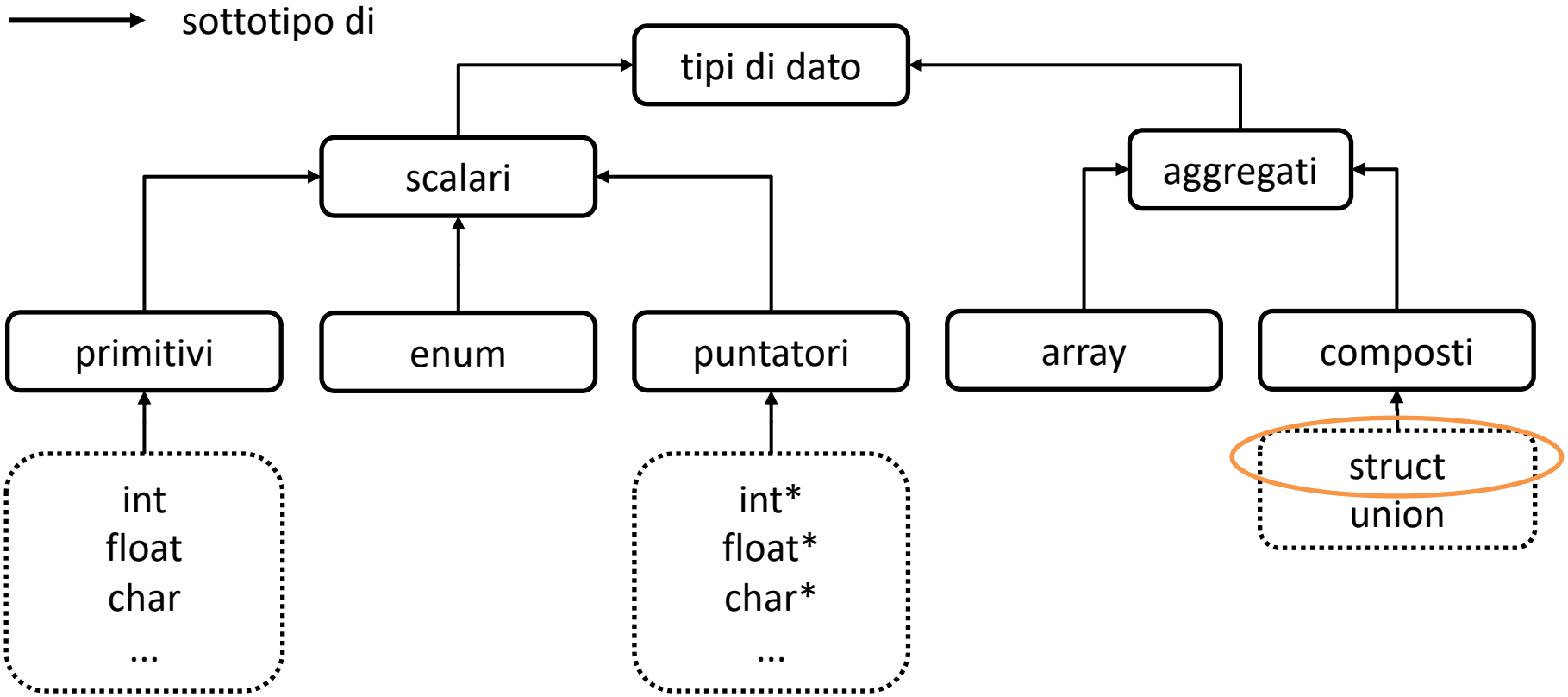
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Strutture in C

Tipi di Dato

Vediamo ora un secondo tipo di dato aggregato in C

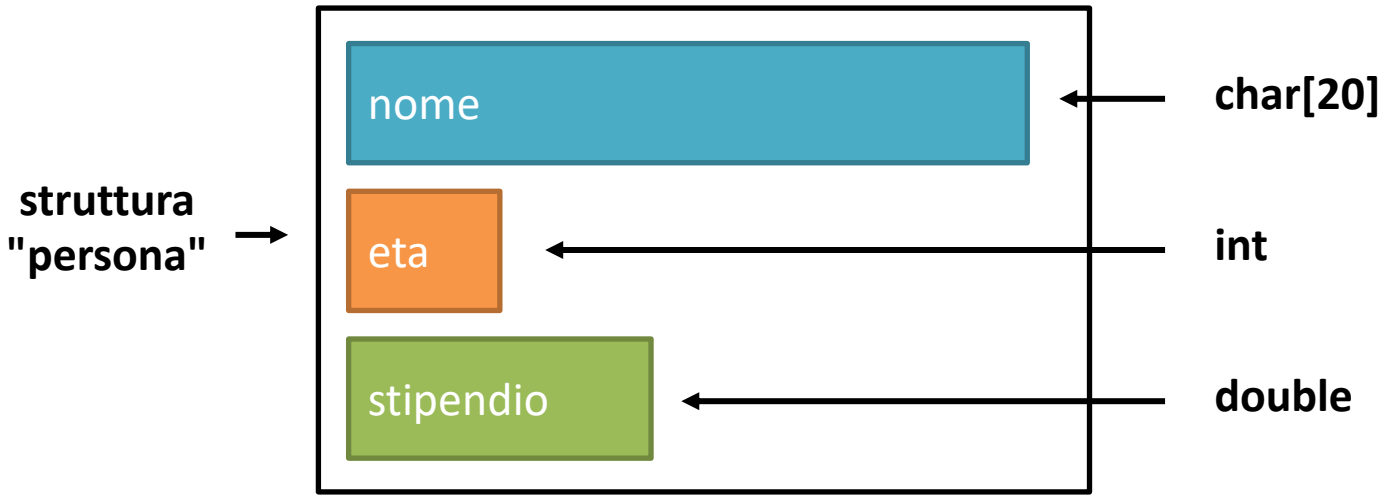


Si chiama struttura ed è un tipo di dato composto

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Strutture

Una struttura è una collezione di dati di tipo non-omogeneo



- Ogni elemento della collezione si dice "campo"
- Ogni campo è identificato da un nome
  - E.g. nome, eta, stipendio

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Strutture in C

In C si possono definire variabili di tipo struttura

Si usa la sintassi

```
struct [<etichetta>] { {<def. variabile>} } <nome>;
```

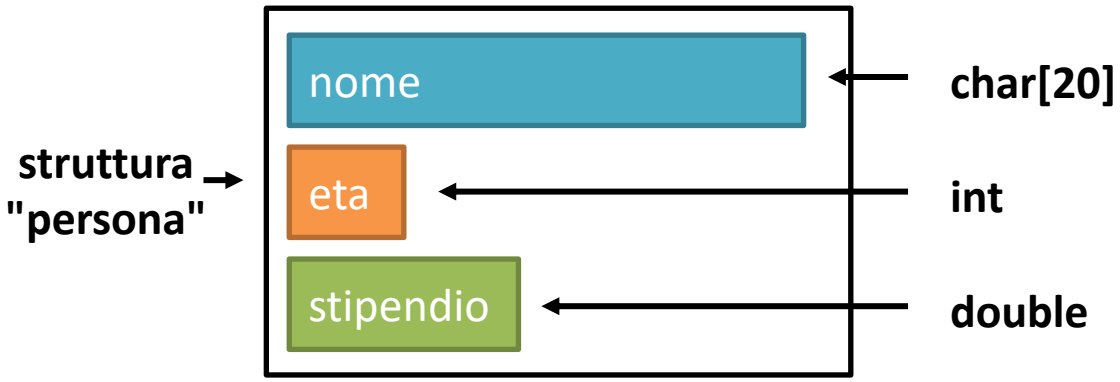
- Si usa la parola chiave "struct"
- <def. variabile> è una normale definizione di variabile
- <nome> è il nome della variabile struttura

Semantica:

- Definisce una variabile di nome <nome>
- Alloca memoria per tutti i campi!

Strutture in C

Per il nostro esempio




```
struct persona {
   char nome[20];
   int eta; // niente accento (solo caratteri ASCII)
   double stipendio;
} p;
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Strutture in C

Un secondo esempio:

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

- p1 e p2 sono due variabili
- Ciascuna contiene due campi float, di nome x e y

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Strutture in C

Un terzo esempio:

```
struct data {
   int giorno, mese, anno;
} ieri, oggi;
```

- "ieri" e "oggi" sono due variabili
- Ciascuna contiene tre campi interi (giorno, mese ed anno)

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Strutture in C

L'etichetta serve ad evitare di ripetere l'elenco dei campi:

```
struct data {
   int giorno, mese, anno;
} ieri;

struct data oggi;
```

- Definisco la variabile "ieri" ed indica il contenuto della struttura
- Associa alla struttura una etichetta
- Quindi in seguito posso definire la variabile "oggi"...
- ...Senza ripetere la lista dei campi

Strutture in C

Si può anche descrivere una struttura, senza definire variabili

```
struct data {
   int giorno, mese, anno;
};

struct data ieri, oggi;
```

- Stabilisco il contenuto della struttura, senza alcuna variabile
  - Si parla di dichiarazione, anziché di definizione
- Associo alla struttura una etichetta
- In seguito posso definire le variabili "ieri" e "oggi"

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Accesso a Strutture in C

Per accedere ad una struttura:

Come una normale variabile, ma si usa la notazione puntata

```
struct data {
   int giorno, mese, anno;
} ieri, oggi;

ieri.giorno = 2;
ieri.mese = 4;
ieri.anno = 2020;
```

- Ogni campo è una normale variabile
- Vi si può accedere sia in lettura che in scrittura

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Accesso a Strutture in C

Per accedere ad una struttura:

Con un puntatore si può usare l'operarore "*" come al solito:

```
struct data {
   int giorno, mese, anno;
} ieri;
struct data *pieri;

(*pieri).giorno = 2;
(*pieri).mese = 4;
(*pieri).anno = 2020;
```

- Le parentesi servono per avere la priorità corretta

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Accesso a Strutture in C

Per accedere ad una struttura:

Con un puntatore si può usare anche l'operatore speciale "->"

```
struct data {
   int giorno, mese, anno;
} ieri;
struct data *pieri;

pieri->giorno = 2;
pieri->mese = 4;
pieri->anno = 2020;
```

- È solo una sintassi semplificata! Ma fa molto comodo

Accesso a Strutture in C

```c
#include <stdio.h>

int main() {
    struct persona {
        char nome[32], cognome[32];
    } p1, p2;

    printf("Nome e cognome della prima persona: ");
    scanf("%s%s", p1.nome, p1.cognome);
    printf("Nome e cognome della seconda persona: ");
    scanf("%s%s", p2.nome, p2.cognome);
    printf("I due nomi sono: %s %s\n", p1.nome, p2.nome);
}
```

- Ognuna delle due strutture ha il suo campo nome e cognome
- Non c'è quindi ambiguità di accesso

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Semantica di una Struttura

Una variabile struttura denota la struttura nel suo complesso

- Si comporta come i tipi primitivi (int, float, etc.)
- ...E non come gli array (che rappresentano indirizzi)

Conseguenza: si possono assegnare strutture

```
struct punto { float x, y; } p1;
...
struct punto p2 = p1;
```

- L'assegnamento copia l'intera area di memoria

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Semantica di una Struttura

Conseguenza: si possono restituire strutture

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

struct punto mezzo(struct punto a, struct punto b) {
    struct punto res;
    res.x = 0.5 * (a.x + b.x);
    res.y = 0.5 * (a.y + b.y);
    return res; // si restituisce come un int!
}
```

Ci vuole però un accortezza:

- Il compilatore deve sapere che un tipo struttura esiste...
- ...Prima di poterlo utilizzare

Quindi la struttura va dichiarata prima della funzione!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Array in Strutture

Se la struttura contiene un array, la semantica è la stessa

Dati per esempio:

```
struct studente {
     char matricola[81];
     int voto;
} s1, s2;
```

L'assegnamento: `s1 = s2;`

- Copia l'intera area di memoria di s2 in s1
- Tutti gli elementi dell'array vengono copiati!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Lettura e Scrittura di Strutture

Si dichiari una struttura di nome "persona" con i campi:

- nome: una stringa di al più 63 caratteri
- cognome: una stringa di al più 63 caratteri

Si definiscano le due procedure:

```
void stampa_persona(struct persona p);
```

- Che stampi a video nome e cognome

```
void leggi_persona(struct persona *p);
```

- Che legga da terminale un nome ed un cognome...
- ...E li inserisca quindi nella struttura puntata da "p"

Si scriva quindi un main di prova

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Lettura e Scrittura di Strutture

```c
#include <stdio.h>
struct persona { char nome[64], cognome[64]; };

void stampa_persona(struct persona p) {
    printf("%s %s\n", p.nome, p.cognome);
}
void leggi_persona(struct persona *p) {
    printf("Inserire nome e cognome: ");
    scanf("%s%s", p->nome, p->cognome);
}
int main() {
    struct persona p1;
    leggi_persona(&p1);
    stampa_persona(p1);
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Confronto di Date

Si dichiari una struttura di nome "data" con i campi:

- giorno, mese, anno: interi

Si definiscano la procedura e la funzione:

```
void leggi_data(struct data *d);
```

- Che legga da termina le informazioni per inizializzare d

```
int date_uguali(struct data d1, struct data d2);
```

- Che restituisca "vero" se le date d1 e d2 sono uguali
- ...E restituisca "falso" altrimenti

Si scriva quindi un main di prova

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Confronto di Date

```c
#include <stdio.h>
struct data { int giorno, mese, anno; };
int date_uguali(struct data d1, struct data d2) {
    return (d1.giorno == d2.giorno) && (d1.mese == d2.mese) &&
           (d1.anno == d2.anno);
}
void leggi_data(struct data *d) {
    printf("Inserire gg/mm/aa: ");
    scanf("%d/%d/%d", &d->giorno, &d->mese, &d->anno);
}
int main() {
    struct data d1, d2;
    leggi_data(&d1);
    leggi_data(&d2);
    if (date_uguali(d1, d2)) printf("Uguali\n");
    else printf("Diverse\n");
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Alias per Tipi di Dato

Motivazione

Consideriamo il codice appena visto:

```
struct data { int giorno, mese, anno; };
int date_uguali(struct data d1, struct data d2) {
    ...
}
void leggi_data(struct data *d) {
    ...
}
int main() {
    struct data d1, d2;
    ...
}
```

Ripetere "struct data" tutte le volte è scomodo!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Motivazione

Sarebbe molto più intuitivo:

- Definire "data" come alias per "struct {int giorno,mese,anno; }"
- Usare "data" al posto di "struct data"

```
// Stabilisco un alias
int date_uguali(data d1, data d2) {
    ...
}
void leggi_data(data *d) {
    ...
}
int main() {
    data d1, d2;
    ...
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Parola Chiave Typedef

In C si possono definire alias per tipi di dato

Si usa la sintassi:

typedef <definizione tipo> <nuovo nome>

- Si usa la parola chiave "typedef"
- <definizione tipo> può essere una qualunque definizione di tipo
  - Spesso si tratta di una struttura (ma non sempre)
- <nuovo nome> è il nome da dare al tipo

Dopo la typedef, ci si può riferire al tipo usando il nuovo nome

Parola Chiave Typedef: Esempio

Torniamo al nostro esempio:

```
typedef struct {int giorno, mese, anno; } data;
```

- La specifica di tipo è "struct { ... }"
  - Notate: non abbiamo specificato un'etichetta
  - Quando si usa la typedef, l'etichetta è superflua

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Parola Chiave Typedef: Esempio

Torniamo al nostro esempio:

```
typedef struct {int giorno, mese, anno; } data;
```

- La specifica di tipo è "struct { ... }"
  - Notate: non abbiamo specificato un'etichetta
  - Quando si usa la typedef, l'etichetta è superflua
- L'alias per il nuovo tipo è "data"

Dopo la typedef:

- Ogni volta che si scrive "data", il compilatore...
- ...Assume il tipo "struct {int giorno, mese, anno; }"

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Parola Chiave Typedef: Esempio

```c
#include <stdio.h>
typedef struct { int giorno, mese, anno; } data;
int date_uguali(data d1, data d2) {
    return (d1.giorno == d2.giorno) && (d1.mese == d2.mese) &&
           (d1.anno == d2.anno);
}
void leggi_data(data *d) {
    printf("Inserire gg/mm/aa: ");
    scanf("%d/%d/%d", &d->giorno, &d->mese, &d->anno);
}
int main() {
    data d1, d2;
    leggi_data(&d1);
    leggi_data(&d2);
    if (date_uguali(d1, d2)) printf("Uguali\n");
    else printf("Diverse\n");
}
```

Parola Chiave Typedef: Altri Esempi

La typedef si può usare per introdurre livelli di astrazione

- E.g. possiamo definire un tipo per indicare un colore RGB

```
typedef int[3] colore; // valori di rosso, verde e blu
```

- Possiamo poi definire funzioni che lavorino con "colore"
- Se ad un certo punto ci interessa anche il livello di trasparenza

```
typedef int[4] colore; // RGB + trasparenza (alpha)
```

- Il codice sviluppato può funzionare senza modifiche
- ...Perché lavora con l'alias (tipo astratto)!

Parola Chiave Typedef: Altri Esempi

Dal C99 in poi, la typedef è usata per definire tipi addizionali!

Molti tipi numerici in C non hanno una dimensione fissa

- E.g. int può essere 16 o 32 bit

Può essere problematico!

- E.g. sistemi embedded con poca memoria, calcolo vettoriale

Dal C99 in poi sono disponibili tipi numerici con dimensione fissa:

- E.g. int8_t, int16_t, int32_t, int64_t...

Sono tutti ottenuti usando typedef!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Fondamenti di Informatica T

Andrea Acquaviva < andrea.acquaviva@unibo.it>
Michele Lombardi <michele.lombardi2@unibo.it
Andrea Borghesi <andrea.borghesi3@unibo.it>
Giuseppe Tagliavini <giuseppe.tagliavini@unibo.it>
Allegra De Filippo < allegra.defilippo@unibo.it>