

...quando il gioco si fa duro

- Sono considerate “applicazioni di piccola dimensione”, applicazioni con qualche migliaio di linee di codice
- Un’applicazione anche “di piccola dimensione” non può essere sviluppata in un unico file
→INGESTIBILE
- Deve necessariamente essere strutturata su più file sorgente
 - Compilabili separatamente
 - Da fondere insieme successivamente per costruire l’applicazione

Funzioni come componenti SW

- Una funzione è un *componente software (servitore) riutilizzabile...*
- ...che costituisce una *unità di traduzione*:
 - può essere definita in un file a sé stante
 - compilata per proprio conto
 - pronta per essere usata da chiunque

Funzioni come componenti SW

Per usare tale componente software, il cliente:

- non ha bisogno di sapere *come è fatto* (cioè, di conoscerne la *definizione*)
- deve conoscerne solo l'interfaccia:
 - nome
 - numero e tipo dei parametri
 - tipo del risultato

3

Dichiarazione di funzione

La dichiarazione di una funzione è costituita dalla sola interfaccia, *senza corpo* (sostituito da un `;`)

```
<dichiarazione-di-funzione> ::=  
  <tipoValore>  <nome>(<parametri>) ;
```

4

Dichiarazione di funzione

- Per usare una funzione non occorre conoscere tutta la *definizione*
- È sufficiente conoscere la *dichiarazione*, perché essa specifica il *contratto di servizio*
- La *definizione* di una funzione costituisce l'effettiva realizzazione del componente
- La *dichiarazione* specifica il *contratto di servizio* fra cliente e servitore, esprimendo le proprietà essenziali della funzione.

5

Dichiarazione VS Definizione

- *Definizione*: dice come è fatto il componente
- *Dichiarazione*: specifica il *contratto di servizio* – come si usa il componente
- *Contratto di servizio*: per usare una funzione non serve sapere come è fatta... anzi...

6

Dichiarazione di funzione

- La *dichiarazione* specifica:
 - il nome della funzione
 - numero e tipo dei parametri (non necessariamente *il nome*)
 - il tipo del risultato

Nota: il nome dei parametri non è necessario, se c'è viene ignorato... perché?!

→ Avrebbe significato solo nell'*environment* della funzione che non esiste non essendoci la definizione

7

Funzioni & File

- Un programma C è, in prima battuta, una collezione di funzioni
 - Una di queste funzioni è SEMPRE `main()`
- Il codice deve essere scritto in uno o più file di testo
 - Attenzione: file è un concetto di sistema operativo e non del linguaggio C

Quali regole osservare?

8

Dichiarazione VS Definizione

- **Definizione:** dice come è fatto il componente
 - costituisce l'effettiva realizzazione del componente
 - NON può essere DUPLICATA!
 - Un'applicazione può contenere una e una sola definizione di una funzione
 - La compilazione di una definizione genera il codice macchina corrispondente alla funzione

9

Dichiarazione VS Definizione

- La **dichiarazione** di una funzione costituisce solo una specifica delle proprietà del componente
 - Può essere duplicata senza problemi
 - Un'applicazione può contenerne più di una
 - La compilazione di una dichiarazione non genera alcun codice macchina

10

Funzioni & File

- Il main può essere scritto dove si vuole nel file
 - Viene invocato dal sistema operativo, il quale sa come identificarlo
- Una funzione deve rispettare una regola fondamentale di visibilità
 - Prima che qualcuno possa invocarla, la funzione deve essere stata **dichiarata** (va bene anche definizione – contiene una dichiarazione)
 - ...altrimenti → **errore di compilazione!**

11

Esempio (singolo file)

File `prova.c`

```
int fact(int);  
  
int main()  
{  
    int y = fact(3);  
    printf("%d", y);  
    return 0;  
}  
  
int fact(int n)  
{  
    return (n<=1) ? 1 : n * fact(n-1);  
}
```

*Dichiarazione (prototipo):
deve precedere l'uso*

Uso (invocazione)

Definizione

12

Progetti su più file

- Per strutturare un'applicazione su più file sorgente, occorre che ogni file possa essere compilato **separatamente** dagli altri
 - Successivamente avverrà la fusione
- Affinché un file possa essere compilato, tutte le funzioni usate devono essere **dichiarate** prima dell'uso
 - Non necessariamente definite!

13

Esempio su due file

File main.c

```
int fact(int);

int main()
{
    int y = fact(3);
    printf("%d", y);
    return 0;
}
```

File fact.c

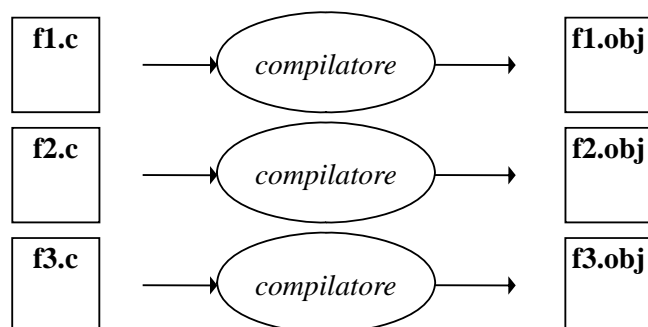
```
int fact(int n)
{
    return (n<=1) ? 1 : n * fact(n-1);
}
```

14

Compilazione di un'applicazione

1. Compilare i singoli file che costituiscono l'applicazione

- File *sorgente*: estensione `.c`
- File *oggetto*: estensione `.o` o `.obj`

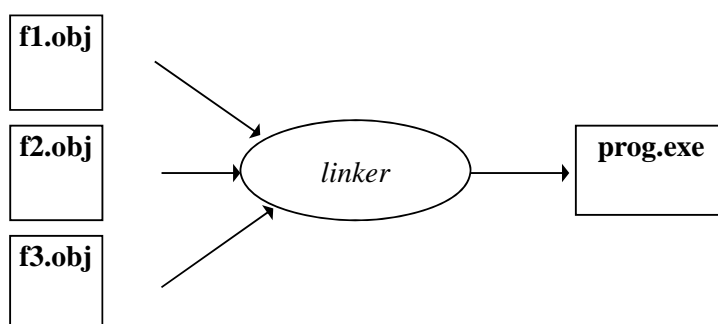


15

Compilazione di un'applicazione

1. Collegare i file oggetto fra loro e con le librerie di sistema

- File *oggetto*: estensione `.o` o `.obj`
- File *eseguibile*: estensione `.exe` o nessuna



16

Riassumendo

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
 - se la definizione manca, si ha errore di linking
- ogni cliente che *usi* una funzione deve incorporare la dichiarazione opportuna
 - se la dichiarazione manca, si ha errore di compilazione nel file del cliente (..forse...!!)

17

Linker?

Perché, esattamente, serve il linker?

- Il compilatore deve “*lasciare in bianco*” i riferimenti alle chiamate di funzione che non sono definite nel medesimo file
- Compito del linker è *risolvere* tali riferimenti, riempiendo gli “spazi bianchi” con l’indirizzo effettivo del codice della funzione.

18

Progetti complessi...

- Ogni cliente deve contenere le dichiarazioni delle funzioni che utilizza
- In una applicazione complessa non è pensabile che questo venga fatto a mano (*copy & paste?*) “file per file”

AUTOMATISMO!

19

Header File

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

- Scopo: evitare ai clienti di dover trascrivere riga per riga le dichiarazioni necessarie
 - il progettista di un componente software (un file .c) predispone un *header file* contenente tutte le dichiarazioni relative alle funzioni definite
 - i clienti non dovranno più ricopiarsi a mano le dichiarazioni: basterà includere l'header file tramite una direttiva `#include`

20

Header File

Il *file di intestazione (header)*

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

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**

21

Header File

Due formati:

#include <libreria.h>

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

#include "miofile.h"

include uno header scritto da noi
occorre indicare dove reperirlo

(attenzione al formato dei percorsi..!!)

22

Header File

- Attenzione: un *header file* dovrebbe contenere solo dichiarazioni e non definizioni
 - In caso contrario è possibile che una definizione sia compilata più volte generando poi errori di link
 - E per le variabili globali, come si fa?
 - **extern!!**

23

Conversione °F / °C

- Versione su singolo file

```
float fahrToCelsius(float f) {  
    return 5.0/9 * (f-32);  
}  
main() {  
    float c = fahrToCelsius(86);  
}
```

24

Conversione °F / °C

- Suddivisione *cliente* e *servitore* su 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);  
}
```

25

Conversione °F / °C

- Si introduce un file header per includere automaticamente la dichiarazione

File `main.c` (*cliente*)

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

File `f2c.h` (*header*)

```
float fahrToCelsius(float);
```

26

Conversione °F / °C

- **Struttura finale dei file dell'applicazione**
 - Un file main.c contenente il main
 - Un file f2c.c contenente la funzione di conversione
 - Un file header f2c.h contenente la dichiarazione della funzione di conversione
 - Incluso da main.c

27

Convenzione – Header file

- Se un componente è definito in xxx.c
 - Le dichiarazioni (parte pubblica) di tale componente sono contenute in xxx.h
- il nome del file header contenente le dichiarazioni è uguale al nome del file c contenente le definizioni

28