



Capitolo 7

Puntatori e Riferimenti

pag. 193-219

Presenta: Prof. Misael Mongiovì

puntatori

- una variabile di tipo *puntatore al tipo x* contiene l'indirizzo di memoria di una variabile di *tipo x*

```
int n;
```

```
int* p; // p è variabile di tipo puntatore ad int
```

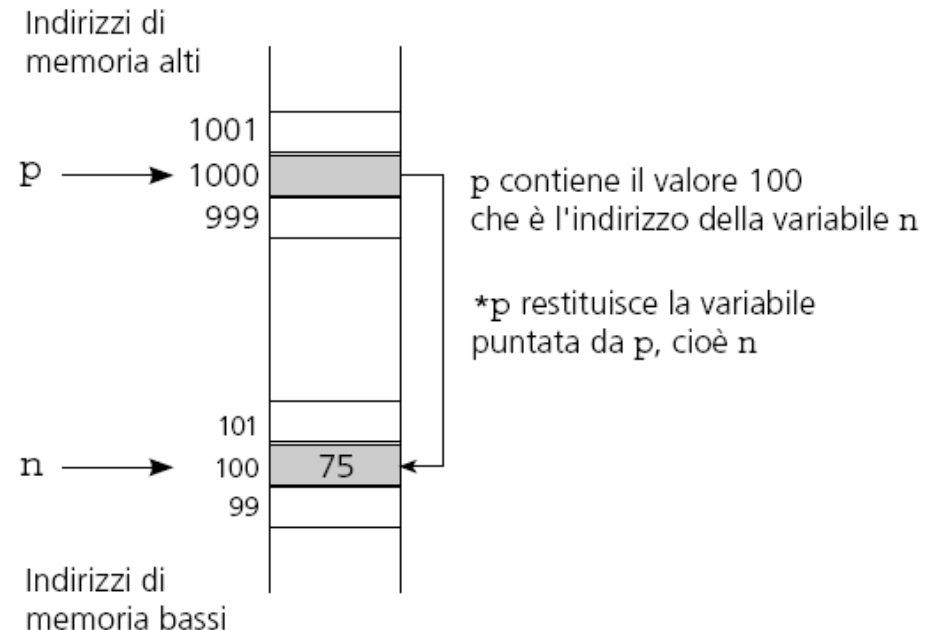
```
p = &n; // p contiene il valore dell'indirizzo di n
```

- dereferenziare* il puntatore significa andare alla variabile puntata partendo da un suo puntatore

operatore di **indirizione**

- ```
*p = 75;
```

  
// scrive 75 nella  
// variabile n





## operazioni coi puntatori

```
int i,j;
int* p;
int* q;
p = &i; // p contiene il valore dell'indirizzo di i
j = *p; // j contiene p che indirizza i e dunque j = 33
q = p; // assegno il puntatore p a al puntatore q
*p = 3; // *p è un left value, operazione di indirezione
*q = *&j; // equivale a i = j
*&j = 4; // equivale a j = 4
if (p == &i) cout << "p punta la variabile i";
if (q == p) cout << "q e p puntano la stessa variabile";
if (p != 0) cout << "p punta a qualche variabile ...";
if (p != NULL) cout << "... e quindi ...";
if (p) cout << "... si può dereferenziare";
cout << p; // stampa l'indirizzo in esadecimale
```

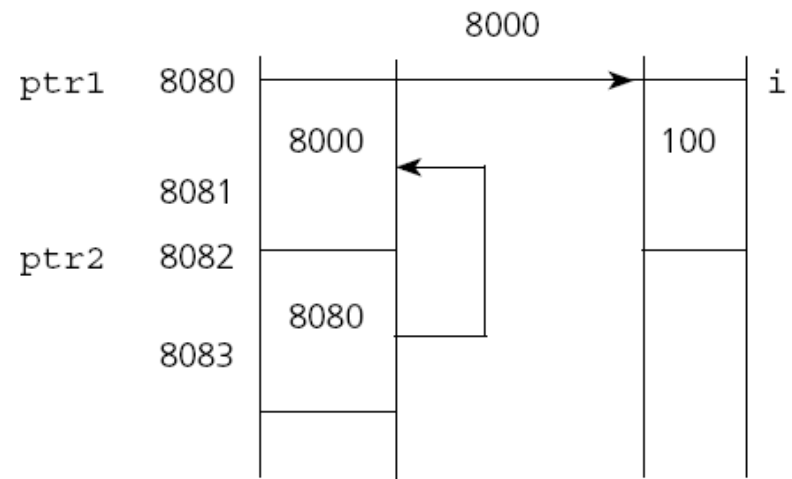


## puntatori a puntatori

- un tipo puntatore può puntare qualunque tipo, anche un altro puntatore

```
int i = 100;
int* ptr1 = &i;
int** ptr2 = &ptr1;
**ptr2 = 4; // equivale a i = 4
```

- `ptr1` è un puntatore ad interi e punta la variabile di tipo `int`  
`ptr2` è un puntatore a puntatore  
ad interi e punta la variabile `ptr1`



## passaggio di puntatori a funzioni



```
emac: scambia2.cpp
File Edit Mule Apps Options Buffers Tools C++ Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News

#include <iostream.h>
void scambia (int* a, int* b)
{ int t = *a;
 *a = *b; *b = t;
}
int main()
{ int x = 1, y = 2;
 scambia(&x, &y);
 cout << "x = " << x << " y = " << y << '\n';
 return 0;
}

ISO8--*-XEmacs: scambia2.cpp (C++ Font Abbrev)--
[root@localhost FondInfo]# ./a.out
x = 2 y = 1
[root@localhost FondInfo]#

ISO8--*-XEmacs: *shell* (Shell:run)----L20--Bot-
```

la funzione  
dereferenzia i  
puntatori

alla chiamata  
si passano gli  
indirizzi delle  
variabili come  
valore

## restituzione argomenti puntatori

- le funzioni possono restituire puntatori

```
int* max(int* pa, int* pb)
{
 if (*pa >= *pb) return pa;
 return pb;
}

int main()
{ int x, y;
 cin >> x >> y;
 *max(&x, &y) = 0;
 cout << "x = " << x << " y = " << y << '\n';
 return 0;
}
```



## puntatori costanti e puntatori a costante



- puntatore costante

- `<tipo_dato>* const <nome_puntatore> = <indirizzo_variabile>;`
  - `int x, e;`
  - `int* const p = &x;`
  - `*p = e; // corretto`
  - `p = &e // scorretto; p non puo' cambiar valore`

- puntatore a costante

- `const <tipo_dato>* <nome_puntatore> = <indirizzo_const>;`
  - `const int x = 25;`
  - `const int e = 50;`
  - `const int* p = &x;`
  - `*p = 15; // scorretto; ciò che p punta non puo' cambiar valore`
  - `p = &e // corretto`

## puntatori costanti e puntatori a costante – esempi



- Puntatori costanti: l'indirizzo non può essere modificato

```
int i, j;
```

```
int* const p = &i;
```

```
p = &j;
```

- Puntatori a costanti: il valore puntato non può essere modificato

```
int i, j;
```

```
const int* p = &i;
```

```
*p = 5;
```



## puntatori a oggetti non modificabili

- `const_cast`

```
int* maxp(const int* pa, const int* pb)
{
 if (*pa >= *pb) return const_cast<int*>(pa);
 return const_cast<int*>(pb);
}

int main()
{
 int x, y;
 cin >> x >> y;
 *maxp(&x, &y) = 0;
 cout << "x = " << x << " y = " << y << '\n';
 return 0;
}
```

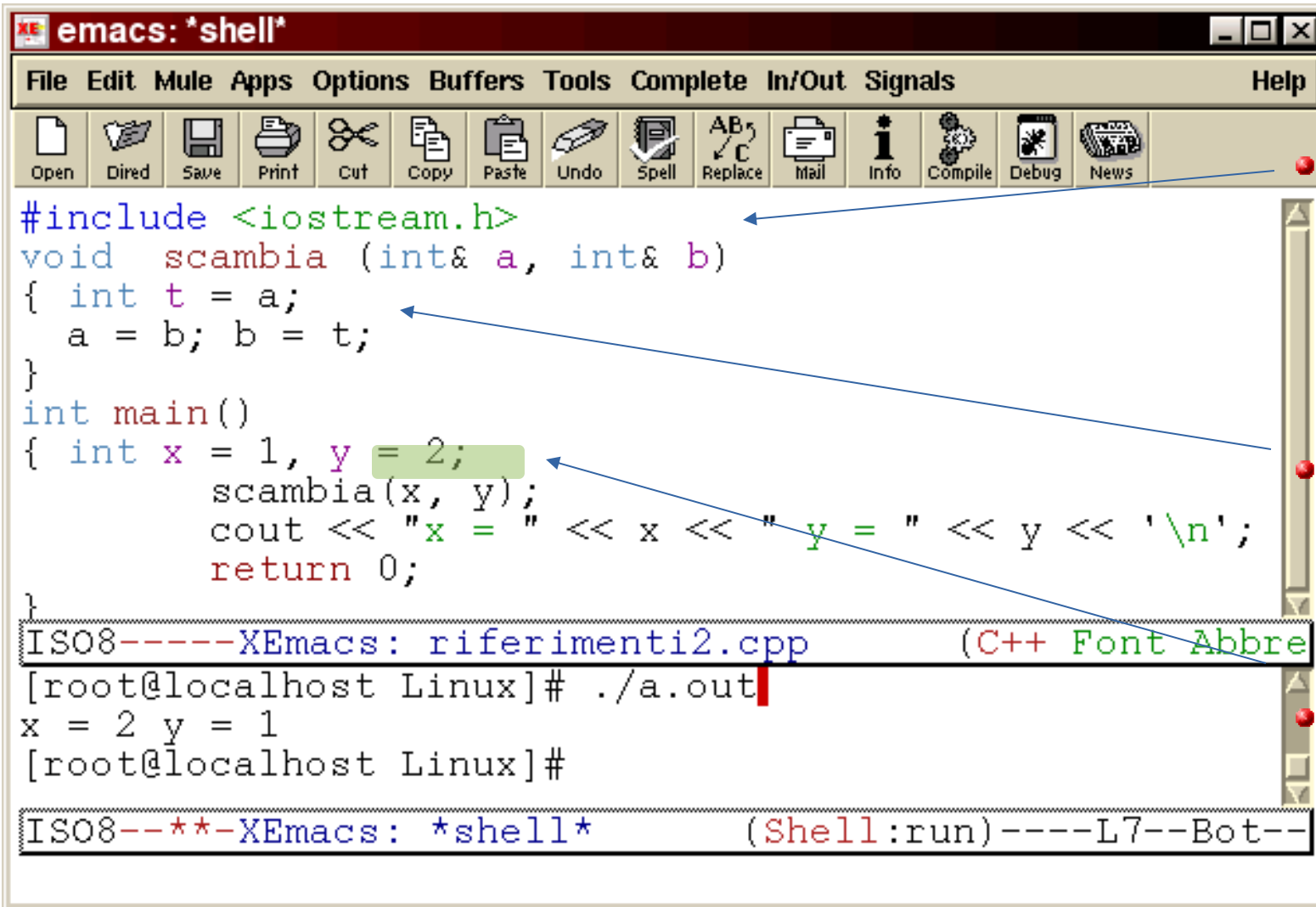
conversione di tipo costante → variabile



## riferimenti

- una variabile di tipo *riferimento al tipo x* è un *ulteriore nome* per una variabile del tipo x
  - `int n;`
  - `int& r = n;`
- una variabile di tipo “riferimento al tipo” deve essere inizializzata al momento della sua definizione
  - ```
void main()
{
    int n = 75;
    int& r = n;          // r è un riferimento per n
    cout << "n = " << n << ", r = " << r << endl;
    cout << "&n = " << &n << ", &r = " << &r << endl;
}
```
 - esecuzione:
`n = 75, r = 75`
`&n = 0x4fffd34, &r = 0x4fffd34`

passaggio di riferimenti a funzioni



```
emac: *shell*

File Edit Mule Apps Options Buffers Tools Complete In/Out Signals Help

Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News

#include <iostream.h>
void scambia (int& a, int& b)
{ int t = a;
  a = b; b = t;
}
int main()
{ int x = 1, y = 2;
  scambia(x, y);
  cout << "x = " << x << " y = " << y << '\n';
  return 0;
}

ISO8-----XEmacs: riferimenti2.cpp (C++ Font Abbre
[root@localhost Linux]# ./a.out
x = 2 y = 1
[root@localhost Linux]#

ISO8--*-XEmacs: *shell* (Shell:run)----L7--Bot--
```

i parametri
formali sono
dichiarati
come
riferimenti

la funzione
non
dereferenzia
puntatori

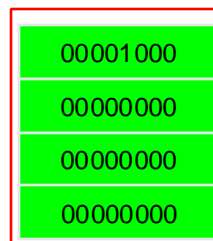
alla chiamata
si passano i
nomi delle
variabili

confronto passaggi

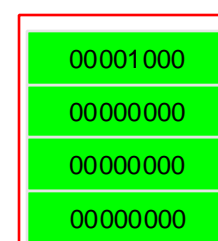
- passaggio per valore

```
int mia_funzione (int a) { ... }  
...  
int x = 8;  
...  
mia_funzione(x)
```

a



x



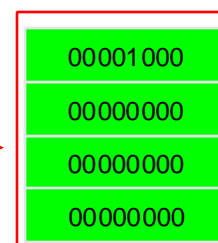
- passaggio per riferimento

```
int mia_funzione (int& a) { ... }  
...  
int x = 8;  
...  
mia_funzione(x)
```

a



x



restituzione argomenti riferimento

- le funzioni possono restituire riferimenti

```
int& max(int& ra, int& rb)
{
    if (ra >= rb) return ra;
    return rb;
}

int main()
{ int x, y;
  cin >> x >> y;
  max(x, y) = 0;
  cout << "x = " << x << " y = " << y << '\n';
  return 0;
}
```



restituzione argomenti riferimento



```
emacs: massimo.cpp
File Edit Mule Apps Options Buffers Tools C++ Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News

#include <iostream.h>
void max(int*& rpa, int* pb)
{ if (*rpa <= *pb) rpa = pb;
}
int main()
{
    int x, y;
    int* px = &x;
    int* py = &y;
    cout << "Immetti i due numeri \n";
    cin >> x >> y;
    cout << " x: " << x << " ind: " << &x << "\n";
    cout << "px: " << px << " ind: " << &px << "\n";
    cout << " y: " << y << " ind: " << &y << "\n";
    cout << "py: " << py << " ind: " << &py << "\n";
    max(px, py);
    cout << "Il maggiore è " << *px << '\n';
    cout << " x: " << x << " ind: " << &x << "\n";
    cout << "px: " << px << " ind: " << &px << "\n";
    cout << " y: " << y << " ind: " << &y << "\n";
    cout << "py: " << py << " ind: " << &py << "\n";
    return 0;
}

[root@localhost FondInfo]# ./a.out
Immetti i due numeri
345 234
x: 345 ind: 0xbffffb04
px: 0xbffffb04 ind: 0xbffffafc
y: 234 ind: 0xbffffb00
py: 0xbffffb00 ind: 0xbffffaf8
Il maggiore è 345
x: 345 ind: 0xbffffb04
px: 0xbffffb04 ind: 0xbffffafc
y: 234 ind: 0xbffffb00
py: 0xbffffb00 ind: 0xbffffaf8
[root@localhost FondInfo]# ./a.out
Immetti i due numeri
456 897
x: 456 ind: 0xbffffb04
px: 0xbffffb04 ind: 0xbffffafc
y: 897 ind: 0xbffffb00
py: 0xbffffb00 ind: 0xbffffaf8
Il maggiore è 897
x: 456 ind: 0xbffffb04
px: 0xbffffb00 ind: 0xbffffafc
y: 897 ind: 0xbffffb00
py: 0xbffffb00 ind: 0xbffffaf8
[root@localhost FondInfo]#
```



riferimento: dietro le quinte

- un riferimento è un puntatore costante, il cui nome non è accessibile al programmatore, che ha per valore l'indirizzo dell'oggetto riferito: l'operazione che coinvolge l'oggetto riferito viene realizzata effettuando un'indirizzazione sul puntatore nascosto
- i riferimenti risultano utili per il programmatore, ma non aggiungono alcuna potenzialità ai programmi rispetto all'uso dei puntatori



confronto fra puntatori e riferimenti

```
int* maxp(int* pa, int* pb)
{ if (*pa >= *pb) return pa;
  return pb;
}

int& maxr(int& ra, int& rb)
{ if (ra >= rb) return ra;
  return rb;
}

int main()
{ int x, y;
  . . .
  *maxp(&x, &y) = 0;
  maxp(&x, &y) = 0; // errore!!!
  y = *maxp(&x, &y);
  y = maxr(x, y);
  . . .
}
```




puntatori a funzioni

- non solo i dati, ma anche le istruzioni stanno in memoria a certi indirizzi; è possibile creare puntatori che puntino a funzioni, cioè al nome della funzione, che altro non è che un puntatore alla prima istruzione della funzione:
- *tipo_restituito (*PuntatoreFunzione) (argomenti);*
- `int f(int) { ... };` // definisce la funzione f
- `int (*pf)(int);` // definisce il puntatore pf alla funzione
- `pf = f;` // assegna l'indirizzo di f a pf

