siano a sinistra o a destra del simbolo di assegnazione:

- a sinistra del simbolo "=" il nome di una variabile indica proprio il contenitore che ha quel nome;
- a destra del simbolo "=" il nome di una variabile indica il contenuto di quella variabile.

```
a = b
```

vuol dire:

copia il contenuto di b nel contenitore a
(cancellandone il contenuto precedente)

Non vuol dire che a e b diventano lo stesso contenitore!
Non vuol dire che a e b diventano sinonimi!

Assegnazione e uguaglianza

Non confondere

```
a = b;
```

che è una istruzione, la cui esecuzione cambia lo stato della macchina, cioè la sua memoria

con

```
a == b
```

che è una espressione, che non viene "eseguita" bensì valutata, e il suo valore è un valore booleano: True o False.

Scrivere `a == b` in una riga al posto di un'istruzione non ha alcun effetto: durante l'esecuzione l'uguaglianza viene valutata, ma il suo valore (True o False) non viene utilizzato.

Invece, come abbiamo già visto, scrivere `if a = b` oppure `while a = b` è un errore, perché `a = b` è un'istruzione, non un'espressione booleana.

Terminologia

In informatica una sequenza di caratteri viene comunemente detta stringa (è un inglesismo, dall'inglese string).

I francesi e gli spagnoli, più inclini alla protezione delle loro lingue, dicono rispettivamente "chaîne" e "cadena".

D'altra parte, i francesi e gli spagnoli chiamano rispettivamente "la souris" e "el ratón" quello che in italiano si chiama "il mouse"!

Stringhe e nomi

In C++, come in tutti i linguaggi di programmazione, per indicare una stringa all'interno di un programma non si può scrivere semplicemente la stringa, perché non si distinguerebbe dai nomi di variabile e da tutte le altre sequenze di caratteri che compongono il programma.

Per scrivere una stringa in un programma occorre racchiuderla fra virgolette. Esempio.

```
string nome = "Adele";
```

`cout << "nome";` visualizza sullo schermo la parola *nome*

`cout << nome;` visualizza sullo schermo la parola *Adele*

L'uso assomiglia a quello della lingua naturale, quando si vuole menzionare o parlare di un'espressione linguistica:

Adele ha 5 lettere dal fidanzato, ma "Adele" ha 5 lettere;
la frase "l'informatica è roba da nerds" è un luogo comune; ecc.

Stringhe : il + come operatore di concatenazione.

```
string a = "Buon";
string b = "giorno";
cout << a + b;
```

scrive sullo schermo **Buongiorno**

```
string s = "25";
string t = "10";
cout << s + t;
```

scrive sullo schermo **2510**

Invece, ovviamente:

```
int m = 25;
int n = 10;
cout << m + n;
```

scrive sullo schermo **35**

Funzioni: invocare funzioni predefinite.

In C++ (come in quasi tutti i linguaggi di programmazione), una funzione è una sequenza di istruzioni, generalmente dotata di un nome, che serve a eseguire un compito specifico.

In C++, come in ogni linguaggio, vi sono molte funzioni predefinite.

Per usare (o invocare) una funzione bisogna scrivere:

- il nome della funzione, ad es. pow (potenza), cos (coseno), oppure sqrt (radice quadrata), ecc.;
- i dati che servono alla funzione per eseguire il suo compito, detti argomenti, racchiusi dentro un'unica coppia di parentesi tonde, e separati l'uno dall'altro mediante virgole; il numero di argomenti dipende dalla funzione.

Valutare gli argomenti

Come in matematica, gli argomenti di una funzione non devono essere necessariamente dei valori: possono essere espressioni, che l'esecutore valuta prima di passarli alla funzione.

Esempio.

```
cout << sqrt(30 + 19);
```

scrive sullo schermo `7`

Funzioni che danno un risultato.

La funzione sqrt (abbreviazione di square root, radice quadrata) può essere usata come in matematica:

```
double r2 = sqrt(2);
```

Il significato è, in realtà, lievemente diverso:

- in un testo matematico una scrittura come la precedente afferma che, da quel punto in poi, il nome r2 è un sinonimo del numero reale che è la radice quadrata di 2;
- in un programma una tale scrittura è un'istruzione eseguendo la quale la funzione sqrt dà come risultato il valore 1.4142… e tale valore viene, per mezzo dell'operatore di assegnazione, depositato nella cella di nome r2.

Restituire … ciò che non si è preso in prestito.

Nella terminologia informatica inglese "dare un risultato" si dice "to return a result", cioè, letteralmente, "restituire un risultato" o anche, con un abbastanza diffuso inglesismo, "ritornare un risultato".

Diremo quindi che la funzione sqrt, invocata con argomento 121, restituisce il numero 11.0, ecc.

Metaforicamente, invocare una procedura è come chiamare un aiutante perché esegua un certo compito; l'aiutante se ne va, esegue il compito (mentre chi l'ha chiamato rimane fermo in attesa, o si addormenta) e poi ritorna al chiamante portandogli il risultato.

A quel punto il chiamante si sveglia e riprende il proprio lavoro.

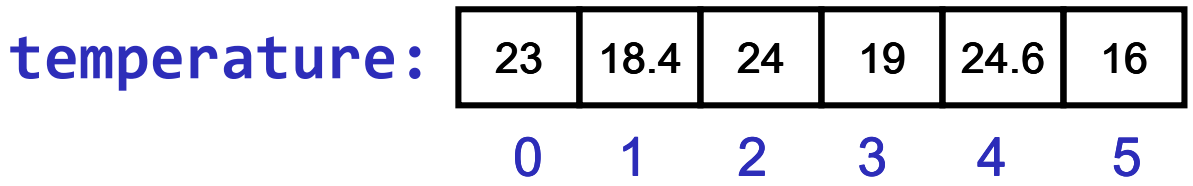
Array

Un array unidimensionale è una sequenza di celle di memoria contigue ognuna delle quali è accessibile tramite un indice in un tempo costante indipendente dalla posizione.

Gli elementi si scrivono fra parentesi graffe separate da virgole.

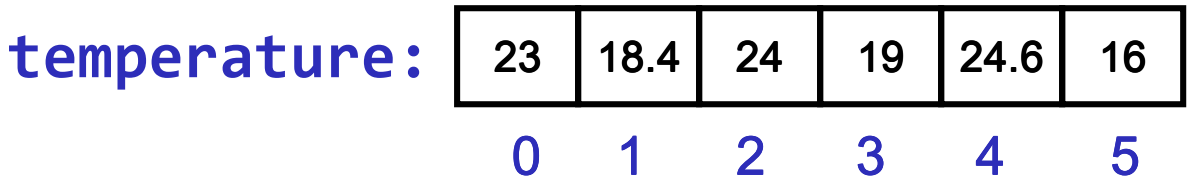
```
double temperature[] = {23, 18.4, 24, 19, 24.6};
```

In C++, come nella maggior parte dei linguaggi moderni, l'indicizzazione comincia da 0 e non da 1.



Un array può metaforicamente vedersi come un indirizzo postale che possiede dei "numeri di interno": nell'esempio l'indirizzo generale è `temperature`, i numeri interi da 0 a 5 sono i sei "numeri di interno" di `temperature`.

Array



Quindi `cout << temperature[1];` scrive sullo schermo 18.4

I contenuti delle celle che compongono un array possono venire modificati, sono cioè delle variabili. Esempio:

```
temperature[2] = 17
cout << temperature[2];
```

scrive sullo schermo 17.

Tipico ciclo su un array:

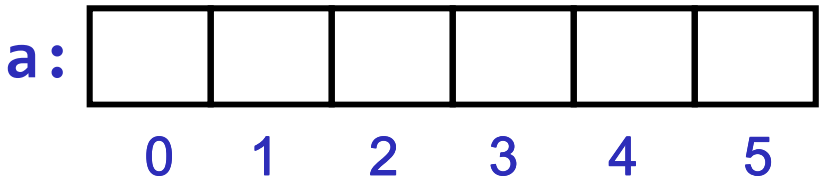
```
int n = 6;
for(int i = 0; i < n; i++)
  cout << temperature[i] << " ";
cout << endl;
```

Array "vuoti".

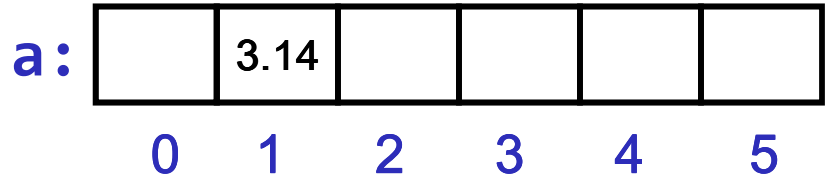
Così come attraverso una dichiarazione di variabile si può creare una cella "vuota" (ad es. `double a;`), analogamente si può creare un array "vuoto" di una data lunghezza, nel modo seguente:

```
double a[6];
```

crea un array di 6 celle (contigue), indiciate da 0 a 5, che possono essere successivamente riempite e modificate:




```
a[1] = 3.14;
```



Input di array da tastiera, output su terminale.

```
int n;
cout << "quanti numeri vuoi immettere? ";
cin >> n;
double numeri[n];
for(int i = 0; i < n; i++) {
  cout << "immetti elemento n." << i << ": ";
  cin >> numeri[i];
}

for(int i = 0; i < n; i++) {
  cout << numeri[i] << " ";
}
cout << endl;
```

Esercizi

1. Scrivi due funzioni

```
void fillFromInput(double a[], int n)
void fillFromInput(string a[], int n)
```

   che riempiano l'array `a` con `n` numeri o rispettivamente stringhe immessi da tastiera.

2. Scrivi due funzioni

```
void print(double a[], int n)
void print(string a[], int n)
```

   che scrivano sullo schermo in sequenza gli elementi dell'array.

3. Riscrivi un programma analogo a quello della slide precedente usando le quattro funzioni definite qui sopra.

Esercizi

4. Scrivi una funzione la quale, dato un array di numeri e la sua lunghezza, dia come risultato la somma:

```
double somma(double a[], int n)
```

5. Scrivi una funzione che, dati due array di stringhe nomi e cognomi di uguale lunghezza n, riempie un terzo array risult di lunghezza n cui ciascun elemento è la stringa composta da nome e cognome corrispondente:

```
void associa(string nomi[], string cognomi[],
             string risult[], int n)
```

Esempio:

```
string nomi[] = {"Ada", "Alan", "John", "Kurt"};
string cognomi[] = {"Byron", "Turing", "von Neumann", "Gödel"}
```

l'array risultato conterrà gli elementI;

```
"Ada Byron", "Alan Turing", "John von Neumann", "Kurt Gödel"
```

Vector

Un vector è una sorta di array impacchettato con i fiocchi. Di fatto, è un array parzialmente riempito cui si possono continuare ad aggiungere elementi al fondo, tramite il metodo push_back; quando l'array si riempie, esso viene automaticamente copiato in un array più lungo.

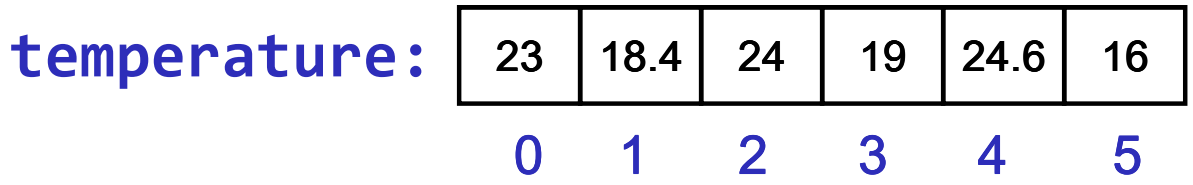
Per il programmatore, quindi, un vector, a differenza di un array, non ha una lunghezza immutabile.

Inoltre il numero di elementi logicamente presenti nel vector è un attributo del vector stesso.

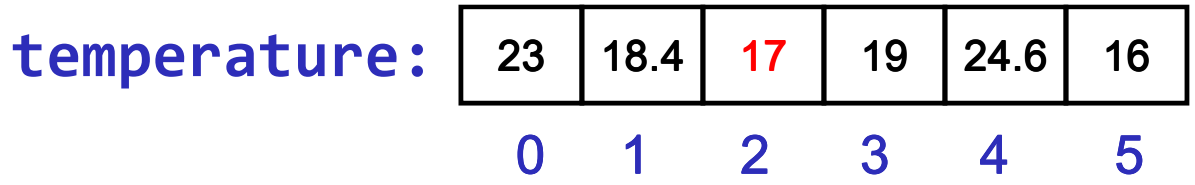
Quindi a una funzione che opera su vector non è necessario passare esplicitamente la dimensione del vettore, stando attenti che sia quella giusta: la funzione chiamata può infatti ricavarla dal vettore stesso, accedendo all'attributo size.

Vettori.

I contenuti delle celle che compongono un vettore possono venire modificati, esattamente come per gli array. Esempio:




```
temperature[2] = 17
```



Esempio

```
int n;
cout << "quante parole vuoi immettere? ";
cin >> n;
vector<string> parole(n);
for(int i = 0; i < n; i++) {
  cout << "immetti elemento n." << i << ": ";
  cin >> parole[i];
}
parole.push_back("pippo");
parole.push_back("pluto");

n = parole.size();
for(int i = 0; i < n; i++)
  cout << parole[i] << " ";
```

Compito alternativo.

Fare gli esercizi 1, 2, 3, 4, 5 (slides 30 e 31) usando i vettori invece degli array.

Templates: introduzione.

Invece di definire tante funzioni sugli array, tutte simili, una per ciascun tipo di elemento …:

```
void print(double a[], int n) {
  for(int i=0; i < n; i++)
    cout << a[i] << endl;
  cout << endl;
}

void print(string a[], int n) {
  for(int i=0; i < n; i++)
    cout << a[i] << endl;
  cout << endl;
}
void print(int a[], int n) {
  ...
```

Esempio di template di funzione che agisce su array

... possiamo definire un *template* (schema, modello, maschera) di funzione per un tipo generico:

```
template<typename E> void print(E a[], int n) {
  for(int i=0; i < n; i++) {
    cout << a[i] << endl;
  }
  cout << endl;
}
```

Quest'unica print permette di scrivere sullo schermo array di elementi di qualsiasi tipo:

```
string as[10]; double ad[15]; ...
print(as, 10); print(ad, 15); ...
```

Esempio di template di funzione che agisce su vector

I vettori, a differenza degli array, sono passati per valore e non per riferimento, cioè sono ricopiati all'interno della funzione.

Se si vuole un passaggio per riferimento, occorre indicarlo esplicitamente con **&**:

```
template<typename E> void print(vector<E>& v) {
  int n = v.size();
  for(int i=0; i < n; i++) {
    cout << v[i] << endl;
  }
  cout << endl;
}
```

Quest'unica print permette di scrivere sullo schermo vettori di elementi di qualsiasi tipo.

Esercizio

Definisci un template di funzione

```
template<typename E>
void fillFromInput(E a[], int n) {
```

che riempia l'array a, per la lunghezza n, di elementi immessi da tastiera.

Esercizio alternativo

Definisci un template di funzione

```
template<typename E>
void fillFromInput(vector<E> &v)
```

che riempia il vettore v, per tutta la sua lunghezza, di elementi immessi da tastiera.