Astrazione e Modularizzazione

Come organizziamo il codice ?

- ***Progettazione***: l'insieme delle attività relative al concepimento della soluzione informatica di un problema
  - **dall'architettura, ai dati da manipolare, alle tecniche algoritmiche**
  - **si sviluppa a partire da una specifica …**
    - **… dal cosa al come**

- ***Modularizzazione***: dividere per gestire la complessità
  - **unità di programma**
    - **Alcune già note: funzioni, procedure, …**

Moduli

- Una unità di programma che mette a disposizione risorse e servizi computazionali (dati, funzioni, …)
- Fondamentale nella realizzazione dei concetti di:
  - **Astrazione**
  - **Information Hiding**
- Riuso di componenti già costruite e verificate
  - **Ad esempio: una volta definite delle funzioni che consentono di risolvere sotto-problemi di utilità generale, come è possibile riusarle nella soluzione di altri problemi ?**

Astrazione

- Procedimento mentale che consente da una parte di evidenziare le caratteristiche pregnanti di un problema e dall'altra di offuscare o addirittura ignorare gli aspetti che si ritengono secondari rispetto ad un determinato obiettivo
  - *La nozione, mutuata dalla psicologia, di "astrazione" permette di concentrarsi su un problema ad un determinato livello di generalizzazione, senza perdersi nei dettagli irrilevanti dei livelli inferiori; l'uso dell'astrazione permette anche di lavorare con concetti e termini che sono familiari all'ambiente di definizione del problema, senza doverli forzatamente trasformare in strutture non altrettanto note … (Wasserman, '83)*

Astrazione

- Già fatto largo uso nello sviluppo di programmi procedurali mediante l'applicazione della decomposizione funzionale …

- Astrazione funzionale: concentrare l'attenzione su *cosa* fa un certo sottoprogramma, astraendo dal *come* esso realizza il suo compito

- Non è il solo tipo di astrazione possibile …

Tipi di astrazione

- **Astrazione funzionale**
  - una funzionalità è totalmente definita ed usabile indipendentemente dall'algoritmo che la implementa (es. algoritmi di ordinamento di un array)
- **Astrazione sui dati**
  - un dato o un tipo di dato è totalmente definito insieme alle operazioni che sul dato possono essere fatte; pertanto, sia le operazioni che il dato (o il tipo di dato) sono usabili a prescindere dalle modalità di implementazione
- **Astrazione sul controllo**
  - un meccanismo di controllo è totalmente definito ed usabile indipendentemente dalle modalità e dalle tecniche con cui è realizzato

… l'enfasi in questo corso sarà sul secondo tipo …

Information hiding

- Parnas, 1972: occultamento dell'informazione
  - La realizzazione di alti livelli di astrazione passa attraverso la definizione di strutture capaci di mettere a disposizione (esportare) risorse e servizi occultando, ovvero rendendo inaccessibili, i dettagli implementativi
- ***Astrazione***: definire le entità funzionali o dati che compongono un sistema
- ***Information hiding***: definire ed imporre vincoli di inaccessibilità ai dettagli funzionali e di rappresentazione della struttura dei dati

Modulo

- Una unità di programma costituita da:
  - Una Interfaccia
    - **definisce le risorse ed i servizi (astrazioni) messi a disposizione dei "clienti" (programma o altri moduli)**
    - **completamente visibile ai clienti**
  - Una sezione implementativa (body)
    - **implementa le risorse ed i servizi esportati**
    - **completamente occultato**
- Un modulo può usare altri moduli
- Un modulo può essere compilato indipendentemente dal modulo (o programma) che lo usa

Modulo *nome*
Usa *nomi di moduli*
Interfaccia

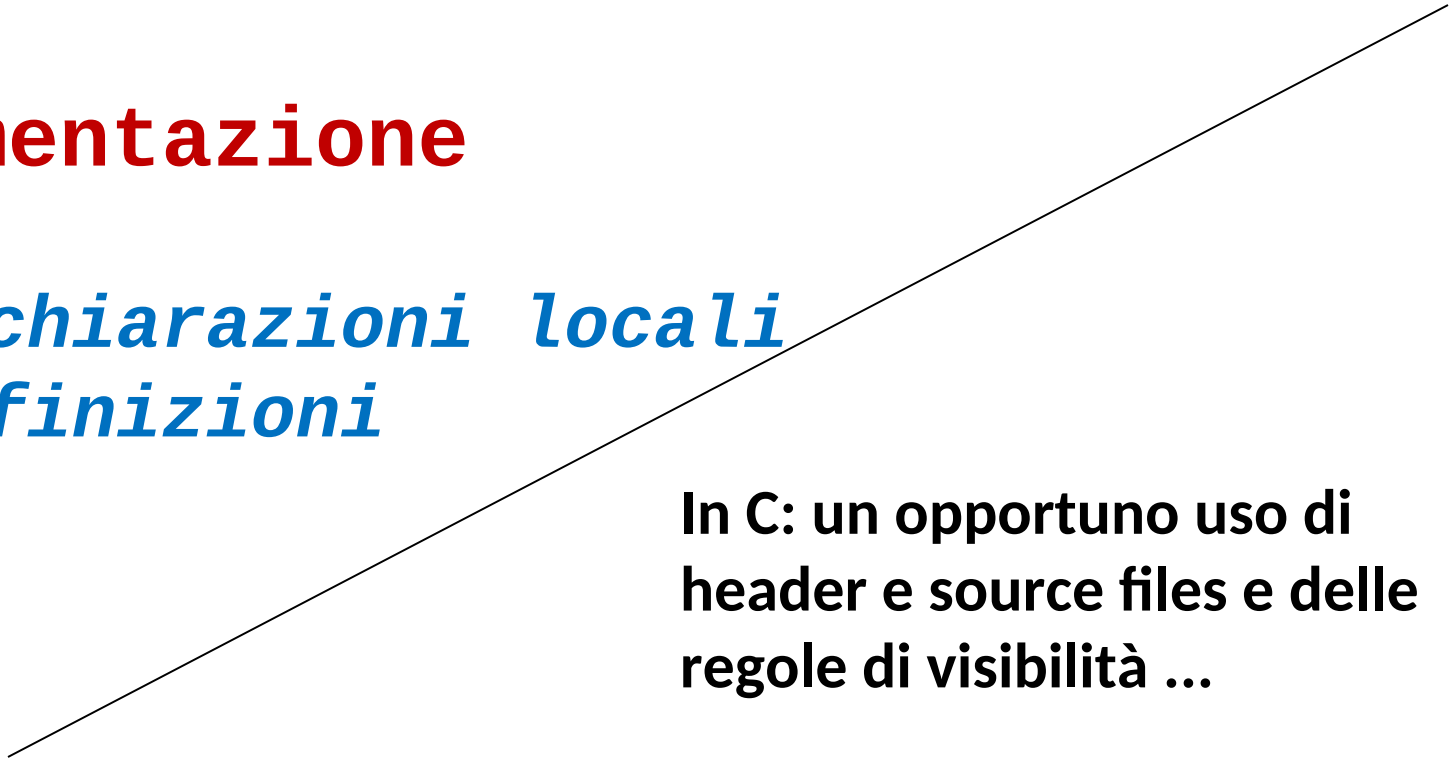
dichiarazioni

Implementazione

dichiarazioni locali
definizioni

Fine

Una visione astratta



In C: un opportuno uso di header e source files e delle regole di visibilità ...

Moduli e C

- In C non esiste un apposito costrutto per realizzare un modulo; di solito un modulo coincide con un file
- Per esportare le risorse definite in un file (modulo), il C fornisce un particolare tipo di file, chiamato ***header*** file (estensione ***.h***)
  - **Un header file rappresenta l'interfaccia di un modulo verso gli altri moduli**
- Per accedere alle risorse messe a disposizione da un modulo bisogna ***includere*** il suo header file
  - **Concetto già incontrato per le librerie predefinite: ad esempio #include <stdio.h> …**
  - **Per i moduli definiti dall'utente: #include "modulo.h" …**

Moduli e C

```
Modulo nome
Usa nomi di moduli
Interfaccia

    dichiarazioni

Implementazione

    dichiarazioni locali
    definizioni

Fine
```

include

Header file: dichiarazioni e prototipi

C file:
dichiarazioni *static* ...

definizioni

corpo delle funzioni

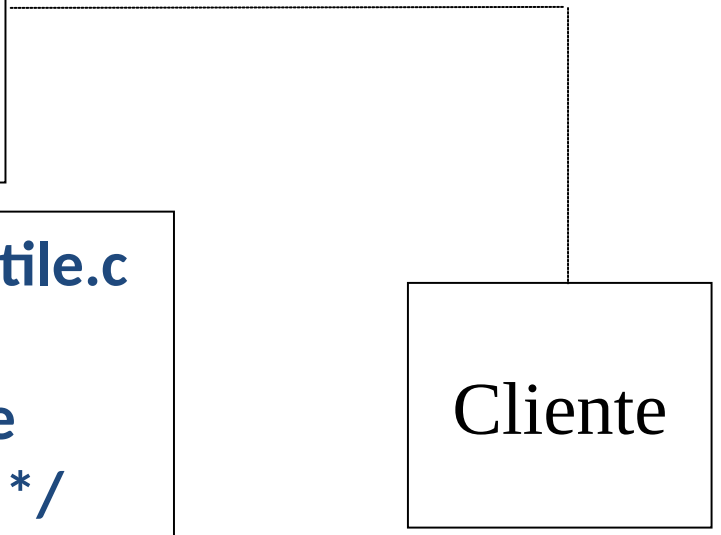
Moduli e librerie di funzioni

- Il modulo implementa astrazioni funzionali e mette a disposizione attraverso la sua interfaccia funzioni e procedure che realizzano le astrazioni
  - il modulo si presenta come una “libreria” di funzioni
  - per l’information hiding
    - nessun effetto collaterale
    - nessuna variabile globale
    - funzioni di servizio nascoste

Modulo utile

```
// Interfaccia del modulo: file utile.h

/* Specifica della funzione scambia */
void scambia(int * x, int * y);

// dichiarazione altre funzioni ...
```

```
// Implementazione del Modulo: file utile.c

/* commenti relativi alla progettazione
e realizzazione della funzione scambia */
void scambia(int * x, int * y)
{       int temp = *x;
        *x = *y;
        *y = temp;
}

// definizione altre funzioni ...
```



Cliente: può usare le risorse e i servizi esportati dal modulo

Uso dei commenti

- I commenti relativi alla specifica di una funzione possono essere inseriti nell'header file prima del prototipo della funzione
  - … serve da documentazione per chi dovrà usare la funzione (modulo client)
- I commenti relativi alla progettazione e realizzazione di una funzione possono essere inseriti nel file .c prima della definizione della funzione
  - … serve da documentazione per chi dovrà eventualmente modificare la funzione

Modulo vettore

vettore.h

```
void input_array(int a[], int n);
void output_array(int a[], int n);
void ordina_array(int a[], int n);
int ricerca_array(int a[], int n, int elem);
int minimo_array(int a[], int n);
…
```

vettore.c

```
#include <stdio.h>
#include “utile.h” // contiene funzione scambia

int minimo_i(int a[], int i, int n); // dichiarazione locale


void input_array(int a[], int n) { … }
void output_array(int a[], int n) { … }
void ordina_array(int a[], int n) { … }
int ricerca_array(int a[], int n, int elem) { … }
int minimo_array(int a[], int n) { … }
int minimo_i(int a[], int i, int n) { … } // usata da ordina_array
…
```

Il programma principale

```
// file ordina_array.c

# include <stdio.h>
# include “vettore.h”
# define MAXELEM 100

int main()
{
    …
}
```

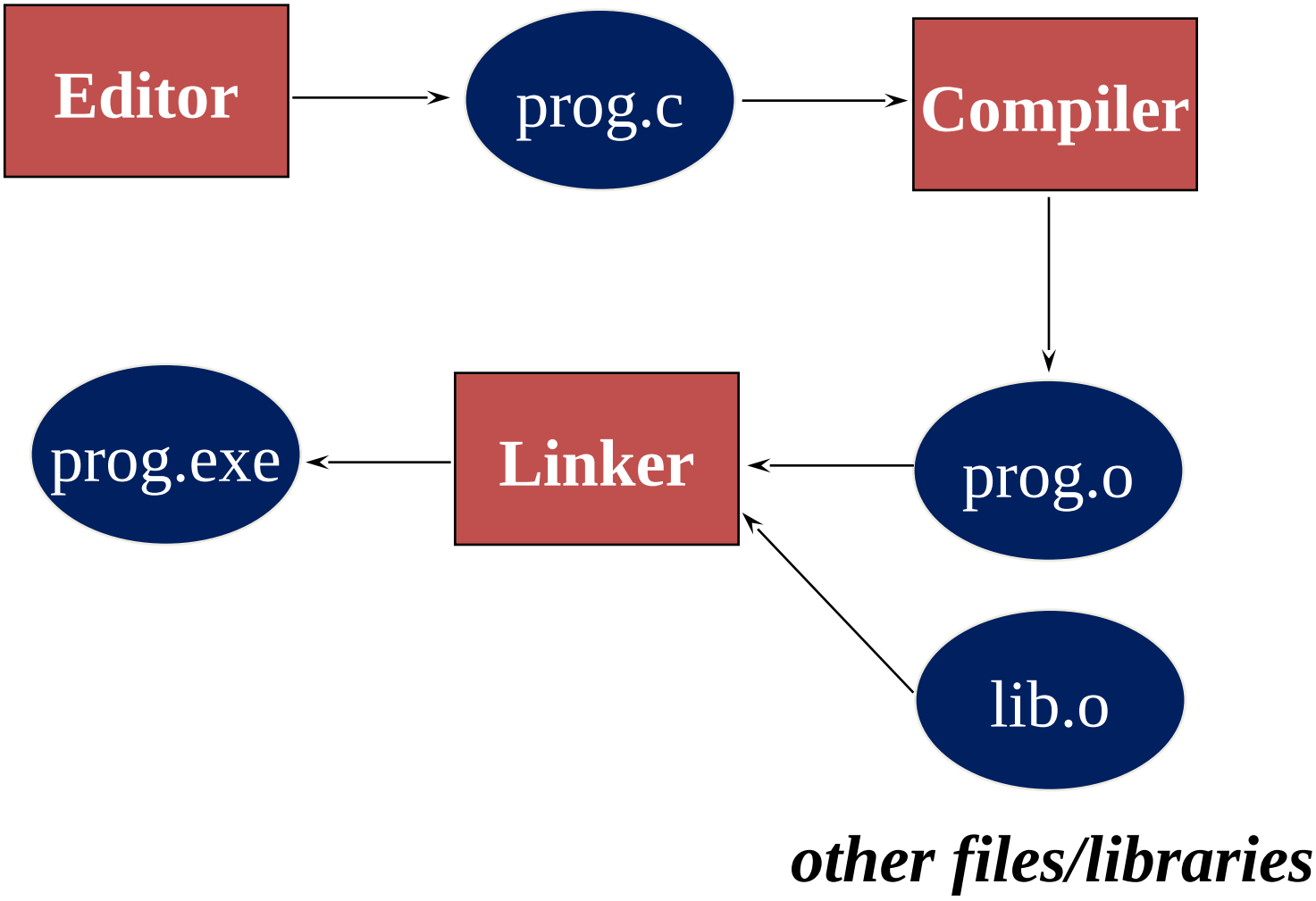
Modulo client del modulo vettore

Compilazione …

- I due moduli possono essere compilati indipendentemente (*)
  - **gcc -c utile.c**
  - **gcc -c vettore.c**
  - **gcc -c ordina_array.c**
- Possibile anche compilarli insieme
  - **gcc -c utile.c vettore.c ordina_array.c**
- In entrambi i casi si ottengono tre file con estensione .o
- Per collegare (link) i tre moduli e produrre l'eseguibile
  - **gcc utile.o vettore.o ordina_array.o -o ordina_array.exe**
- possibile compilazione e collegamento in un sol passo
  - **gcc utile.c vettore.c ordina_array.c -o ordina_array.exe**

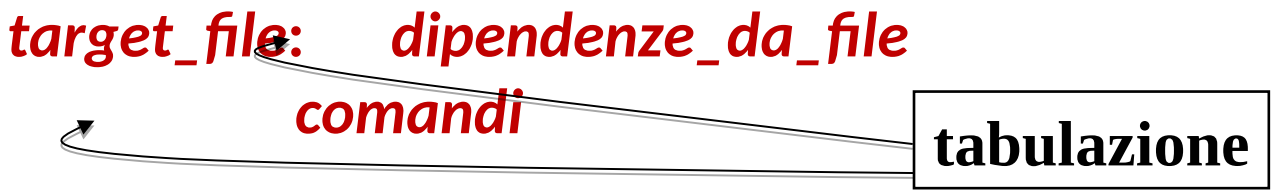
(*) L'opzione *–c* del comando *gcc* indica al compilatore di non eseguire il *linker*, così il risultato sarà il codice oggetto e non l'eseguibile

Dal programma sorgente al programma eseguibile



Progetto: Makefile e comando make

- Tutti gli ambienti di programmazione consentono di costruire un progetto
  - **il comando make: compilazione e collegamento dei vari moduli che compongono il progetto**
- In ambiente UNIX (Linux): Makefile e comando make
  - **Il Makefile è costituito da specifiche del tipo:**



  - **esecuzione della specifica: make *target_file***

Comando make: alcune osservazioni …

- L’ordine delle specifiche non è importante …
- … ma è buona norma inserire come prima specifica quella per la costruzione del programma eseguibile
  - **In questo caso per lanciare il processo basta digitare il comando make**

… il Makefile del nostro esempio

```
ordina_array.exe:       utile.o  vettore.o  ordina_array.o
        gcc utile.o  vettore.o  ordina_array.o  -o ordina_array.exe

utile.o:            utile.c
        gcc -c utile.c

vettore.o:       vettore.c  utile.h
        gcc -c vettore.c

ordina_array.o:          vettore.h  ordina_array.c
        gcc -c ordina_array.c
```

Makefile e comando make

- Nel nostro esempio, come effetto del comando make:
  - **bisogna produrre ordina_array.exe, ma occorrono utile.o vettore.o ordina_array.o**
  - **se i due file non esistono, si cercano le specifiche per produrli, e così via …**
- Ordine di esecuzione:
  - **gcc –c utile.c**
  - **gcc -c vettore.c**
  - **gcc -c ordina_array.c**
  - **gcc utile.o vettore.o ordina_array.o -o ordina_array.exe**