Informatica

Terzo anno

Prof. A. Longheu

Architetture multi-componente

- In generale, una applicazione informatica complessa è organizzata come insieme di **componenti software** che cooperano per raggiungere un fine comune
- Un programma quindi NON è un unico blocco di istruzioni inserite nel main, ma spesso viene diviso in varie parti (qui, le funzioni)
- Ogni componente *fornisce servizi* ad altri componenti, e *usa i servizi* forniti da altri componenti.

  - Quali modelli per costruire una architettura multi-componente?
  - Quali astrazioni, quali modelli per costruire i singoli componenti?
  - Quali protocolli di comunicazione fra componenti?

Modello cliente/servitore

- Un modello molto generale per impostare un'architettura a componenti è il cosiddetto modello "cliente/servitore".

- ogni componente *fornisce servizi* ad altri componenti
  - **E' UN SERVITORE per essi**
- e *usa i servizi* forniti da altri componenti
  - **E' UN CLIENTE di servizi altrui**

Modello cliente/servitore

Servitore:

- Un qualunque ente computazionale capace di nascondere la propria organizzazione interna
- **presentando ai clienti una precisa *interfaccia*** per lo scambio di informazioni

Cliente:

- qualunque ente in grado di **invocare uno o più servitori** per svolgere il proprio compito

Modello cliente/servitore

Un servitore può:

- servire *molti clienti* oppure costituire la risorsa privata di uno *specifico cliente*
  - in particolare:
    - può servire un cliente alla volta, *in sequenza*,
    - oppure più clienti per volta, *in parallelo*
- *trasformarsi a sua volta in cliente*, invocando altri servitori o anche *se stesso*
- essere *passivo* o *attivo*

Comunicazione cliente/servitore

- Lo scambio di informazioni tra un cliente e un servitore può avvenire
  - in *modo esplicito* tramite le *interfacce* stabilite dal servitore
  - in *modo implicito* tramite *aree-dati* accessibili ad entrambi, ossia l'ambiente condiviso

Funzioni

Una funzione:

- è un gruppo di istruzioni che insieme compiono un certo compito
- Per compiere il lavoro assegnato, una funzione potrebbe avere bisogno di dati in ingresso, chiamati *parametri*
- Alla fine del lavoro assegnato, potrebbe restituire un qualche valore, frutto del lavoro; tale valore è il valore di uscita
- I parametri e/o il valore di uscita potrebbero essere assenti (ad es. una funzione che stampa “ciao” non richiede dati dall’esterno né restituisce valori)

Funzioni come servitori

- Una funzione è un servitore
  - *passivo*
  - che serve *un cliente per volta*
  - che può trasformarsi in cliente *invocando altre funzioni o se stessa*
- Una funzione è un servitore dotato di *nome* che incapsula le istruzioni che realizzano un certo *servizio*
- Il cliente chiede al servitore di svolgere il servizio
  - chiamando tale servitore (per nome)
  - *fornendogli le necessarie informazioni*
- Nel caso di una funzione, cliente e servitore comunicano mediante *l'interfaccia* della funzione

Interfaccia di una funzione

- L'**interfaccia** (o firma o *signature* o prototipo) di una funzione comprende
  - *nome della funzione*
  - *lista dei parametri*
  - *tipo del valore da essa denotato*

- *Esplicita il contratto di servizio* fra cliente e servitore
- Cliente e servitore comunicano quindi mediante
  - i *parametri* trasmessi dal cliente al servitore all'atto della chiamata
  - il *valore restituito* dal servitore al cliente

Funzioni: esempio

```
int massimo (int x, int y ){
    if (x>y) return x ;
    else return y;
}
```

- Il simbolo **massimo** denota il nome della funzione
- Le variabili intere **x** e **y** sono i parametri della funzione
- Il valore restituito è un intero **int**

Comunicazione cliente/servitore

- Il cliente passa informazioni al servitore mediante una serie di ***parametri***

Parametri formali:

- sono specificati nella *dichiarazione* del servitore
- esplicitano *il contratto* fra servitore e cliente
- indicano *che cosa il servitore si aspetta dal cliente*

Parametri attuali:

- sono *trasmessi dal cliente* all'atto della chiamata
- devono corrispondere ai parametri formali in *numero*, *posizione* e *tipo*

Funzioni: esempio parametri

Parametri formali

```
int massimo (int x, int y ){
    if (x>y) return x ;
          else return y;
}
```

SERVITORE

Definizione della funzione

```
main() {
     int z = 8;
     int m;

     m = massimo (z, 4);
}
```

CLIENTE

Chiamata della funzione

Parametri attuali

Comunicazione cliente/servitore

Legame tra parametri attuali e parametri formali:

- effettuato *al momento della chiamata,* in modo dinamico

Tale legame:

- vale solo per l'invocazione corrente
- vale solo per la durata della funzione

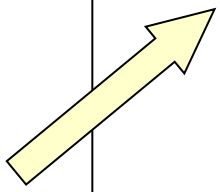
Esempio

Parametri formali

```
int massimo (int x, int y ){
    if (x>y) return x ;
          else return y;
}
```

```
main() {
     int z = 8;
     int m1, m2;

     m1 = massimo (z, 4);
     m2 = massimo (5, z);
}
```



All'atto di questa chiamata della funzione, si effettua un legame tra:

x e **z**

y e **4**

Esempio

Parametri formali

```
int massimo (int x, int y ){
    if (x>y) return x ;
          else return y;
}
```

```
main() {
     int z = 8;
     int m1, m2;

     m1 = massimo (z, 4);
     m2 = massimo (5, z);
}
```

All'atto di questa chiamata della funzione, si effettua un legame tra:

x e **5**

y e **z**

Information hiding

- La *struttura interna* (corpo) di una funzione è *completamente inaccessibile dall'esterno*
- Così facendo si garantisce *protezione dell'informazione* (*information hiding*)
- Una funzione è accessibile **solo** attraverso la sua interfaccia

Definizione di funzione

```
<definizione-di-funzione> ::=
  <tipoValore> <nome>(<parametri-formali>) {
      <corpo>
  }
```

La forma base è:
`return <espressione>;`

<parametri-formali>

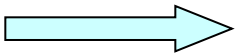
- o una **lista vuota**: `void`
- o una **lista di variabili** (separate da virgole) *visibili solo entro il corpo della funzione*

<tipoValore>

- deve coincidere con il tipo del valore restituito dalla funzione

Definizione di funzione

```
<definizione-di-funzione> ::=
  <tipoValore> <nome>(<parametri-formali>) {
      <corpo>
  }
```



La forma base è:
`return <espressione>;`

- Nella parte **corpo** possono essere presenti definizioni e/o dichiarazioni locali (***parte dichiarazioni***) e un insieme di istruzioni (***parte istruzioni***)
- I dati riferiti nel corpo possono essere **costanti**, **variabili**, oppure **parametri formali**
- All'interno del corpo, i parametri formali vengono trattati come variabili

Funzioni: nascita e morte

- All'atto della chiamata, **l'esecuzione del cliente viene sospesa e il controllo passa al servitore**
- Il servitore *"vive"* solo per il tempo necessario a svolgere il servizio
- Al termine, il servitore *"muore"*, e *l'esecuzione torna al cliente*

Chiamata di funzione

- La chiamata di funzione è un'espressione della forma

```
<nomefunzione> ( <parametri-attuali> )
```

- dove:

```
<parametri-attuali> ::=
    [ <espressione> ] { , <espressione> }
```

Funzioni: esempio

SERVITORE
Definizione
della funzione

Parametri formali

```
int massimo (int x, int y ){
    if (x>y) return x ;
          else return y;
}
```

```
main() {
    int z = 8;
    int m;

    m = massimo (z, 4);
}
```

Parametri attuali

CLIENTE
Chiamata
della funzione

Risultato di una funzione

- L'istruzione **return** provoca la *restituzione del controllo al cliente*, unitamente al valore dell'espressione che la segue
- Eventuali istruzioni successive alla return *non saranno mai eseguite*

```
int massimo (int x, int y ){
      if (x>y) return x ;
      else return y;
}
```

Funzioni: esempio

Parametri formali

```
int massimo (int x, int y ){
    if (x>y) return x ;
          else return y;
}
```

SERVITORE

Definizione della funzione

```
main() {
     int z = 8;
     int m;

     m = massimo (z, 4);
}
```

CLIENTE

Chiamata della funzione

Risultato

Riassumendo

All'atto dell'invocazione di una funzione:

- si crea una ***nuova attivazione (istanza)*** *del servitore*
- si alloca la ***memoria per i parametri*** (e le eventuali variabili locali)
- si trasferiscono i parametri al servitore
- si trasferisce il controllo al servitore
- si esegue il codice della funzione
- Alla fine, il controllo ritorna al chiamante

Progetto di una funzione

- *Scegliere un nome significativo per la funzione*
- *La funzione deve ricevere qualche dato dalla funzione chiamante?*
  - Se sì, elencare ed identificare tutti i tipi di dato da passare alla funzione (lista dei parametri)
  - Se no, la lista dei parametri è void
- *La funzione deve restituire un valore alla funzione chiamante?*
  - Se sì, identificare il tipo di dato
  - Se no, il tipo di ritorno della funzione è void
- *La funzione deve restituire più di un valore alla funzione chiamante?*
  - Se sì, bisogna inserire nella lista dei parametri dei parametri dati come “vuoti”, che saranno riempiti dalla funzione
  - Altrimenti basta il valore di ritorno

Progetto di una funzione: lista dei parametri

- **Parametri formali:** argomenti dichiarati nella definizione di funzione
- Devono essere variabili:
  - Non appena l'ambiente della funzione chiamata viene attivato, i parametri formali vengono dichiarati (come variabili locali all'ambiente della funzione) ed inizializzati al valore del corrispondente parametro attuale
  - La corrispondenza tra parametri formali ed attuali è sia posizionale sia di tipo. Ovvero si presume che la lista dei parametri formali e la lista dei parametri attuali abbia lo stesso numero e tipo di elementi
  - I 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

Progetto di una funzione: lista dei parametri

- **Parametri attuali:** argomenti inseriti al momento della chiamata di funzione
  - Possono essere espressioni *(costanti, variabili, espressioni aritmetiche, …)* di qualunque tipo, purchè compatibile con il corrispondente tipo del parametro formale