Corso di Laurea in Ingegneria Civile

Corso di Elementi di Informatica

A.A. 2016-17

Docente: Alessandro Amirante

alessandro.amirante@unina.it

Università degli Studi di Napoli Federico II

Facoltà di Ingegneria

Agenda

- Le funzioni
- Il passaggio dei parametri per valore e per riferimento
- Parametri di ingresso, di uscita e di ingresso-uscita
- Il passaggio di parametri const
- Il passaggio di parametri vettore

Funzioni: cosa sono

- Le funzioni rappresentano dei contenitori per dei frammenti di codice
- Esse possono essere invocate ogni qual volta si desideri eseguire uno specifico frammento
- I frammenti di codice che costituiscono una funzione vengono spesso definiti “sotto-programmi” (sub-routines).
- Una chiamata a funzione provoca:
  - l’esecuzione del codice contenuto nella funzione;
  - una volta che la funzione sia terminata, l’esecuzione della istruzione immediatamente successiva alla chiamata.

Esempio di programma che usa funzioni

```
//Programma che, utilizzando funzioni, realizza
//l'ordinamento di un vettore inserito dall'utente

int main() {
  TVettore V;
  int nelem;

  CaricaVettore(…); //Chiamata a funz.

  cout << "Il vettore non ordinato e': " << endl;
  StampaVettore(…); //Chiamata a funz.

  OrdinaVettore(…); //Chiamata a funz.

  cout << "Il vettore ordinato e': " << endl;
  StampaVettore(…); //Chiamata a funz.
}
```

Vantaggi nell'uso delle funzioni

- maggiore sinteticità
  - una funzione può essere scritta una sola volta ed usata tante volte (p.es. la funzione StampaVettore(…)). Inoltre programmi più brevi sono più facilmente manutenibili.
- maggiore leggibilità
  - da un veloce sguardo del listato appare subito chiaro "cosa fa" il programma
- possibilità di scomporre ogni problema in sottoproblemi più semplici
  - si può realizzare il programma in più fasi successive non dovendo preoccuparsi di "tutto e subito"
  - si può delegare la stesura di parti di programma ad altri individui
  - ciò è diretta conseguenza del maggiore grado di modularità raggiunto
- pur sapendo "cosa fa" una funzione (p.es. OrdinaVettore(…)), non è detto che si sappia "come lo fa"
  - questo permette di concentrarsi più sulla logica del problema che sull'implementazione della sua soluzione

Esempio: la funzione somma

senza le funzioni

```
int main() {
  int a, b;
  int s;

  cout << "Inserisci a: ";
  cin >> a;
  cout << "Inserisci b: ";
  cin >> b;

  s = a + b;

  cout << "La somma vale: " << s;
  cout << endl;
}
```

con le funzioni

```
int Somma(int x, int y); //prototipo

int main() {
  int a, b;
  int s;

  cout << "Inserisci a: ";
  cin >> a;
  cout << "Inserisci b: ";
  cin >> b;

  s = Somma(a, b);

  cout << "La somma vale: " << s;
  cout << endl;
}

int Somma(int x, int y) {
  return x + y;
}
```

Il prototipo di una funzione

- Il prototipo di una funzione serve a definire il nome della funzione ed il tipo di parametri che tratta, siano essi di ingresso o di uscita

```
<tipo ris.> <NomeFunz>(<tipo> <nome>, <tipo> <nome>, <tipo> <nome>, …)
```

- Esempi:

```
bool DataValida(int giorno, int mese, int anno);
bool NumeroPari(int n);
float Dividi(int a, int b);
```

Il valore restituito da una funzione

- Dal prototipo di una funzione è possibile dedurre il tipo del valore che assumerà la funzione in seguito alla sua chiamata
- Il valore assunto dalla funzione può essere utilizzato, ad esempio, all'interno di espressioni più complesse

```
d = Somma(a,b) * Dividi(a,c); // d=(a+b)*a/c
```

- L'argomento stesso di una funzione potrebbe essere il risultato restituito da un'altra funzione

```
e = Somma( a, Somma(b, c) ); // e=a+b+c
```

Le funzioni void

- Una funzione più anche non restituire alcun valore
- In questo caso il suo prototipo sarà del tipo:

```
void Func(…)
```

- Le funzioni che non restituiscono alcunché vengono anche dette *procedure*
- La chiamata ad una procedura è del tipo:

```
Func(…)
```

- e non ha più senso qualcosa del tipo:

```
a = Func(…)
```

- Esempi di procedure sono **`StampaVettore`** e **`CaricaVettore`**

La keyword return

- Dall'interno di una funzione, per restituire il giusto valore al chiamante, si usa la keyword **`return`**
- Essa provoca anche l'immediata terminazione della funzione
- Esempio:

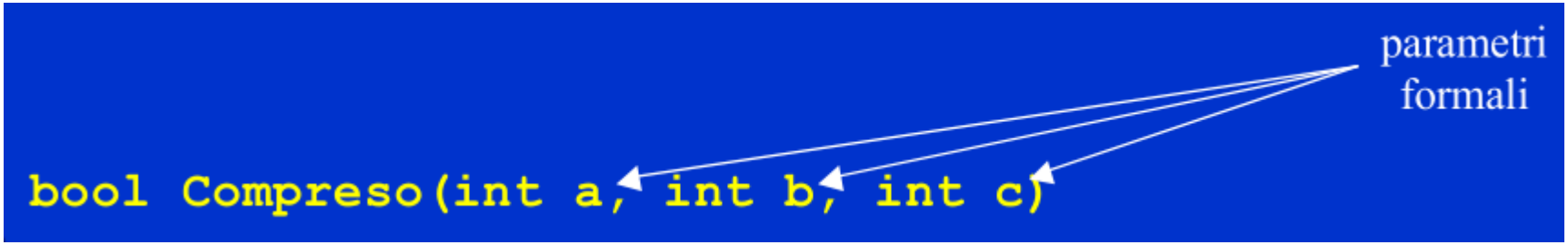
```
//funzione che indica se c è compreso nell'intervallo
//a..b estremi inclusi (implementazione poco felice)
bool Compreso(int a, int b, int c) {
  if (c < a) return false;
  if (c > b) return false;
  return true;
}
```

- La stessa funzione poteva essere più felicemente realizzata nel modo seguente:

```
bool Compreso(int a, int b, int c) {
  return ( (c >= a) && (c <= b) );
}
```

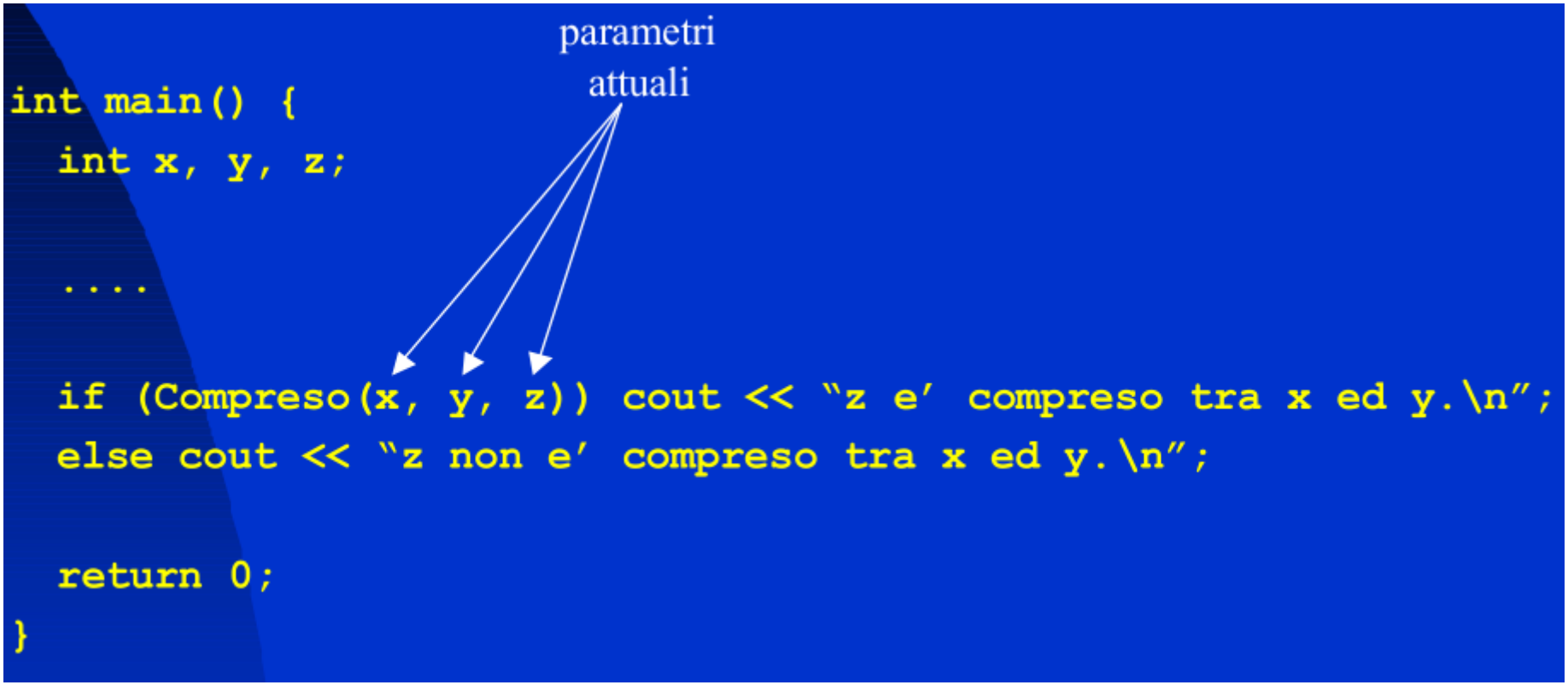
Parametri formali e parametri attuali

- I parametri **formali** sono quelli che la funzione utilizza al suo interno
- È come se fosse stata essa stessa ad averli dichiarati



- Al momento della definizione della funzione non è necessario conoscere quali saranno i veri parametri su cui essa opererà

Parametri formali e parametri attuali



- All'atto della chiamata alla funzione, x, y, e z sono detti parametri **attuali**, e non sono tenuti ad avere lo stesso nome dei parametri formali

Parametri formali e parametri attuali

- All’atto della chiamata:
  - Il compilatore controlla se il tipo dei parametri attuali corrisponde con quello dei parametri formali, segnalando eventualmente un errore;
  - Sostituisce ciascun parametro formale con il corrispondente parametro attuale;
  - Esegue la funzione.
- La corrispondenza non avviene per nome, ma per posizione
- In altre parole, al primo parametro formale viene sostituito il primo parametro attuale, al secondo il secondo, e così via…

La funzione main()

- Anche il **`main()`** è una funzione al pari di tutte le altre
- La sua unica caratteristica peculiare è che, per convenzione, è la prima funzione invocata in un programma
- È il sistema operativo ad invocare la funzione **`main()`** ed a recuperare, al suo termine, il valore restituito attraverso il **`return`**
- Si noti anche che, così come è possibile dichiarare delle variabili locali al **`main()`** utili al suo funzionamento, è ugualmente possibile dichiarare delle variabili all'interno di tutte le altre funzioni

Esercizi

- Scrivere il prototipo di una funzione che accetti in ingresso un numero intero e restituisca un valore che indichi se il numero è un numero primo.
  Si implementi poi la funzione.
  Si realizzi infine un programma chiamante che verifichi il corretto comportamento della funzione.
  Per la verifica della proprietà si utilizzi la definizione di numero primo: "un numero intero positivo è numero primo se è divisibile solo per 1 e per sé stesso".

- Utilizzando la funzione per la rivelazione dei numeri primi, si realizzi un programma che, dato un numero intero dall'utente, stampi tutti i numeri primi minori o uguali ad esso.

- Scrivere il prototipo di una funzione che accetti in ingresso tre interi e, interpretandoli come giorno, mese ed anno, restituisca un valore che indichi se la data specificata è una data valida. Si implementi poi la funzione. Si realizzi infine un programma chiamante (main) che verifichi il corretto comportamento della funzione.

- Realizzare, attraverso l'uso di un'apposita funzione, un programma che calcoli la distanza tra due punti di cui l'utente inserisce le coordinate in un sistema di riferimento cartesiano bidimensionale

Il passaggio dei parametri per valore

- Provare ad eseguire il seguente programma:

```
void Azzera(int numero);

int main() {
  int a;
  cout << "Inserisci a: ";
  cin >> a;
  cout << "a = " << a << endl;

  cout << "Ora azzero a.";
  Azzera(a);

  cout << "a = " << a << endl;
}

void Azzera(int numero) {
  numero = 0;
}
```

Il passaggio dei parametri per valore

- Il comportamento apparentemente strano dell'esempio precedente si spiega attraverso la modalità di passaggio dei parametri impiegata in questo caso particolare
- Il passaggio dei parametri in C++, in assenza di esplicite direttive, avviene secondo la modalità detta *per valore*
- Questo significa che, all'atto della chiamata di una funzione, il compilatore realizza una copia dei parametri attuali e la associa ai parametri formali
- La funzione lavora dunque su delle copie dei parametri attuali, localizzate in aree di memoria completamente diversi, e non sui parametri attuali veri e propri
- Le copie vengono distrutte al termine della funzione: del loro valore, eventualmente alterato all'interno della funzione, non resta traccia

Il passaggio dei parametri per valore

- Vantaggi:
  - Il chiamante di una funzione può essere certo che i parametri ad essa passati non potranno essere restituiti alterati in seguito alla chiamata
  - La funzione, se lo crede opportuno, può modificare i parametri a suo piacimento con la certezza che le modifiche non saranno visibili all'esterno di essa
- Svantaggi:
  - L'occupazione di memoria risulta doppia rispetto al necessario
  - Il tempo per effettuale la copia dei parametri, specialmente nel caso in cui questi siano appartenenti a tipi strutturati di grosse dimensioni, può degradare le prestazioni di un programma

Il passaggio dei parametri per valore

- Al di là dei vantaggi e svantaggi elencati, ci sono dei casi in cui il comportamento descritto è del tutto indesiderato
  - Ad es. la funzione Azzera() vista in precedenza
- Se si desidera che il parametro attuale venga modificato in seguito alla chiamata alla funzione, il meccanismo di protezione dovuto al passaggio dei parametri per valore impedisce che questo possa avvenire
- In altri termini, con il passaggio dei parametri per valore è possibile realizzare solo parametri di ingresso e non di uscita o di ingresso-uscita

Il passaggio dei parametri per riferimento

- Per risolvere il problema è necessario esplicitamente richiedere al compilatore che all'interno della funzione di possa lavorare non su delle copie, ma sui parametri veri e propri
- Questo è possibile attraverso il passaggio dei parametri *per riferimento*
- In questo caso alla funzione viene passata non una copia del parametro attuale, ma il riferimento ad esso, cioè l'indirizzo di memoria
- Per richiedere questo tipo di passaggio bisogna aggiungere il carattere **&** tra il tipo ed il nome del parametro in questione

Il passaggio dei parametri per riferimento

- La versione corretta della funzione Azzera() risulta dunque:

```
void Azzera(int& numero) {
  numero = 0;
}
```

- In questo caso il compilatore permette alla funzione di lavorare direttamente sull'area di memoria in cui è contenuto il parametro attuale che corrisponderà al parametro formale **`numero`**, senza quindi che di esso ne venga fatta una copia

Il passaggio dei parametri per riferimento

- Vantaggi:
  - Il passaggio è più efficiente dal momento che, a prescindere dalle dimensioni del dato, quello che deve essere passato è sempre e solo un indirizzo di memoria
- Svantaggi:
  - Si perde il meccanismo di protezione garantito dal passaggio dei parametri per valore
    - A meno che non si utilizzi un artificio particolare…

Valore o riferimento?

- Abbiamo visto che entrambi i tipi di passaggi sono indispensabili per un corretto comportamento delle funzioni in tutti i casi che si possono verificare
- I parametri di una funzione convogliano informazioni tra il chiamante e la funzione chiamata, in entrambe le direzioni
- È possibile individuare il tipo di passaggio che deve essere di volta in volta utilizzato attraverso l'analisi della "direzione" che le informazioni hanno rispetto alla funzione chiamata
- In altre parole bisogna capire se le informazioni trasportate dalle variabili di passaggio sono di ingresso alle funzioni chiamate, di ingresso-uscita o di uscita

Parametri di input, output, input-output

- Definiamo i parametri di una funzione:
  - parametri di ingresso (input), se ai fini della corretta esecuzione della funzione è sufficiente, per la funzione stessa, esclusivamente leggere il loro valore; p.es.: NumeroPari(n);
  - parametri di uscita (output), se essi rappresentano esclusivamente un supporto per convogliare informazioni verso l'esterno della funzione; p.es.: CheOreSono(t);
  - parametri di ingresso-uscita (input-output), se il loro valore all'ingresso della funzione è significativo ai fini della elaborazione che essa realizza ma vengono anche alterati per convogliare informazioni verso il chiamante; p.es.: Raddoppia(n).

Parametri di input, output, input-output

- Le precedenti osservazioni ci indicano dunque che:
  - i parametri di ingresso sono preferibilmente scambiati per valore.
  - i parametri di uscita non possono essere scambiati per valore, altrimenti le modifiche apportate ad essi non sarebbero visibili all'esterno della funzione. Devono dunque essere scambiati per riferimento.
  - i parametri di ingresso-uscita, analogamente a quelli di uscita, non possono essere scambiati per valore, ma devono essere scambiati per riferimento.
- Il C++, pertanto, nell'ottica della modalità di passaggio dei parametri, non fa grande differenza tra parametri di uscita e di ingresso-uscita.
- Si tenga presente che una generica funzione può avere più parametri di ingresso, uscita e ingresso-uscita contemporaneamente.

Il passaggio di parametri const

- Se si vuole passare un dato di grosse dimensioni conservando l'efficienza e proteggendolo comunque da modifiche indesiderate, è possibile utilizzare la clausola **`const`**
- Durante lo scambio dei parametri, se si antepone al parametro la keyword **`const`**, si impedisce del tutto alla funzione di modificare il parametro all'interno di essa

```
int NumeroPari(const int& n);
int QuantiHanniHa(const Tpersona& p);
```

- Nel caso di passaggio per riferimento, la clausola **`const`** risolve il problema di modifiche indesiderate ai parametri, consentendo contemporaneamente di sfruttare l'efficienza intrinseca di questa modalità

Il passaggio dei vettori

- I vettori passati come parametri alle funzioni assumono un comportamento apparentemente strano
- Si osservi l'output apparentemente inspiegabile del seguente programma:

```
typedef int TVettore[5];

void Funzione(TVettore vet) {
  vet[0] = 3;
}

int main() {
  TVettore v = {0, 0, 0, 0, 0};
  cout << "v[0] = " << v[0] << endl;
  Funzione(v);
  cout << "v[0] = " << v[0] << endl;
}
```

- Questo strano comportamento discende dalla particolare proprietà dei vettori secondo la quale il nome di un vettore rappresenta un puntatore alla locazione in cui si trova il primo elemento dell'array.
- Il parametro che quindi viene scambiato per valore, e pertanto viene copiato all'atto del passaggio, non è il vettore ma il suo puntatore.
- Ecco perché le modifiche interne alla funzione si propagano anche all'esterno: in entrambi i casi si lavora sempre sulla stessa area di memoria.
- Ancora una volta, volendo evitare questo inconveniente è possibile passare il vettore con la clausola **`const`**.
- Ciò avviene tipicamente nel caso di funzioni che accettino vettori di ingresso. P.es.:

```
void StampaVettore(const TVettore v, int nelem);
```

- Si noti che l'unico modo per conoscere all'interno di una funzione il numero di elementi significativi di un vettore passato è quello di indicarlo esplicitamente come parametro di passaggio aggiuntivo

Riepilogo

Nel prototipo	Modalità di passaggio	Quando si usa	Note
`int n`	Passaggio per valore	Parametro di ingresso	Le modifiche non sono propagate all'esterno
`const int n`	Passaggio const per valore	Parametro di ingresso	La clausola const è pleonastica e serve solo ad enfatizzare il ruolo di parametro di ingresso
`int& n`	Passaggio per riferimento	Parametro di uscita o di ingresso/uscita	Le modifiche si propagano all'esterno
`const TPersona& p`	Passaggio const per riferimento	Parametro di ingresso di grosse dimensioni	Non può essere modificato
`TVettore v`	Passaggio per valore di un vettore.	Vettore di uscita o di ingresso-uscita	Viene copiato puntatore al vettore e non il vettore. Le modifiche si propagano
`TVettore& v`	Passaggio per riferimento di un vettore.	Vettore di uscita o di ingresso-uscita	È equivalente al caso precedente
`const TVettore v`	Passaggio const per valore di un vettore	Vettore di ingresso	Il vettore non può essere modificato

Esercizi

- Scrivere una funzione che calcoli il massimo e il minimo di un vettore
- Scrivere una funzione che realizzi il concatenamento di due vettori
- Scrivere una funzione che realizzi il prodotto scalare di due vettori

Domande?