Dipartimento di Elettronica ed Informazione
Politecnico di Milano

Informatica e CAD (c.i.) - ICA
Prof. Pierluigi Plebani
A.A. 2008/2009

Linguaggio C: le funzioni. Visibilità variabili e passaggio parametri

27/11/2008

Introduzione

- Introducendo le funzioni è stato introdotto un meccanismo per definire dei piccoli programmi all'interno di altri programmi
- Questi piccoli programmi sono autonomi con il resto del codice
- Al loro interno sono definite le proprie variabili ed il canale di comunicazione con il codice chiamante (passaggio paramentri, return)
- A supporto di questo meccanismo deve esistere una gestione degli identificatori delle variabili
- È infatti possibile avere due variabili con lo stesso nome a patto che abbiano raggio di visibilità disgiunto

Visibilità

- Visibilità di un identificatore: indicazione della parte del programma in cui tale identificatore può essere usato
- Ambiente globale del programma
  - insieme di identificatori (tipi, costanti, variabili) definiti nella parte dichiarativa globale
  - regole di visibilità: visibili a tutte le funzioni del programma
- Ambiente locale di una funzione
  - insieme di identificatori definiti nella parte dichiarativa locale e degli identificatori definiti nella testata (parametri formali)
  - Regole di visibilità: visibili alla funzione e ai blocchi in essa contenuti
- Ambiente di blocco
  - insieme di identificatori definiti nella parte dichiarativa locale del blocco
  - regole di visibilità: visibili al blocco e ai blocchi in esso contenuti

Esempi

```
int x;
f()
{
   int y;
   y=1;
}
```

- La visibilità di y si estende dal punto di dichiarazione fino alla fine del blocco di appartenenza

```
int x;
g(int y, char z)
{
   int k;
   int l;
   …
}
```

- **`y`** e **`z`** locali alla funzione **`g`**, con visibilità nel blocco racchiuso da parentesi graffe
- **`k`** e **`l`** hanno la stessa visibilità

```
f(int x)
{
   int x;
}
```

- Errata! Si tenta di definire due volte la variabile locale x nello stesso blocco

Mascheramento (shadowing)

- Un nome ridefinito all'interno di un blocco nasconde il significato precedente di quel nome
- Tale significato è ripristinato all'uscita del blocco più interno

```
int x;                          /*nome globale*/
int f() {
   int x;                  /*x locale che nasconde x globale*/
   x=1;             /*assegna 1 al primo x locale*/
   {
      int x;               /* nasconde il primo x locale*/
      x=2;                 /*assegna 2 al secondo x locale*/
   }
   x=3;             /*assegna 3 al primo x locale*/
 }
scanf ("%d", &x);       /*inserisce un dato in x globale*/
```

- In caso di omonimia di identificatori in ambienti diversi è visibile quello dell'ambiente più "vicino"

Ambiente di esecuzione

- L'ambiente di esecuzione di una funzione viene creato al momento della chiamata e rilasciato quando la funzione termina
- In una sequenza di chiamate, l'ultima chiamata è la prima a terminare
- La zona di memoria di lavoro che contiene l'ambiente di esecuzione di un sottoprogramma è gestito con la logica di una pila (stack)
  - L'ultimo elemento inserito nello stack è il primo ad essere estratto
  - Logica LIFO (Last In First Out)

Record di attivazione

- Alla chiamata di una funzione
  - si alloca uno spazio di memoria (record di attivazione) in cima allo stack per contenere i parametri formali e le variabili locali
  - lo spazio viene rilasciato quando la funzione termina
- Il record di attivazione contiene:
  - L'ambiente locale della funzione
  - L'indirizzo di ritorno al chiamante

Funzionamento del record di attivazione

- Ad ogni attivazione viene allocato un record di attivazione
- Al termine dell'attivazione il record viene rilasciato (l'area di memoria è riutilizzabile)
- La dimensione del record di attivazione è già nota in fase di compilazione
- Il numero di attivazioni della funzione non è noto
- Il primo record di attivazione è destinato al main()

Lo stack

- Nello stack, i record vengono allocati "uno sopra l'altro"; il primo record dello stack è relativo all'ultima funzione attivata e non ancora terminata
- Lo stack cresce dal basso verso l'alto
- Stack pointer: registro della CPU che contiene l'indirizzo della cima della pila
- Una parte della RAM è destinata a contenere lo stack
  - Stack overflow: quando l'area della RAM destinata allo stack viene superata (troppi annidamenti di chiamate)

Dipartimento di Elettronica ed Informazione
Politecnico di Milano

Informatica e CAD (c.i.) - ICA
Prof. Pierluigi Plebani
A.A. 2008/2009

Linguaggio C: le funzioni. Passaggio dei parametri

27/11/2008

Passaggio dei parametri

- Il passaggio dei parametri consiste nell'associare, all'atto delle chiamata di un sottoprogramma, ai parametri formali i parametri attuali
  - Se il prototipo di una funzione è

    ```
    float circonferenza (float raggio);
    ```

  - Invocare questa funzione significa eseguire l'istruzione

    ```
    c = circonferenza(5.0);
    ```

  - In questo modo la variabile raggio (il parametro formale) assumerà per quella particolare invocazione il valore 5 (il parametro attuale).
- Lo scambio di informazioni con passaggio dei parametri tra chiamante e chiamato può avvenire in due modi:
  - Passaggio per valore
  - Passaggio per indirizzo

Passaggio per valore

- All'atto della chiamata il valore del parametro attuale viene **copiato** nelle celle di memoria del corrispondente parametro formale. In altre parole il parametro formale e il parametro attuale si riferiscono a due diverse celle di memoria
- Il sottoprogramma in esecuzione lavora nel suo ambiente e quindi sui parametri formali
- I parametri attuali non vengono modificati

Passaggio per indirizzo

- All'atto della chiamata l'indirizzo dei parametri attuali viene associato ai parametri formali. In altre parole il parametro attuale e il parametro formale si **riferiscono alla stessa cella di memoria**
- Il sottoprogramma in esecuzione lavora nel suo ambiente sui parametri formali (e di conseguenza anche sui parametri attuali) e ogni modifica sul parametro formale è una modifica del corrispondente parametro attuale
- Gli effetti del sottoprogramma si manifestano nel chiamante con modifiche al suo ambiente locale di esecuzione

Passaggio dei parametri in C

- In C non esiste un costrutto sintattico per distinguere tra passaggio dei parametri per valore e per indirizzo
- Il passaggio è sempre per valore

```
float circonferenza(float raggio);
                              /*passaggio per valore*/
```

- Per ottenere il passaggio per indirizzo è necessario utilizzare parametri formali di tipo indirizzo (puntatori)

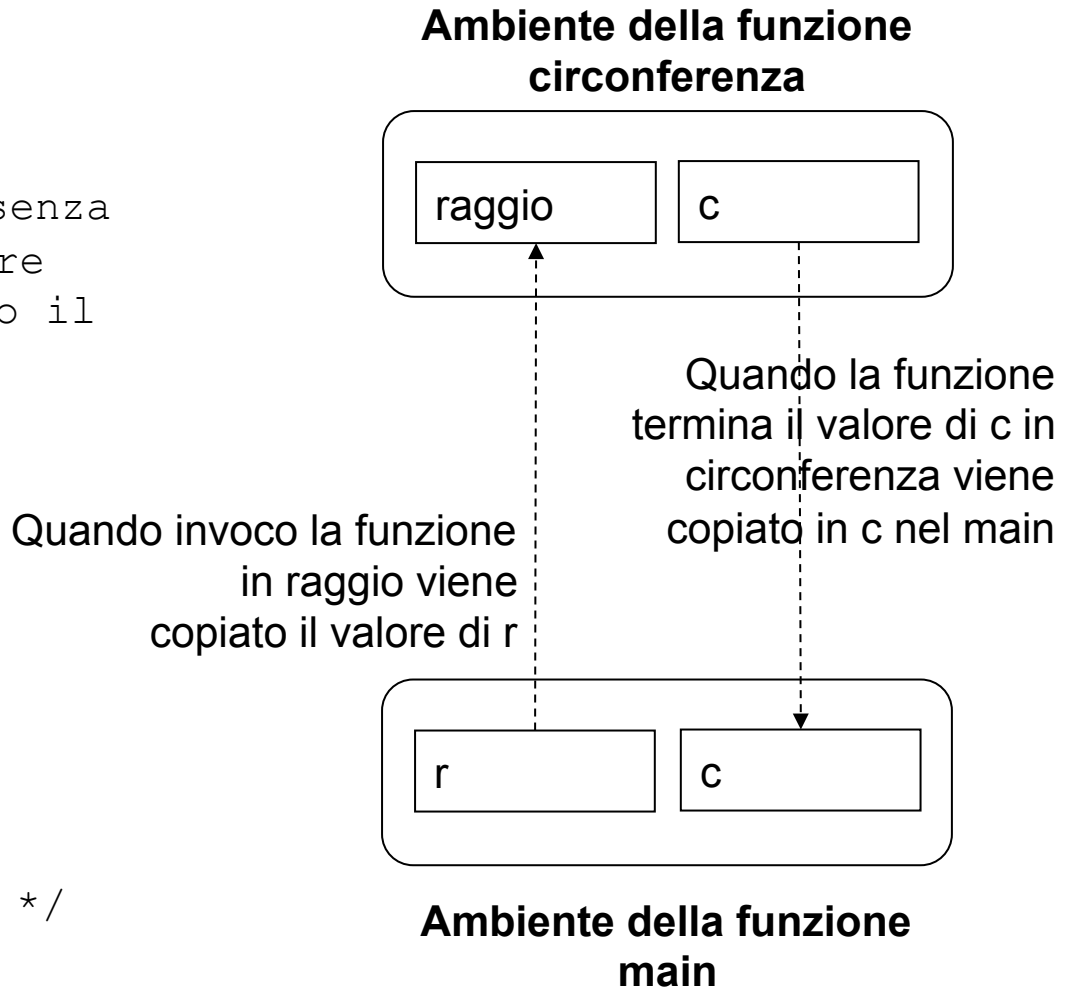
```
float circonferenza(float *raggio);
                              /*passaggio per indirizzo*/
```

Esempio: passaggio per valore

```
float circonferenza(float raggio)
   {
    float c;
    c = raggio * 3.14;
    raggio = 7; /*istruzione senza
       senso, voglio solo vedere
       cosa succede modificando il
       valore di un paramentro
       formale
    return c;
}

/* nel main */
float c,r=5;

c=circonferenza(r);
/*Attenzione! r vale sempre 5 */
```



Passaggio parametri per indirizzo

- Si utilizza:
  - il costruttore di tipo puntatore per la definizione dei parametri formali della funzione
  - l'operatore di deferenziazione all'interno della funzione
  - alla chiamata della funzione, si passa un indirizzo di variabile come parametro attuale
- Attenzione! Gli array sono SEMPRE passati per indirizzo. Una variabile di tipo array, infatti, è per definizione un puntatore

Esempio: passaggio per indirizzo

```
float circonferenza(float *raggio)
   {
    float c;
    c = *raggio * 3.14;
    *raggio = 7; /*istruzione senza
       senso, voglio solo vedere
       cosa succede modificando il
       valore di un paramentro
       formale
    return c;
}

/* nel main */
float c,r=5;

c=circonferenza(&r);
/*attenzione! D'ora in poi
r vale 7 */
```