

Array e Strutture

Prof. Marco Gavanelli

Dipartimento di Ingegneria

Università di Ferrara

1

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!

2 2

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

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

3

TIPI DI DATO: ESEMPIO

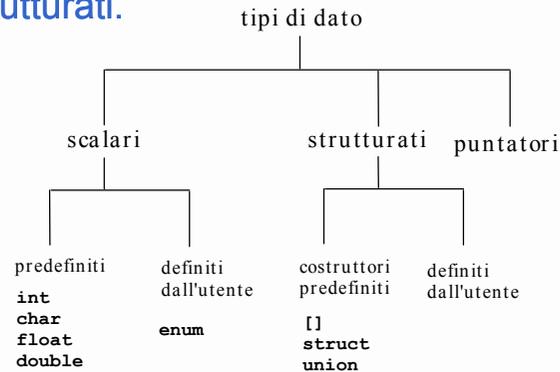
Il *tipo di dato* **INTERO** è definito come:

- 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...

4

TIPI DI DATO

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



5

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**

6

TIPI DI DATO STRUTTURATO



In C si possono definire tipi strutturati.

Costruttori fondamentali:

[] (array)

struct (strutture)

union (unioni)

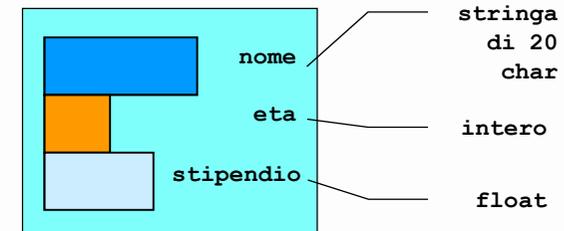
7

STRUTTURE



Una **struttura** è una collezione finita di variabili non necessariamente dello stesso tipo, ognuna identificata da un nome.

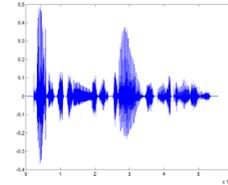
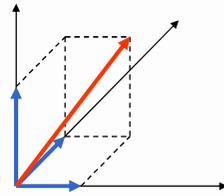
struct
persona



8

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
Marco	Via Saragat 1	4833
Giacomo	Via Saragat 1	2009
...

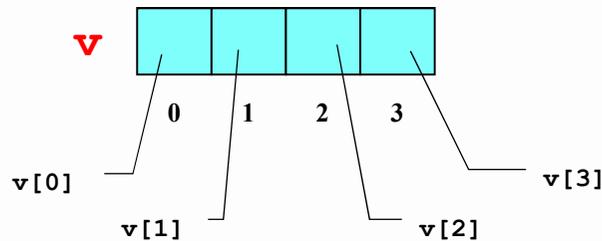
$$\det \begin{pmatrix} 1 & 2 & 1 \\ 3 & 2 & 0 \\ 2 & 0 & 1 \end{pmatrix}$$

9



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 $N-1$



11

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);
```

10



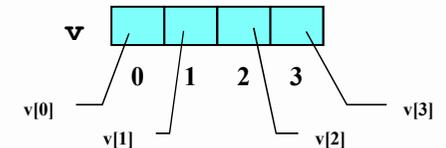
ARRAY (VETTORI)

Definizione di una variabile di tipo array:

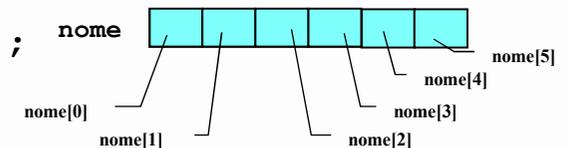
```
<tipo> <nomeArray> [ <costante> ] ;
```

Esempi:

```
int v[4];
```



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



12

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
```

13

inizializzare un vettore con il quadrato degli indici



A	0	1	4	9	16	25	36	49	64
	0	1	2	3	4	5	6	7	8

14

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]);  
}
```

15

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

16

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'elemen
    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)

17

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: #



18

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)

19

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;
}
```

20

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]);
}
```

21

#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 ';'

22

#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;];
  // 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]);
}
```

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

missing ')' before ';'

23



ESEMPIO

Problema: leggere da tastiera gli elementi di un vettore

```
#include <stdio.h>
#define N 3
main()
{ int k;
  int A[N];
  for(k=0; k < N; k++)
  { printf("Dammi elemento %d: ", k);
    scanf("%d", &A[k]);
  }
}
```

24



ESEMPIO

Problema: scrivere un programma che, dato un vettore di N interi, determini il valore massimo.

Specifica di I livello:

Inizialmente, si assuma come massimo di tentativo il primo elemento. $m_0 = v[0] \rightarrow m_0 \geq v[0]$

Poi, si confronti via via il massimo di tentativo con gli elementi del vettore: nel caso se ne trovi uno maggiore del massimo di tentativo attuale, si aggiorni il valore del massimo.

$m_i = \max(m_{i-1}, v[i]) \rightarrow m_i \geq v[0], v[1] \dots v[i]$

Al termine, il valore del massimo di tentativo coincide col valore massimo ospitato nel vettore. $m_{n-1} \geq v[0], v[1] \dots v[n-1]$ cioè m_{n-1} è il max cercato.

25

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"

12	6	2	0	8	4	2
0	1	2	3	4	5	6

26

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

27

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*!

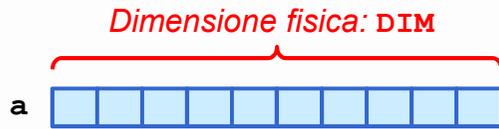
28



DIMENSIONE FISICA

- ... e se ho bisogno di memorizzare un numero di dati non noto a priori?
- Un array è una collezione finita di celle; questo non significa che si debbano per forza usare sempre tutte!

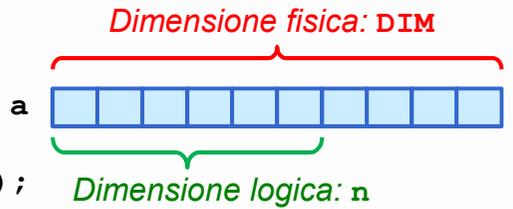
```
#define DIM 10
...
int a[DIM];
```




DIMENSIONE FISICA VS. LOGICA

- Spesso la porzione di array realmente utilizzata dipende dai dati di ingresso
- La **Dimensione Logica** di un array può essere inferiore alla sua **Dimensione Fisica**

```
#define DIM 10
...
int a[DIM];
printf("Num. dati?");
scanf("%d", &n);
for (i=0; i<n; i++)
    scanf("%d", &a[i]);
```



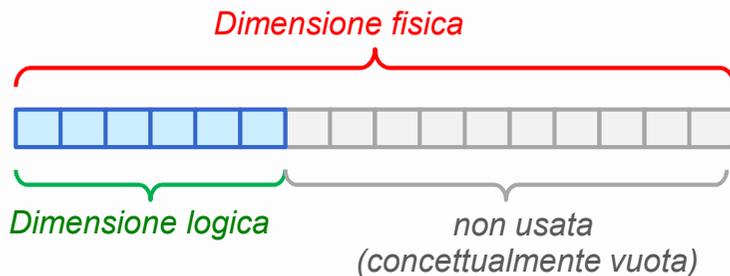
• È responsabilità del programmatore mantenere la **dimensione logica** di un array

• In genere, viene salvata in una variabile apposita



DIMENSIONE FISICA VS. LOGICA

- La **Dimensione Logica** di un array può essere inferiore (mai superiore!) alla sua **Dimensione Fisica**



- Quindi: se non sappiamo esattamente quanti sono i dati da inserire, scegliamo una dimensione fisica sufficientemente grande, poi useremo solo una parte dell'array



DIMENSIONE FISICA VS. LOGICA

Esempio

È data una serie di rilevazioni di temperature espresse in gradi Kelvin.

La serie è composta di al più 10 valori, ma può essere più corta. Il valore "-1" indica che la serie delle temperature è finita.

Scrivere un programma che, data una serie di temperature memorizzata in un vettore, calcoli la media delle temperature fornite.



ESEMPIO

- Il vettore deve essere dimensionato per 10 celle (caso peggiore)...
- ... ma la porzione realmente usata può essere minore!

Specifica di I livello:

- calcolare la somma di tutti gli elementi del vettore, e nel frattempo contare quanti sono
- il risultato è il rapporto fra la somma degli elementi così calcolata e il numero degli elementi.

33



Codifica

```
#include <stdio.h>
#define DIM 10
main()
{ int k=0, v[DIM], s=0;
  float media;
  do // lettura dei dati da tastiera
  { printf("Inserisci temp: ");
    scanf("%d",&v[k]);
    k++;
  } while (v[k-1]>0 && k<DIM);
  for (k=0; k<DIM && v[k]>=0; k++)
    s += v[k];
  media = s / (float)k;
  printf("media=%f\n",media);
}
```

Dimensione fisica = 10

Dimensione logica stabilita dall'utente

Condizione di prosecuzione del ciclo: la serie di dati non è finita ($v[k] \geq 0$) e ci sono ancora altre celle nell'array ($k < DIM$)

35



ESEMPIO

Specifica di II livello:

Inizialmente, poni uguale a 0 una variabile S che rappresenti la somma corrente e poni uguale a 0 un indice k che rappresenti l'elemento corrente

$$s_0 = 0, k_0 = 0$$

A ogni passo, aggiungi l'elemento corrente a S e incrementa k

$$\begin{aligned} s_{k+1} &= s_k + v[k_k], \\ k_{k+1} &= k_k + 1, \quad k < N \end{aligned}$$

Al termine (quando o un elemento vale -1, oppure hai esaminato N elementi), l'indice K rappresenta il numero totale di elementi: il risultato è il rapporto S/K.

$$\begin{aligned} s_N &= s_{N-1} + v[N-1], \\ k_N &= N \end{aligned}$$

34



DIMENSIONE FISICA VS. LOGICA

Esempio

- 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 legga le N rilevazioni.
- Infine, si visualizzi il giorno del mese in cui è stata rilevata la temperatura massima

36

Primo livello di specifica



- Definisci un array di dimensione sufficiente
 - Se non ci dicono quanti sono i dati, come faccio a sapere se è sufficiente?
 - Decido io una dimensione massima. Nel caso del mese, al massimo saranno 31 giorni
- Leggi N
- Leggi le N rilevazioni
- Calcola il massimo ed il giorno in cui si è verificato
 - stabilisci come giorno di temp massima temporaneo il primo giorno del mese: `GiornoPiuCaldo=1`
 - per ogni giorno da 2 a N
 - se questo giorno è stato più caldo del `GiornoPiuCaldo`, aggiorna `GiornoPiuCaldo`
- Visualizza `GiornoPiuCaldo`

37

Codifica



```
#include <stdio.h>
#define DIM 32 // Definisci un array di dim sufficiente
main()
{ int temp[DIM]; // Dimensione fisica: 32 (da 0 a 31)
  int N, GiornoPiuCaldo, i; // Dimensione logica: N (da 1 a N)
  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);
}
```

38

DIMENSIONE FISICA VS. LOGICA: ES CON 2 ARRAY



- Scrivere un programma che legga da tastiera un numero N di dati da inserire (si sa che N è al massimo 10), poi si legga un array di N interi.
- Si ricopino su un secondo array i soli valori positivi; i dati nell'array devono essere in celle consecutive
- Infine, si visualizzi il secondo array

12	6	-2	0	8	-4	2
0	1	2	3	4	5	6

12	6	8	2
0	1	2	3

39

Codifica



```
#include <stdio.h>
#define DIM 10 // Dimensione fisica: 10 (da 0 a 9)
main()
{ int A[DIM], pos[DIM]; // Dimensione logica array A: N (da 0 a N-1)
  int N, Npos=0, i;
  printf("Inserisci N: ");
  scanf("%d", &N);
  for (i=0; i<N; i++) // Leggi gli N valori
    scanf("%d", &A[i]);
  for (i=0; i<N; i++) // Dimensione logica array pos: Npos
    if (A[i]>0)
    { pos[Npos]=A[i];
      Npos++;
    }
  for (i=0; i<Npos; i++) // Visualizza l'array pos
    printf("%d ",pos[i]);
}
```

40

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

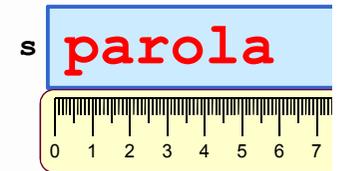
```
for (i=0; i<10; i++)
    printf("%c", s[i]);
```
- Però così posso leggere/scrivere solo un numero predefinito di caratteri

41

Array di caratteri

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

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



lung 6

42

Array di caratteri

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

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



43

Stringhe



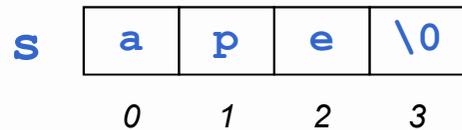
- Nel linguaggio C si utilizza questa seconda visione.
- Viene utilizzato un **codice speciale** che non può comparire in nessuna stringa: il carattere con **codice ASCII 0**, indicato anche con: `'\0'`
- Come vedremo, se utilizziamo questa convenzione, il C ci fornisce alcune istruzioni comode già fatte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

44



STRINGHE: ARRAY DI CARATTERI

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



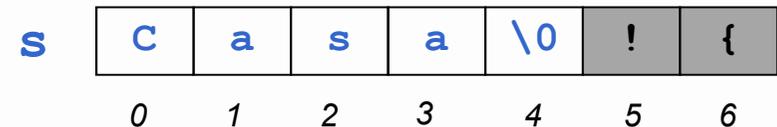
- Un vettore di N caratteri può dunque ospitare stringhe lunghe al più N-1 caratteri, perché una cella è destinata al terminatore '\0'.

45



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 (dove k è la lunghezza della stringa) sono concettualmente vuote: praticamente sono inutilizzate e contengono un valore casuale.

46

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!

47



STRINGHE: LETTURA E SCRITTURA

- Una stringa si può leggere da tastiera (o 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

48

Costanti di tipo stringa

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

```
main()      i n i z i o \0
{ char s[10]="inizio";
  printf("%s %s",s,"fine");
}  % s % s \0      f i n e \0
```

Sono quindi delle sequenze di caratteri con il terminatore in fondo

49

Quine

```
main()
{char c[]="main(){char c[]={%c%s%c;printf(c,34,c,34);}";
  printf(c,34,c,34);
}
```

50

Esempio

- Date due stringhe di caratteri, decidere quale precede l'altra in ordine alfabetico
- L'algoritmo deve produrre un intero
 - =0 se le stringhe sono uguali
 - <0 se la prima stringa viene prima della seconda
 - >0 se la seconda stringa viene prima della prima

```
main()
{ char s1[10],s2[10];
  int ris;
  scanf("%s %s",s1,s2);
  ...
  if (ris==0) printf("Uguali");
  else if (ris<0) printf("%s prima di %s",s1,s2);
  else printf("%s prima di %s",s2,s1);
```

51

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.

52



ESEMPIO

Specifica:

- scandire uno a uno gli elementi di egual posizione delle due stringhe, o fino alla fine delle stringhe, o fino a che se ne trovano due diversi
 - nel primo caso, le stringhe sono uguali
 - nel secondo, sono diverse
- nel secondo caso, confrontare i due caratteri così trovati, e determinare qual è il minore
 - la stringa a cui appartiene tale carattere precede l'altra

53



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 ↔ s1 precede s2
positivo ↔ s2 precede s1
zero ↔ s1 è uguale a s2

54



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**, dicono che quella variabile è un array
 - in fase di **utilizzo**, dicono di selezionare un certo elemento dell'array (e non tutto l'array)
- Se si vuole **utilizzare** l'intero array, non si mettono le parentesi quadre

55



Stringhe e caratteri

	Caratteri	Stringhe
Definizione	<code>char c;</code>	<code>char s[10];</code>
Contenuto	un carattere	una sequenza di caratteri, terminata dal carattere '\0'
Costante	un carattere fra apici singoli: 'a'	una sequenza di caratteri fra apici doppi: "ciao"
codice nelle stringhe formato	<code>%c</code>	<code>%s</code>

56

Esercizio



- Dire se le seguenti dichiarazioni sono corrette
 - `char a='a';`
 - `char b="ciao";`
 - `char v[10]="ciao";`
 - `char s[4]='a';`
- 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("%s", c);`
 - `s[5]=t[5];`
 - `printf("%c", s[5]);`
 - `printf("%s", s[5]);`
 - `printf("%s", s[0]);`
 - `c = s;`
 - `c = s[2];`
- Nelle istruzioni scorrette, si dica se si ha errore di compilazione

57

COPIA DI UNA STRINGA



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.

58

COPIA DI UNA STRINGA



Codifica: copia della stringa carattere per carattere

```
main()
{
  char s[] = "Nel mezzo del cammin di";
  char s2[40];
  int i=0;
  while (s[i]!='\0')
  {
    s2[i] = s[i];
    i++;
  }
  s2[i] = '\0';
}
```

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

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

59

COPIA DI UNA STRINGA



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 INTERESSA !

Vedremo più avanti il perché

60

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"`).

61

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"`

62

`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");
```

63

`string.h: strlen`

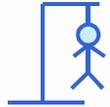


- `strlen (Stringa)`
- fornisce la lunghezza della `Stringa`
- Altre istruzioni e funzioni possono essere trovate nell'help del Visual Studio (cercare `"string.h"`)

64

Esercizio (adattato dal compito 13 set 2007) ★

- 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.



65

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?



67

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

66

Es: note

- Potrei usare una variabile 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

68

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;
  ...
}
```

69

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;
  ...
}
```

70

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];
  ...
}
```

71

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]* ...

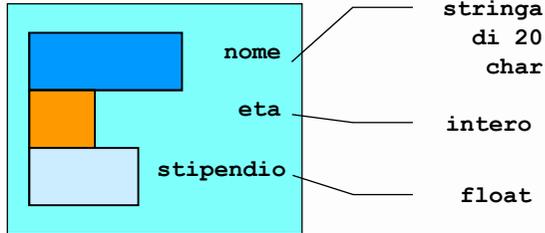
72

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



73

STRUTTURE



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

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

74

ESEMPIO

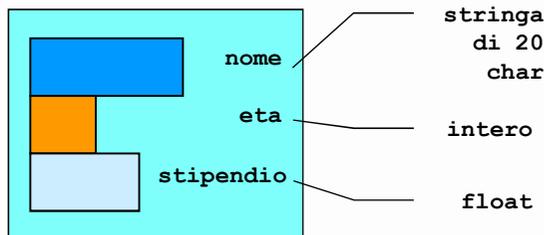


Etichetta

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

variabile

Definisce una variabile **pers** di tipo **struct persona** strutturata nel modo illustrato.



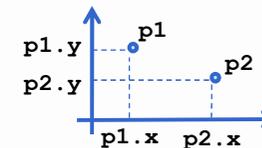
75

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**



76



STRUTTURE: Notazione puntata

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

Ad esempio:

```
struct persona
{ char nome[20];
  int eta;
  float stipendio;
} direttore, impiegato;
```

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

```
impiegato.eta = 30;
if (direttore.stipendio < impiegato.stipendio)
  direttore.stipendio = 2*impiegato.stipendio;
```

77

Esercizio

- Si leggano da tastiera due strutture "data" e si dica quale delle due precede l'altra in ordine cronologico
- Suggestione: 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

78

Esercizio: differenza orari

- Si leggano da tastiera due orari, ciascuno costituito da un numero intero di ore e di minuti, e li si inseriscano in opportune strutture
- Si calcoli il tempo trascorso tra il primo e il secondo orario (supponendo che il secondo sia successivo nel tempo)

```
Inserisci i due orari:
6 40
10 30
differenza: 3:50
```

79



Esempio

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

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

```
struct persona direttore, impiegati[100];
```

- Non è ancora esattamente come definire un nuovo tipo (bisogna comunque mettere la parola chiave **struct**)

80

Esercizio: compleanno

- Si legga da tastiera una struttura costituita da 3 campi interi contenente la data di oggi
- Si legga una struttura contenente il nome di una persona e la sua data di nascita
- Se oggi è il compleanno della persona, si stampi "Auguri " seguito dal nome della persona

81

STRUTTURE

```
main()
{ struct frutto
  { char nome[20]; int peso;
  } f1;
  struct frutto f2 ;
  ...
}
```

Non occorre ripetere l'elenco dei campi perché è implicito nell'etichetta *frutto*, che è già comparsa sopra.

82

Definizione di nuovi tipi



- Per definire un nuovo tipo di dato, si utilizza la `typedef`
`typedef <TipoEsistente> <NuovoTipo>;`
- Es:

```
typedef struct
{ char nome[20];
  int eta;
  float stipendio;
} persona;
```
- Ora posso usare `persona` esattamente come utilizzo i tipi predefiniti
`persona p, impiegati[100];`

83

Esercizio: compleanno

- Si modifichi il programma dell'esercizio precedente definendo i tipi tramite la `typedef`.
- Inoltre, l'utente dovrà inserire *il nome del mese*, invece del numero

```
inserisci data di oggi:
26 ottobre 2017
inserisci dati persona:
Gigi 1 giugno 2001
non e` il tuo compleanno
```

84



Inizializzazione di strutture

- Per inizializzare una variabile di tipo struttura, si elencano i valori dei campi fra parentesi graffe
- Es:

```
typedef struct
{ int giorno;
  char mese[10];
  int anno;
} data;

main()
{ data Liberazione = {25,"aprile",1945};
...
}
```

85

Esercizio

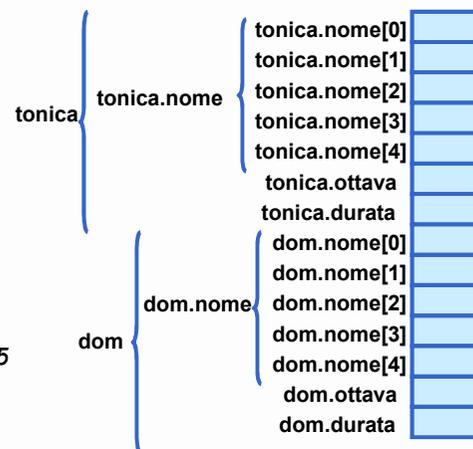
- Si definiscano due variabili di tipo struttura "numero complesso"
- Si leggano da tastiera due numeri complessi
- Si calcoli e si visualizzi il loro prodotto

86

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
 - ...

87

ESEMPIO

```
main()
{
  struct frutto
  { char nome[20]; int peso;
  } f1 = {"mela", 70};
  struct frutto f2 = {"arancia", 50};

  int peso = f1.peso + f2.peso;
}
```

Non c'è alcuna ambiguità perché peso nella propria struct è diverso da peso variabile intera

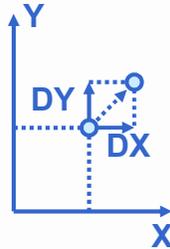
88

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

89

Esercizio: Commissione Paritetica Docenti-Studenti

- La **Commissione Paritetica Docenti - Studenti** si attiva per ricevere segnalazioni provenienti dai corsi di studio del Dipartimento di riferimento e dagli studenti, per approfondire gli aspetti critici legati al percorso di formazione (esperienza dello studente) offrendo un ulteriore canale oltre ai tradizionali questionari di valutazione, per proporre sinergicamente informazioni che il corso di studio e il suo Gruppo di Riesame potrebbero non ricevere tramite altri canali.

Nome	Cognome	Ruolo	Corso di Studio
Nicola	Prodi	Professore	Civile
Gian Luca	Garagnani	Professore	Industriale
Silvio	Simani	Professore	Informazione
Khadija	Suleiman	Studente	Informazione
Nicholas	Menegatti	Studente	Industriale
Manzari	Marco	Studente	Civile

```
#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);
}
```

Esercizio: Commissione Paritetica Docenti-Studenti

- Si definisca una opportuna struttura dati per contenere i membri della Commissione Paritetica Docenti-Studenti
- Si **inizializzi** la struttura dati con i seguenti dati:

Nome	Cognome	Ruolo	Corso di Studio
Nicola	Prodi	P	Civile
Gian Luca	Garagnani	P	Industriale
Silvio	Simani	P	Informazione
Khadija	Suleiman	S	Informazione
Nicholas	Menegatti	S	Industriale
Manzari	Marco	S	Civile

- Si scriva un programma C che
 - legge da tastiera il ruolo e corso di studio di una persona
 - visualizza nome e cognome del membro della CPDS

92



Esercizio

- Un array di strutture contiene i risultati degli esami di Fondamenti di Informatica (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.

Nome	Matricola	Voto
Francesco	31234	18
Sofia	41323	26
Alessandro	55235	15
Aurora	23411	10
Leonardo	32144	17
Giulia	34331	24
Lorenzo	12334	30
Emma	52341	31
Mattia	53139	18
Giorgia	36932	18
Andrea	25672	15

93



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
- ...

94



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

95



Esempio

nome	inizio	durata
informatica	14	2
analisi1	11	2
geometria	8	3

- Nuova lezione:
 - nome: "fisica"
 - inizio: 12
 - durata: 3

```
si sovrappone con
analisi1
informatica
```

96

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

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

97

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

98

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;
```

99

Esempio

- Che cosa fa questo programma?

```
main()
{ int i, pagaOraria=...;
  int paga[7], ore[7]={...};
  for (i=0; i<=6; i++)
  { if ((i==5) || (i==6))
      paga[i]=ore[i]*pagaOraria*1.5;
    else
      paga[i]=ore[i]*pagaOraria;
  }
}
```

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" a_1, \dots, a_N un numero naturale $(0, 1, \dots)$, che viene usato nella valutazione di espressioni che coinvolgono il nuovo tipo.

101

TIPI ENUMERATIVI

- Gli "identificativi di valore" a_1, \dots, a_N 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 */
```

102

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.

103

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.

104

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!

105

TIPI ENUMERATIVI

- È anche possibile specificare i valori naturali cui associare i simboli a_1, \dots, a_N

- qui, $\text{lun} \leftrightarrow 0, \text{mar} \leftrightarrow 1, \text{mer} \leftrightarrow 2, \text{etc.}$:

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

- qui, invece, $\text{lun} \leftrightarrow 1, \text{mar} \leftrightarrow 2, \text{mer} \leftrightarrow 3, \text{etc.}$:

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

- qui, infine, l'associazione è data caso per caso:

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

106

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

107

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!!!

108



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!!!

Union

- Una union è un tipo di dato che può contenere campi di tipo diverso, come una struct
- La differenza è che nelle union solo uno dei campi viene tenuto in memoria: i campi sono in alternativa
- tutti i campi di una union condividono lo stesso spazio in memoria

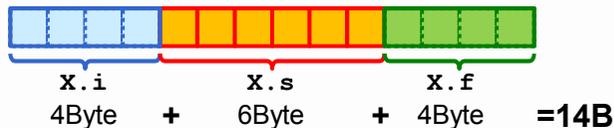
struct

struct

```
{ int i;
char s[6];
float f;
} X;
```

- Tutti e 3 i campi della struttura x sono presenti in memoria contemporaneamente

- La struttura contiene un dato int, una stringa e un float



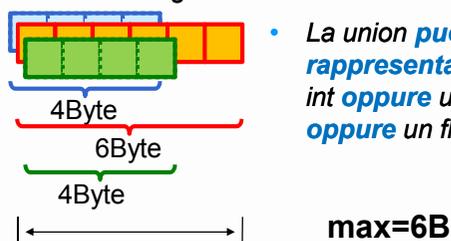
union

union

```
{ int i;
char s[6];
float f;
} X;
```

- Un solo campo della union x è presente in memoria in ogni momento

- La union può rappresentare un dato int oppure una stringa oppure un float



Union vs struct

- La union permette quindi di risparmiare spazio in memoria quando abbiamo campi alternativi

Struct

I campi di una struct hanno ciascuno un diverso indirizzo in memoria

La memoria occupata da una struct è la somma della memoria occupata dai singoli campi (non considerando l'allineamento)

Union

I campi di una union condividono tutti lo stesso indirizzo in memoria

La memoria occupata da una union è la memoria occupata dal campo di lunghezza massima (non considerando l'allineamento)

Assegnamenti in una union

```
union u
{ int i;
  char s[6];
  float f;
} X;

X.i=65;
printf("%d %s %f",X.i,X.s,X.f);
strcpy(X.s,"ABCD");
printf("%d %s %f",X.i,X.s,X.f)
X.f=3.1415
printf("%d %s %f",X.i,X.s,X.f)
```

- Visto che i campi sono memorizzati nella stessa area di memoria, se quando si utilizza un dato si usa il campo sbagliato, si ottiene un risultato privo di senso

```
65      A      0
1145258561 ABCD 781.0352
1078529622 VI@ 3.141500
```

Valori system-dependent!

113

Come ricordare il tipo inserito?

- I dati inseriti in una `union` hanno senso solo se sono interpretati correttamente
 - Se ho inserito un dato di tipo `int` nel campo di tipo intero, poi ha senso utilizzare solo il campo di tipo `int`, ecc.
- Come faccio a ricordarmi il tipo del dato che è inserito?
- Il linguaggio C non fornisce alcun aiuto: è responsabilità del programmatore ricordarsi il tipo del dato inserito
- Di solito, si utilizza un'altra variabile, fuori dalla `union`, che memorizza di che tipo è il dato inserito
 - Il tipo enumerativo è molto adatto a questo scopo

114

Come ricordare il tipo inserito?

```
typedef enum {intero, stringa, reale} TipoEnum;
```

```
typedef union
```

```
{ int i;
  char s[5];
  float f;
} TipoUnione;
```

```
main()
```

```
{TipoUnione X;
  TipoEnum T;
```

```
  X.i=5;
  T=intero;
```

```
  ...
```

```
  if (T==intero) X.i++;
```

Ogni volta in cui assegno un valore alla union `X`, memorizzo nella variabile `T` il tipo del valore

In questo modo, in seguito posso verificare il contenuto della variabile `T` per capire quali operazioni possono essere effettuate

115

Come ricordare il tipo inserito?

```
typedef enum {intero, stringa, reale} TipoEnum;
```

```
typedef union
```

```
{ int i;
  char s[5];
  float f;
} TipoUnione;
```

```
typedef struct
```

```
{ TipoEnum T;
  TipoUnione U;
} StrutturaUnione;
```

```
main()
```

```
{ StrutturaUnione X;
```

```
  X.U.i=5;
```

```
  X.T=intero;
```

```
  if (X.T==intero) X.U.i++;
```

Ma allora, visto che la variabile `T` e la `union` vanno sempre di pari passo, tanto vale unirle in una `struct`

116

Esercizio

- *Un'università deve memorizzare in un array i dati degli studenti e dei docenti*
- *Per entrambi viene inserito il nome ed il cognome (stringhe di 10 char max) e il ruolo (stringa "studente" o "docente")*
- *Per gli studenti è presente un array di 10 interi, che rappresenta i voti conseguiti agli esami*
- *Per i docenti è presente un float che rappresenta lo stipendio*
- *Si desidera sapere*
 - *1. qual è il cognome più frequente*
 - *2. qual è lo studente con la media più alta*