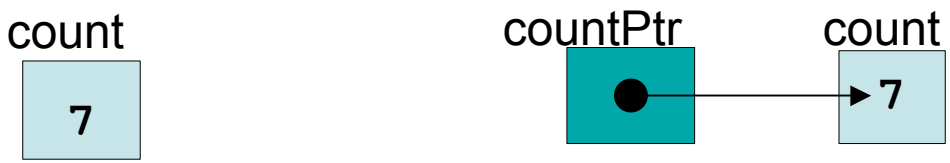
I puntatori

DD Cap.7 pp.245-280

KP Cap.6 pp.219-235

Cos’è un puntatore

- Un puntatore è una variabile che assume come valore un indirizzo di memoria.
- Il nome di una variabile fa quindi riferimento ad un valore direttamente, mentre un puntatore lo fa indirettamente.



Dichiarare un puntatore

I puntatori, come ogni altra variabile, devono essere dichiarati, prima di essere usati:
Esempio: `int* ptr, count;`

Questa dichiarazione dichiara una variabile di tipo puntatore ed una variabile intera.

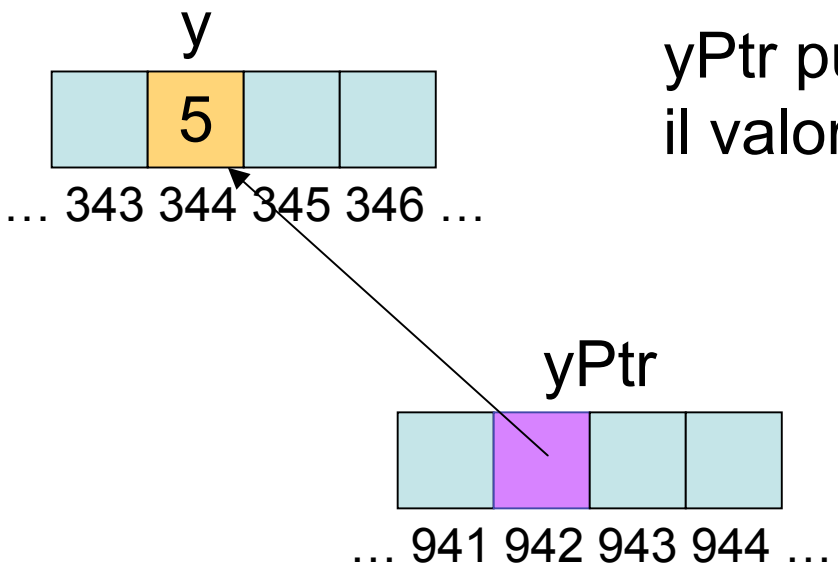
Se vogliamo dichiarare più puntatori, dobbiamo usare l'operatore * per ciascuno di essi.
Esempio: `int* ptr1, *ptr2;`
Si può inizializzare un puntatore a:

NULL (costante simbolica definita in stddef.h)=nessun dato
0, equivalente a NULL (ma è meglio quest'ultimo)
N.B. 0 è l'unico valore che si può asegnare ad un puntatore direttamente
un indirizzo…

Operatori sui puntatori (1) DD p. 246

- operatore unario di indirizzo: &

```
int y = 5;
int *yPtr;
yPtr = &y;     /* yPtr prende l'indirizzo di y */
```



yPtr punta ad y, cioè
il valore di yPtr è 344

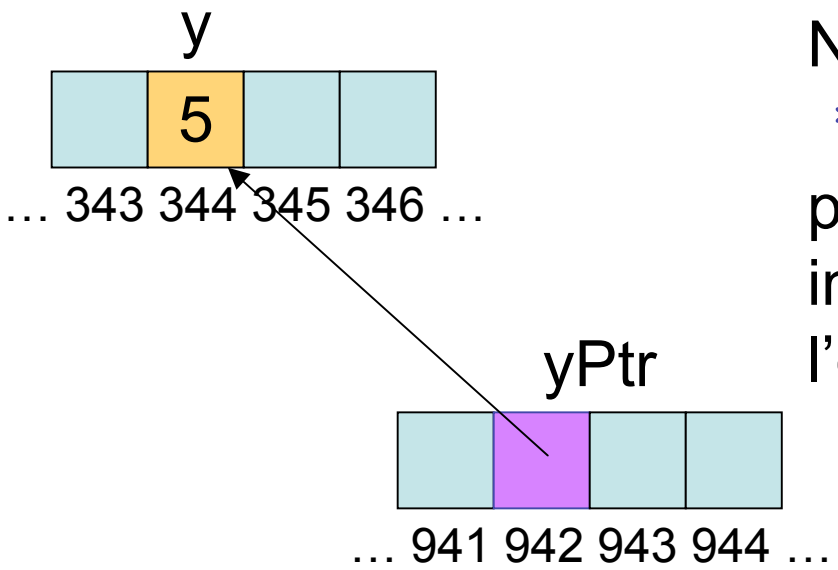
N.B. l'operando dell'operatore & deve essere una variabile; & non può quindi essere applicato a costanti, espressioni e var. register.

Operatori sui puntatori (2)

- operatore unario di deriferimento: *

```
printf("%d", *yPtr); /* stampa la variabile puntata da yPtr */
```

*yPtr è 5



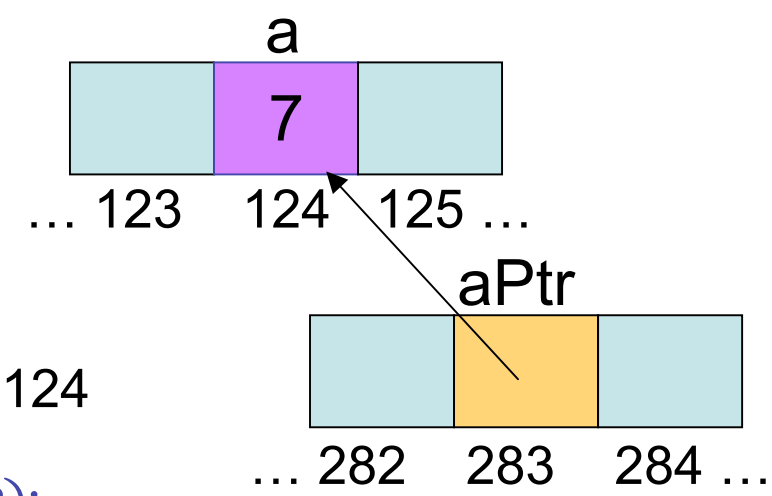
N.B. l' operatore & e l'operatore * sono uno l'inverso dell'altro, perciò applicandoli entrambi, in qualunque ordine, si riottiene l'operando.

Esempio

```
#include <stdio.h>
    int main()
    {
      int a, *aPtr;

    a=7; aPtr=&a;
    printf("%p %p", &a, aPtr);
    printf("%d %d", a, *aPtr);
    printf("%p %p", &*aPtr, *&aPtr);
    return 0;
    }
```

124 124
7 7
124 124



Cosa stampa questo programma?

Assegnare un puntatore

Un puntatore può essere assegnato ad un altro puntatore, SOLO se sono entrambi dello stesso tipo.

L'unica eccezione a questa regola è il puntatore a void (void *), poiché questo è generico e può rappresentare qualunque puntatore.

Un puntatore a void può essere assegnato a tutti gli altri tipi di puntatori e questi possono essere assegnati ad un puntatore a void. In entrambi i casi, non è necessaria alcuna conversione.

Non è possibile risolvere un riferimento di un puntatore a void (quanti bytes??). Il compilatore deve conoscere il tipo di dato per risolvere il riferimento di un puntatore.

Esempio

Siano i e j interi, p e q puntaori ad interi.
Quali, delle seguenti espressioni di assegnamento, sono corrette?

`p=&i;`	ok
`p=&*&i;`	ok
`i=(int)p;`	legale, ma???
`q=&p;`	tipi errati: q punta ad int, &p ad un puntatore
`*q=&j;`	no, *q è un int, &j è un puntatore
`i=(*&)j;`	ok
`i=*&*&j;`	ok
`i=*p++ + *q;`	ok, ma attenzione: il ++ si applica a p (vedremo poi cosa significa), non a *p perché gli operatori unari si associano da dx a sx

Stampare un puntatore

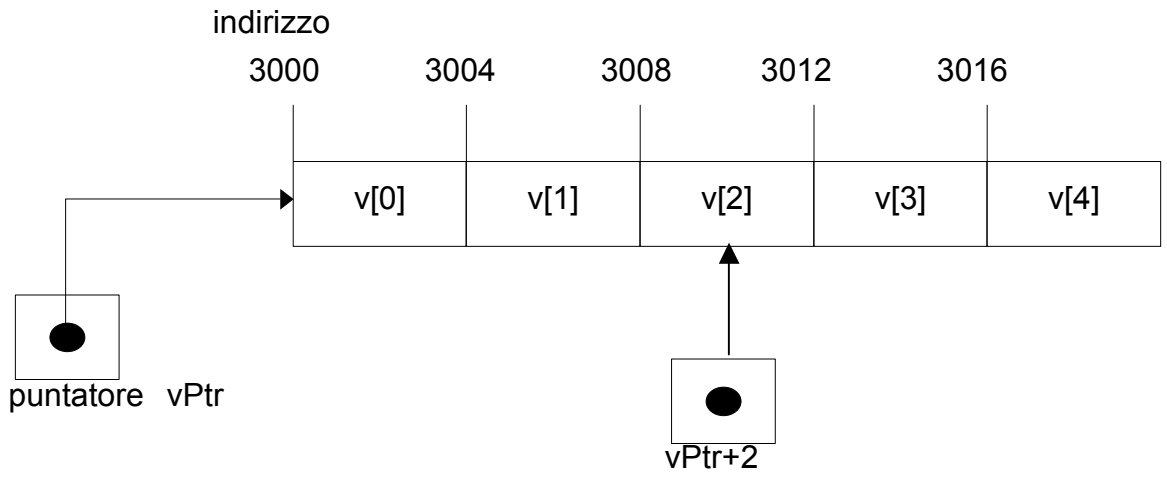
- Per stampare il contenuto di una variabile puntatore con una printf di usa `%p`. Il risultato è un numero espresso in esadecimale.
  Esempio. 0xbffffb94
- Per stampare gli indirizzi in decimale, un metodo consiste nell'impiego di un cast ad `unsigned long` e nell'utilizzo del formato `%lu`. Infatti, i puntatori usano 4 bytes, e se gli interi ne usano 2, la conversione in intero semplice può provocare la stampa di un numero negativo.
  Esempio. %d: -1073742956
  %ul: 3221224340l

L’aritmetica dei puntatori (1) DD p.267

- I puntatori sono operandi validi per le espressioni aritmentiche, quelle di assegnamento e quelle di confronto.
- Ma non tutte le operazioni sono ammissibili sui puntatori…
- Un puntatore può essere incrementato (++) o decrementato (--), vi si può sommare (+ o +=) o sottrarre (- o -=) un intero, si può infine sottrarre un puntatore da un altro.

L'aritmetica dei puntatori (2) DD p.267

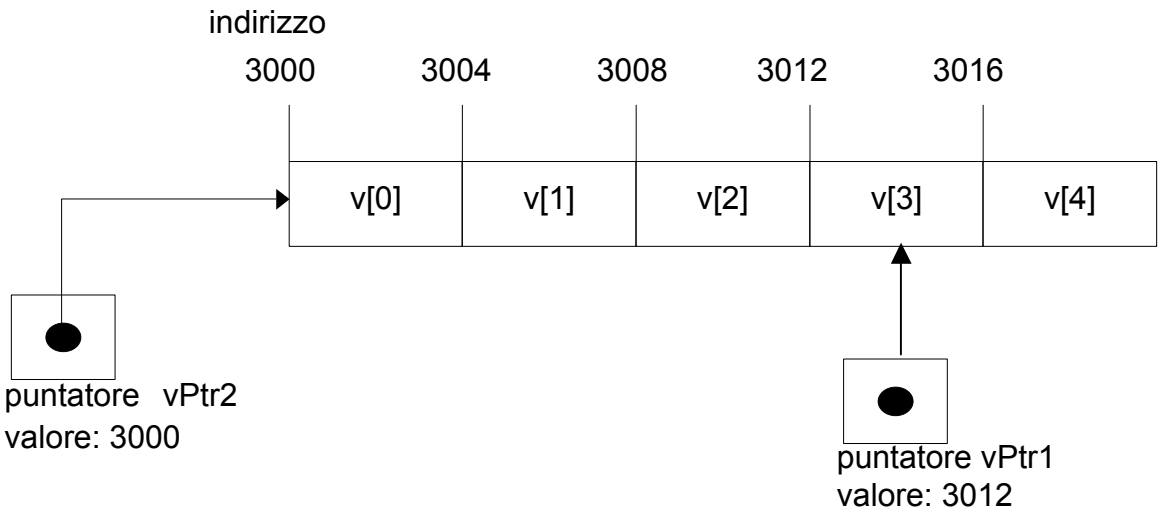
Esempio: `int v[5];`
`vPtr=&v[0];`
`vPtr+=2;`



Nell'aritmetica convenzionale la somma 3000+2 restituisce il valore 3002. Questo non avviene nella aritmetica dei puntatori. L'incremento o il decremento deve essere moltiplicato per la dimensione (sizeof) dell'oggetto cui il puntatore fa riferimento.

L'aritmetica dei puntatori (3)

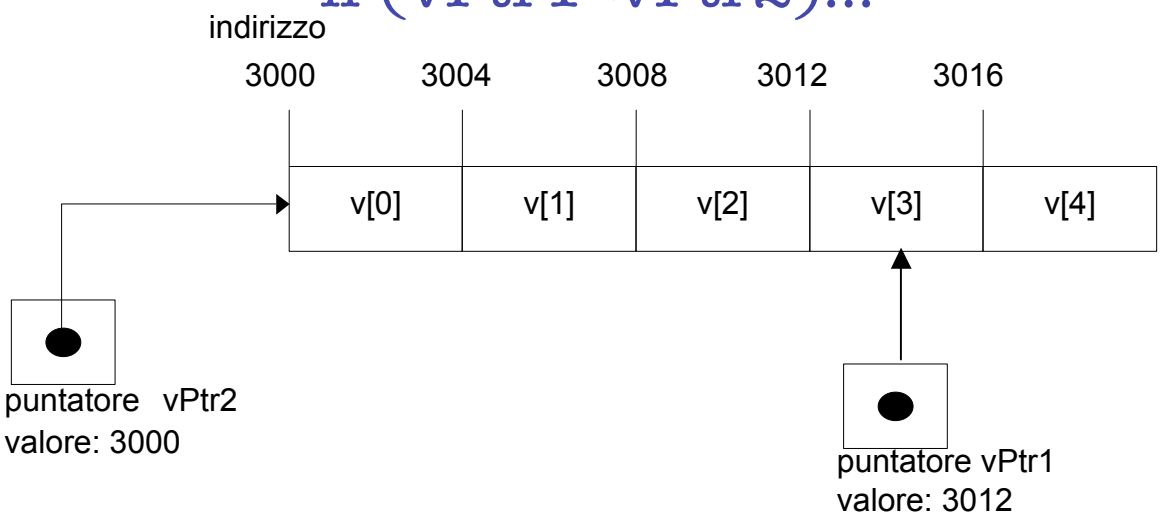
Esempio: `vPtr1=&v[3];`
`vPtr2=&v[0];`
`x=vPtr1-vPtr2;`



La terza assegnazione pone in x il numero degli elementi compresi tra vPtr2 e vPtr1. In questo caso il risultato è 3 (non 12).
N.B. Questa espressione non ha senso se non utilizzata con un vettore, infatti non possiamo presumere nulla sulla posizione di due variabili qualunque in memoria.

Le relazioni tra puntatori

Esempio: `if (vPtr1==vPtr2)...`
`if (vPtr==NULL)...`
`if (vPtr1<vPtr2)...`



I puntatori possono essere confrontati utilizzando gli operatori di uguaglianza e quelli relazionali, ma questi ultimi confronti non hanno significato se i puntatori non fanno riferimento allo stesso vettore.

Un confronto tra due puntatori che fanno riferimento allo stesso vettore mostra che un puntatore punta ad un elemento con un indice minore di un altro.

Relazione tra puntatori e vettori

- I vettori e i puntatori sono strettamente correlati in C e potranno essere usati in modo *quasi* interscambiabile.
- Esempi:
  - `vPtr=&V[0];`
  - `*(vPtr+3)` equivale a `V[3]`
    N.B. le parentesi in `*(vPtr+3)` sono necessarie perché `*` ha priorità maggiore di `+`
  - `vPtr+3` equivale a `&V[3]`
- Il nome di un vettore è un puntatore all'inizio del vettore, quindi `&V[0]` equivale a `V`, ma non si può scrivere `V+=3;` perché `V` è un puntatore costante.

Esempio

```
#include <stdio.h>
    int main()
    {int b[] = {10,20,30,40};
     int *bPtr = b; int i;

    for(i=0;i<4;i++)                          /*primo modo: b[i]*/
      printf("b[%d]=%d\n",i,b[i]);
    for(i=0;i<4;i++ )                         /*secondo modo: *(b+i)*/
      printf("*(b+%d)=%d\n",i,*(b+i));
    for(i=0;i<4;i++)                          /*terzo modo: bPtr[i]*/
    printf("bPtr[%d]=%d\n",i,bPtr[i]);
    for(i=0;i<4;i++)                          /*quarto modo: *(bPtr+i)*/
      printf("*(bPtr+%d)=%d\n",i,*(bPtr+i));
    return 0;
    }
```

N.B. La notazione *(bPtr+i) al posto di b[i], se pure di meno immediata comprensione, genera un codice più veloce.

Un altro esempio

Problema: copiare un vettore di caratteri su un altro vettore di caratteri

```
char string1[]="ciao"; char string2[10];
...copiax(string1, string2);...

void copia1(char* sA, char* sB)
{ int i=0;
  while (sA[i] !='\0')
    {sB[i]=sA[i];
     i++;
     }
  sB[i]='\0';
}
```

```
void copia3(char* sA, char* sB)
{ int i;
  for (; (*sB=*sA) !='\0'; sA++,sB++);
}
```

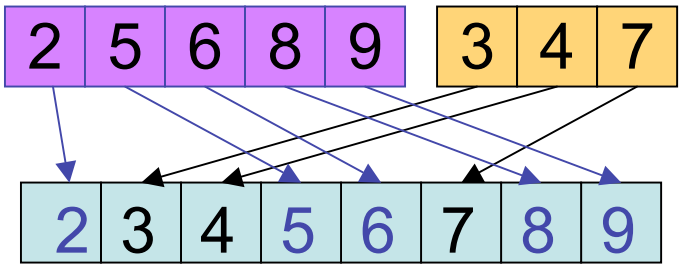
```
void copia2(char* sA, char* sB)
    { int i;
      for (i=0; (sB[i]=sA[i]) !='\0'; i++);
    }
```

Ancora un esempio

Problema: dati due vettori ordinati in senso crescente A e B, di dimensioni n ed m, costruire un terzo vettore C in cui compaiano tutti gli elementi di A e B ordinati in senso crescente.

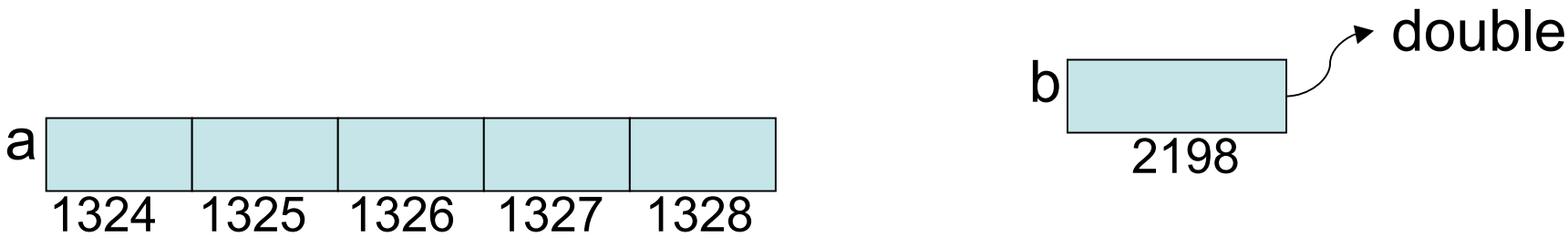
```
void fondi(int A[], long n, int B[], long m, int C[])
{
  if (n>0 && (m==0 || (m>0 && A[0]<B[0])))
    {
      C[0]=A[0];
      fondi(A+1,n-1,B,m,C+1);
    }
    else if (m>0)
          {
            C[0]=B[0];
            fondi(A,n,B+1,m-1,C+1);
          }
}
```

Per esercizio: versione iterativa



Allocazione dinamica della memoria (1) KP p.232

- Dato un vettore A di reali, i due seguenti prototipi sono equivalenti:
  ```
  double sum(double a[], int n);
  double sum(double *a, int n);
  ```
- Tale equivalenza NON vale nel caso di dichiarazioni, infatti le due dichiarazioni:
  ```
  double a[5];
  double *b;
  ```
  allocano due oggetti molto diversi!!



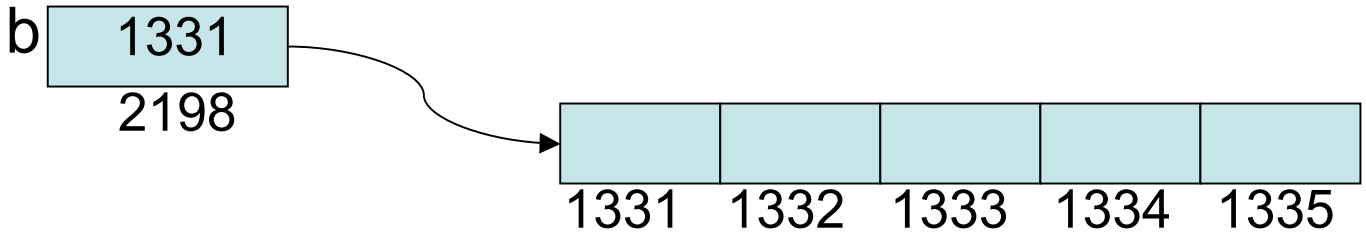
Allocazione dinamica della memoria (2)

- Per allocare un vettore a partire dal puntatore b si usa la funzione malloc (memory allocation), il cui prototipo è contenuto nella libreria stdlib.h:

```
double *b;
...
b=malloc(5*sizeof(double));
```

- Lo spazio così allocato NON viene restituito al sistema all'uscita delle funzioni, come accade per le altre variabili; esso va restituito esplicitamente tramite la funzione free (anche il suo prototipo è nella libreria stdlib.h):

```
free(b);
```



Allocazione dinamica della memoria (3)

- La funzione malloc viene usata per allocare i vettori dinamici, cioè quei vettori la cui dimensione debba essere definita da input:

```
...
printf("dammi la lungh. del vettore");
scanf("%d", &n);
int A[n];          /*ERRATO! */
int *a;
a=malloc(n*sizeof(int)); /*CORRETTO */
...
free(a);
```

Esercizi

Se a è un vettore di 3 interi, e b un vettore di 4 double, cosa produce la funzione sizeof applicata ad a ed a b, rispettivamente?

Diagramma a blocchi o pseudocodice + C per i seguenti problemi:

- Una sequenza si dice palindroma se essa può essere letta allo stesso modo da destra e da sinistra. Leggete un certo numero (dato in input) di interi e stabilite se la sequenza risulta palindroma (usare vettori dinamici)
- Il valore memorizzabile in un long int è circa 2mld. Per molte applicazioni, tale ordine di grandezza non è abbastanza grande. Scrivete una funzione che sommi due interi molto grandi, le cui cifre vengano memorizzate all'interno di vettori.