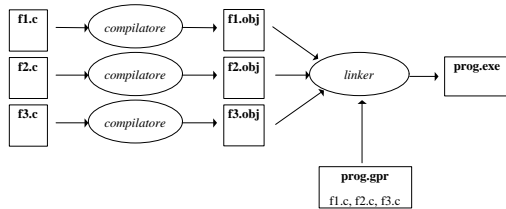


CREAZIONE DI UN'APPLICAZIONE



FUNZIONI & FILE

- Un programma C è, in prima battuta, una collezione di funzioni
 - una delle quali è il *main*
- Il testo del programma deve essere scritto in uno o più *file di testo*
 - il file è un concetto *del sistema operativo*, non del linguaggio C. Il linguaggio C non fornisce

Quali regole osservare ?

FUNZIONI & FILE: compilazione

- Il *main* può essere scritto *dove si vuole*
 - viene chiamato dal sistema operativo, il quale sa come identificarlo
- Una funzione, invece, deve rispettare una *regola fondamentale di visibilità*
 - prima che qualcuno possa *chiamarla*, la funzione deve essere stata *dichiarata*, non necessariamente *definita*
 - altrimenti ➡ *errore di compilazione*.

DALL'ESEMPIO SU UN SOLO FILE...

File `prova1.c`

```

int fact(int);

main() {
    int y = fact(3);
}

int fact(int n) {
    return (n<=1) ? 1 : n*fact(n-1);
}
  
```

... ALL'ESEMPIO SU DUE FILE

File `main.c`

```

int fact(int);
main() {
    int y = fact(3);
}
  
```

Dichiarazione della funzione

Uso (chiamata)

File `fact.c`

```

int fact(int n) {
    return (n<=1) ? 1 : n*fact(n-1);
}
  
```

Definizione della funzione

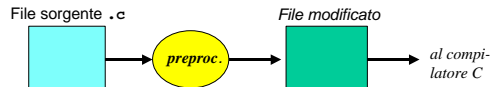
COSTRUZIONE DI UN'APPLICAZIONE

Perché la costruzione vada a buon fine:

- ogni funzione deve essere *definita* *una e una sola volta* in *uno e uno solo* dei file sorgente. Altrimenti ➡ *errore di Link*
- ogni cliente che *usi* una funzione deve incorporare la *dichiarazione* opportuna

↓
trascrivere riga per riga le dichiarazioni necessarie?

IL PRE-PROCESSORE C



Il pre-processor *non* è un compilatore C

- modifica il testo del programma.
- *non conosce il linguaggio C*: non può interpretare le istruzioni C, né controllarne la correttezza
- potrebbe manipolare *qualunque* testo, non solo programmi C

IL PRE-PROCESSORE C

Cosa può fare?

- includere altre porzioni di testo, prese da altri file
- effettuare *ricerche e sostituzioni* (più o meno sofisticate) sul testo
- *inserire o sopprimere parti del testo* a seconda del verificarsi di certe condizioni da noi specificate.

IL PRE-PROCESSORE C

Come si controlla il suo funzionamento?

- mediante *direttive* inserite nel testo.

Attenzione: le direttive *non sono istruzioni C*

- non ne hanno la sintassi
- infatti, *non sono destinate al compilatore*, che non le vedrà mai
- vengono soppresse dal pre-processor dopo essere state da esso interpretate.

DIRETTIVE AL PRE-PROCESSORE C

- includere altre porzioni di testo
`#include nomefile`
- effettuare *ricerche e sostituzioni*
`#define testo1 testo2`
- *inserire o sopprimere parti del testo*
`#ifdef cond #ifndef cond`
`...testo... ...testo...`
`#endif #endif`

LA DIRETTIVA #include

Include il contenuto del file specificato *esattamente nella posizione* in cui si trova la direttiva stessa.

`#include <libreria.h>`

include l'header di una *libreria di sistema*
il sistema sa già dove trovarlo

`#include "miofile.h"`

include un file scritto da noi
occorre indicare dove reperirlo
(attenzione al formato dei percorsi)

FILE HEADER

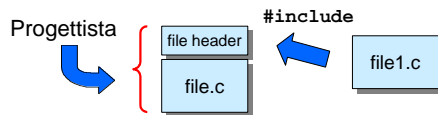
Per automatizzare la gestione delle dichiarazioni, si introduce il concetto di *header file* (file di intestazione). Scopo:

- Separare l'*Interfaccia* di un modulo dalla sua *Realizzazione*
- evitare ai clienti di dover trascrivere riga per riga le dichiarazioni necessarie

FILE HEADER

In pratica:

- il progettista di un componente software (un file .c) predispone un header file contenente *tutte le dichiarazioni* relative alle funzioni definite nel modulo
- I clienti *non dovranno più ricopiarsi a mano le dichiarazioni*: basterà *includere l'header file* tramite una direttiva `#include`.



FILE HEADER

Il file di intestazione (header)

- ha estensione **.h**
- ha (per convenzione) **nome uguale al file .c** di cui fornisce le dichiarazioni

Ad esempio:

- se la funzione `f` è definita nel file `f2c.c`
- il corrispondente header file, che i clienti potranno includere per usare la funzione `f`, dovrebbe chiamarsi `f2c.h`

ESEMPIO

Il problema della conversione °F / °C

1ª versione: singolo file

```

float fahrToCelsius(float f) {
    return 5.0/9 * (f-32);
}

main() {
    float c = fahrToCelsius(86);
}
    
```

ESEMPIO

Vogliamo suddividere cliente e servitore su due file separati

File `main.c` (cliente)

```

float fahrToCelsius(float);
main() { float c = fahrToCelsius(86); }
    
```

File `f2c.c` (servitore)

```

float fahrToCelsius(float f) {
    return 5.0/9 * (f-32);
}
    
```

ESEMPIO

Perché la dichiarazione sia *inclusa automaticamente*, occorre *introdurre un file header*

File `main.c` (cliente)

```

#include "f2c.h"
main() { float c = fahrToCelsius(86); }
    
```

File `f2c.h` (header)

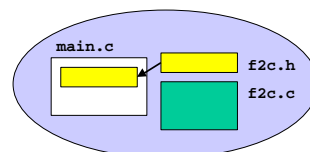
```

float fahrToCelsius(float);
    
```

ESEMPIO

Struttura finale dell'applicazione:

- un main definito in `main.c`
- una funzione definita in `f2c.c`
- un file header `f2c.h` incluso da `main.c`



FILE HEADER - CAUTELE D'USO

- **ATTENZIONE!!** Un file header deve contenere *solo dichiarazioni* !
- Se contiene anche solo una definizione possono crearsi situazioni di *errore* (rischio di *definizioni duplicate*).

FILE HEADER - CAUTELE D'USO

Esempio

- un main usa le funzioni f1 e f2
- sia f1 sia f2 usano la funzione f
- lo header di f contiene la definizione invece della dichiarazione

È un falso header !



FILE HEADER - CAUTELE D'USO

