

# TIPI DI DATO

---

***Il concetto di **tipo di dato** viene introdotto per raggiungere due obiettivi:***

- ***esprimere in modo sintetico***
  - *la loro rappresentazione in memoria, e*
  - *un insieme di operazioni ammissibili*
- ***permettere di effettuare controlli statici (al momento della compilazione) sulla correttezza del programma***
  - *se facciamo un errore, vogliamo che il compilatore ce lo segnali: tutti gli errori che trova il compilatore non dobbiamo trovarli noi!*

# TIPI DI DATO

---

**Un tipo di dato  $T$  è definito come:**

- **un dominio di valori,  $D$**
- **un insieme di funzioni  $F_1, \dots, F_n$  sul dominio  $D$**
- **un insieme di predicati  $P_1, \dots, P_m$  sul dominio  $D$**

$$T = \{ D, \{F_1, \dots, F_n\}, \{P_1, \dots, P_m\} \}$$

# ***TIPI DI DATO: ESEMPIO***

---

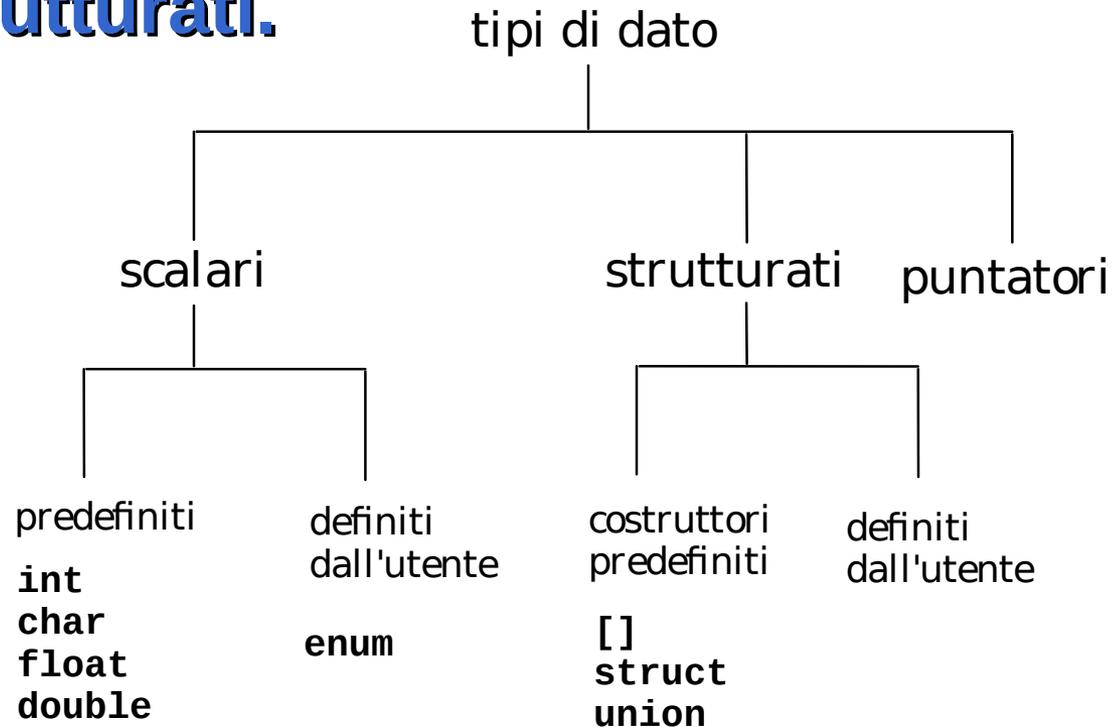
***Il tipo di dato **INTERO** è definito come:***

- ***un dominio di valori,  $Z$***
- ***un insieme di funzioni  $F_1, \dots, F_n$  sul dominio  $D$*** 
  - ***esempio SOMMA, SOTTRAZIONE, PRODOTTO ....***
- ***un insieme di predicati  $P_1, \dots, P_m$  sul dominio  $D$*** 
  - ***ad esempio MAGGIORE, MINORE, UGUALE...***

# TIPI DI DATO

---

*I tipi di dato si differenziano in **scalari e strutturati**.*



# ***RIASSUNTO TIPI PRIMITIVI***

---

*Il C prevede quattro tipi primitivi:*

**char**      *(caratteri)*

**int**      *(interi)*

**float**      *(reali)*

**double**      *(reali in doppia precisione)*

*I tipi possono essere modificati dai qualificatori:*

**signed**                      **unsigned**

*che possono essere applicati ai tipi char e int*

**short**                      **long**

*che possono essere applicati al tipo int*

**long**

*che può essere applicato anche al tipo double*



# ***TIPI DI DATO STRUTTURATO***

---

***In C si possono definire tipi strutturati.***

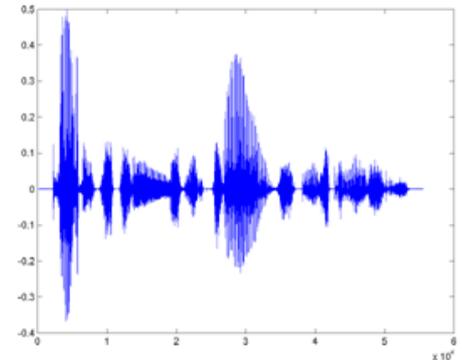
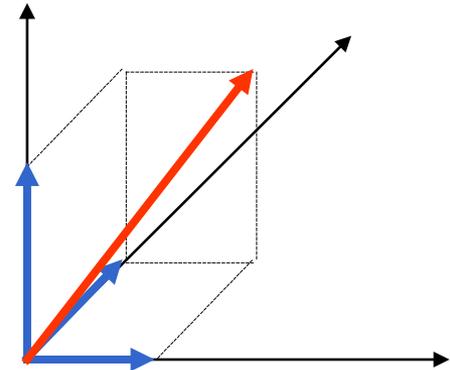
***Vi sono due costruttori fondamentali:***

**[ ]** (array)

**struct** (strutture)

# Array: esempi

- *Se devo rappresentare dati vettoriali, ho bisogno di avere più coordinate*
- *Posso aver bisogno di rappresentare tabelle (elenco del telefono, database di ogni tipo)*
- *Uno spartito è una sequenza di note*
- *Parti di successioni*
- *Parole e frasi sono sequenze di caratteri*
- *Un suono è una sequenza di valori di pressione sonora*
- *Una matrice è una sequenza bidimensionale (sequenza di sequenze)*



nome	indirizzo	tel
Mario	Via Garibaldi 1	4922

$$\det \begin{pmatrix} 1 & 2 & 1 \\ 3 & 2 & 0 \\ 2 & 0 & 1 \end{pmatrix}$$

# ARRAY (VETTORI)

---

*Se vogliamo usare 1000 variabili intere, è scomodo dichiararle così*

```
int a0, a1, a2, a3, a4, ... a999;
```

*Poi se vogliamo visualizzarle tutte, non vogliamo scrivere*

```
printf("%d ", a0);
```

```
printf("%d ", a1);
```

```
printf("%d ", a2);
```

```
...
```

```
printf("%d ", a999);
```

*ma ci piacerebbe scrivere qualcosa del tipo*

```
for (i=0; i<1000; i++)
```

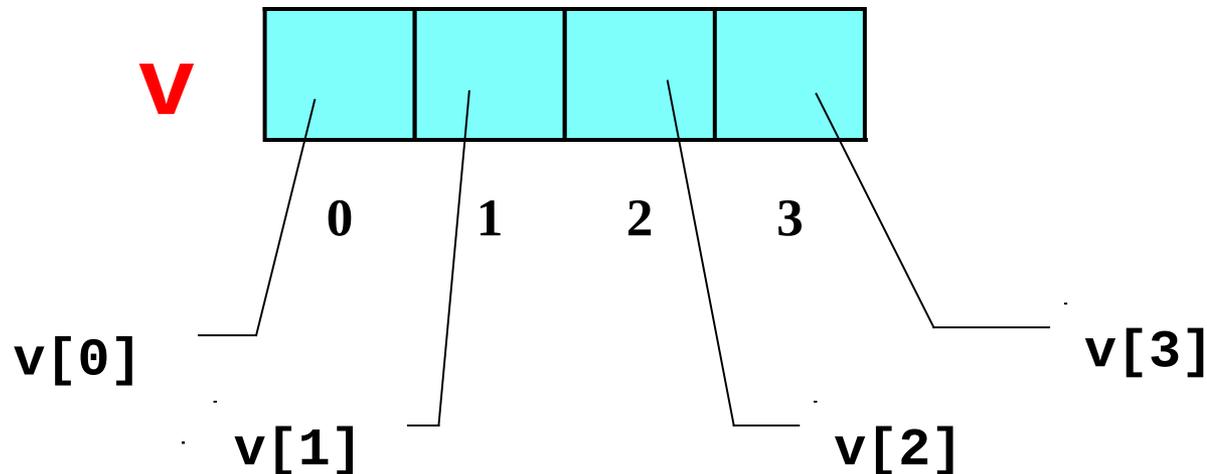
```
    printf("%d ", ai);
```



# ARRAY (VETTORI)

---

**Un array di  $N$  elementi è una collezione finita di  $N$  variabili dello stesso tipo, ognuna identificata da un indice compreso fra 0 e  $N-1$**





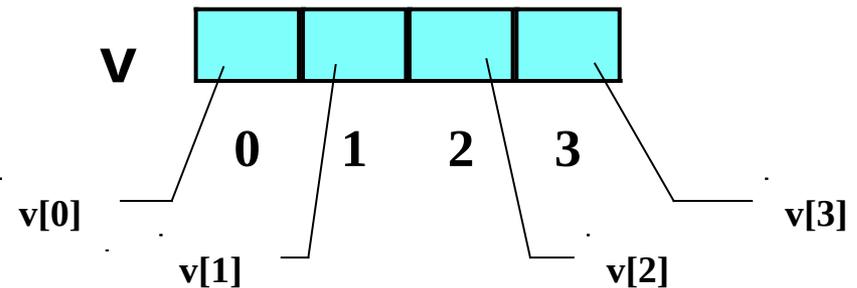
# ARRAY (VETTORI)

## Definizione di una variabile di tipo array:

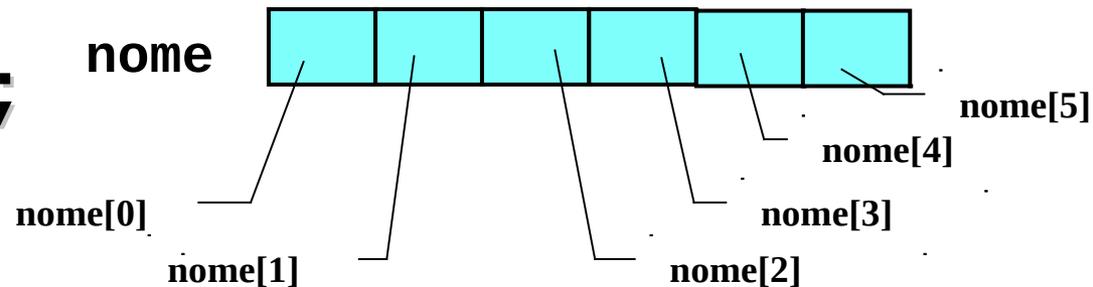
**<tipo> <nomeArray> [ <costante> ] ;**

**Esempi:**

**int v[4];**



**char nome[2\*3];**





# Array

- **Una volta dichiarato un array,**  

```
int a[4], media, i;
```
- **posso usare le variabili che lo costituiscono come delle normalissime variabili di quel tipo:**

- **Assegnarvi dei valori:**

```
a[0]=10;  
a[1]=20;
```

- **usarle nelle espressioni**

```
media = (a[0]+a[1])/2;
```

- **leggerle da tastiera**

```
scanf("%d", &a[2]);
```

- **visualizzarle:**

```
printf("a[2]=%d", a[2]);
```

- **Inoltre posso riferirle con un indice che può essere un'espressione:**

```
i=2;  
a[i+1]=a[i];
```

a[0]	10
a[1]	20
a[2]	3
a[3]	3
media	15
i	2

```
3  
a[2]=3
```

# *inizializzare un vettore con il quadrato degli indici*

---



```
#include <stdio.h>

main()
{ int i=0;
  int A[3];

  while (i<3)
  {
    A[i]=i*i;   /*gli elementi del vettore sono 0,1,4*/
    i++;
  }
}
```

# Visualizzazione in ordine inverso



- leggere una sequenza di 10 interi e visualizzarla in ordine inverso

```
#include <stdio.h>
main()
{ int i, a[10];
  // lettura dell'array
  for (i=0; i<10; i++)
  { printf("inserisci l'elemento %d:", i);
    scanf("%d", &a[i]);
  }
  // stampa in ordine inverso
  for (i=9; i>=0; i--)
    printf("%d ", a[i]);
}
```



## *Dimensione degli array*

---

*Il compilatore deve sapere qual è l'occupazione di memoria di una variabile, quando la crea*

```
<tipo> <nomeArray> [ <costante> ] ;
```

~~**ATTENZIONE: Sbagliato !!**~~

```
int N;  
char nome [N];
```

Il compilatore non sa come dimensionare l'array

# Visualizzazione in ordine inverso

- leggere una sequenza di 10 interi e visualizzarla in ordine inverso

```
#include <stdio.h>
main()
{ int i,a[10];
  // lettura dell'array
  for (i=0; i<10; i++)
  { printf("inserisci l'elemento ");
    scanf("%d",&a[i]);
  }
  // stampa in ordine inverso
  for (i=9; i>=0; i--)
    printf("%d ",a[i]);
}
```

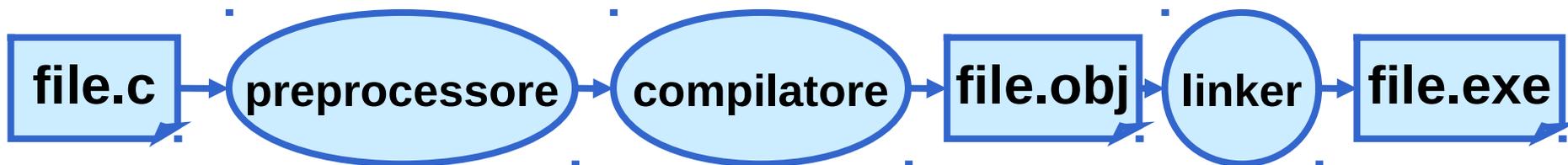
**Problema:**

se voglio cambiare la dimensione dell'array, devo intervenire in tanti punti del programma 😞 (facile che me ne dimentichi qualcuno)

# Costanti

---

- **Ho bisogno di definire una costante: un simbolo che rappresenta un valore che non viene mai cambiato**
  - *in questo modo il compilatore potrebbe usarlo a tempo di compilazione, ad es. per definire la dimensione degli array*
  - *se voglio cambiare il valore, lo faccio in un solo punto del programma*
- **Per definire le costanti in linguaggio C si utilizza il **Preprocessore****
- **Il Preprocessore viene invocato prima della fase di compilazione ed effettua delle **sostituzioni testuali****
- **Il preprocessore accetta delle **direttive****
  - *vengono scritte nel programma, ma non sono istruzioni C*
  - *sono riconoscibili dal simbolo del cancelletto: #*



# LA DIRETTIVA `#define`

---

## Sintassi:

**`#define`** `<testo1>``<testo2>`

## Effetto:

*definisce una regola di ricerca e sostituzione: ogni occorrenza di `testo1` verrà sostituita da `testo2`*

## Scopo:

*definire costanti simboliche (per convenzione, `testo1` è maiuscolo)*

# ESEMPIO

---

*Prima del pre-processing:*

```
#define RADICEDI2 1.4142  
  
main()  
{ float lato = 18;  
  float diagonale = lato * RADICEDI2;  
}
```

*Dopo il pre-processing:*

```
main()  
{ float lato = 18;  
  float diagonale = lato * 1.4142;  
}
```

# *Visualizzazione in ordine inverso*

---

*leggere una sequenza di 10 interi e visualizzarla in ordine  
inverso*

```
#include <stdio.h>
#define N 10
main()
{ int i,a[N];
  // lettura dell'array
  for (i=0; i<N; i++)
  { printf("inserisci l'elemento %d: ",i);
    scanf("%d",&a[i]);
  }
  // stampa in ordine inverso
  for (i=N-1; i>=0; i--)
    printf("%d ",a[i]);
}
```

# #define non è un'istruzione

- Siccome non è un'istruzione compresa dal compilatore, non va terminata dal punto e virgola

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

```
#define N 10;
```

```
main()
```

```
{ int i, a[N];
```

```
// lettura dell'array
```

```
for (i=0; i<N; i++)
```

```
{ printf("inserisci l'elemento %d: ", i);
```

```
scanf("%d", &a[i]);
```

```
}
```

```
// stampa in ordine inverso
```

```
for (i=N-1; i>=0; i--)
```

```
printf("%d ", a[i]);
```

```
}
```

error C2143: syntax error :  
missing ']' before ';'

missing ')' before ';'

# #define non è un'istruzione

- Il compilatore vede un programma in cui il preprocessore ha sostituito "N" con "10;"

```
#include <stdio.h>
main()
{ int i,a[10;];
```

error C2143: syntax error :  
missing ']' before ';'

```
    // lettura dell'array
```

```
    for (i=0; i<10;; i++)
```

```
    { printf("inserisci l'elemento %d: ",i);
      scanf("%d",&a[i]);
    }
```

```
    // stampa in ordine inverso
```

```
    for (i=10;-1; i>=0; i--)
```

```
        printf("%d ",a[i]);
```

```
}
```

missing ')' before ';'



# **RICERCA**

---

- ***Dato un vettore ed un elemento  $X$ , trovare se  $X$  è un elemento del vettore.***
- ***Se  $X$  appartiene al vettore, visualizzarne l'indice, altrimenti visualizzare "non trovato"***

# RICERCA

---

Codifica:

```
#define DIM 4  
main()  
{  
    int v[DIM] = {43, 12, 7, 86};  
    int i=0, x=7;  
    while ((i<DIM) && (v[i]!=x))  
        i++;  
    if (i<DIM) printf("%d", i);  
    else printf("non trovato");  
}
```

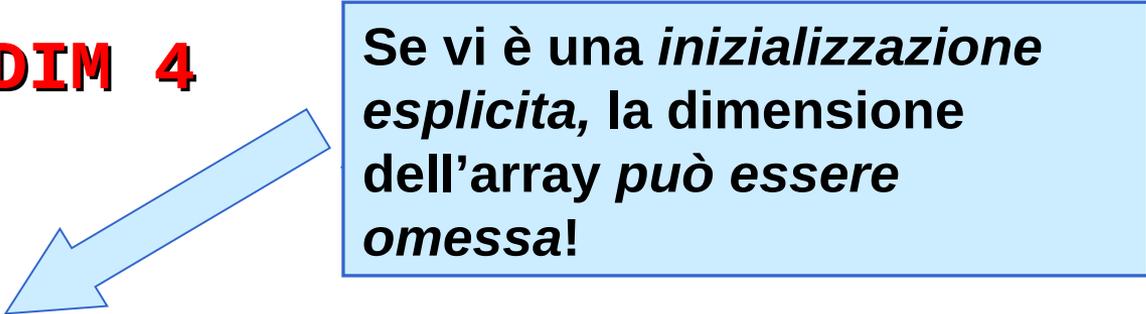
Espressione di  
*inizializzazione*  
di un array

# ESEMPIO

---

Codifica:

```
#define DIM 4  
main()  
{  
    int v[] = {43, 12, 7, 86};  
    int i=0, x=7;  
    while ((i<DIM) && (v[i]!=x))  
        i++;  
    if (i<DIM) printf("%d", i);  
    else printf("non trovato");  
}
```



Se vi è una *inizializzazione esplicita*, la dimensione dell'array può essere *omessa*!



# ***DIMENSIONE FISICA VS. LOGICA***

---

- ***... e se ho bisogno di memorizzare un numero di dati non noto a priori?***
- ***Un array è una collezione finita di N celle dello stesso tipo; questo non significa che si debbano per forza usare sempre tutte!***
- ***La **dimensione logica** di un array può essere inferiore (mai superiore!) alla sua dimensione fisica***
- ***Spesso, la porzione di array realmente utilizzata dipende dai dati d'ingresso.***
- ***Quindi: se non sappiamo esattamente quanti sono i dati da inserire, scegliamo una dimensione fisica sufficientemente grande, poi useremo solo una parte dell'array***



## ***Esercizio***

---

- ***È data una serie di rilevazioni di temperature espresse in gradi Celsius, corrispondenti ai giorni di un mese***
- ***Scrivere un programma che legga da tastiera il numero  $N$  di giorni del mese, poi memorizzi le  $N$  rilevazioni.***
- ***Infine, si visualizzi il giorno del mese in cui è stata rilevata la temperatura massima***



# Codifica

```
#include <stdio.h>
#define DIM 32 // Definisci un array di dim sufficiente
main()
{ int temp[DIM];
  int N, GiornoPiuCaldo, i;
  printf("Inserisci N: ");
  scanf("%d",&N);
  for (i=1; i<=N; i++) // Leggi le N rilevazioni
    scanf("%d",&temp[i]);
  GiornoPiuCaldo=1; // Calcola il massimo
  for (i=2; i<=N; i++)
    if (temp[i]>temp[GiornoPiuCaldo])
      GiornoPiuCaldo=i;
  // Visualizza il massimo
  printf("Il giorno piu` caldo e` %d\n",GiornoPiuCaldo);
}
```

Dimensione fisica: 32 (da 0 a 31)

Dimensione logica: N (da 1 a N)



# ***DIMENSIONE FISICA VS. LOGICA: ES CON 2 ARRAY***

---

- *È data una serie di rilevazioni di temperature espresse in gradi Celsius, corrispondenti ai giorni di un mese*
- *Scrivere un programma che legga da tastiera il numero N di dati da inserire, poi legga le N rilevazioni.*
- *Si ricopino su un secondo array le sole temperature positive*
- *Infine, si visualizzi il secondo array (delle temperature positive)*

12	6	-2	0	8	-4	2
----	---	----	---	---	----	---

12	6	8	2
----	---	---	---



# Codifica

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

Dimensione fisica: 32 (da 0 a 31)

```
#define DIM 32 // Definisci un array di dim sufficiente
```

```
main()
```

```
{ int temp[DIM], pos[DIM];
```

```
int N, Npos=1, i;
```

```
printf("Inserisci N: ");
```

```
scanf("%d",&N);
```

Dimensione logica  
array temp: N (da 1 a  
N)

```
for (i=1; i<=N; i++) // Leggi le N rilevazioni
```

```
scanf("%d",&temp[i]);
```

```
for (i=1; i<=N; i++)
```

```
if (temp[i]>0)
```

```
{ pos[Npos]=temp[i];
```

```
Npos++;
```

```
}
```

Dimensione logica  
array pos: Npos

```
for (i=1; i<Npos; i++) // Visualizza l'array pos
```

```
printf("%d ",pos[i]);
```

```
}
```



# Esercizi

---

- ***Scrivere un programma che richieda in ingresso N numeri interi e li stampi dal più piccolo al più grande***
- ***Scrivere un programma che richieda in ingresso i voti ottenuti agli esami da uno studente (terminando quando viene inserito il voto -1) e scriva il voto minimo, il voto massimo, la media e i voti più frequenti. Si tenga presente che possono essere registrati anche voti insufficienti. Si ignorino eventuali lodi.***
- ***Scrivere un programma che richieda in ingresso un massimo di DIMENSIONE numeri interi positivi (fermandosi quando viene inserito un numero negativo) e stampi il numero minimo, il numero massimo, la media aritmetica e i numeri più frequenti.***



# Stringhe

---

- *Finora sappiamo leggere, scrivere, elaborare singoli caratteri*
- *Avremmo bisogno anche di elaborare parole, frasi, ...*
- *In C, si possono usare le **stringhe**, che sono un caso particolare di **array di caratteri**.*
- *Per quello che sappiamo finora, possiamo leggere un array di caratteri uno alla volta:*

```
int i;  
char s[10];  
for (i=0; i<10; i++)  
    scanf("%c",&s[i]);
```

- *e stamparlo con*

```
for (i=0; i<10; i++)  
    printf("%c",s[i]);
```

- *Però così posso leggere/scrivere solo un numero predefinito di caratteri*

# Array di caratteri

---

- *Se non so quanti sono a priori i caratteri, potrei usare una variabile per contenere la lunghezza*

```
int lung, i;
char s[100];
printf("Immetti il numero di caratteri:");
scanf("%d", &lung);
printf("immetti i caratteri");
for (i=0; i<lung; i++)
    scanf("%c", &s[i]);
for (i=0; i<lung; i++)
    printf("%c", s[i]);
```

# Array di caratteri

---

- *Oppure potrei usare un carattere speciale per indicare che i caratteri significativi finiscono lì*

```
int i;
char s[100];
printf("immetti i caratteri (@ per terminare):");
i=-1;
do
{
    i++;
    scanf("%c",&s[i]);
} while (s[i]!='@');
i=0;
while (s[i]!='@')
{
    printf("%c",s[i]);
    i++;
}
```



# Stringhe

---

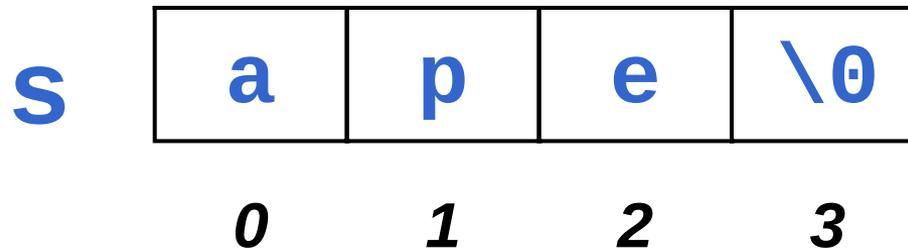
- *Nel linguaggio C si utilizza questa seconda visione.*
- *Viene utilizzato un **codice speciale** che non può comparire in nessuna stringa: il carattere con **codice ASCII 0**, indicato anche con: `'\0'`*
- *Come vedremo, se utilizziamo questa convenzione, il C ci fornisce alcune istruzioni comode già fatte (non dobbiamo ricostruirle noi).*



# STRINGHE: ARRAY DI CARATTERI

---

- **Una stringa di caratteri in C è un array di caratteri terminato dal carattere `'\0'`**



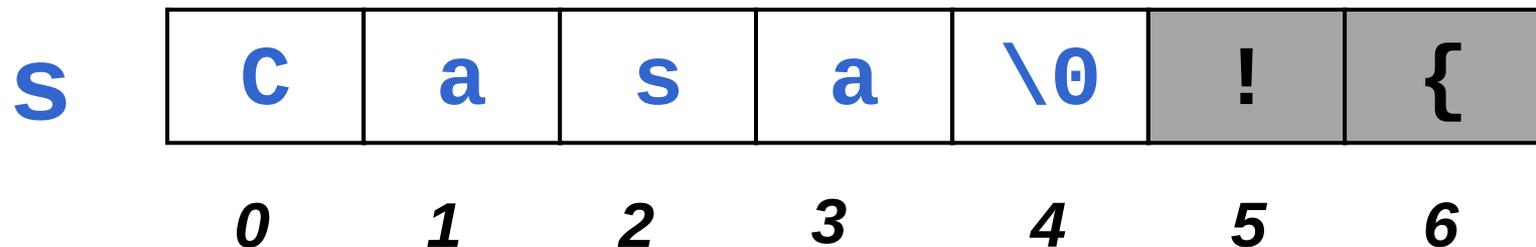
- **Un vettore di  $N$  caratteri può dunque ospitare stringhe lunghe al più  $N-1$  caratteri, perché una cella è destinata al terminatore `'\0'`.**



# STRINGHE: ARRAY DI CARATTERI

---

- *Un array di  $N$  caratteri può essere usato per memorizzare stringhe più corte*



- *In questo caso, le celle oltre la  $k$ -esima ( $k$  essendo la lunghezza della stringa) sono concettualmente vuote: praticamente sono inutilizzate e contengono un valore casuale.*

# STRINGHE: Inizializzazione

---

- Una stringa si può **inizializzare**, come ogni altro array, elencando le singole componenti:

```
char s[4] = {'a', 'p', 'e', '\0'};
```

- oppure anche, più brevemente, con la forma compatta seguente:

```
char s[4] = "ape" ;
```

*Il carattere di terminazione '\0' è automaticamente incluso in fondo.*

*Attenzione alla lunghezza!*



# STRINGHE: LETTURA E SCRITTURA

---

- *Una stringa si può leggere da tastiera e stampare, come ogni altro array, elencando le singole componenti:*

```
...
char str[4];
int i;
for (i=0; i < 3; i++)
    scanf("%c", &str[i]);
str[3] = '\0';
```

- *oppure anche, più brevemente, con la forma compatta seguente:*

```
...
char str[4];
scanf("%s", str);
```

Per motivi che studieremo più avanti, nella scanf non si usa la & per le stringhe

Quando voglio parlare della stringa nella sua totalità non metto le parentesi quadre

# ***Costanti di tipo stringa***

---

- ***In generale, le doppie virgolette rappresentano le **costanti di tipo stringa*****

```
main()
```

```
{ char s[10]="inizio";  
  printf("%s %s", s, "fine");  
}
```

***Sono quindi delle sequenze di caratteri con il terminatore in fondo***



# ESEMPIO

---

## Problema:

*Date due stringhe di caratteri, decidere quale precede l'altra in ordine alfabetico.*

## Rappresentazione dell'informazione:

- *poiché vi possono essere tre risultati ( $s1 < s2$ ,  $s1 == s2$ ,  $s2 < s1$ ), un boolean non basta*
- *possiamo usare:*
  - *due boolean (uguale e precede)*
  - *tre boolean (uguale,  $s1$  precede  $s2$ ,  $s2$  precede  $s1$ )*
  - *un intero (negativo, zero, positivo)*

*scegliamo la terza via.*



# CONFRONTO FRA STRINGHE

---

## Codifica:

```
main()
{
    char s1[] = "Maria";
    char s2[] = "Marta";
    int i=0, stato;
    while(s1[i]!='\0' && s2[i]!='\0' && s1[i]==s2[i])
        i++;
    stato = s1[i]-s2[i];
    .....
}
```

negativo  $\leftrightarrow$  s1 precede s2  
positivo  $\leftrightarrow$  s2 precede s1  
zero  $\leftrightarrow$  s1 è uguale a s2



# L'operatore [ ]

---

- **Le parentesi quadre si usano per**
  - **dichiarare una variabile array:**  
`char s[10];`
  - **identificare un elemento di una variabile array**  
`printf("%c", s[3]);`
- **Ovvero:**
  - **in fase di *dichiarazione***, mi dicono che quella variabile è un array
  - **in fase di *utilizzo***, mi dicono di prendere un certo elemento dell'array (e non tutto l'array)
- **Se voglio *utilizzare* l'intero array, non metto le parentesi quadre**



# Stringhe e caratteri

---

## Caratteri

- **definizione:**  
`char c;`
- **contenuto:**  
***un carattere***
- **costante:**  
***un carattere fra apici singoli: 'a'***
- **codice nelle stringhe formato: %c**

## Stringhe

- **definizione:**  
`char s[10];`
- **contenuto:**  
***una sequenza di caratteri, terminata dal carattere '\0'***
- **costante:**  
***una sequenza di caratteri fra apici doppi: "ciao"***
- **Codice nelle stringhe formato: %s**



# Esercizio

---

- **Dire se le seguenti dichiarazioni sono corrette**
  - `char a='a';`
  - `char b="ciao";`
  - `char s[]='a';`
  - `char t[]="ciao";`
  - `char v[10]="ciao";`
- **Dire se le seguenti istruzioni sono corrette, sapendo che vale la dichiarazione: `char c, s[5], t[5];`**
  - `printf("%c", c);`
  - `printf("%s", s);`
  - `printf("%c", s);`
  - `printf("%s", c);`
  - `s[5]=t[5];`
  - `printf("%c", s[5]);`
  - `printf("%s", s[5]);`
  - `printf("%c", s[0]);`
  - `printf("%s", s[0]);`
  - `c = s;`
  - `c = s[2];`
- **Nelle istruzioni scorrette, si dica se si ha errore di compilazione**



# ESEMPIO

---

## Problema:

***Data una stringa di caratteri, copiarla in un altro array di caratteri (di lunghezza non inferiore).***

## Ipotesi:

***La stringa è “ben formata”, ossia correttamente terminata dal carattere '\0'.***

## Specifica:

- ***scandire la stringa elemento per elemento, fino a trovare il terminatore '\0' (che esiste certamente)***
- ***nel fare ciò, copiare l'elemento nella posizione corrispondente dell'altro array.***



# ESEMPIO

Codifica: copia della stringa carattere per carattere

```
main()
```

```
{char s[] = "Nel mezzo del cammin di";
```

```
char s2[40];
```

La dimensione deve essere tale da garantire che la stringa non ecceda

```
int i=0;
```

```
for (i=0; s[i]!='\0'; i++)
```

```
    s2[i] = s[i];
```

```
    s2[i] = '\0';
```

Al termine, occorre garantire che anche la nuova stringa sia “ben formata”, inserendo esplicitamente il terminatore.

```
}
```



# ESEMPIO

---

## Perché non fare così?

```
main()
```

```
{char s[] = "Nel mezzo del cammin di";
```

```
char s2[40];
```

```
s2 = s;
```

```
}
```

**ERRORE DI COMPILAZIONE:**  
*left operand must be l-value*

**PERCHÉ GLI ARRAY NON POSSONO  
ESSERE MANIPOLATI NELLA LORO INTEREZZA !**

Vedremo più avanti il perché



# Esercizi

---

- ***Leggere da tastiera una stringa e dire qual è la sua lunghezza***
- ***Date due stringhe, mettere in una terza stringa la concatenazione delle due***
  - ***Es: char a[]="gian", b[]="luca", c[20];  
mettere nella stringa c: "gianluca")***
- ***Letti una stringa ed un carattere, verificare se il carattere compare nella stringa***
- ***Date due stringhe, verificare se una contiene l'altra (es "zio" è contenuta in "Tizio").***



# Libreria sulle stringhe: *strcpy* e *strcat*

---

- Il C ha una libreria sulle stringhe:

```
#include <string.h>
```

- Alcune istruzioni utili:

- `strcpy(Destinazione, Sorgente)`

- copia la stringa *Sorgente* sulla *Destinazione*

- es:

```
char d[6]="pippo", s[5]="ciao";
```

```
strcpy(d,s);
```

```
printf("%s",d); → stampa "ciao"
```

- `strcat(Destinazione, Sorgente)`

- aggiunge in fondo alla stringa *Destinazione* la *Sorgente*

- es:

```
char d[20]="gian", s[]="luca";
```

```
strcat(d,s);
```

```
printf("%s",d); → stampa "gianluca"
```



# *string.h: strcmp*

---

- `strcmp(Stringa1, Stringa2)`
- *fornisce un valore*
  - $=0$  se le due stringhe sono uguali
  - $<0$  se *Stringa1* viene prima di *Stringa2* in ordine alfabetico
  - $>0$  se *Stringa2* viene prima di *Stringa1*
- *Es*

```
char s1[10], s2[10]; int diverse;
scanf("%s", s1);
scanf("%s", s2);
diverse = strcmp(s1, s2);
if (diverse!=0)
    if (diverse>0)
        printf("%s precede %s", s2, s1);
    else printf("%s precede %s", s1, s2);
else printf("sono uguali");
```



# ***string.h: strlen***

---

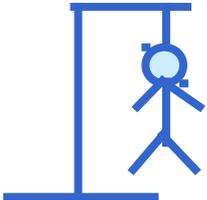
- **strlen(Stringa)**
- ***fornisce la lunghezza della Stringa***
- ***Altre istruzioni e funzioni possono essere trovate nell'help del Visual Studio (cercare "string.h")***



# Esercizio

---

- **Inizialmente, si legga una parola da tastiera; poi un altro utente cerca di indovinare la parola selezionata. Per indovinare la parola, l'utente inserisce una lettera. Se la lettera appartiene alla parola, il programma visualizza in quali posizioni si trova la lettera. Altrimenti (se la lettera non compare nella parola), viene segnalato un tentativo di insuccesso. Dopo 5 tentativi infruttuosi, l'utente perde. Se, invece, riesce ad indovinare tutte le lettere della parola, vince.**



- **Variante: dopo ogni tentativo, scrivere la parola segreta, mostrando il carattere underscore (\_) al posto delle lettere non ancora indovinate.**
- **Ulteriore variante: mostrare in maiuscolo (per evidenziarla) l'ultima lettera tentata.**

# Strutture

---

- ***In molti casi, un oggetto del mondo esterno è rappresentato da più dati all'interno del calcolatore:***
  - ***In una rubrica telefonica, ogni persona ha nome, cognome, indirizzo, numero di telefono***
  - ***In un mazzo di carte, ogni carta ha***
    - ***numero (da 1 a 13)***
    - ***seme (cuori, quadri, fiori, picche)***
- ***A volte, può servire creare strutture dati che contengono più informazioni:***
  - ***numeri complessi: parte reale, parte immaginaria***
  - ***frazioni: numeratore, denominatore***

# Esempio: Note musicali

- **Vogliamo rappresentare all'interno del calcolatore una musica (es. suoneria del cellulare)**
- **Dovrò rappresentare tante note**
- **Ogni nota ha**
  - **nome della nota ("do", "do#", "re", ...)**
  - **ottava es 1, 2, ...**
  - **durata (ad es in ottavi: 1/8, 2/8, ...)**
  - **...**
- **Come rappresentarla?**



do re mi fa sol la si do re mi

1 1 1 1 1 1 1 2 2 2



# ***Es: note***

---

- ***Potrei usare una varabile per ogni caratteristica della nota***
- ***Per la prima nota ho***
  - ***nome = “do”***
  - ***ottava = 5***
  - ***durata = 4***
- ***Per la seconda nota ho***
  - ***nome = “sol#”***
  - ***ottava = 5***
  - ***durata = 2***

# Codifica

---

- ***Potrei usare 6 variabili: nome1, ottava1, durata1, nome2, ottava2, durata2***
- ***Poi sono io che mi devo ricordare che nome1 è il nome della prima nota, etc.***

```
main()
```

```
{ char nome1[5], nome2[5];
```

```
  int ottava1, ottava2;
```

```
  int durata1, durata2;
```

```
...
```

```
}
```

# ***Codifica: aggiungo una nota***

---

- ***Però se mi accorgo che ho bisogno di un'altra nota, devo definire tutte le variabili che la compongono ☹***
  - ***noioso***
  - ***facile dimenticarsene qualcuna***

```
main()  
{ char nome1[5], nome2[5], nome3[5];  
  int ottava1, ottava2, ottava3;  
  int durata1, durata2, durata3;  
  ...  
}
```

# Codifica: aggiungo una nota

- *Se poi ho bisogno di uno spartito, devo farne un array 😞*
  - *ancora più complicato*

```
#define N 100
```

```
main()
```

```
{ char nome1[5], nome2[5], nome3[5], spartitoNome[N]  
  [5];
```

```
  int ottava1, ottava2, ottava3, spartitoOttava[N];
```

```
  int durata1, durata2, durata3, spartitoDurata[N];
```

```
  ...
```

```
}
```

# ***Sarebbe meglio ...***

---

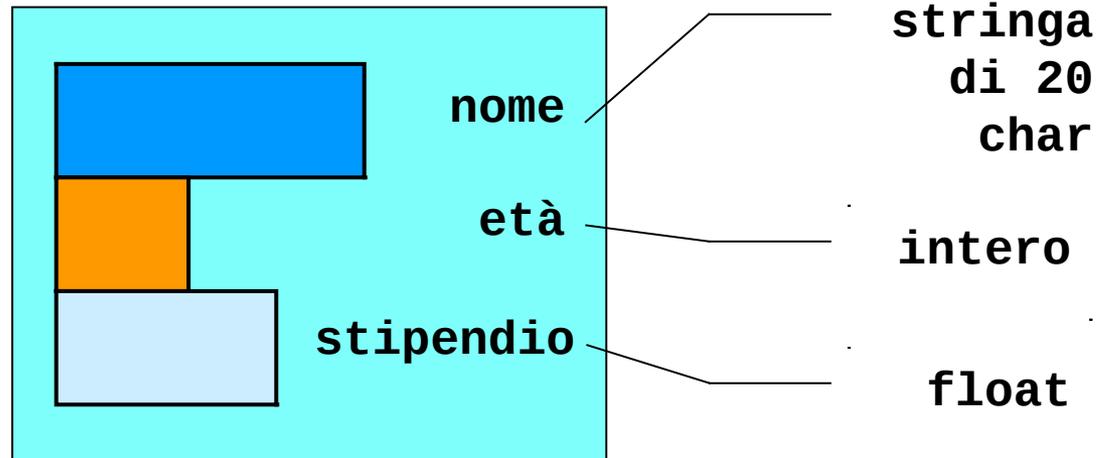
- ***Sarebbe meglio poter definire un nuovo tipo di dato: la nota***
  - ***poi potrei definire variabili ed array di quel tipo:***  
`nota n;`  
`nota spartito[100];`
- ***poi dovrei avere un modo per identificare le singole caratteristiche delle note. Ho bisogno di scrivere frasi del tipo***
  - ***assegna al nome di n il valore "fa#"***
  - ***se la durata di n è maggiore della durata di spartito[4] ...***



# STRUTTURE

- *Per raggruppare dati di tipo diverso ma che logicamente devono essere insieme, esistono le strutture*
- **Una struttura è una collezione finita di variabili non necessariamente dello stesso tipo, ognuna identificata da un nome.**

**struct  
persona**





# STRUTTURE

---

*Definizione di una **variabile di tipo** struttura:*

```
struct [<etichetta>  
  
{  
    { <definizione-di-variabile> }  
}  
<nomeStruttura> ;
```



# ESEMPIO

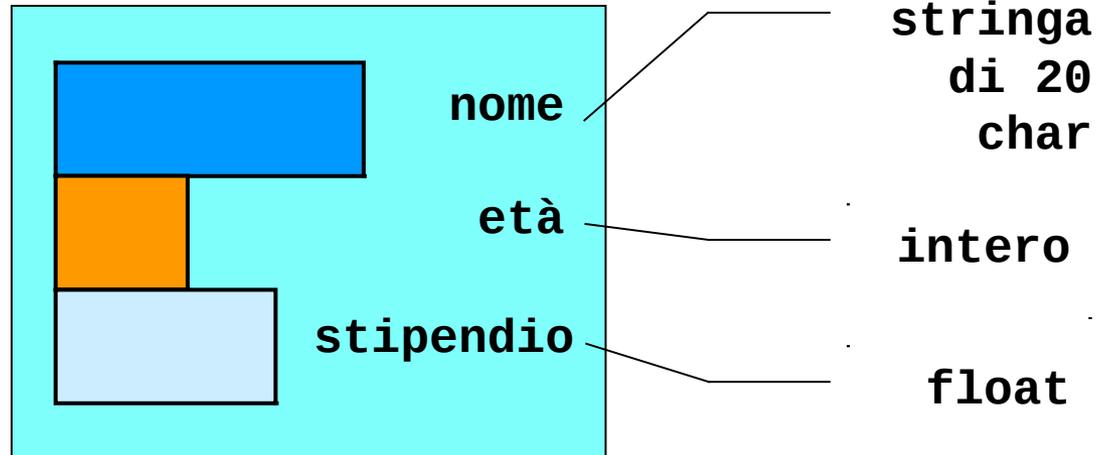
Etichetta

```
struct persona  
{ char nome[20];  
  int eta;  
  float stipendio;  
} pers ;
```

Definisce una variabile pers strutturata nel modo illustrato.

variabile

struct  
persona

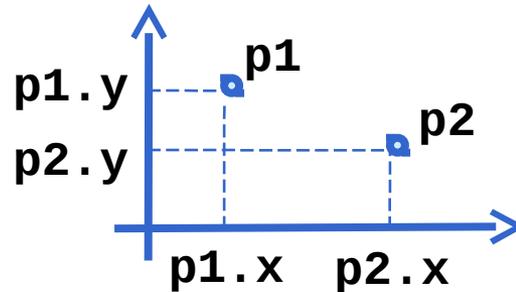




# ESEMPIO

```
struct punto  
{ int x, y;  
} p1, p2 ;
```

*p1 e p2 sono strutture fatte ciascuna da due campi interi di nome x e y*



```
struct data
```

```
{ int giorno, mese, anno;  
} d ;
```

*d è costituita da tre campi interi di nome giorno, mese e anno*





# STRUTTURE: Notazione puntata

---

- *Una volta definita una variabile struttura, si accede ai singoli campi mediante la **notazione puntata**.*

*Ad esempio:*

```
struct nota
{ char nome[5];
  int ottava;
  int durata;
} n1, n2;
```

Ogni campo si usa come una normale variabile del tipo corrispondente al tipo del campo.

```
n1.ottava = 5;
if (n1.ottava > n2.ottava)
{ ... }
n2.durata = n1.durata/2;
```

# ***Esercizio***

---

- ***Si leggano da tastiera due strutture “data” e si dica quale delle due precede l’altra in ordine cronologico***
- ***Suggerimento: si crei una variabile intera dif tale che***
  - ***dif = 0 se le date sono uguali***
  - ***dif < 0 se la prima data precede la seconda***
  - ***dif > 0 se la seconda data precede la prima***



# Esempio

---

```
struct nota
{
    char nome[5];
    int ottava;
    int durata;
} n;
```

- ***Una volta definita una variabile con una etichetta, si possono definire altre variabili usando la stessa etichetta***

**struct nota tonica, spartito[100];**

- ***però non è ancora esattamente come definire un nuovo tipo (devo metterci comunque struct)***



# ***Esercizio***

---

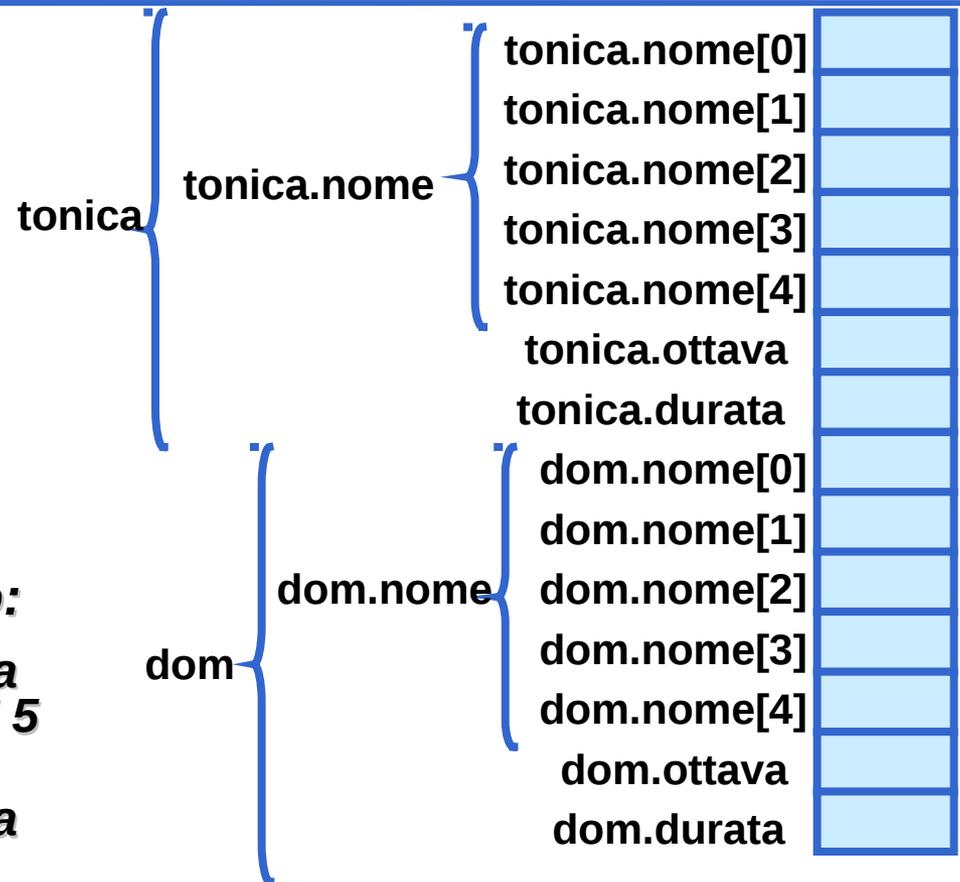
- ***Si definiscano due variabili di tipo struttura “numero complesso”***
- ***Si leggano da tastiera due numeri complessi***
- ***Si calcoli e si visualizzi il loro prodotto***



# Rappresentazione

```
typedef struct
{ char nome[5];
  int ottava;
  int durata;
} nota;
nota tonica, dom;
```

- **è come se avessimo creato:**
  - **una variabile che si chiama n1.nome, di tipo stringa di 5 caratteri**
  - **una variabile che si chiama n1.ottava → int**
  - **una variabile che si chiama n1.durata → int**
  - **una var che si chiama n2.durata → int**
  - ...



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

```
main()
```

```
{ struct punto{float x,y;} P;
```

```
  unsigned int op;
```

```
  float Dx, Dy;
```

```
  printf("ascissa? "); scanf("%f",&P.x);
```

```
  printf("ordinata? "); scanf("%f",&P.y);
```

```
  printf("operazione(0,1,2,3)?\n");
```

```
  scanf("%d",&op);
```

```
  if (op==1) P.y= 0;
```

```
  if (op==2) P.x= 0;
```

```
  if (op==3)
```

```
  {printf("%s", "Traslazione?");
```

```
    scanf("%f%f",&Dx,&Dy);
```

```
    P.x=P.x+Dx;
```

```
    P.y=P.y+Dy;
```

```
  }
```

```
  printf("%s\n", "nuove coordinate sono");
```

```
  printf("%f%s%f\n", P.x, " ", P.y);
```

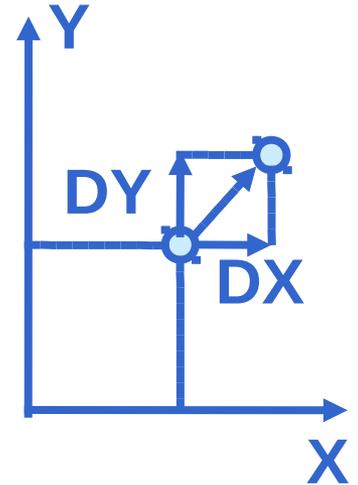
```
}
```



# ESEMPIO

**PROBLEMA:** leggere le coordinate di un punto in un piano e modificarle a seconda dell'operazione richiesta:

- proiezione sull'asse X
- proiezione sull'asse Y
- traslazione di  $DX$  e  $DY$



## Specifica:

- leggere le coordinate di input e memorizzarle in una struttura
- leggere l'operazione richiesta
- effettuare l'operazione
- stampare il risultato



## ***TIPI DEFINITI DALL'UTENTE***

---

- ***In C, l'utente può introdurre nuovi tipi tramite una **definizione di tipo*****
- ***La definizione associa a un identificatore (nome del tipo) un tipo di dato***
  - ***aumenta la leggibilità del programma***
  - ***consente di ragionare per astrazioni***
- ***Il C consente, in particolare, di:***
  - ***ridefinire tipi già esistenti***
  - ***definire dei nuovi tipi enumerativi***
  - ***definire dei nuovi tipi strutturati***

# TIPI RIDEFINITI

---

- *Un nuovo identificatore di tipo viene dichiarato identico a un tipo già esistente*
- *Schema generale:*

```
typedef TipoEsistente NuovoTipo ;
```

- *Esempio*

```
typedef int MioIntero;
```

```
MioIntero X,Y,Z;
```

```
int W;
```



# Definizione di nuovi tipi

---

- *Per definire un nuovo tipo di dato, si utilizza la typedef*

**typedef** <TipoEsistente> <NuovoTipo>;

- *Es:*

```
typedef struct
    { char nome[5];
      int ottava;
      int durata;
    } nota;
```

- *Ora posso usare nota esattamente come utilizzo i tipi predefiniti*

```
nota n, spartito[100];
```

# TIPI ENUMERATIVI

---

- *Un tipo enumerativo viene specificato tramite l'elenco dei valori che i dati di quel tipo possono assumere.*

- *Schema generale:*

```
typedef enum  
{ a1, a2, a3, ... , aN } EnumType;
```

- *Il compilatore associa a ciascun “identificativo di valore” a1,...,aN un numero naturale (0,1,...), che viene usato nella valutazione di espressioni che coinvolgono il nuovo tipo.*

# TIPI ENUMERATIVI

---

- Gli “identificativi di valore” **a1, ..., aN** sono a tutti gli effetti delle nuove costanti.
- *Esempi*

```
typedef enum  
{ lun, mar, mer, gio, ven, sab, dom } Giorni;
```

```
typedef enum  
{ cuori, picche, quadri, fiori } Carte;
```

```
Carte C1, C2, C3, C4, C5;
```

```
Giorni Giorno;
```

```
if (Giorno == dom) /* giorno festivo */
```

```
else /* giorno feriale */
```

# TIPI ENUMERATIVI

---

- Un “*identificativo di valore*” può comparire **una sola volta** nella definizione di **un solo tipo**, altrimenti si ha ambiguità.
- *Esempio*

```
typedef enum  
{lun, mar, mer, gio, ven, sab, dom} Giorni;
```

```
typedef enum  
{gen, feb, mar, apr, mag, giu, lug, ago, set, ott  
, nov, dic} Mesi;
```

*La definizione del secondo tipo enumerativo è scorretta, perché l'identificatore **mar** è già stato usato altrove.*

# TIPI ENUMERATIVI

---

- *Un tipo enumerativo è **totalmente ordinato**: vale l'ordine con cui gli identificativi di valore sono stati elencati nella definizione.*
- *Esempio*

```
typedef enum  
{lun, mar, mer, gio, ven, sab, dom} Giorni;
```

*Data la definizione sopra,*

`lun < mar`      *è vera*

`lun >= sab`     *è falsa*

*in quanto* `lun ↔ 0`, `mar ↔ 1`, `mer ↔ 2`, *etc.*

# TIPI ENUMERATIVI

---

- *Poiché un tipo enumerativo è, per la macchina C, indistinguibile da un intero, è possibile in linea di principio mischiare interi e tipi enumerativi*

- *Esempio*

```
typedef enum
{lun, mar, mer, gio, ven, sab, dom} Giorni;
Giorni g;
g = 5;    /* equivale a g = sab */
```

- **È una pratica da evitare ovunque possibile!**

# TIPI ENUMERATIVI

---

- È anche possibile specificare i valori naturali cui associare i simboli **a1, . . . , aN**

- *qui, lun ↔ 0, mar ↔ 1, mer ↔ 2, etc.:*

```
typedef enum {lun, mar, mer, gio, ven, sab, dom}  
Giorni;
```

- *qui, invece, lun ↔ 1, mar ↔ 2, mer ↔ 3, etc.:*

```
typedef enum {lun=1, mar, mer, gio, ven, sab, dom}  
Giorni;
```

- *qui, infine, lun ↔ 1, mar ↔ 2, mer ↔ 7, gio ↔ 8...:*

```
typedef enum { lun=1, mar, mer=7, gio, ven,  
sab, dom} Giorni;
```

# IL TIPO BOOLEAN

---

- *Il boolean non esiste in C, ma si può facilmente definirlo:*

```
typedef enum { false, true } Boolean;
```

- *così:*

**false ↔ 0, true ↔ 1**

**false < true**

- *logica positiva*



# Tipi e variabili

tipo

```
int a;
```

variabile

```
char c[10];
```

- **Ad un tipo possono essere associate variabili**
  - **Ad un tipo non è associata alcuna area di memoria**
  - **Un tipo non ha un indirizzo, né un valore. Non posso assegnargli un valore. Non posso stampare il suo valore.**  
es.:
- **Una variabile ha sempre un tipo**
  - **Ad una variabile è associata un'area di memoria**
  - **essa ha un indirizzo e contiene un valore**
  - **La quantità di memoria associata ad una variabile dipende dal suo tipo**

```
int = 7;  
char[8] = 'A';  
printf("%f", float);
```

**ERRORE!!!**



# Tipi e variabili

tipo

```
typedef struct {char  
nome[5]; int durata;} nota
```

variabile

```
nota n1, brano[20];
```

- *Ad un tipo possono essere associate variabili*
- *Ad un tipo non è associata alcuna area di memoria*
- *Un tipo non ha un indirizzo, né un valore. Non posso assegnargli un valore. Non posso stampare il suo valore.*  
es.:
- *Una variabile ha sempre un tipo*
- *Ad una variabile è associata un'area di memoria*
- *essa ha un indirizzo e contiene un valore*
- *La quantità di memoria associata ad una variabile dipende dal suo tipo*

```
nota.durata = 7;  
nota.nome[8] = 'A';  
printf("%s", nota.nome);
```

**ERRORE!!!**



# **Esercizio**

---

- ***Un array di strutture contiene i risultati degli esami di Programmazione (voto da 0 a 31).***
- ***Per ogni studente che sostiene l'esame viene riportato***
  - ***Nome***
  - ***Matricola***
  - ***Voto***
- ***Si scriva un programma che visualizza il Nome degli studenti che hanno preso almeno 18.***



# Esercizio

---

- ***Dato l'array di strutture dell'esercizio precedente, si stampi la frequenza di ogni voto (cioè per ogni possibile voto quanti studenti hanno avuto quel voto)***
- ***18 → 5 studenti***
- ***19 → 7 studenti***
- ***20 → 10 studenti***
- ***...***



# ***Esercizio***

---

- ***Un array di strutture contiene l'orario delle lezioni del lunedì. Per ogni lezione si hanno***
  - ***nome del corso (stringa di 20 caratteri, incluso terminatore)***
  - ***ora di inizio (intero)***
  - ***durata (intero)***
- ***Il numero delle lezioni è contenuto in una variabile NL***
- ***Si legga da tastiera una nuova lezione (nome, ora, durata) e si mostrino a video le lezioni che si sovrappongono con questa***



# Esempio

nome	inizio	durata
Programmazione	14	2
Matematica	11	2
Inglese	8	3

- ***Nuova lezione:***
  - ***nome: "Fisica"***
  - ***inizio: 12***
  - ***durata: 3***

si sovrappone con  
Programmazione  
Matematica



# Esercizio

---

- Sapendo che

```
typedef struct
{ char nome[5];
  int ottava;
  int durata;
} nota;

main()
{ nota tonica, dom; ... }
```

*dire quali delle seguenti operazioni sono corrette, quali danno errore di compilazione, quali errore a tempo di esecuzione*

- dom.ottava=1
- strcpy(tonica, "do#")
- strcpy(tonica.nome, "do#")
- strcpy(tonica.nome, nota.nome)
- tonica.ottava=dom.nome[2]
- tonica[2].nome='#'
- nota.ottava=nota.durata
- dom.nome=1
- tonica.nome[1]='\0'
- strcpy(tonica.nome, dom.nome)
- tonica.nome[5]=dom.nome[5]
- nota.ottava++
- dom.nome++
- tonica.ottava++