

# Client - Interfaccia - Implementazione

Violetta Lonati

Università degli studi di Milano  
Dipartimento di Informatica

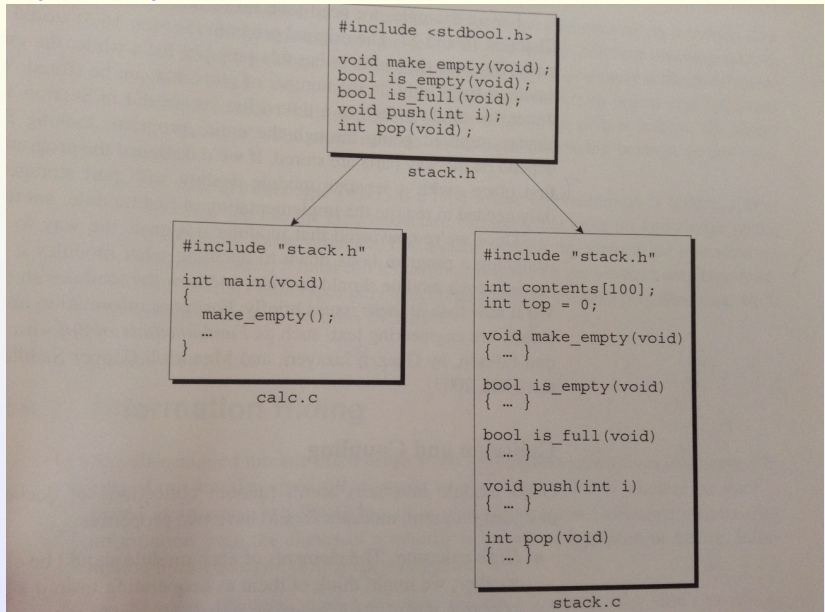
Laboratorio di algoritmi e strutture dati  
Corso di laurea in Informatica

# Paradigma Client - Interfaccia - Implementazione

Quando si progetta un programma di grandi dimensioni, può essere utile vederlo come un insieme di più *componenti* o *moduli* indipendenti.

- ▶ Ciascun modulo fornisce dei **servizi** ad altre parti del programma, che per questo vengono chiamate **client**.
- ▶ Ogni modulo ha un'**interfaccia** che descrive i servizi che mette a disposizione.
- ▶ I dettagli relativi al modulo (incluso il codice che realizza i servizi stessi) ne definiscono l'**implementazione**.

# Esempio: la pila



# Vantaggi dell'uso di moduli

**Astrazione** non è necessario sapere come sono implementati i moduli, possiamo usarli astrattamente sulla base di quanto specificato dall'interfaccia.

**Riusabilità** ogni modulo che fornisce servizi è potenzialmente riutilizzabile da programmi diversi, con scopi diversi.

**Manutenibilità** un baco avrà effetti relativi all'implementazione di un solo modulo, sarà quindi tendenzialmente più facile da individuare e correggere; una volta corretto non sarà necessario ricompilare tutto il programma ma solo l'implementazione del modulo coinvolto (a patto poi di ripetere il *link* dell'intero programma).

# Come progettare un modulo

Un modulo deve essere:

1. coeso – i suoi elementi devono essere fortemente correlati fra loro;
2. indipendente – meno relazioni ha con gli altri moduli, più sarà riusabile e manutenibile.

## Esempi di moduli

- ▶ Famiglia di dati: una collezione di variabili e macro correlate (es: `float.h` o `limits.h`).
- ▶ Libreria: una collezione di funzioni correlate (es: `string.h` è l'interfaccia di una libreria per la gestione di stringhe).
- ▶ Oggetto astratto: una collezione di moduli che operano su una struttura dati *nascosta* (es: la pila vista prima)
- ▶ Tipo di dato astratto: un tipo di dato la cui rappresentazione è nascosta. I programmi client possono usare il tipo astratto per dichiarare una variabile, e applicarvi le funzioni specificate nell'interfaccia, ma non sanno come questo tipo sia implementato.

## Encapsulation e Information hiding

Un modulo ben progettato **nasconde** delle informazioni ai suoi client. Questo garantisce:

- ▶ Sicurezza: se un client non sa come è rappresentata e memorizzata la pila, non potrà accedervi direttamente, ma soltanto attraverso le funzioni messe a disposizione dall'interfaccia, funzioni che sono state scritte e testate a parte.
- ▶ Flessibilità: è possibile modificare l'implementazione di un modulo senza che questo abbia effetti sul client (purché non venga modificata l'interfaccia).

## Oggetti astratti e tipi di dati astratti

Un oggetto astratto, come nell'esempio della pila visto prima, ha uno svantaggio importante: *non c'è modo di avere più istanze dell'oggetto*. Per risolvere questo problema è necessario fare un passo ulteriore e creare un **nuovo tipo astratto**.

### Come cambia il client?

- ▶ Potranno essere definiti più oggetti dello stesso tipo astratto `Stack s1, s2;`
- ▶ Per il client, `s1` e `s2` sono astrazioni che rispondono a quanto specificato nell'interfaccia.

### Come cambia l'interfaccia?

- ▶ Sarà definito il nuovo tipo, es `Stack`.
- ▶ Nella lista dei parametri delle funzioni, ci sarà un nuovo argomento, che indica a quale istanza di `Stack` ci stiamo riferendo.



# Come fare tutto ciò in C?

## File di intestazione

Con la direttiva `#include` del preprocessore, è possibile condividere informazioni (prototipi di funzioni, definizioni di macro, ecc). La direttiva dice al preprocessore di aprire il file specificato e di inserire il suo contenuto nel file corrente. I file inclusi in questo modo sono chiamati **header file** o **file di intestazione**. Per convenzione hanno estensione `.h`

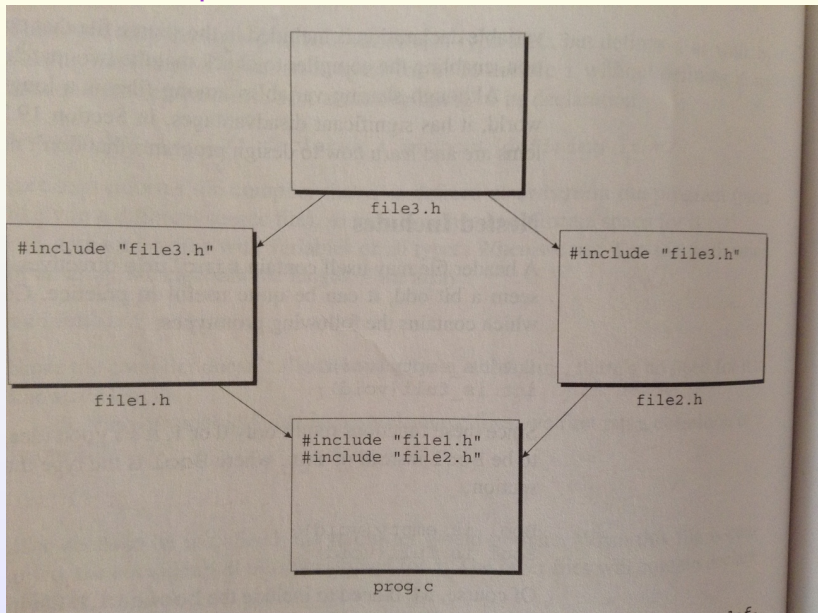
`#include <filename>` per i file della libreria standard;

`#include "filename"` per gli altri file, compreso quelli scritti da te.

## Protezione dei file di intestazione

Se un file sorgente include un header file più di una volta, possono verificarsi errori in fase di compilazione.

# Inclusione multipla di file di intestazione



# Protezione dei file di intestazione

Per proteggere un file si può usare la direttiva del preprocessore `#ifndef`.

Esempio: protezione del file `boolean.h`

```
#ifndef BOOLEAN_H
#define BOOLEAN_H

/* contenuto vero e proprio del file boolean.h */
#define TRUE 1
#define FALSE 0
typedef int Bool;
/* fine contenuto */

#endif
```