Gestione progetti C

Classi di memorizzazione

- **Quando si dichiarano delle variabili, senza specificare alcuna classe di memorizzazione (cioè quando lo si fa normalmente come negli esempi visti fino a questo punto), il loro campo d'azione è relativo alla posizione della dichiarazione:**
  - **le variabili dichiarate all'esterno delle funzioni sono globali, cioè accessibili da parte di tutte le funzioni, a partire dal punto in cui vengono dichiarate**
  - **le variabili dichiarate all'interno delle funzioni sono locali, cioè accessibili esclusivamente dall'interno della funzione in cui si trovano**
- **Le stesse regole di visibilità valgono per i blocchi**

Classi di memorizzazione

- **Si distinguono quattro tipi di classi di memorizzazione, a ciascuna delle quali corrisponde una parola chiave per la loro dichiarazione:**
  - ***automatica*** **auto**
  - ***registro*** **register**
  - ***statica*** **static**
  - ***esterna*** **extern**
- **auto: vale in modo predefinito e non occorre indicarla quando si dichiarano le variabili (automatiche)**
- **Una variabile dichiarata come appartenente alla classe register viene posta in un registro del microprocessore**

Classi di memorizzazione

- **Si distinguono quattro tipi di classi di memorizzazione, a ciascuna delle quali corrisponde una parola chiave per la loro dichiarazione:**
  - ***automatica*** **auto**
  - ***registro*** **register**
  - ***statica*** **static**
  - ***esterna*** **extern**
- **static genera due situazioni distinte a seconda della posizione in cui viene dichiarata la variabile**
  - **Se è definita al di fuori delle funzioni, risulterà accessibile solo all'interno del file sorgente in cui viene dichiarata**
  - **Se invece è interna a una funzione, la variabile mantiene il suo valore tra una chiamata e l'altra della funzione**

Classi di memorizzazione

- **Si distinguono quattro tipi di classi di memorizzazione, a ciascuna delle quali corrisponde una parola chiave per la loro dichiarazione:**
  - ***automatica*** **auto**
  - ***registro*** **register**
  - ***statica*** **static**
  - ***esterna*** **extern**

- **La dichiarazione extern è usata**
  - **quando da un file sorgente si vuole accedere a variabili globali dichiarate in modo normale in un altro file**
  - **quando in un file si vuole poter accedere a variabili dichiarate in una posizione più avanzata dello stesso**
  - **In questo modo si informa esplicitamente il compilatore e il linker della presenza di queste**

Classi di memorizzazione

```
extern int iTuaVariabile;
...
int miafunzione(...) {
iTuaVariabile = ...
}
...
```

- **Questo caso rappresenta la situazione in cui una variabile dichiarata in un altro file sorgente diventa accessibile alle funzioni del file attuale attraverso la dichiarazione extern**
  - **Perché ciò possa funzionare, occorre che la variabile iTuaVariabile sia stata dichiarata in modo normale, senza la parola chiave static**

Programmi in un singolo modulo

- **Principali inconvenienti:**
  - **se si fanno anche solo delle piccole modifiche, bisogna sempre ricompilare tutto il programma**
  - **le funzioni definite nel programma non possono essere utilizzate facilmente in altri programmi**
- **Il primo inconveniente porta inevitabilmente a tempi di compilazione piuttosto elevati**
- **Il secondo inconveniente non può efficacemente essere risolto semplicemente ricopiando le funzioni perché**
  - **se ad un certo punto si trova un codice più efficiente per la funzione, si dovrebbe rimpiazzare ogni copia della vecchia funzione con la nuova**
  - **ciascuna copia della funzione occupa spazio disco**

Suddivisione in più moduli

- **I programmi C spesso sono articolati in più file sorgenti separati che vengono compilati in modo indipendentemente e infine collegati in un unico eseguibile**
- **Nella produzione di un eseguibile, vengono ricompilati solo i file sorgente che hanno subito modifiche successivamente alla generazione del file oggetto relativo**
- **Per permettere la condivisione del codice bisogna**
  - **rimuovere il codice delle funzioni che si vogliono rendere meglio fruibili**
  - **compilarlo separatamente e, quindi,**
  - **fare il linking (collegamento) del modulo oggetto risultante in qualsiasi programma che lo voglia utilizzare**

File di intestazione (header)

- **Quando un programma è suddiviso in più moduli sorgente:**
  - **se si utilizzano delle macro del preprocessore, queste dovranno essere dichiarate in tutti i sorgenti che ne fanno uso;**
  - **se all'interno di un sorgente si fa riferimento ad una funzione dichiarata altrove, il sorgente deve contenere una dichiarazione del prototipo della funzione (le variabili globali vanno trattate in modo analogo)**
- **Per facilitarne l'inclusione in tutti i sorgenti che lo necessitano, conviene predisporre dei file di intestazione (con suffisso *.h*) con le dichiarazioni delle macro del preprocessore e dei prototipi delle funzioni**

Make
(in ambiente «consolle»…)

- **La ricompilazione di un programma suddiviso in più moduli può essere un'operazione alquanto laboriosa: si potrebbe predisporre uno script per eseguire tutte le operazioni necessarie ma la tradizione impone di usare Make**
- **Make è una utility che aggiorna un file sulla base di regole di dipendenza contenute in un file con un formato speciale detto makefile**
  - **Per default, make si aspetta che le regole di dipendenza si trovino nel file makefile o, preferibilmente, Makefile**
  - **È possibile specificare un nome diverso per il file contenente le regole (il cui nome è consigliabile abbia suffisso .make)**
- **Il makefile contiene la lista di tutte le interdipendenze che esistono tra i file che sono usati per creare l'eseguibile**

Make
(in ambiente integrato)

- **Makefile è molto utilizzato in ambito scientifico, all'interno di sistemi UNIX**
  - **Vale la pena conoscerne le caratteristiche, ma al momento del suo eventuale utilizzo**
- **In un ambiente integrato (come lcc) il controllo delle dipendenze viene effettuato «in automatico»**
  - **Il risultato verrà trascritto in un makefile (ricordate il «generate makefile» delle opzioni di compilazione…) che poi verrà eseguito**
- **Vediamo qualche esempio**