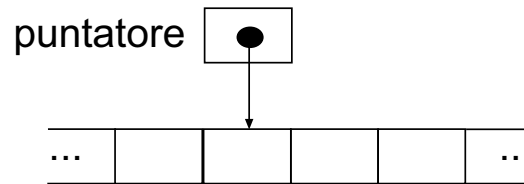


# I puntatori

- Un **puntatore** è una variabile che contiene l'**indirizzo** di un'altra variabile.



- L'operatore `&` fornisce l'indirizzo di un oggetto:  
`p = &c;` assegna a `p` l'indirizzo di `c`, i.e., `p` “punta a” `c`.
- L'operatore `*` (indirizzazione/deriferimento) si applica ad un puntatore e fornisce come risultato l'oggetto puntato da quest'ultimo.

- Esempi di dichiarazione ed uso di puntatori:

```
int x=1, y=2, z[10];
int *ip;          /* ip è un puntatore a interi */
ip = &x;          /* ip punta a x */
y = *ip;          /* y vale 1 */
ip = &z[0];        /* ip punta a z[0] */
*ip = *ip + 10;   /* incrementa di 10 il valore di z[0] */
```

# Puntatori e argomenti di funzioni

Gli **argomenti** delle funzioni passati come **puntatori (e vettori)** sono **modificabili** dalle funzioni (al contrario degli altri casi in cui il passaggio avviene per valore, creando una copia locale dell'argomento).

Esempio:

```
void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;    /* lavora su una copia */
    y = temp; /* degli argomenti */
}
```

```
void swap(int *px, int *py)
{
    int temp;
    temp = *px;
    *px = *py; /* lavora */
    *py = temp; /* sugli argomenti */
}
```

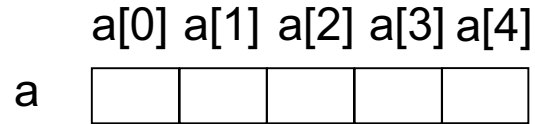
```
main()
{
    int a=5, b=10;
    swap(a,b); /* non modifica */
                /* a, b */
    return 0;
}
```

```
main()
{
    int a=5, b=10;
    swap(&a,&b); /* modifica a,b */
    return 0;
}
```

# Puntatori e vettori (I)

Puntatori e vettori sono strettamente correlati:

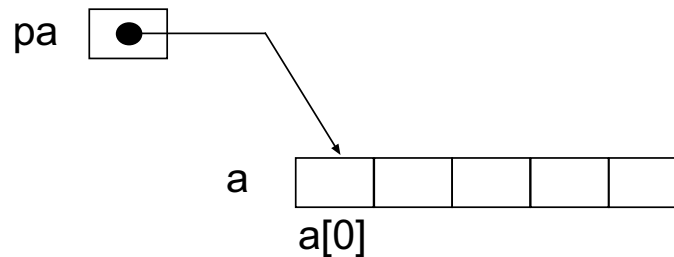
`int a[5];` dichiara un array di 5 elementi; `a` è l'indirizzo del primo elemento `a[0]`.



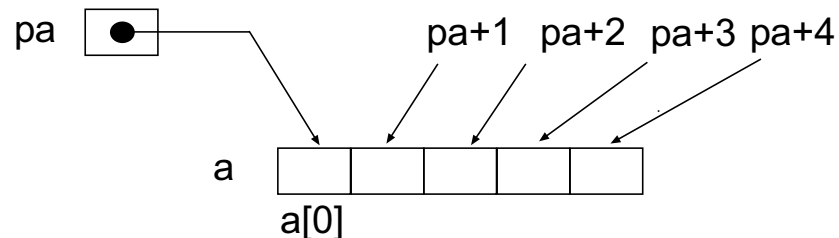
Il codice

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

dichiara ed assegna al puntatore `pa` l'indirizzo dell'elemento `a[0]`.



- Il comando `x = *pa;` copia `a[0]` in `x`.
- L'espressione `(pa+i)` è l'indirizzo dell'elemento `a[i]` (per valori di `i` uguali a 0, 1, 2, 3, 4).



## Puntatori e vettori (II)

- Poiché in C `a` è l'indirizzo di `a[0]`, eseguendo l'assegnamento  
`pa = &a[0];`  
oppure l'assegnamento  
`pa = a;`  
otteniamo che `pa` e `a` hanno lo stesso valore, i.e., puntano entrambi a `a[0]`.
- Da quanto detto precedentemente abbiamo inoltre che `*(a+i)` corrisponde a `a[i]`.
- Tuttavia esiste la seguente differenza tra puntatori e vettori:
  - un puntatore è una variabile, quindi sono legali le espressioni seguenti:  
`pa = a`  
`pa++`
  - un vettore **non** è una variabile, quindi **non** sono legali le espressioni seguenti:  
`a = pa`  
`a++`

# Puntatori a caratteri

In C una stringa costante come "ciao, mondo" è un vettore di caratteri terminato dal carattere speciale null (`\0`):

c	i	a	o	,		m	o	n	d	o	\0
---	---	---	---	---	--	---	---	---	---	---	----

Il carattere speciale null svolge il ruolo di **terminatore** della stringa. In questo modo è possibile memorizzare stringhe di lunghezze diverse all'interno di un vettore di caratteri (purché ovviamente il numero di caratteri di tali stringhe più il carattere `\0` non superi il numero di elementi del vettore).

Un puntatore può essere utilizzato direttamente per allocare la memoria necessaria alla memorizzazione di una stringa costante nel modo seguente:

```
char *pmessage;  
pmessage = "ciao, mondo";
```

In seguito è possibile utilizzare `pmessage` alla stregua di un normale vettore di caratteri, ad esempio in una chiamata alla funzione di libreria `printf`:

```
printf("%s", pmessage);
```

# Vettori di puntatori e vettori multidimensionali (I)

- Essendo i puntatori delle variabili, possono essere a loro volta memorizzati in vettori; ad esempio la dichiarazione seguente:

```
char *line[MAXLINE];
```

crea un vettore di MAXLINE elementi, ognuno dei quali è un puntatore a carattere.

- In particolare, se ogni `line[i]` punta al primo elemento di un ulteriore vettore di caratteri, il risultato complessivo sarà che il vettore `line` potrà essere utilizzato per memorizzare delle righe di testo.

- In C è possibile dichiarare vettori multidimensionali, ad esempio:

```
int a[3][4];
```

dichiara una matrice di 3 righe e 4 colonne i cui elementi sono interi. In C un vettore bidimensionale come `a` è trattato come un vettore (unidimensionale) di elementi che, a loro volta, sono vettori. Quindi l'espressione `a[i][j]` è corretta, mentre l'espressione `a[i,j]` **non** è corretta.

## Vettori di puntatori e vettori multidimensionali (II)

Le dichiarazioni

```
int a[10][20];
```

```
int *b[10];
```

differiscono per i motivi seguenti:

- nel caso della prima dichiarazione vengono allocate le locazioni di memoria necessarie per contenere 200 interi ( $10 \times 20$ ), mentre nel caso della seconda vengono allocate le locazioni necessarie per contenere 10 puntatori ad interi;
- nonostante le espressioni `a[3][4]` e `b[3][4]` denotino entrambe un numero intero, nel caso di `b` ogni elemento `b[i]` può puntare ad un vettore di lunghezza arbitraria (non necessariamente di 20 elementi);

Quindi è conveniente utilizzare dei vettori di puntatori (rispetto a dei vettori multidimensionali) quando la lunghezza dei “vettori puntati” è variabile (per ragioni di risparmio di memoria).

# Inizializzazione di vettori e puntatori a caratteri

- È possibile inizializzare un vettore (oltre che con un ciclo che assegni un valore ad ogni singolo elemento) direttamente in un colpo solo:

```
int v[4]={10, 20, 30, 40};
```

- È anche possibile inizializzare un array in fase di dichiarazione senza specificare esplicitamente il numero di elementi:

```
char string[]="ciao, mondo";
```

In questo caso, verrà automaticamente allocato lo spazio necessario per contenere i caratteri della stringa `ciao, mondo` più il carattere terminatore `\0`.

- Nel caso di un puntatore a caratteri possiamo utilizzare la seguente dichiarazione

```
char *pstring="ciao, mondo";
```

- Un esempio più complesso:

```
char *line[]={ "abc", "def", "ghi"};
```



# Argomenti sulla linea di comando

Analogamente a quanto succede con i comandi Unix, è possibile scrivere dei programmi C che accettano argomenti sulla linea di comando:

```
#include <stdio.h>
```

```
main(int argc, char *argv[]) /* argc: argument count (n. argomenti+1) */
                               /* argv: argument vector (puntatore ad      */
                               /* un vettore di stringhe che contiene     */
                               /* gli argomenti).                          */
{
    int i;

    for(i=1; i<argc; i++)
        printf("%s\n",argv[i]);

    printf("\n");
    return 0;
}
```

Il programma precedente emula il comando Unix `echo`, nel senso che accetta un numero arbitrario di comandi e li stampa sullo standard output.

**N.B.:** `argv[0]` è il nome del programma C compilato, mentre `argv[argc]` è la costante 0 (i.e., un **null pointer**).

## Input formattato: scanf e sscanf (I)

- La funzione di libreria

```
int scanf(char *format,...);
```

legge i caratteri dallo standard input interpretandoli secondo il formato specificato dal primo argomento e memorizzando i risultati nei rimanenti argomenti, che **devono** essere dei **puntatori**.

Ad esempio `scanf("%f %d", &x, &i);` legge dallo standard input un numero a virgola mobile ed un intero e li assegna, rispettivamente, alle variabili `x` e `i` (si noti l'uso dell'operatore `&` per ottenere gli indirizzi delle variabili argomento).

- L'esecuzione di `scanf` termina quando esaurisce l'argomento `format`, quando l'input non soddisfa le specifiche od in caso di end-of-file.
- Il valore restituito da `scanf` rappresenta il numero di elementi in input che sono stati memorizzati con successo negli argomenti corrispondenti. In caso di end-of-file viene invece restituito il valore EOF.
- La funzione  

```
int sscanf(char *string, char *format,...);
```

si comporta esattamente come `scanf` tranne per il fatto di leggere i caratteri dalla stringa puntata da `string` invece che dallo standard input.

## Input formattato: scanf e sscanf (II)

- La stringa di formato può essere costituita dai seguenti elementi:
  - spazi o tabulazioni (vengono ignorati);
  - caratteri ordinari (diversi da %) che dovranno poi corrispondere esattamente ai caratteri in input diversi dai **white space characters**;
  - **specifiche di conversione** che iniziano con il carattere % seguito da un “suppression character” \* (opzionale), da un numero indicante la lunghezza massima del campo (opzionale), da un carattere fra h, l, L indicante la “grandezza” del valore (opzionale) e da un **carattere di conversione**:

d	intero decimale
i	intero (eventualmente ottale, se preceduto da uno 0, o esadecimale, se preceduto da 0x o 0X).
o, x	intero ottale o esadecimale rispettivamente
c	carattere (eventualmente anche white space)
s	stringa di caratteri
e, f, g	numero a virgola mobile

- Normalmente gli input field (a meno che non si usi il carattere di conversione c) sono sequenze consecutive di caratteri **non** white space che si estendono fino al prossimo carattere white space o fino alla lunghezza massima di campo specificata.

## Esempi d'uso di scanf

- Nel caso si vogliono leggere linee del tipo

5 feb 2003

si può utilizzare la scanf seguente:

```
scanf("%2d %3s %4d", &giorno, mese, &anno);
```

dove giorno, mese ed anno sono dichiarati come segue:

```
int giorno, anno;
```

```
char mese[4];
```

Si noti che il vettore `mese` ha dimensione 4 per permettere la memorizzazione di nomi abbreviati di mesi dell'anno (3 caratteri) più il carattere nullo di terminazione.

- L'istruzione

```
scanf("%f %*f %f", &x, &y);
```

legge tre numeri in virgola mobile e memorizza il primo in `x` ed il terzo in `y`, saltando il secondo (si noti l'uso del "suppression character" `*`).

# Esercizi

- Scrivete le seguenti funzioni (utilizzando i puntatori):
  - `strcat(s,t)` che copia la stringa `t` al termine della stringa `s` (supponete che la lunghezza del vettore di caratteri che contiene `s` sia sufficiente a contenere la concatenazione di `s` e `t`).
  - `reverse(s)` che inverte la stringa `s` (e.g. `abc`  $\rightsquigarrow$  `cba`).
- Scrivete un programma `sort` che, se chiamato senza argomenti sulla linea di comando, ordina le linee di un testo in ordine alfabetico, se chiamato con l'opzione `-r`, le ordina in senso inverso.  
**Suggerimento:** si sfrutti la funzione di libreria `strcmp` (il prototipo è specificato nel file header `string.h`).
- Scrivete un programma che legga dallo standard input dei numeri e ne stampi la somma totale.