Salvatore Cuomo

Il linguaggio C – funzioni e puntatori

Lezione n. 8

Parole chiave:
Linguaggio C, procedure, funzioni.

Corso di Laurea:
Informatica

Insegnamento:
Programmazione II, modulo di Laboratorio

Email Docente:
salvatore.cuomo@unina.it

A.A. 2009-2010

Problema.

scrivere una funzione di tipo **main** che richiami una funzione di tipo **int** denominata **pot_ennesima** che calcoli la potenza n-ma di assegnato numero intero.

Mostra codice

```c
#include <stdio.h>
/* dichiarazione della funzione power */
int pot_ennesima(int base, int n);

main()
{
 int i, potenza;
     for(i=0 ; i < 10 ; i++){
              potenza = pot_ennesima(2,i);
              printf("%d %d \n", i, potenza);
     }
}

/* funzione potenza */
int pot_ennesima(int base , int n)
{
    int i, p=1;
    for (i=1 ; i <=n ; i++)
     p=p*base;
return p;
}
```

Struttura di una funzione

Di seguito si riporta la specifica della funzione che si vuole costruire. Si osservi che prima di iniziare il **`main`** la funzione viene dichiarata secondo le proprie caratteristiche strutturale. Ovvero “in gergo” diremo che abbiamo scritto il prototipo della funzione.

```
int pot_ennesima(int base, int n)
```

1. **`int`** è il tipo del valore che la funzione deve ritornare
2. **`pot_emmesima`** è il nome della funzione ed memorizza il valore di ritorno
3. **`int base, int n`** rappresentano i parametri di ingresso, ovvero i valori di input, passati alla funzione. In questo esempio abbiamo due parametri interi in input.

La funzione

Un funzione viene richiamata attraverso una istruzione del tipo:

```
potenza = pot_ennesima(2,i);
```

nel programma di tipo `main` . L'inzio e la fine della funzione sono individuati dalla apertura e dalla chiusura di parentesi graffe

```
int pot_ennesima(int base, int n){
…
return p;
}
```

L'istruzione **`return p`** restituisce in output alla funzione chiamante main() i il valore di output p attraverso il nome della funzione.

N.B. Si osservi che se la funzine è di fissato tipo (ad esempio int) il valore ritornato (quello a fianco a return) deve essere dello stesso tipo (nel nostro caso int).

Problema.

scrivere una funzione di tipo **main** che richiami una funzione di tipo **int** denominata **scambio** che scambi il valore du due variabili di tipo float

Mostra codice

```c
#include <stdio.h>
int scambio(float x , float y);

/* funzione main */
main() {
  float a, b;
  a=3; b=5;
  scambio(a,b);
  printf("a=%f  b=%f \n", a, b);
}

/* funzione scambio */
  int scambio(float x, float y){
  float t;
  t=x; x=y;    y=t;
  return 0;
  }
```

Passaggio di parametri ad una funzione

Se si prova ad eseguire il precedente codice avviene che programma non dà il risultato atteso. Si ottiene infatti:

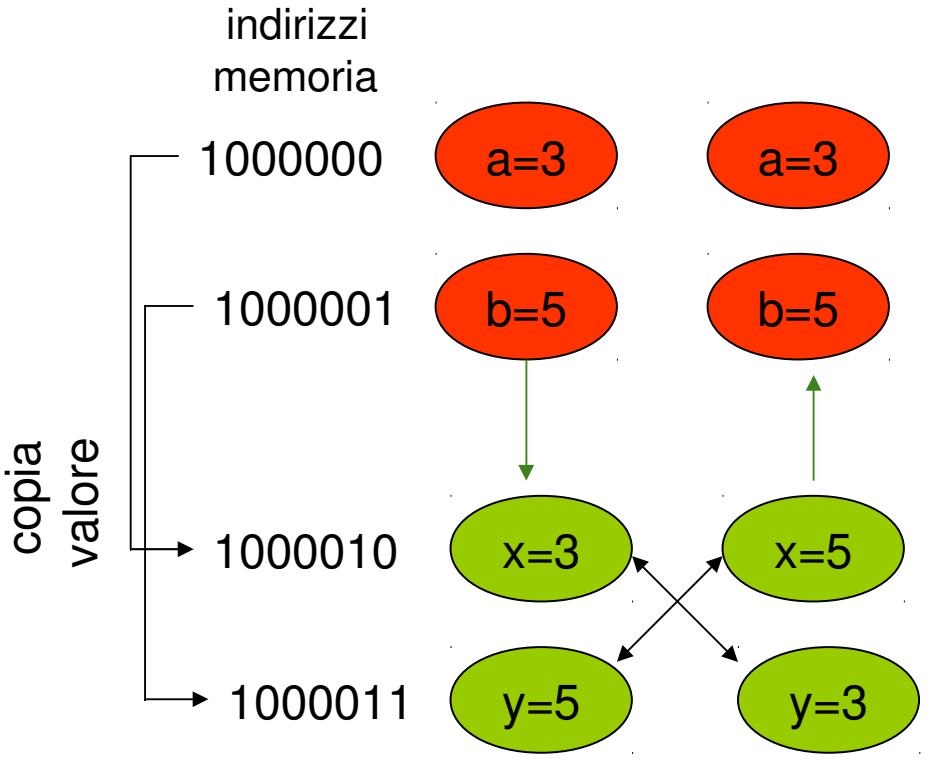
a=3 b=5

Quello che avviene è mostrato in figura.

Ovvero:

1. le variabili a e b in fissati indirizzi di memoria vengono "copiate" nelle variabili x e y che risiedono in differenti indirizzi
2. la funzione effettua lo scambio in memoria delle variabili
3. la funzione ritorna il controllo al main.

In C il passaggio dei parametri avviene per valore non per indirizzo di memoria



Passaggio di parametri ad una funzione

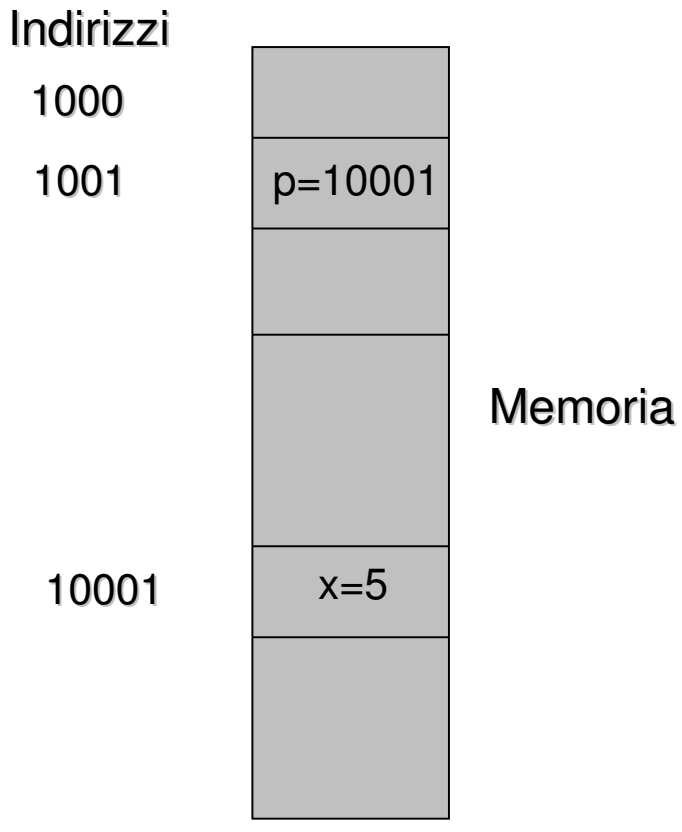
In C il passaggio dei parametri avviene per valore non per indirizzo di memoria

I Puntatori:

Nel linguaggio C esistono della “variabili speciali” dette **puntatori** che contengono l’indirizzi di memoria.

Nel esempio in figura si vede che la variabile p contiene l’indirizzo in memoria della variabile x. Potremo dire che p è una variabile che memorizza il numero civico dell’abitazione di x.

In generale, si dice che p punta ad x o anche che p è un puntatore per x.



p è un puntatore alla variabile x, ovvero p è una variabile che contiene l'indirizzo della variabile x.

L'istruzione

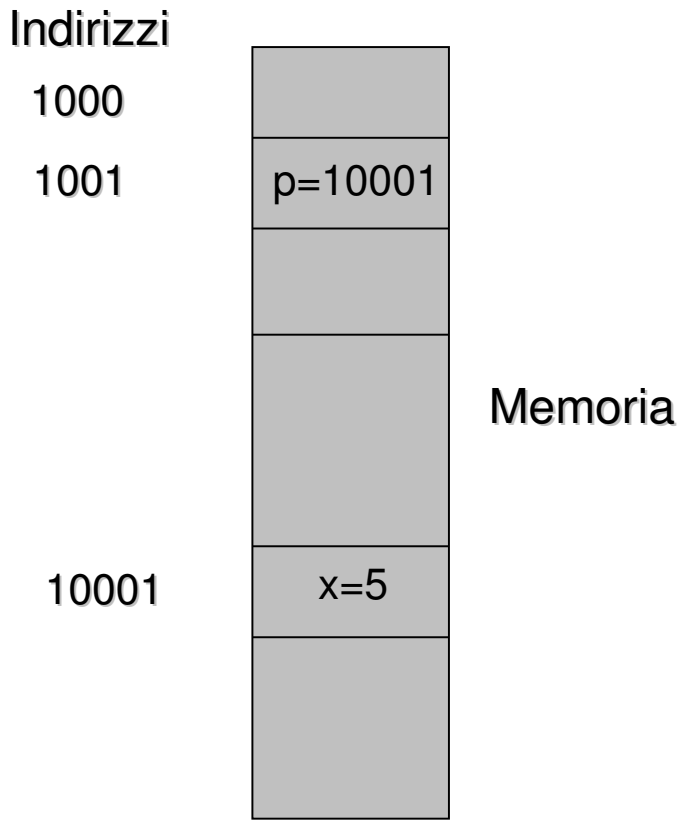
```
p = &x;
```

assegna l'indirizzo di x (nell'esempio 10001) alla variabile puntatore p da questo momento p punta a x.

Per conoscere il valore della variabile puntata da p (nell'esempio il valore di x che è 5) si utilizza l'operatore **unario** *. In particolare:

```
*p
```

è il contenuto dell'area di memoria puntata da p, cioè nel nostro esempio *p vale 5 ossia il valore della variabile x



Una variabile puntatore deve essere dichiarata necessariamente allo scopo a cui è adibita.

Ad esempio se si dichiara all'inizio di una funzione:

int *p;

p è una variabile puntatore ad una locazione di memoria di tipo intero.

ESEMPIO:

1. int x=5; /* dichiarazione della variabile x */
2. int *p; /* dichiarazione del puntatore p */
3. p=&x; /* assegnazione a p dell'indirizzo di x */
4. *p=3;/* assegnazione al contenuto della variabile puntata da p del valore 3*/

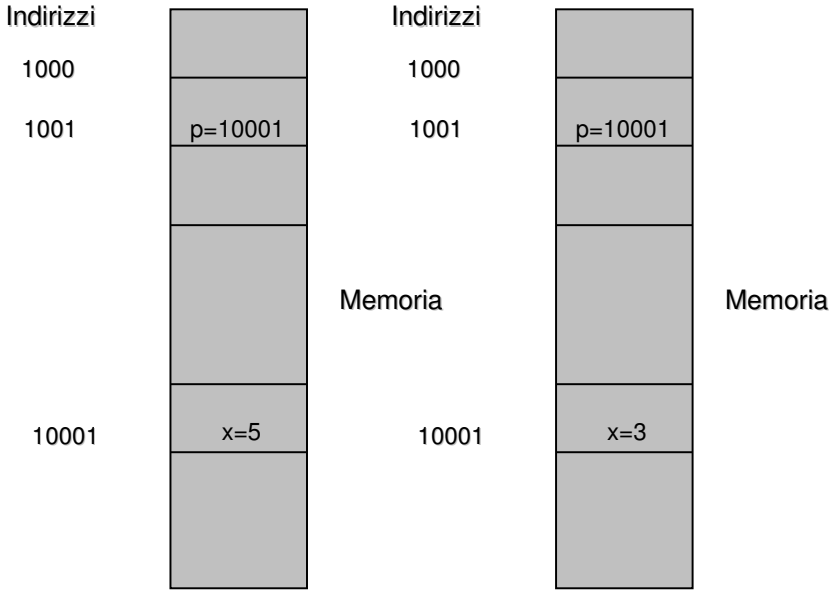


Figura: A sinistra situazione della memoria ai passi 1. 2. 3. dell'esempio. A destra cosa avviene con l'istruzione 4.

Esempio di utilizzo dei puntatori

Vediamo con qualche esempio come sia possibile manipolare dei puntatori

```
1.int x=1, y=2, z[10];
2.int *p;    /* dichiarazione del puntatore p */
3.p = &x;    /* p ora punta a x */
4.y = *p;    /* y ora vale 1 */
5.*p = 0;    /* x ora vale 0 */
6.p = &z[0]; /* p ora punta a z[0] */
```

Nell' esempio p è un puntatore utilizzato al punto 3. per individuare la variabile x

ed al punto 6. per puntare alla variabile z[0].

Esempio di utilizzo dei puntatori

I puntatori possono apparire nelle espressioni, ad esempio:

```
1.int x,y, *p;
2.p=&x;
3.y = *p +1; /* equivale a y = x+1; */
4.*p += 1;   /* equivale a x = x+1; */
5.*p = 10;   /* equivale a x=10; */
```

Se q è un altro puntatore ad una variabile intera, l'istruzione

```
q = p;
```

copia il contenuto di p in q. In base a quanto detto questo significa che q punta alla stessa variabile a cui punta p.

Esempio 2 versione corretta

Problema.

scrivere una funzione di tipo **main** che richiami una funzione di tipo **int** denominata **scambio** che scambi il valore due variabili di tipo float

Mostra codice

```c
#include <stdio.h>
int scambio(float *px , float *py);

/* funzione main */
main() {
  float a, b;
  a=3; b=5;
  scambio(&a,&b);
  printf("a=%f  b=%f \n", a, b);
}

/* funzione scambio */
  int scambio(float *px, float *py){
  float temp;
    temp = *px;
    *px = *py;
    *py = temp;
  return 0;
  }
```

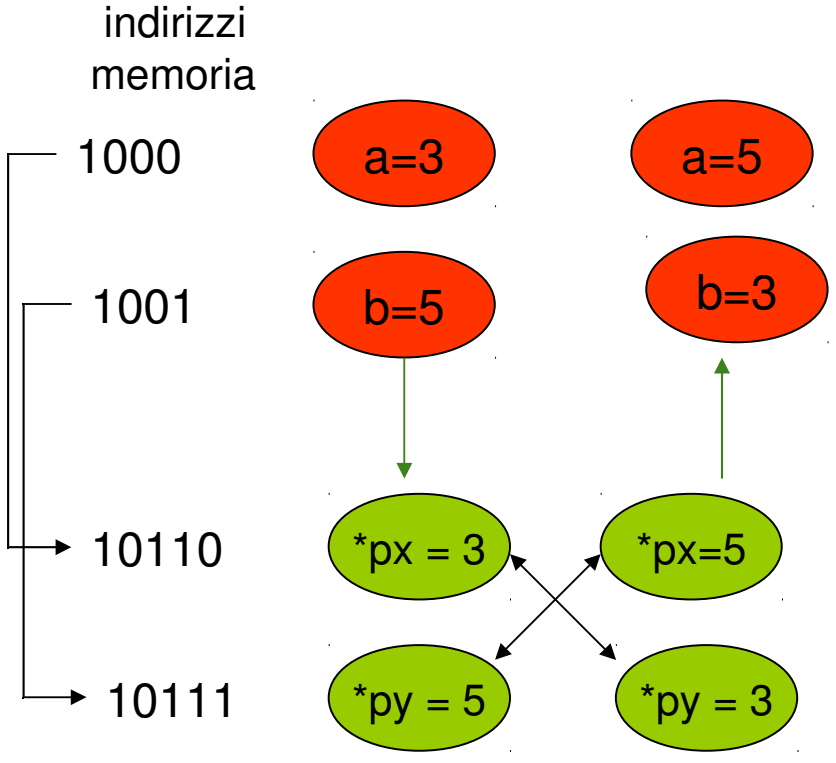
Passaggio di parametri ad una funzione per indirizzo

Se si prova ad eseguire il precedente codice con la chiamata:

scambio(&a, &b)

1. gli indirizzi delle variabili a e b vengono “copiati” nelle variabili px e py.
2. i puntatori px e py scambiano il loro valore ovvero gli indirizzi che contengono
3. la funzione ritorna il controllo al main.

In C il passaggio dei parametri per indirizzo si ottine utilizzando nella chiamata l’operatore &



Array

Esiste una stretta correlazione tra array e puntatori. Con l'istruzione:

```
int a[10], *pa;
```

Viene allocato un vettore di interi di lunghezza 10 ed un puntatore ad intero. Per puntare all'array è sufficiente l'struzione:

```
pa = &a[0];
```

```
int a[10];
```

E' possibile accedere agli elementi del vettore a attraverso il puntatore pa. Inoltre:

- pa+1 è l'indirizzo di a[1] mentre *(pa+1) è il contenuto di a[1]
- pa+i è l'indirizzo di a[i] mentre *(pa+i) è il contenuto di a[i]

La figura mostra schematicamente come pa punta ad a

Array

Per definizione il nome di un array è l'indirizzo del primo elemento dell'array.per cui nell'esempio il nome:

a

rappresenta l'indirizzo del vettore a[10], è dunque esso stesso un puntatore. Riportiamo alcuni esempi di utilizzo di array e puntatore

- a[i] equivalente *(a+i)
- &a[i] equivalente a+i
- pa = &a[0]; equivalente pa = a;
- *(pa+i) equivalente pa[i]

Nota

Si deve però tenere presente che:

1. un puntatore è una variabile;
2. il nome di un array è una costante.

Secondo tale osservazione si evince che :

```
pa=a;   pa++;
```

sono istruzioni consentite

Mentre

```
a=pa;    a++;
```

Non sono dellle istruzioni consentite.

Passaggio di un array ad una funzione.

Se il nome di un array è passato come argomento ad una funzione, in realtà è passato l'indirizzo del primo elemento dell'array. Per cui ad esempio si puo' ragionare nella seguente maniera:

```
main()
{
    int a[3];
     …
    fun(a);
     …
}
```

Il prototipo della funzione può essere strutturato in uno dei seguenti modi:

- `int fun(int arr[3]);`
- `int fun(int arr[]);`
- `int fun(int *arr);`

Passaggio di un array ad una funzione.

Se il nome di un array è passato come argomento ad una funzione, in realtà è passato l'indirizzo del primo elemento dell'array. Per cui ad esempio si puo' ragionare nella seguente maniera:

```
main()
{
    int a[3];
     …
    fun(a);
     …
}
```

Il prototipo della funzione può essere strutturato in uno dei seguenti modi:

- `int fun(int arr[3]);`
- `int fun(int arr[]);`
- `int fun(int *arr);`

Problema.

scrivere una funzione di tipo **main** che sommi due vettori di reali memorizzati secondo in strutture dati di tipo array statico.

Mostra codice

```c
#include <stdio.h>
main()
{
  float a[10],b[10],c[10];
  int i,n;
  scanf("%d",&n);
  for(i=0;i<n;i++)
  scanf("%f %f",&a[i],&b[i]);
  for(i=0;i<n;i++)
  {
    c[i]=a[i]+b[i];
    printf("%f\n",c[i]);
  }
}
```

Linguaggio C – esempio 3 versione puntatori

Problema.

scrivere una funzione di tipo **main** che sommi due vettori di reali memorizzati secondo in strutture dati di tipo array statico.

Mostra codice

```c
#include <stdio.h>
main()
{
  float a[10],b[10],c[10];
  int i,n;
  scanf("%d",&n);
  for(i=0;i<n;i++)
    scanf("%f %f",a+i,b+i);
  for(i=0;i<n;i++)
  {
    *(c+i)=*(a+i)+*(b+i);
    printf("%f\n",*(c+i));
  }
}
```

Sitografia essenziale

[1] Guida all’utilizzo dei puntatori **(visitato ottobre 2009)**

http://www.science.unitn.it/~fiorella/guidac/guidac025.html