

Capitolo 7 – I puntatori in C

Outline

Introduzione

Dichiarazione e inizializzazione dei puntatori

Gli operatori sui puntatori

Chiamata di funzioni per riferimento

Utilizzare il qualificatore const con i puntatori

Bubble Sort utilizzando la chiamata per riferimento

Espressioni puntatore e puntatori aritmetici

La relazione fra puntatori ed array

Array di puntatori

Un caso di studio: simulazione di un mescolatore e distributore di carte

Puntatori a funzioni

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Obiettivi

- In questo capitolo, impareremo a
 - Utilizzare i puntatori;
 - Utilizzare i puntatori per il passaggio di argomenti a funzioni utilizzando la chiamata per riferimento;
 - Capire la stretta relazione fra puntatori, array e stringhe;
 - Capire l'uso di puntatori a funzioni;
 - Definire ed utilizzare array di stringhe.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Introduzione

- Puntatori
 - Potenti, ma difficili da gestire
 - Simulazione del call-by-reference
 - Stretta relazione fra array e stringhe

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Dichiarazione e inizializzazione di puntatori

- Variabili puntatore
 - Contengono gli indirizzi di memoria come valore
 - Le normali variabili contengono uno specifico valore (riferimento diretto) `count`
`7`
 - I puntatori contengono gli indirizzi di una variabile che ha uno specifico valore (riferimento indiretto)
 - Referenziare – far riferimento al valore di un puntatore



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Dichiarazione e inizializzazione di puntatori

- Dichiarazione di puntatori

- * utilizzato con le variabili puntatore

```
int *myPtr;
```

- Definisce un puntatore ad un int (puntatore di tipo int *)
- Puntatori multipli richiedono l'uso di un * prima di ogni definizione di variabile

```
int *myPtr1, *myPtr2;
```

- Si possono definire puntatori ad ogni tipo di dato
- Inizializzare un puntatore a 0, NULL, o ad un indirizzo
 - 0 o NULL – puntano a niente (NULL è preferito)

Operatori su puntatori

- & (operatore di indirizzo)

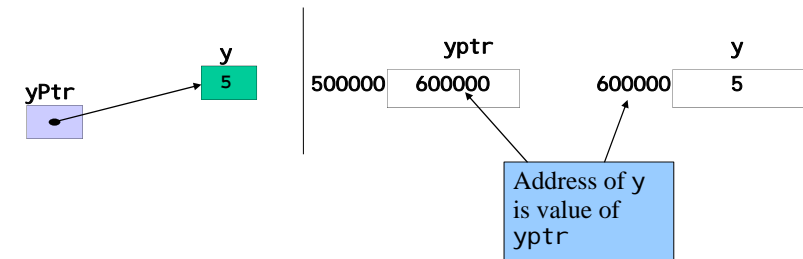
- Restituisce l'indirizzo dell'operando

```
int y = 5;
```

```
int *yPtr;
```

```
yPtr = &y; /* yPtr gets address of y */
```

```
yPtr "points to" y
```



Operatori su puntatori

- * (operatore di deriferimento o operatore di risoluzione del riferimento)

- Restituisce il valore dell'oggetto puntato dal suo operando
 - *yPtr restituisce y (poiché yPtr punta a y)
 - * può essere utilizzato per l'assegnamento
 - Restituisce l'alias ad un oggetto
- ```
yPtr = 7; / changes y to 7 */
```
- puntatore dereferenziato (operando di \*) deve essere un lvalue (non una costante)

- \* e & sono l'uno il complemento dell'altro

```
1 /* Fig. 7.4: fig07_04.c
2 Using the & and * operators */
3 #include <stdio.h>
4
5 int main()
6 {
7 int a; /* a is an integer */
8 int *aPtr; /* aPtr is a pointer to an integer */
9
10 a = 7;
11 aPtr = &a; /* aPtr set to address of a */
12
13 printf("The address of a is %p\n", &a);
14 printf("\nThe value of aPtr is %p, &a, aPtr);
15
16 printf("\n\nThe value of a is %d\n", a);
17 printf("\nThe value of *aPtr is %d, a, *aPtr);
18
19 printf("\n\nShowing that * and & are complements of "
20 "each other\n\n*aPtr = %p\n", &*aPtr, **aPtr);
21
22 return 0; /* indicates successful termination */
23 } /* end main */
```

The address of a is the value of aPtr.

The \* operator returns an alias to what its operand points to. aPtr points to a, so \*aPtr returns a.



Notice how \* and & are inverses

```
The address of a is 0012FF7C
The value of aPtr is 0012FF7C
```

```
The value of a is 7
The value of *aPtr is 7
```

```
Showing that * and & are complements of each other.
&*aPtr = 0012FF7C
*&aPtr = 0012FF7C
```

 [Outline](#)  
 **Program Output**

 [Outline](#)  


## Operatori sui puntatori

| Operators |    |    |    |    |    |   |        |  | Associativity | Type           |
|-----------|----|----|----|----|----|---|--------|--|---------------|----------------|
| ()        | [] |    |    |    |    |   |        |  | left to right | highest        |
| +         | -  | ++ | -- | !  | *  | & | (type) |  | right to left | unary          |
| *         | /  | %  |    |    |    |   |        |  | left to right | multiplicative |
| +         | -  |    |    |    |    |   |        |  | left to right | additive       |
| <         | <= | >  | >= |    |    |   |        |  | left to right | relational     |
| ==        | != |    |    |    |    |   |        |  | left to right | equality       |
| &&        |    |    |    |    |    |   |        |  | left to right | logical and    |
|           |    |    |    |    |    |   |        |  | left to right | logical or     |
| ?:        |    |    |    |    |    |   |        |  | right to left | conditional    |
| =         | += | -- | *= | /= | %= |   |        |  | right to left | assignment     |
| ,         |    |    |    |    |    |   |        |  | left to right | comma          |



Fig. 7.5 Operator precedence.

## Chiamata di funzioni per riferimento

- La chiamata per riferimento con argomenti puntatore
  - Si passa l'indirizzo dell'argomento utilizzando l'operatore &
  - Permette di modificare la reale locazione di memoria
  - Gli array non sono passati con & poiché il nome di un array è già un puntatore
- operatore \*
  - Utilizzato come alias/nickname per la variabile in una funzione
 

```
void raddoppia(int *number)
{
 *number = 2 * (*number);
}
```
  - \*number usato come nickname per la variabile passata

```
1 /* Fig. 7.6: fig07_06.c
2 Cube a variable using call-by-value */
3 #include <stdio.h>
4
5 int cubeByValue(int n); /* prototype */
6
7 int main()
8 {
9 int number = 5; /* Initialize number */
10
11 printf("The original value of number is %d", number);
12
13 /* pass number by value to cubeByValue */
14 number = cubeByValue(number);
15
16 printf("\nThe new value of number is %d\n", number);
17
18 return 0; /* Indicates successful termination */
19 } /* end main */
20
21
22 /* calculate and return cube of Integer argument */
23 int cubeByValue(int n)
24 {
25 return n * n * n; /* cube local variable n and return result */
26 } /* end function cubeByValue */
```

 [Outline](#)  
 **fig07\_06.c**

The original value of number is 5  
The new value of number is 125

Outline  
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1 /* Fig. 7.7: fig07_07.c
2 Cube a variable using call-by-reference with a pointer argument */
3
4 #include <stdio.h>
5
6 void cubeByReference(int *nPtr); /* prototype */
7
8 int main()
9 {
10 int number = 5; /* initialize number */
11
12 printf("The original value of number is %d", number);
13
14 /* pass address of number to cubeByReference */
15 cubeByReference(&number);
16
17 printf("\nThe new value of number is %d\n", number);
18
19 return 0; /* indicates successful termination */
20
21 } /* end main */
22
23 /* calculate cube of *nPtr; modifies variable number in main */
24 void cubeByReference(int *nPtr)
25 {
26 *nPtr = *nPtr * *nPtr * *nPtr; /* cube *nPtr */
27 } /* end function cubeByReference */
```

Outline  
fig07\_07.c

Notice that the function prototype takes a pointer to an integer.

Notice how the address of number is given - cubeByReference expects a pointer (an address of a variable).

Inside cubeByReference, \*nPtr is used (\*nPtr is number).

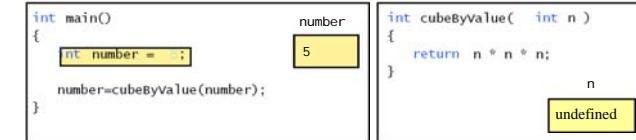
© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

The original value of number is 5  
The new value of number is 125

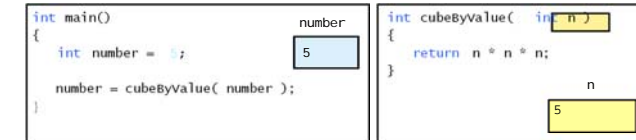
Outline  
Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

Before main calls cubeByValue :



After cubeByValue receives the call:



After cubeByValue cubes parameter n and before cubeByValue returns to main:

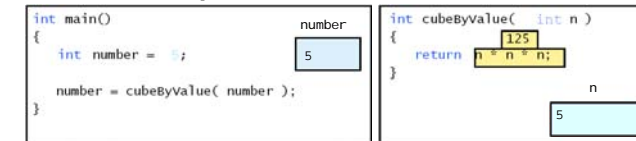
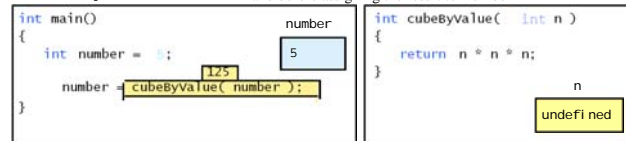


Fig. 7.8 Analysis of a typical call-by-value. (Part 1 of 2.)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

After cubeByValue returns to main and before assigning the result to number:



After main completes the assignment to number:

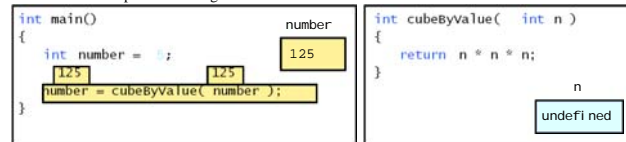
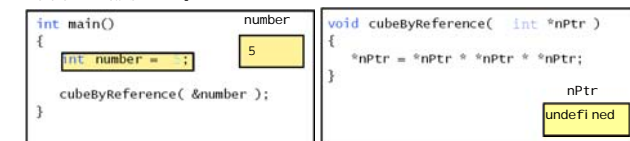
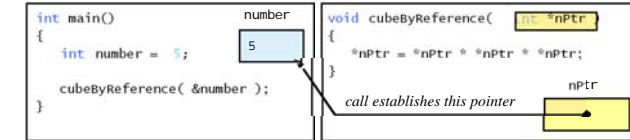


Fig. 7.8 Analysis of a typical call-by-value. (Part 2 of 2.)

Before main calls cubeByReference:



After cubeByReference receives the call and before \*nPtr is cubed:



After \*nPtr is cubed and before program control returns to main:

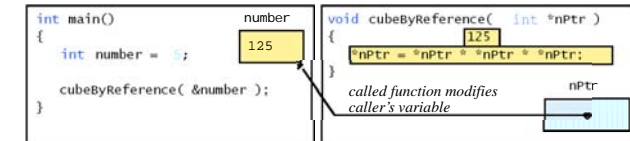


Fig. 7.9 Analysis of a typical call-by-reference with a pointer argument.



## 7.5 Usare il qualificatore const con i puntatori

- il qualificatore const
  - La variabile non può essere cambiata
  - Usare const se la funzione non necessita di modificare una variabile
  - Cercare di modificare una variabile const provoca un errore
- puntatori const
  - Puntano ad una locazione di memoria costante
  - Devono essere inizializzati quando definiti
  - `int *const myPtr = &x;`
    - Type `int *const` – puntatore costante ad un `int`
  - `const int *myPtr = &x;`
    - Puntatore generico a un `const int`
  - `const int *const Ptr = &x;`
    - `const` punta a `const int`
    - `x` può essere cambiato, ma non `*Ptr`

```

1 /* Fig. 7.10: fig07_10.c
2 Converting lowercase letters to uppercase letters
3 using a non-constant pointer to non-constant data */
4
5 #include <stdio.h>
6 #include <ctype.h>
7
8 void convertToUpper(char *sPtr); /* prototype */
9
10 int main()
11 {
12 char string[] = "characters and $32.98"; /* initialize char array */
13
14 printf("The string before conversion is: %s", string);
15 convertToUpper(string);
16 printf("\nThe string after conversion is: %s\n", string);
17
18 return 0; /* indicates successful termination */
19
20 } /* end main */
21

```

 [Outline](#)  
 **fig07\_10.c (Part 1 of 2)**


```

22 /* convert string to uppercase letters */
23 void convertToUpper(char *sPtr)
24 {
25 while (*sPtr != '\0') { /* current character is not '\0' */
26
27 if (!islower(*sPtr)) { /* if character is lowercase, */
28 *sPtr = toupper(*sPtr); /* convert to uppercase */
29 } /* end if */
30
31 ++sPtr; /* move sPtr to the next character */
32 } /* end while */
33
34 } /* end function convertToUpper */

```

The string before conversion is: characters and \$32.98  
The string after conversion is: CHARACTERS AND \$32.98

 Outline

 fig07\_10.c (Part 2 of 2)


Program Output

```

1 /* Fig. 7.11: fig07_11.c
2 Printing a string one character at a time using
3 a non-constant pointer to constant data */
4
5 #include <stdio.h>
6
7 void printCharacters(const char *sPtr);
8
9 int main()
10 {
11 /* Initialize char array */
12 char string[] = "print characters of a string";
13
14 printf("The string is:\n");
15 printCharacters(string);
16 printf("\n");
17
18 return 0; /* Indicates successful termination */
19
20 } /* end main */
21

```

 Outline

 fig07\_11.c (Part 1 of 2)


```

22 /* sPtr cannot modify the character to which it points,
23 i.e., sPtr is a "read-only" pointer */
24 void printCharacters(const char *sPtr)
25 {
26 /* Loop through entire string */
27 for (; *sPtr != '\0'; sPtr++) { /* no initialization */
28 printf("%c", *sPtr);
29 } /* end for */
30
31 } /* end function printCharacters */

```

The string is:  
print characters of a string

 Outline

 fig07\_11.c (Part 2 of 2)

Program Output

```

1 /* Fig. 7.12: fig07_12.c
2 Attempting to modify data through a
3 non-constant pointer to constant data. */
4
5 #include <stdio.h>
6
7 void f(const int *xPtr); /* prototype */
8
9 int main()
10 {
11 int y; /* define y */
12
13 f(&y); /* f attempts illegal modification */
14
15 return 0; /* Indicates successful termination */
16 } /* end main */
17
18 /* xPtr cannot be used to modify the
19 value of the variable to which it points */
20 void f(const int *xPtr)
21 {
22 *xPtr = 100; /* error: cannot modify a const object */
23 } /* end function f */


```

 Outline

 fig07\_12.c

```
Compiling...
FIG07_12.c
d:\books\2003\chtp4\examples\ch07\fig07_12.c(22) : error C2166: l-value
 specifies const object
Error executing cl.exe.

FIG07_12.exe - 1 error(s), 0 warning(s)
```

 [Outline](#)  
 **Program Output**

```
1 /* Fig. 7.13: fig07_13.c
2 Attempting to modify a constant pointer to non-constant data */
3 #include <stdio.h>
4
5 int main()
6 {
7 int x; /* define x */
8 int y; /* define y */
9
10 /* ptr is a constant pointer to an integer that can be modified
11 through ptr, but ptr always points to the same memory location */
12 int *const ptr = &x;
13
14 *ptr = 7; /* allowed: *ptr is not const */
15 ptr = &y; /* error: ptr is const; cannot assign new address */
16
17 return 0; /* indicates successful termination */
18
19 } /* end main */
```

Changing \*ptr is allowed – x is not a constant.

Changing ptr is an error – ptr is a constant pointer.

```
Compiling...
FIG07_13.c
D:\books\2003\chtp4\examples\ch07\FIG07_13.c(15) : error C2166: l-value
 specifies const object
Error executing cl.exe.



FIG07_13.exe - 1 error(s), 0 warning(s)
```

 [Outline](#)  
 **fig07\_13.c**  
**Program Output**

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1 /* Fig. 7.14: fig07_14.c
2 Attempting to modify a constant pointer to constant data. */
3 #include <stdio.h>
4
5 int main()
6 {
7 int x = 5; /* initialize x */
8 int y; /* define y */
9
10 /* ptr is a constant pointer to a constant integer. ptr always
11 points to the same location; the integer at that location
12 cannot be modified */
13 const int *const ptr = &x;
14
15 printf("%d\n", *ptr);
16
17 *ptr = 7; /* error: *ptr is const; cannot assign new value */
18 ptr = &y; /* error: ptr is const; cannot assign new address */
19
20 return 0; /* indicates successful termination */
21
22 } /* end main */
```

 [Outline](#)  
 **fig07\_14.c**

```
Compiling...
FIG07_14.c
D:\books\2003\chtp4\examples\ch07\FIG07_14.c(17) : error C2166: l-value
 specifies const object
D:\books\2003\chtp4\examples\ch07\FIG07_14.c(18) : error C2166: l-value
 specifies const object
Error executing cl.exe.

FIG07_14.exe - 2 error(s), 0 warning(s)
```

 [Outline](#)  
 **Program Output**

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Bubble Sort utilizzando la chiamata per riferimento

- Implementazione del bubblesort utilizzando i puntatori
  - Scambio di due elementi
  - La funzione swap deve ricevere l'indirizzo (usando &) di un array di elementi
  - Usando i puntatori e l'operatore \*, swap può scambiare elementi di un array

- Pseudo codice

*Initialize array*

*print data in original order*

*Call function bubblesort*

*print sorted array*

*Define bubblesort*

## Bubble Sort utilizzando la chiamata per riferimento

- si zeof
  - Restituisce la dimensione in byte dell'operando
  - Per gli array: size di 1 elemento \* numero di elementi
  - se si zeof( int ) è uguale a 4 byte, allora

```
int myArray[10];
printf("%d", si zeof(myArray));
```

    - stamperà 40
- si zeof può essere usato con
  - Nomi di variabili
  - Tipi di nomi
  - Valori costanti

```
1 /* Fig. 7.15: fig07_15.c
2 This program puts values into an array, sorts the values into
3 ascending order, and prints the resulting array. */
4 #include <stdio.h>
5 #define SIZE 10
6
7 void bubbleSort(int *array, const int size); /* prototype */
8
9 int main()
10 {
11 /* Initialize array a */
12 int a[SIZE] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
13
14 int i; /* counter */
15
16 printf("Data items in original order\n");
17
18 /* Loop through array a */
19 for (i = 0; i < SIZE; i++) {
20 printf("%4d", a[i]);
21 } /* end for */
22
23 bubbleSort(a, SIZE); /* sort the array */
24
25 printf("\nData items in ascending order\n");
26
```



Outline



fig07\_15.c (Part 1 of 3)

Bubble sort gets passed the address of array elements (pointers). The name of an array is a pointer.

```
27 /* Loop through array a */
28 for (i = 0; i < SIZE; i++) {
29 printf("%4d", a[i]);
30 } /* end for */
31
32 printf("\n");
33
34 return 0; /* Indicates successful termination */
35
36 } /* end main */
37
38 /* sort an array of integers using bubble sort algorithm */
39 void bubbleSort(int *array, const int size)
40 {
41 void swap(int *element1Ptr, int *element2Ptr); /* prototype */
42 int pass; /* pass counter */
43 int j; /* comparison counter */
44
45 /* Loop to control passes */
46 for (pass = 0; pass < size - 1; pass++) {
47
48 /* Loop to control comparisons during each pass */
49 for (j = 0; j < size - 1; j++) {
50
```



Outline



fig07\_15.c (Part 2 of 3)



```

51 /* swap adjacent elements if they are out of order */
52 if (array[j] > array[j + 1]) {
53 swap(&array[j], &array[j + 1]);
54 } /* end if */
55
56 } /* end inner for */
57
58 } /* end outer for */
59
60 } /* end function bubbleSort */
61
62 /* swap values at memory locations to which element1Ptr and
63 element2Ptr point */
64 void swap(int *element1Ptr, int *element2Ptr)
65 {
66 int hold = *element1Ptr;
67 *element1Ptr = *element2Ptr;
68 *element2Ptr = hold;
69 } /* end function swap */


```

```

Data items in original order
2 6 4 8 10 12 89 68 45 37
Data items in ascending order
2 4 6 8 10 12 37 45 68 89

```

 Outline

 fig07\_15.c (Part 3 of 3)

Program Output

```

1 /* Fig. 7.16: fig07_16.c
2 sizeof operator when used on an array name
3 returns the number of bytes in the array. */
4 #include <stdio.h>
5
6 size_t getSize(float *ptr); /* prototype */
7
8 int main()
9 {
10 float array[20]; /* create array */
11
12 printf("The number of bytes in the array is %d"
13 "\nThe number of bytes returned by getSize is %d\n",
14 sizeof(array), getSize(array));
15
16 return 0; /* indicates successful termination */
17
18 } /* end main */
19
20 /* return size of ptr */
21 size_t getSize(float *ptr)
22 {
23 return sizeof(ptr);
24
25 } /* end function getSize */

```

```

The number of bytes in the array is 80
The number of bytes returned by getSize is 4

```

 Outline

 fig07\_16.c


Program Output

```


1 /* Fig. 7.17: fig07_17.c
2 Demonstrating the sizeof operator */
3 #include <stdio.h>
4
5 int main()
6 {
7 char c; /* define c */
8 short s; /* define s */
9 int i; /* define i */
10 long l; /* define l */
11 float f; /* define f */
12 double d; /* define d */
13 long double ld; /* define ld */
14 int array[20]; /* initialize array */
15 int *ptr = array; /* create pointer to array */
16
17 printf(" sizeof c = %d\\sizeof(char) = %d"
18 "\\n sizeof s = %d\\sizeof(short) = %d"
19 "\\n sizeof i = %d\\sizeof(int) = %d"
20 "\\n sizeof l = %d\\sizeof(long) = %d"
21 "\\n sizeof f = %d\\sizeof(float) = %d"
22 "\\n sizeof d = %d\\sizeof(double) = %d"
23 "\\n sizeof ld = %d\\sizeof(long double) = %d"
24 "\\n sizeof array = %d"
25 "\\n sizeof ptr = %d\\n",

```

 Outline

 fig07\_17.c (Part 1 of 2)

 Outline

 fig07\_17.c (Part 2 of 2)

```

26 sizeof c, sizeof(char), sizeof s,
27 sizeof(short), sizeof(int),
28 sizeof l, sizeof(long), sizeof f,
29 sizeof(float), sizeof d, sizeof(double),
30 sizeof ld, sizeof(long double),
31 sizeof array, sizeof ptr);
32
33 return 0; /* indicates successful termination */
34
35 } /* end main */

```

```

sizeof c = 1 sizeof(char) = 1
sizeof s = 2 sizeof(short) = 2
sizeof i = 4 sizeof(int) = 4
sizeof l = 4 sizeof(long) = 4
sizeof f = 4 sizeof(float) = 4
sizeof d = 8 sizeof(double) = 8
sizeof ld = 8 sizeof(long double) = 8
sizeof array = 80
sizeof ptr = 4

```

Program Output

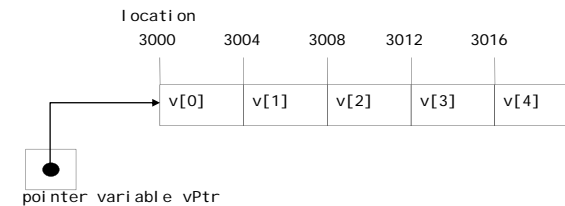
## Espressioni puntatore e puntatori aritmetici

- Si possono effettuare delle operazioni aritmetiche sui puntatori
  - Incremento/decremento di puntatore ( ++ o -- )
  - Aggiungere un intero ad un puntatore ( + o += , - o -= )
  - Un puntatore può essere sottratto da un altro

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Espressioni puntatore e puntatori aritmetici

- 5 elementi di un array di `int` su una macchina a 4 byte `ints`
  - `vPtr` punta al primo elemento `v[ 0 ]`
    - Alla locazione 3000 (`vPtr = 3000`)
  - `vPtr += 2;` setta `vPtr` a 3008
    - `vPtr` punta a `v[ 2 ]` (incrementato di 2), ma la macchina ha 4 byte `ints`, quindi punta all'indirizzo 3008



© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Espressioni puntatore e puntatori aritmetici

- Sottrarre i puntatori
  - Restituisce il numero di elementi. Se  
`vPtr2 = v[ 2 ];`  
`vPtr = v[ 0 ];`
    - `vPtr2 - vPtr` dovrebbe produrre 2
- Confronto di puntatori ( `<`, `==`, `>` )
  - Controlla quale puntatore punta all'elemento più alto dell'array
  - Controlla se un puntatore punta a 0

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Espressioni puntatore e puntatori aritmetici

- Puntatori dello stesso tipo possono essere assegnati l'uno all'altro
  - Se non sono dello stesso tipo bisogna usare un operatore di cast
  - Eccezione: puntatore a void (type void \*)
    - Puntatore generico, rappresenta qualsiasi tipo
    - Non c'è bisogno di casting per convertire un puntatore a un puntatore a void
    - I puntatori void non possono essere dereferenziati

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Relazione fra puntatori ed array

- Gli array e i puntatori sono fortemente correlati
  - Nome di array come puntatore costante
  - I puntatori possono accedere agli elementi di un array
- Definire un array `b[ 5 ]` ed un puntatore `bPtr`
  - Per settarli uguali l'uno all'altro:  
`bPtr = b;`
    - Il nome dell'array (`b`) è in realtà l'indirizzo del primo elemento dell'array `b[ 5 ]`  
`bPtr = &b[ 0 ]`
    - Assegna in modo esplicito `bPtr` all'indirizzo del primo elemento di `b`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Relazione fra puntatori ed array

- L'elemento `b[ 3 ]`
  - si può accedere con `*( bPtr + 3 )`
    - dove `n` è l'. Detta pointer/offset notation
  - si può accedere con `bPtr[ 3 ]`
    - Detta pointer/subscript notation
    - `bPtr[ 3 ]` come `b[ 3 ]`
  - si può accedere effettuando un'operazione aritmetica sull'operatore  
`*( b + 3 )`

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```
1 /* Fig. 7.20: fig07_20.cpp
2 Using subscripting and pointer notations w th arrays */
3
4 #include <stdio.h>
5
6 int main()
7 {
8 int b[] = { 10, 20, 30, 40 }; /* initialize array b */
9 int *bPtr = b; /* set bPtr to point to array b */
10 int i; /* counter */
11 int offset; /* counter */
12
13 /* output array b using array subscript notation */
14 printf("Array b printed w th:\nArray subscript notation\n");
15
16 /* loop through array b */
17 for (i = 0; i < 4; i++) {
18 printf("b[%d] = %d\n", i, b[i]);
19 } /* end for */
20
21 /* output array b using array name and pointer/offset notation */
22 printf("\nPointer/offset notation where\n"
23 "the pointer is the array name\n");
24
```



### Outline



fig07\_20.c (Part 1 of 2)

```
25 /* loop through array b */
26 for (offset = 0; offset < 4; offset++) {
27 printf("*(b + %d) = %d\n", offset, *(b + offset));
28 } /* end for */
29
30 /* output array b using bPtr and array subscript notation */
31 printf("\nPointer subscript notation\n");
32
33 /* loop through array b */
34 for (i = 0; i < 4; i++) {
35 printf("bPtr[%d] = %d\n", i, bPtr[i]);
36 } /* end for */
37
38 /* output array b using bPtr and pointer/offset notation */
39 printf("\nPointer/offset notation\n");
40
41 /* loop through array b */
42 for (offset = 0; offset < 4; offset++) {
43 printf("*(bPtr + %d) = %d\n", offset, *(bPtr + offset));
44 } /* end for */
45
46 return 0; /* indicates successful termination */
47
48 } /* end main */
```



### Outline



fig07\_20.c (Part 2 of 2)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```

Array b printed with:
Array subscript notation
b[0] = 10
b[1] = 20
b[2] = 30
b[3] = 40

```

```

Pointer/offset notation where
the pointer is the array name
*(b + 0) = 10
*(b + 1) = 20
*(b + 2) = 30
*(b + 3) = 40

```

```


Pointer subscript notation
bPtr[0] = 10
bPtr[1] = 20
bPtr[2] = 30
bPtr[3] = 40

```

```

Pointer/offset notation
*(bPtr + 0) = 10
*(bPtr + 1) = 20
*(bPtr + 2) = 30
*(bPtr + 3) = 40

```



 [Outline](#)  
 **Program Output**

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```

1 /* Fig. 7.21: fig07_21.c
2 Copying a string using array notation and pointer notation. */
3 #include <stdio.h>
4
5 void copy1(char *s1, const char *s2); /* prototype */
6 void copy2(char *s1, const char *s2); /* prototype */
7
8 int main()
9 {
10 char string1[10]; /* create array string1 */
11 char *string2 = "Hello"; /* create a pointer to a string */
12 char string3[10]; /* create array string3 */
13 char *string4 = "Good Bye"; /* create a pointer to a string */
14
15 copy1(string1, string2);
16 printf("string1 = %s\n", string1);
17
18 copy2(string3, string4);
19 printf("string3 = %s\n", string3);
20
21 return 0; /* indicates successful termination */
22
23 } /* end main */
24

```

 [Outline](#)  
 **fig07\_21.c (Part 1 of 2)**

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```



25 /* copy s2 to s1 using array notation */
26 void copy1(char *s1, const char *s2)
27 {
28 int i; /* counter */
29
30 /* loop through strings */
31 for (i = 0; (s1[i] = s2[i]) != '\0'; i++) {
32 ; /* do nothing in body */
33 } /* end for */
34
35 } /* end function copy1 */
36
37 /* copy s2 to s1 using pointer notation */
38 void copy2(char *s1, const char *s2)
39 {
40 /* loop through strings */
41 for (; (*s1 = *s2) != '\0'; s1++, s2++) {
42 ; /* do nothing in body */
43 } /* end for */
44
45 } /* end function copy2 */

```

```

string1 = Hello
string3 = Good Bye

```

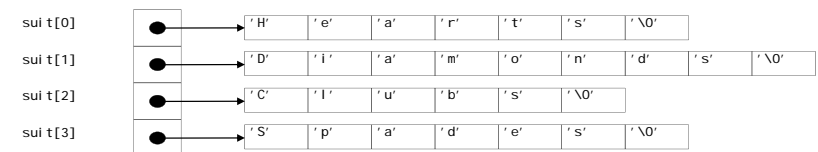
 [Outline](#)  
 **fig07\_21.c (Part 2 of 2)**

**Program Output**

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Array di puntatori

- Gli arrays possono contenere puntatori
- Per esempio: un array di stringhe
  - char \*sui t[ 4 ] = { "Hearts", "Diamonds", "Clubs", "Spades" };
    - Le stringhe sono dei puntatori al primo carattere
    - char \* – ogni elemento di sui t è un puntatore a char
    - Le stringhe non sono in realtà memorizzate nell'array sui t, sono memorizzate solo i puntatori alle stringhe

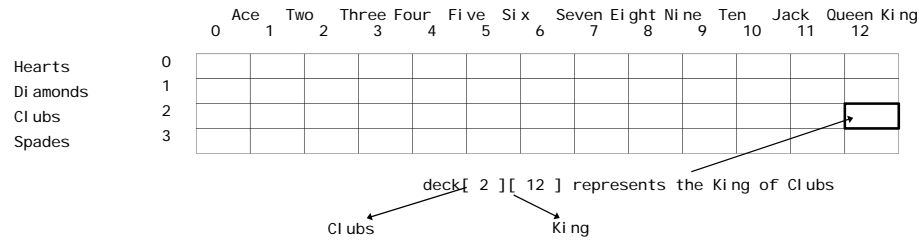


- l'array sui t ha una dimensione fissa, ma le stringhe possono essere di qualsiasi dimensione

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Simulazione di mescolatore e distributore di carte

- Card shuffling program
  - Use array of pointers to strings
  - Use double scripted array (suit, face)



- The numbers 1-52 go into the array
  - Representing the order in which the cards are dealt

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Simulazione di mescolatore e distributore di carte

- Pseudocode
  - Top level:
    - Shuffle and deal 52 cards*
  - First refinement:
    - Initialize the suit array*
    - Initialize the face array*
    - Initialize the deck array*
    - Shuffle the deck*
    - Deal 52 cards*

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Simulazione di mescolatore e distributore di carte

- Second refinement
  - Convert *shuffle the deck* to
    - For each of the 52 cards*
    - Place card number in randomly selected unoccupied slot of deck*
  - Convert *deal 52 cards* to
    - For each of the 52 cards*
    - Find card number in deck array and print face and suit of card*

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

## Simulazione di mescolatore e distributore di carte

- Third refinement
  - Convert *shuffle the deck* to
    - Choose slot of deck randomly*
    - While chosen slot of deck has been previously chosen*
    - Choose slot of deck randomly*
    - Place card number in chosen slot of deck*
  - Convert *deal 52 cards* to
    - For each slot of the deck array*
    - If slot contains card number*
    - Print the face and suit of the card*

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```

1 /* Fig. 7.24: fig07_24.c
2 Card shuffling dealing program */
3 #include <stdio.h>
4 #include <stdlib.h>
5 #include <time.h>
6
7 /* prototypes */
8 void shuffle(int wDeck[][13]);
9 void deal (const int wDeck[][13], const char *wFace[],
10 const char *wsuit[]);
11
12 int main()
13 {
14 /* initialize suit array */
15 const char *suit[4] = { "Hearts", "Diamonds", "Clubs", "Spades" };
16
17 /* initialize face array */
18 const char *face[13] =
19 { "Ace", "Deuce", "Three", "Four",
20 "Five", "Six", "Seven", "Eight",
21 "Nine", "Ten", "Jack", "Queen", "King" };
22
23 /* initialize deck array */
24 int deck[4][13] = { 0 };
25

```



## Outline



fig07\_24.c (Part 1 of 4)

```

26 srand(time(0)); /* seed random-number generator */
27
28 shuffle(deck);
29 deal (deck, face, suit);
30
31 return 0; /* indicates successful termination */
32
33 } /* end main */
34
35 /* shuffle cards in deck */
36 void shuffle(int wDeck[][13])
37 {
38 int row; /* row number */
39 int column; /* column number */
40 int card; /* counter */
41
42 /* for each of the 52 cards, choose slot of deck randomly */
43 for (card = 1; card <= 52; card++) {
44
45 /* choose new random location until unoccupied slot found */
46 do {
47 row = rand() % 4;
48 column = rand() % 13;
49 } while(wDeck[row][column] != 0); /* end do...while */
50

```



## Outline



fig07\_24.c (Part 2 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```

51 /* place card number in chosen slot of deck */
52 wDeck[row][column] = card;
53 } /* end for */
54
55 } /* end function shuffle */
56
57 /* deal cards in deck */
58 void deal (const int wDeck[][13], const char *wFace[],
59 const char *wsuit[])
60 {
61 int card; /* card counter */
62 int row; /* row counter */
63 int column; /* column counter */
64
65 /* deal each of the 52 cards */
66 for (card = 1; card <= 52; card++) {
67
68 /* loop through rows of wDeck */
69 for (row = 0; row <= 3; row++) {
70
71 /* loop through columns of wDeck for current row */
72 for (column = 0; column <= 12; column++) {
73
74 /* if slot contains current card, display card */
75 if (wDeck[row][column] == card) {

```



## Outline



fig07\_24.c (Part 3 of 4)

```

76 printf("%5s of %-8s%c", wFace[column], wsuit[row],
77 card % 2 == 0 ? '\n' : '\t');
78 } /* end if */
79
80 } /* end for */
81
82 } /* end for */
83
84 } /* end for */
85
86 } /* end function deal */

```



## Outline





fig07\_24.c (Part 4 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

|                   |                   |
|-------------------|-------------------|
| Nine of Hearts    | Five of Clubs     |
| Queen of Spades   | Three of Spades   |
| Queen of Hearts   | Ace of Clubs      |
| King of Hearts    | Six of Spades     |
| Jack of Diamonds  | Five of Spades    |
| Seven of Hearts   | King of Clubs     |
| Three of Clubs    | Eight of Hearts   |
| Three of Diamonds | Four of Diamonds  |
| Queen of Diamonds | Five of Diamonds  |
| Six of Diamonds   | Five of Hearts    |
| Ace of Spades     | Six of Hearts     |
| Nine of Diamonds  | Queen of Clubs    |
| Eight of Spades   | Nine of Clubs     |
| Deuce of Clubs    | Six of Clubs      |
| Deuce of Spades   | Jack of Clubs     |
| Four of Clubs     | Eight of Clubs    |
| Four of Spades    | Seven of Spades   |
| Seven of Diamonds | Seven of Clubs    |
| King of Spades    | Ten of Diamonds   |
| Jack of Hearts    | Ace of Hearts     |
| Jack of Spades    | Ten of Clubs      |
| Eight of Diamonds | Deuce of Diamonds |
| Ace of Diamonds   | Nine of Spades    |
| Four of Hearts    | Deuce of Hearts   |
| King of Diamonds  | Ten of Spades     |
| Three of Hearts   | Ten of Hearts     |

 [Outline](#)  
 [Program Output](#)

## Puntatori a funzioni

- Puntatore a funzione
  - Contiene l'indirizzo della funzione
  - Simile a come il nome dell'array è l'indirizzo del primo elemento
  - Il nome della funzione è l'indirizzo iniziale del codice che definisce la funzione
- I puntatori a funzione possono essere
  - passati a funzioni
  - memorizzati in array
  - assegnati ad altri puntatori a funzioni

## Puntatori a funzioni

- Esempio: bubblesort
  - La funzione bubble ha in input un puntatore a funzione
    - bubble chiama questa helper function
    - ciò determina l'ordinamento ascendente e discendente
  - L'argomento in bubblesort per il puntatore a funzione:
 

```
int (*compare)(int a, int b)
```

 dice a bubblesort di aspettarsi un puntatore ad una funzione che ha due ints come input e restituisce un int
  - Se non vengono usate le parentesi:
 



```
int *compare(int a, int b)
```

 • Definisce una funzione che riceve due interi come input e restituisce un puntatore a int

```

1 /* Fig. 7.26: fig07_26.c
2 Multi purpose sorting program using function pointers */
3 #include <stdio.h>
4 #define SIZE 10
5
6 /* prototypes */
7 void bubble(int work[], const int size, int (*compare)(int a, int b));
8 int ascending(int a, int b);
9 int descending(int a, int b);
10
11 int main()
12 {
13 int order; /* 1 for ascending order or 2 for descending order */
14 int counter; /* counter */
15
16 /* Initialize array a */
17 int a[SIZE] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19 printf("Enter 1 to sort in ascending order,\n"
20 "Enter 2 to sort in descending order: ");
21 scanf("%d", &order);
22
23 printf("\nData Items in original order\n");
24

```

 [Outline](#)  
 [fig07\\_26.c \(Part 1 of 4\)](#)

```

25 /* output original array */
26 for (counter = 0; counter < SIZE; counter++) {
27 printf("%5d", a[counter]);
28 } /* end for */
29
30 /* sort array in ascending order; pass functi on ascending as an
31 argument to specifi y ascending sorting order */
32 if (order == 1) {
33 bubble(a, SIZE, ascending);
34 printf("\nData Items in ascending order\n");
35 } /* end if */
36 else { /* pass functi on descending */
37 bubble(a, SIZE, descending);
38 printf("\nData Items in descending order\n");
39 } /* end else */
40
41 /* output sorted array */
42 for (counter = 0; counter < SIZE; counter++) {
43 printf("%5d", a[counter]);
44 } /* end for */
45
46 printf("\n");
47
48 return 0; /* indicates successful termination */
49
50 } /* end main */
51

```



## Outline

fig07\_26.c (Part 2 of 4)

```

52 /* multipurpose bubble sort; parameter compare is a pointer to
53 the comparison function that determines sorting order */
54 void bubble(int work[], const int size, int (*compare)(int a, int b))
55 {
56 int pass; /* pass counter */
57 int count; /* comparison counter */
58
59 void swap(int *element1Ptr, int *element2Ptr); /* prototype */
60
61 /* loop to control passes */
62 for (pass = 1; pass < size; pass++) {
63
64 /* loop to control number of comparisons per pass */
65 for (count = 0; count < size - 1; count++) {
66
67 /* if adjacent elements are out of order, swap them */
68 if ((*compare)(work[count], work[count + 1])) {
69 swap(&work[count], &work[count + 1]);
70 } /* end if */
71
72 } /* end for */
73
74 } /* end for */
75
76 } /* end function bubble */
77

```



## Outline

fig07\_26.c (Part 3 of 4)

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

```

78 /* swap values at memory locations to which element1Ptr and
79 element2Ptr point */
80 void swap(int *element1Ptr, int *element2Ptr)
81 {
82 int hold; /* temporary holding variable */
83
84 hold = *element1Ptr;
85 *element1Ptr = *element2Ptr;
86 *element2Ptr = hold;
87 } /* end function swap */
88
89 /* determine whether elements are out of order for an ascending
90 order sort */
91 int ascending(int a, int b)
92 {
93 return b < a; /* swap if b is less than a */
94
95 } /* end function ascending */
96
97 /* determine whether elements are out of order for a descending
98 order sort */
99 int descending(int a, int b)
100 {
101 return b > a; /* swap if b is greater than a */
102
103 } /* end function descending */

```



## Outline

fig07\_26.c (Part 4 of 4)

Enter 1 to sort in ascending order,  
Enter 2 to sort in descending order: 1

```

Data Items in original order
2 6 4 8 10 12 89 68 45 37
Data Items in ascending order
2 4 6 8 10 12 37 45 68 89

```

Enter 1 to sort in ascending order,  
Enter 2 to sort in descending order: 2

```

Data Items in original order
2 6 4 8 10 12 89 68 45 37
Data Items in descending order
89 68 45 37 12 10 8 6 4 2

```



## Outline

Program Output

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.

© Copyright 1992–2004 by Deitel & Associates, Inc. and Pearson Education Inc. All Rights Reserved.