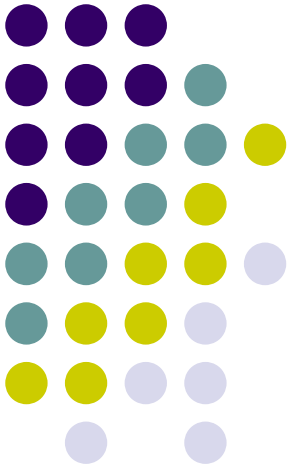
Università di Napoli Federico II - Dip. di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Corso di Laurea in Ingegneria Informatica

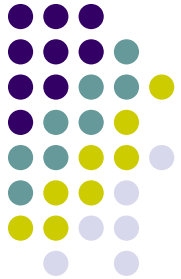
DIE TI

Corso di Programmazione

Sottoprogrammi (Richiami)



Divide et Impera



Problema:
Sviluppo di una applicazione software complessa (anche solo *mediamente* complessa...)

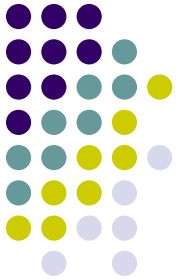
- Un problema complesso può essere diviso in sottoproblemi più semplici risolvibili separatamente
- La soluzione del problema iniziale è ottenuta combinando opportunamente i risultati “parziali”

Metodologie di sviluppo

- Top down: si parte dall'alto, considerando il problema nella sua interezza e si procede verso il basso **per raffinamenti successivi** fino a ridurlo ad un insieme di sottoproblemi elementari

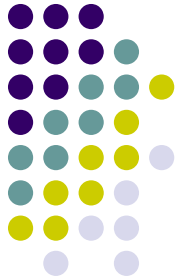
- Bottom up: si risolvono singole parti del problema, senza averne necessariamente una visione d'insieme, per poi risalire procedendo per aggiustamenti successivi fino ad ottenere la soluzione globale.

Sottoprogrammi



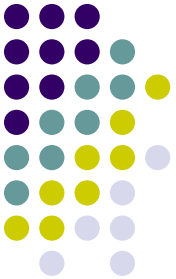
- Costituiscono un meccanismo di astrazione procedurale (astrazione sul controllo)
- Sono sezioni di codice che non vanno in esecuzione autonomamente, ma su “chiamata” di un programma o sottoprogramma esterno
- Possono svolgere funzioni di utilità generale (es: funzioni di libreria) o semplicemente servire a sviluppare codice ben strutturato e modulare (es: funzioni che svolgono task di un particolare programma)

Vantaggi dell'uso di sottoprogrammi



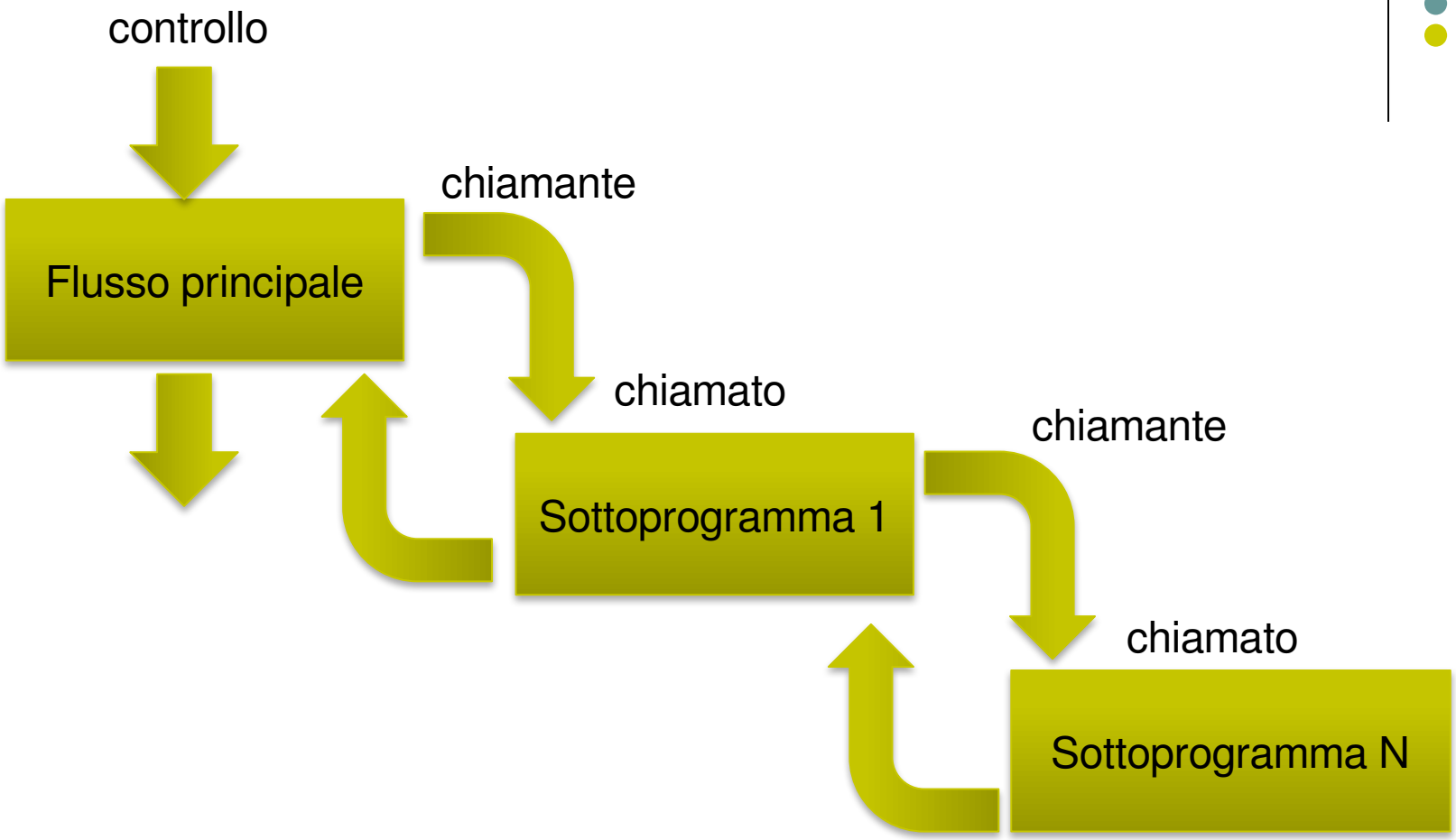
- Testo del programma *suddiviso in unità significative*
- Testo di ogni unità *più breve*
  - *minore probabilità di errori*
  - *migliore verificabilità*
- Riutilizzo di codice
- Migliore leggibilità
- Sviluppo top-down del software

Sottoprogrammi (1/2)



- Ai **sottoprogrammi** si assegnano determinate **responsabilità**, che includono la risoluzione di un sotto-problema:
  - I sottoprogrammi hanno un **nome** e possono essere attivati durante l'esecuzione del programma zero, uno o più volte.
  - I sottoprogrammi hanno un solo punto in ingresso e di uscita (secondo i principi della programmazione strutturata).
  - Il flusso di controllo del programma in esecuzione è ceduto con una **istruzione (semplice) di chiamata** o **invocazione** dal modulo chiamante al sottoprogramma chiamato

Distinzione chiamante-chiamato

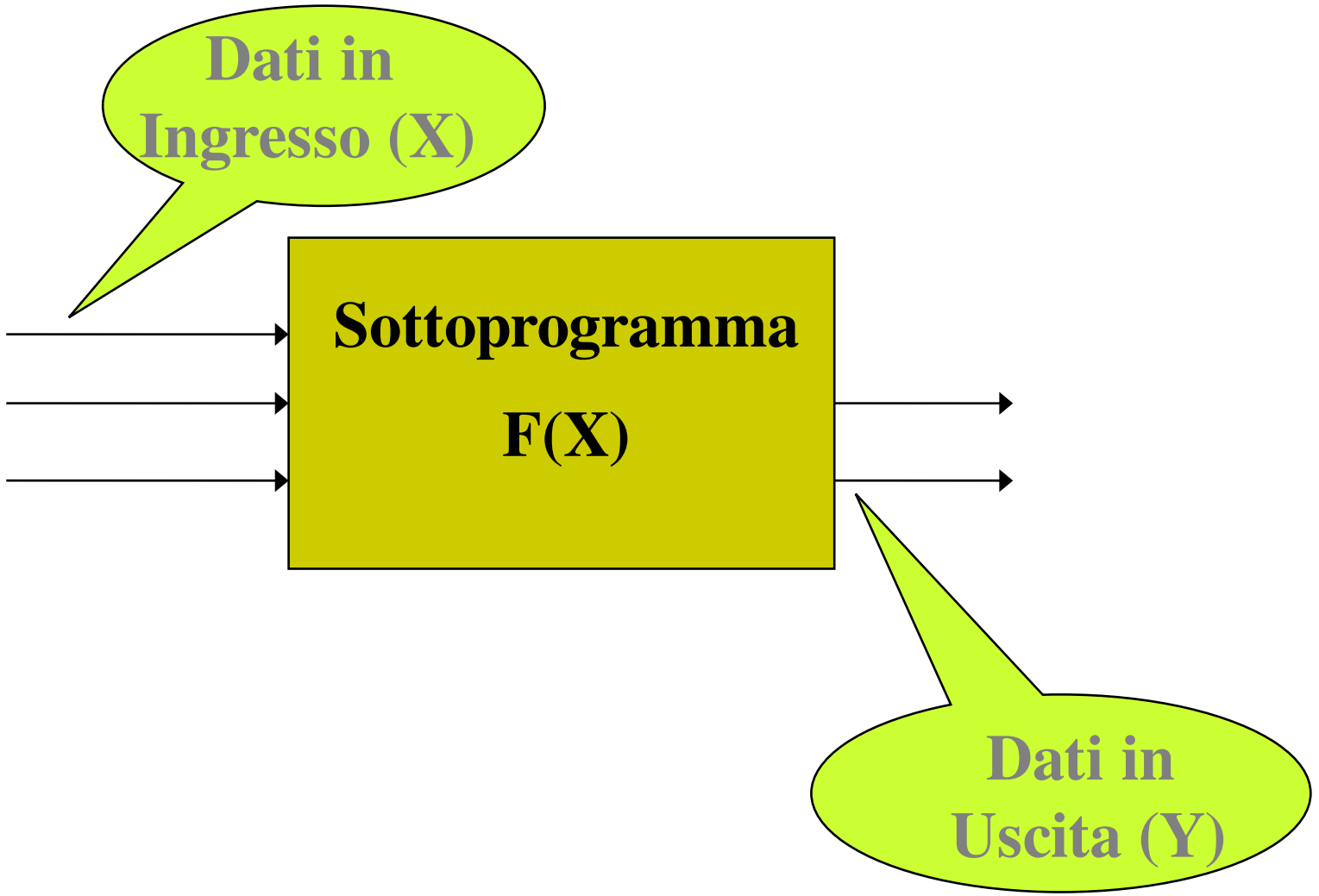


Interazione con il chiamante ed altri sottoprogrammi

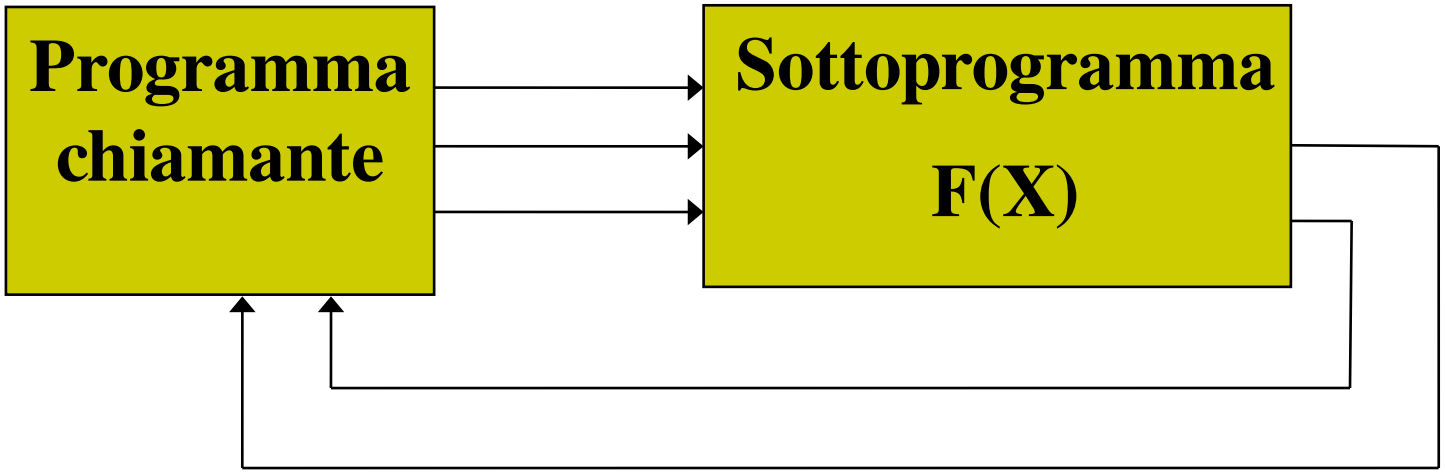
- Funzioni e procedure comunicano con il programma chiamante e con i sottoprogrammi che a loro volta chiamano mediante uno scambio di dati.
- I dati possono essere di ingresso, di uscita o di ingresso/uscita
- Lo scambio avviene mediante parametri:
  - Parametri formali:
    - Nomi delle variabili passate alla funzione
  - Parametri attuali:
    - Valori delle variabili passate alla funzione

Parametri formali

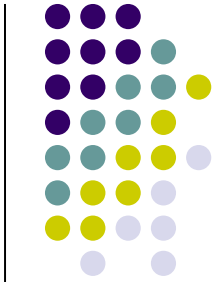
Y=F(X)



Parametri Effettivi/Attuali



Parametri formali ed effettivi/attuali



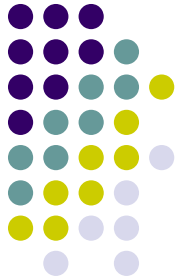
- Devono corrispondersi:
  - In numero
  - In posizione
  - In tipo

Regola della sostituzione

L'effetto di una chiamata di un sottoprogramma è quello che si ottiene eseguendo il sottoprogramma dopo aver sostituito ordinatamente a ciascun parametro formale, ovunque esso compaia nel corpo del programma, il corrispondente parametro effettivo

- N.B. **nomi dei parametri attuali e formali non hanno importanza**. Possono essere gli stessi o diversi. L'importante è la posizione ed il valore che assume un parametro attuale al momento della chiamata

Sottoprogrammi

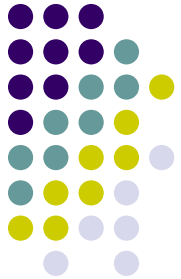


- Funzioni:
  - Esprimono l'astrazione matematica di funzione: calcolano un risultato valutato a partire dai valori che vengono forniti. Una funzione può ritornare **un solo valore di tipo atomico** attraverso l'istruzione ***return***
- Procedure:
  - Esprimono l'astrazione di "insieme di azioni", producono degli "effetti" ma non "ritornano" un risultato

Sottoprogrammi e astrazione sul controllo

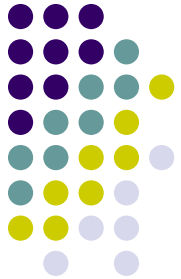
- I sottoprogrammi implementano l'astrazione sul controllo
- Possono pertanto essere visti come black box:
  - Un sottoprogramma svolge un compito offrendo un "servizio" a chi lo utilizza
  - Chi chiama un sottoprogramma non ha visibilità di come il servizio sia implementato
  - L'interazione tra sottoprogramma ed ambiente esterno avviene mediante una precisa interfaccia *costituita dalla intestazione.*

Interfaccia



- L'interfaccia di un sottoprogramma deve specificare:
  - Il nome (deve essere significativo della funzionalità offerta)
  - Quali informazioni sono necessarie al sottoprogramma per svolgere l'elaborazione richiesta
  - Quali risultati/informazioni il sottoprogramma mette a disposizione come risultato dell'elaborazione
  - *Nelle slides seguenti è resa nota all'utente mediante la **intestazione** del sottoprogramma*

Interfaccia e intestazione

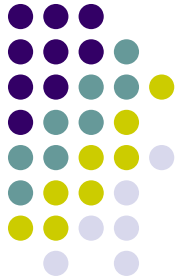


- L'intestazione infatti prevede:
  - Che sia specificato il nome da utilizzare per chiamare il sottoprogramma
  - Che sia specificata una lista (eventualmente vuota) di parametri formali di scambio con l'esterno
  - Il tipo del valore eventualmente ritornato
- Le informazioni da fornire al sottoprogramma costituiscono un sottoinsieme dei parametri formali
- I risultati/informazioni prodotti dal sottoprogramma sono specificate dal tipo del valore ritornato e/o da un sottoinsieme dei parametri formali

Sottoprogrammi in C/C++

- **L'astrazione di *funzione* è presente in tutti i linguaggi di programmazione di alto livello.**
- **In C/C++ non introduce esplicitamente le procedure. I sottoprogrammi in C/C++ sono esclusivamente funzioni**
- **Le procedure vengono *emulate* mediante le funzioni**

Elementi fondamentali



- Definizione della funzione
- Definizione della funzione (Prototipo)
- Uso della funzione (istruzione di chiamata)
- Esecuzione della funzione

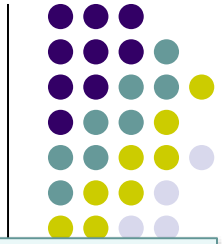
Esempio. Definizione di funzione

- Scrivere una funzione per il calcolo del fattoriale:

```
float fattoriale (int num) {

float fatt=1;
if (num==0 || num==1) fatt=1;
 else
   for (int i=1; i<=num; i++)
     fatt=fatt*i;
return fatt;
}
```

Esempio. Programma chiamante



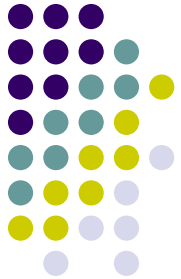
```
float fattoriale(int);  //prototipo

int main(int argc, char *argv[]) {

  int n;
  float ris;

  cout << "\n Calcolo del fattoriale";
  cout << "\n inserisci un numero intero: ";
  cin >> n;
  if (n<0) cout << "\n il fattoriale non e' definito per numeri interi negativi!";
  else {
  ris=fattoriale(n);  // istruzione di chiamata + assegnazione del valore ritornato
  cout << "\n il fattoriale di " << n << " e': " << ris;
  }
  cout << "\n";
  system("PAUSE");
  return 0;
}
```

Modalità di scambio dei parametri



- In C sono possibili due modalità di passaggio dei parametri:
  - Per valore
  - Per indirizzo
- In C++ è prevista una terza modalità
  - Per riferimento

Scambio per valore

- Il *passaggio per valore* di un argomento viene realizzato (dal compilatore) copiando il valore del parametro effettivo nel corrispondente parametro formale
- Se la funzione chiamata modifica un parametro passato per valore, *la funzione chiamante non potrà vedere queste modifiche*. Si tratta quindi di un passaggio unidirezionale

```
void stampa(const int x)
// x parametro formale
{  cout<<x; }
```

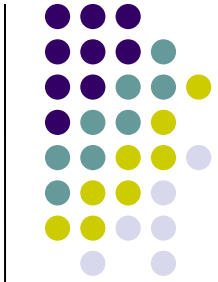
```
int b;
stampa(b); // b param. effett.
```

Scambio per indirizzo o per riferimento

- I parametri di **uscita** e quelli di **ingresso-uscita** vanno scambiati *per indirizzo* o mediante *passaggio per riferimento*
- Se il sottoprogramma modifica un parametro passato per indirizzo o per riferimento, *la modifica sarà visibile anche alla funzione chiamante*. Il passaggio è quindi potenzialmente bidirezionale.

- Il *passaggio per indirizzo* è totalmente a cura del programmatore, e consiste nel passare per valore l'indirizzo della variabile.
- Richiede la conoscenza e l'uso dei puntatori.

I parametri riferimento



- In alternativa, in C++, esiste il *passaggio per riferimento*
- Un *parametro riferimento* è un *alias* della variabile argomento
- Il passaggio per riferimento è efficiente in quanto evita la copia del valore
  - Il parametro formale in realtà è un puntatore che contiene l'indirizzo del parametro effettivo
  - Il passaggio dell'indirizzo (*dereferencing* del riferimento) è a cura del compilatore

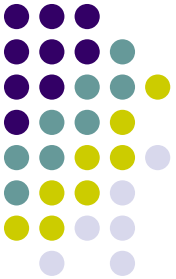
```
void incr(int &v) {
  v += 1;
}
```

```
int a = 4;
incr(a);
// ora "a" vale 5
```

Passaggio dei parametri di scambio

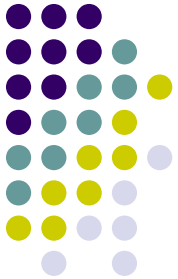
- Possono essere passati per valore solo i parametri di ingresso
- I parametri di uscita e quelli di ingresso/uscita, poiché devono essere modificati, devono essere passati per riferimento
- I parametri effettivi corrispondenti a parametri passati per valore devono essere espressioni
- I parametri effettivi corrispondenti a parametri passati per riferimento devono essere delle variabili

Parametri e modalità di scambio



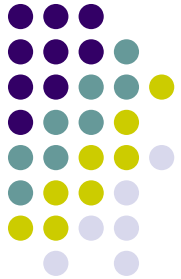
	valore	indirizzo	riferimento
in	sì	possibile/a volte necessario	possibile/a volte necessario
out	no	sì	sì
in/out	no	sì	sì

Di default in C/C++....



- Gli array sono scambiati per indirizzo
- Le variabili atomiche vengono scambiate per valore
- può essere utilizzato il qualificatore **const**

Riferimenti



- Da "Che C serve": Capitolo 6