Progetti su più File Sorgenti

- 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 →***modularità, riutilizzo, leggibilità***
- Deve necessariamente essere strutturata ***su più file sorgente***
  - ***Compilabili separatamente***
  - Da collegare insieme successivamente per costruire l’applicazione

Funzioni come Componenti SW

- Avremmo bisogno di un modulo di codice - ***componente software (servitore) riutilizzabile***
- ***Concetto di funzione*** *(?)*
- Costituisce una *unità di traduzione:*
  - può essere definita in un file a sé stante
  - compilata per proprio conto
  - pronta per essere usata da chiunque

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

Dichiarazione di funzione

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

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

- Per usare una funzione ***non occorre conoscere tutta la definizione***
- È sufficiente conoscerne la ***dichiarazione*** ovvero la specifica del *contratto di servizio*

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…

→ Avrebbe significato solo nell'ambiente di esecuzione (vedi record di attivazione) della funzione, che però al momento non esiste (non essendoci la definizione)

Dichiarazione vs. Definizione

- **Definizione: dice come è fatto il componente**
  - costituisce ***l'effettiva realizzazione*** del componente
  - ***NON può essere DUPLICATA***
  - La compilazione di una definizione genera il codice oggetto corrispondente alla funzione
- 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***

Funzioni e 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?

Funzioni e File

- **`main()`** può essere scritto dove si vuole nel file
  - Viene invocato dal sistema operativo, che lo identifica sulla base del nome
- 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!**

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

Progetti su più file

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

Esempio su due file

File main.c

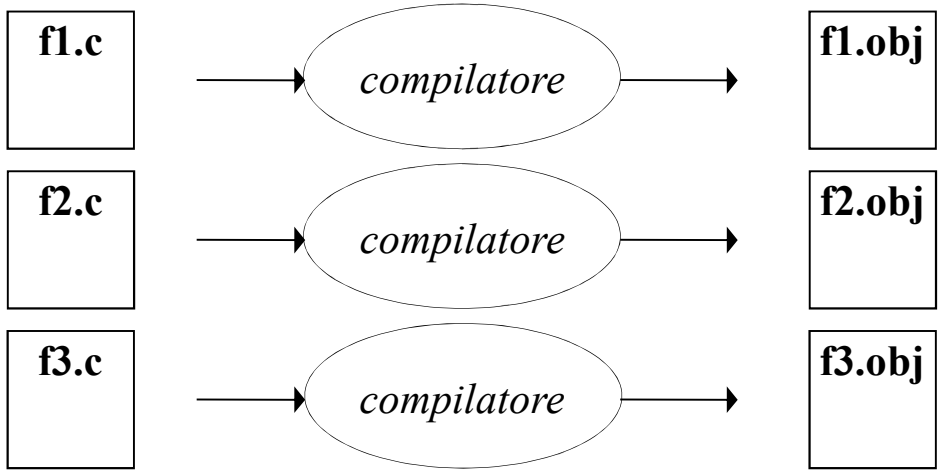
```
int fact(int);

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);
}
```

Compilazione di un'applicazione

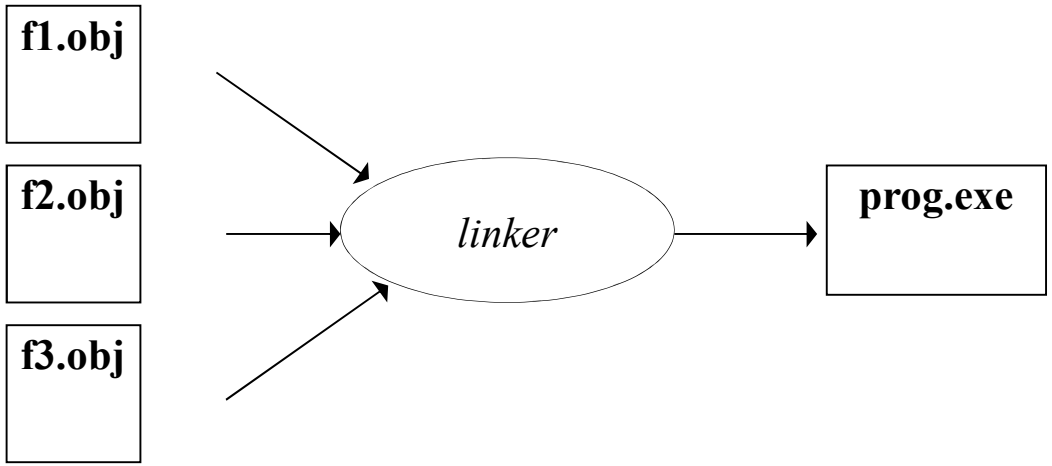
1. Compilare i singoli file che costituiscono l'applicazione
   - File *sorgente*: estensione `.c`
   - File *oggetto*: estensione `.o` oppure `.obj`



Compilazione di un'applicazione

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

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



Riassumendo

Perché la produzione dell'eseguibile 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

Linker?

Perché, esattamente, serve il linker?

- Il compilatore deve "*lasciare indefiniti*" i ***riferimenti alle chiamate di funzione*** che non sono definite nel medesimo file
- ***Compito del linker è risolvere tali riferimenti***, sostituendoli con gli indirizzi effettivi del codice della funzione

Progetti complessi…

- Ogni ***cliente deve contenere le dichiarazioni delle funzioni*** che utilizza

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 predispone un *header file* contenente tutte le dichiarazioni relative alle funzioni definite nel suo componente software (o modulo)
  - i clienti potranno semplicemente includere tale header file tramite una direttiva **`#include`**

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 **`func1`** è definita nel file **`file2c.c`**
- il corrispondente *header file*, che i clienti potranno includere per usare la funzione **`func1`**, dovrebbe chiamarsi **`file2c.h`**

Header File

Due formati:

1) **`#include <`*libreria.h*`>`**

include normalmente l'header di ***una libreria di sistema***
la macchina runtime del C sa dove cercare il file header (all'interno di un elenco di directory predefinite)

2) **`#include`** ***"miofile.h"***

include normalmente un header di "propria produzione"
generalmente occorre indicare alla macchina runtime dove andare a reperirlo *(attenzione al formato dei percorsi, dipende dal sistema operativo, non dal linguaggio)*

Header File

- Attenzione: un ***header file dovrebbe contenere solo dichiarazioni*** e non definizioni (sia di funzioni che di variabili)
  - Possibile problema di una definizione compilata più volte, generando poi anche errori di linking
  - E se servono ***variabili globali utilizzate da più file sorgenti***?
    - Clausola **`extern`**

Conversione ° F / ° C

- Versione su singolo file

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

Conversione ° F / ° C

- Suddivisione su due file separati

File `main.c` *(cliente)*

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

File `f2c.c` *(servitore)*

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

Conversione ° F / ° C

- Si può introdurre un file header per includere automaticamente la dichiarazione

File **`main.c`** *(cliente)*

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

File `f2c.h` *(header)*

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

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