ISIS "Guido Tassinari" di Pozzuoli
Indirizzo Informatico - Articolazione Informatica

Informatica
Prof. Rio Chierego
A.S. 2012/2013

Linguaggio C: le funzioni. Introduzione e sintassi

21/10/2012

Introduzione

- Spesso alcuni gruppi di operazioni vengono ripetute in diverse parti all'interno del medesimo programma
- Si pensi ad esempio a un programma di gestione di matrici
  - È plausibile pensare che una matrice dovrà essere stampata a video in numerosi momenti
  - Il codice relativo alla stampa è sempre lo stesso (due cicli for annidati)
- È quindi utile definire una sola volta questo gruppo di istruzioni e dare un nome ad esso
- Ogni volta che si vuole stampare a video una matrice sarà sufficiente indicare il nome assegnato

Sottoprogrammi

- Un sottoprogramma è quindi:
  - Un insieme di istruzioni dotato di nome
  - descritto (definito) una sola volta,
  - attivabile (richiamabile o invocabile) all'interno del programma o di un altro sottoprogramma
- Alcuni sottoprogrammi che usiamo sono già definiti
  - Si pensi alla scanf e alla printf
  - Dietro a questi nomi vi sono una serie di istruzioni in grado di, rispettivamente, intercettare la pressione dei tasti e di visualizzare uno o più caratteri sullo schermo
  - Chi richiama queste funzioni non si preoccupa di come sono fatte, basta sapere solo cosa fanno (visione black box)

Sottoprogrammi: motivazioni 1/2

- Astrazione e leggibilità:
  - enucleano parti di codice, nascondendo dettagli algoritmici e di codifica.
  - Il nome del programma si presenta come "un'operazione elementare"
- Strutturazione e scomposizione funzionale del programma:
  - consentono una stesura del programma che riflette un'analisi funzionale del problema
- Collaudo:
  - verifica di correttezza della soluzione facilitata dal poter verificare la correttezza prima dei singoli sottoprogrammi e poi dell'intero programma visto come insieme di chiamate che si scambiano informazioni

Sottoprogrammi: motivazioni 2/2

- Compattezza, efficienza e leggibilità del codice:
  - si evita di ripetere sequenze di istruzioni in più parti del programma
- Modificabilità:
  - una sola modifica vale per tutte le attivazioni del sottoprogramma
- Riuso:
  - sottoprogrammi non troppo specifici possono essere raccolti in librerie utilizzabili da programmi diversi

Funzioni e procedure

- In tutti i linguaggi di programmazione di alto livello, i sottoprogrammi possono essere
  - di tipo funzionale (FUNZIONI)
  - di tipo procedurale (PROCEDURE)
- Si differenziano:
  - per la logica di definizione
  - per l'uso e per la modalità di chiamata
- Sottoprogrammi di tipo funzionale (FUNZIONI) possono essere considerati una astrazione di valore
  - l'invocazione della funzione associa al nome della funzione il valore del risultato calcolato dal sottoprogramma
- Sottoprogrammi di tipo procedurale (PROCEDURE) possono essere considerati una astrazione di operazioni
  - l'invocazione della procedura è associata all'esecuzione delle istruzioni del sottoprogramma che realizzano l'operazione specificata dal sottoprogramma

ISIS "Guido Tassinari" di Pozzuoli
Indirizzo Informatico - Articolazione Informatica

Sintassi

Funzioni e procedure in C

- In C esistono solo le FUNZIONI. Le PROCEDURE sono particolari funzioni che non restituiscono nulla (tipo void)
- Quindi parleremo solo di FUNZIONI intendendo sia le funzioni che le procedure
- Definire una FUNZIONE secondo il linguaggio C implica:
  - 1) DICHIARAZIONE del PROTOTIPO della funzione (nella sezione dichiarativa)
  - 2) DEFINIZIONE della funzione (dopo il main)
  - 3) INVOCAZIONE o CHIAMATA della funzione (nel codice che necessita della funzioni)

Dichiarazione del prototipo

- Il PROTOTIPO definisce:
  - il nome della funzione
  - il tipo restituito ("void" per una procedura)
  - il nome ed il tipo dei parametri in input e in output (ev. nessuno)
  - la modalità d'uso (ossia stabilisce come deve essere chiamata)
- Chi utilizzerà la funzione dovrà rispettare la sintassi definita nel prototipo
- Prototipo FUNZIONE

```
<tipo_restituito> <nome_funz> (<lista dei parametri>);
```

- Prototipo PROCEDURA
  void è una parola chiave del C che indica assenza di tipo)

```
void <nome_funz> (<lista dei parametri>);
```

Definizione di una funzione

- La DEFINIZIONE della funzione va messa subito dopo il main.
  Ha la stessa struttura del main.
  (N.B. in realtà anche il main è una funzione chiamata dal S.O.):
  - Una parte dichiarativa
  - Una parte esecutiva

```
<tipo| void> <nome> (<tipo par_f_1> <nome_1>, <tipo par_f_2> <nome_2>  ...)
{
 <parte dichiarativa locale>
 <parte esecutiva>
  return;    oppure   return <valore>;
}
```

- `<tipo> <nome> (<tipo par_f_1> <nome_1>, <tipo par_f_2> <nome_2>  ...)` è la testata della funzione
- `par_f_1, par_f_2` sono i nomi dei parametri formali della funzione, il cui tipo deve corrispondere in modo ordinato ai tipi elencati nel prototipo

Parametri formali

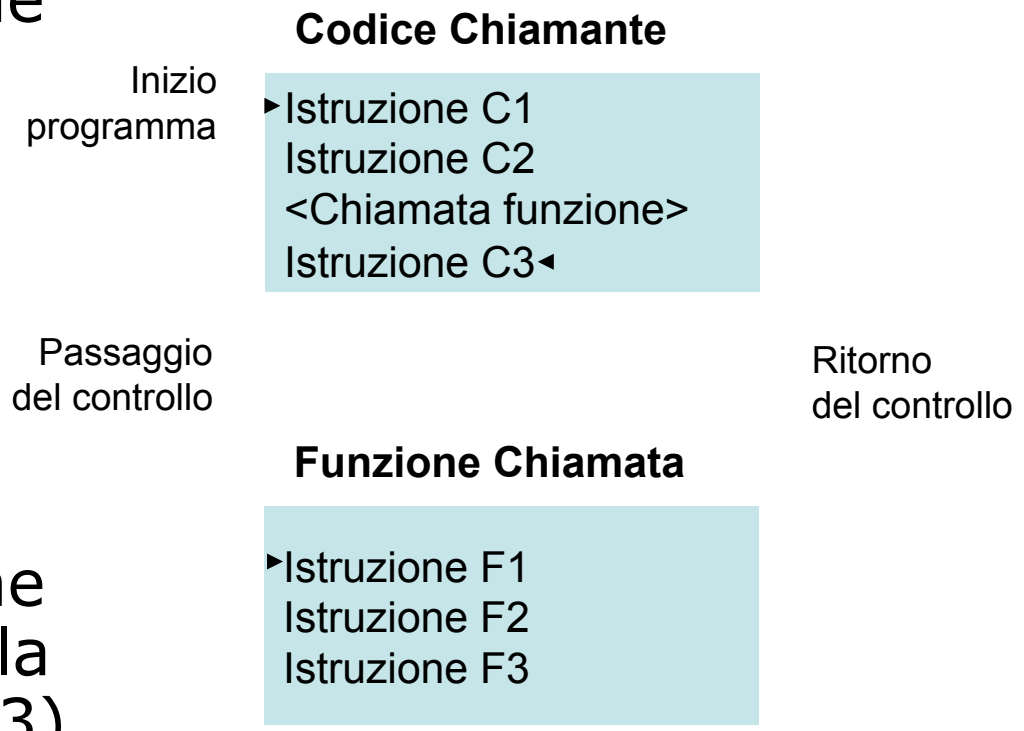
- Rappresentano un riferimento simbolico (identificatori) a oggetti utilizzati all'interno della funzione
- Sono utilizzati dalla funzione come se fossero **variabili** dichiarate localmente
- Il valore iniziale di parametri formali viene definito all'atto della chiamata della funzione dal programma chiamante tramite il passaggio dei parametri attuali. Esso può avvenire:
  a) passaggio per valore o BY VALUE
  b) passaggio per riferimento o BY REFERENCE

Invocazione o chiamata

- Nel corpo del main o di un'altra funzione indica il punto in cui va eseguita la parte del codice presente nella definizione di funzione
- invocazione di funzione
  - come operando in una espressione
  - `<espressione> = <nomefunzione(par_att1, par_att2, …)…>;`
- invocazione di procedura
  - come un'istruzione
  - `nomeprocedura(par_att1, par_att2, …);`
- In entrambi i casi:
  - `par_att1, par_att2, …` sono i parametri attuali che devono corrispondere per ordine, numero e per tipo ai parametri formali.
  - I parametri attuali possono essere variabili, costanti o espressioni definite nell'ambiente chiamante, i cui valori all'atto della chiamata vengono copiati nei parametri formali

Chiamata di una funzione

- All'atto della chiamata, il controllo dell'esecuzione passa dal chiamante al chiamato
- Il codice del chiamato viene eseguito
- Al termine dell'esecuzione il controllo ritorna al chiamante, all'istruzione successiva a quella della chiamata (Istruzione C3)



Scambio di informazioni

- Il programma chiamante deve poter ricevere dei valori attuali (specifici) sui quali eseguire le operazioni definite nel sottoprogramma
- Il programma chiamato deve poter fornire al chiamante i risultati dell'elaborazione
- Lo scambio di informazioni tra programma chiamante e programma chiamato può avvenire in tre modi distinti:
  - 1) tramite valore restituito da una funzione (dal p. chiamato al p. chiamante)
    - vale solo per le funzioni
    - per fornire un valore al programma chiamante
  - 2) tramite passaggio dei parametri (dal p. chiamante al p. chiamato)
    - per fornire i valori in ingresso al chiamato
    - per fornire i risultati di operazioni al chiamante
  - 3) tramite variabili globali (in entrambe le direzioni)
    - fortemente sconsigliato

Istruzione `return`

- Parola chiave C utilizzata solo nelle funzioni
- Sintassi

```
return <espressione>;
```

- È l'ultima istruzione di una funzione e indica:
  - Il punto in cui il controllo torna al chiamante
  - Il valore restituito
- In una funzione
  - Deve esserci almeno un'istruzione di return
    (non necesssria se restituisce un void)
  - Possono esserci più istruzioni di return ma in alternativa
    (fortemente sconsigliato: uso della programmazione strutturata
  - La funzione può restituire un solo valore di un tipo