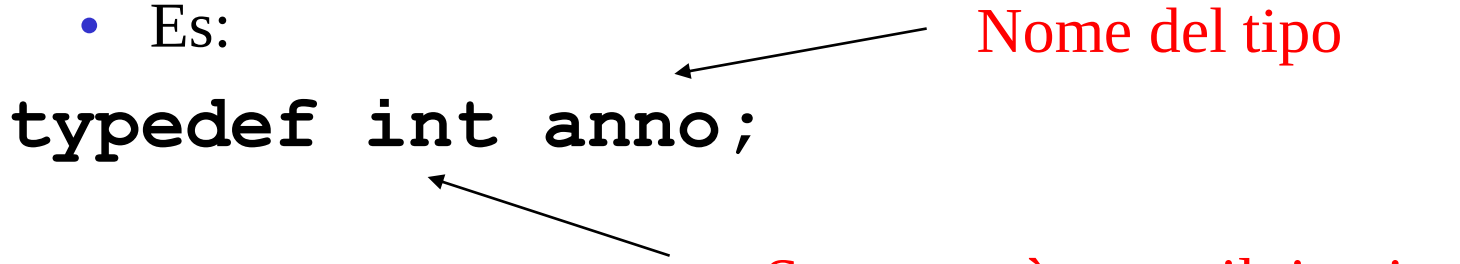
Tipi definiti dall'utente

Tipi definiti

- Il C mette a disposizione un insieme di tipi di dato predefiniti (tipi built-in) e dei meccanismi per denire nuovi tipi (tipi user-defined)
- Possiamo:
  - Ridenominare tipi esistenti ....
  - Crearne di nuovi aggregando tipi esistenti....
- Vediamo come ....

Definizione di tipo

- In C è possibile definire nuovi nomi per tipi esistenti:
  - La definizione è introdotta dalla parola chiave **`typedef`** ed è seguita da
    - la dichiarazione della struttura del nuovo tipo e dal suo nome, cioè come si costruisce a partire dai tipi già esistenti
    - Il nome del tipo
    - Il punto e virgola (;) che chiude la dichiarazione
    - Es:



Definizione di tipo

```
/* esempio: ridenominiamo il tipo int */
typedef int anno;
/* dichiarazione di variabili */
anno x, y;
 /* accesso e modifica */
x = x + 3;
```

```
/* typedef in realtà può essere omessa in
  alcuni casi ma per semplicità e leggibilità
  la useremo sempre */
```

Definizione di tipo

```
/* esempio: ridenominiamo il tipo int */
typedef int anno_t;
/* dichiarazione di variabili */
anno_t x, y;
 /* accesso e modifica */
x = x + 3;
```

```
/* per leggibilità aggiungeremo sempre "_t" al
  nome di un tipo, per distinguerlo più
  semplicemente dai nomi delle variabili*/
```

Tipi enumerati

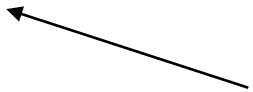
- Servono a rappresentare insiemi finiti:
  - giorni: lunedi, martedi, …
  - mesi: gennaio, febbraio, …
  - bool: true, false.
- Associano un identificatore (costante) ad ogni elemento dell'insieme
- Sono rappresentati internamente con degli interi
  - ... Ma il compilatore controlla che le funzioni siano chiamate con il tipo giusto

Tipi enumerati

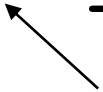
- Vediamo adesso come costruire un tipo nuovo enumerato:

  I tipi enumerati sono introdotti dalla parola chiave enum che consente di definire un tipo enumerando tutti i suoi valori, ad esempio:

```
typedef enum
{LUN, MAR, MER, GIO, VEN, SAB, DOM} giorni_t;
```



Struttura: tutte le costanti del tipo



Nome del tipo

Tipi enumerati

- Vediamo adesso come costruire un tipo nuovo enumerato:

  I tipi enumerati sono introdotti dalla parola chiave enum che consente di definire un tipo enumerando tutti i suoi valori, ad esempio:

```
typedef enum
{LUN, MAR, MER, GIO, VEN, SAB, DOM} giorni_t;

giorni_t a;  /* dichiarazione */
a = DOM ; /* assegnamento di una costante del
  tipo */
```

Esempio: stampa dei giorni

```
typedef enum {LUN, MAR, MER, GIO, VEN, SAB, DOM}
  giorni_t;


void stampa_giorni (giorni_t x) {
  switch ( x ) {
  case LUN: case MAR:
  case MER: case GIO:
  case VEN:
    printf("Giorno Lavorativo\n"); break;
  case SAB:
  case DOM:
     printf("Week-end\n");
  }
}
```

Esempio: stampa dei giorni

```
/* un possibile main */
#include <stdio.h>
typedef enum {LUN, MAR, MER, GIO, VEN, SAB, DOM}
  giorni_t;
void stampa_giorni (giorni_t x) {.... }


int main (void) {
   stampa_giorni(DOM);
   return 0;
}
/* stampa "Week-end" */
```

Perchè funziona ?

- Perchè tutti i valori sono interi!
  - Di default sono assegnati valori progressivi a partire da 0, nell'esempio

```
LUN == 0, MAR == 1, MER == 2 etc ...
```

  - Se vogliamo che assumano valori (interi) diversi possiamo richiederlo all'atto della dichiarazione, es

```
typedef enum {true = 1, false = 0} bool_t;
bool_t trovato;          /* dichiarazione */
trovato = false;
```

…

Tipi enumerati come interi ....

- Possiamo
  - applicare gli operatori */+- definiti sugli interi
  - confrontare due valori con !=, ==, <=, >=
    - (attenzione però il risultato ottenuto dipende dal numero associato, sta a noi capire se ha senso o no applicarlo ...)
  - analizzarli nei case di uno switch
  - usarli come indici di array: es **`a[LUN]`**

Le strutture

- Aggregati di variabili (anche di tipo diverso)
  - dichiarazione di una nuova struttura:

```
#define MAXLEN 40
   typedef struct studente {
     char nome[MAXLEN+1];
     char cognome[MAXLEN+1];
     unsigned matricola;
   } studente_t;
```

Parola chiave
Per definire una struttura

Le strutture

- Aggregati di variabili
  - dichiarazione di una nuova struttura:

```
#define MAXLEN 40
   typedef struct studente {
     char nome[MAXLEN+1];
     char cognome[MAXLEN+1];
     unsigned matricola;
   } studente_t;
```

Etichetta, nome della struttura che stiamo definendo

Le strutture

- Aggregati di variabili
  - dichiarazione di una nuova struttura:

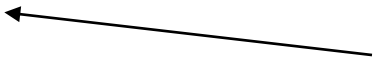
```
#define MAXLEN 40

    typedef struct studente {
        char nome[MAXLEN+1];
        char cognome[MAXLEN+1];
        unsigned matricola;
    } studente_t;
```

Campi, le variabili aggregate dalla struttura

Le strutture

- Aggregati di variabili
  - dichiarazione di una nuova struttura:

```
#define MAXLEN 40
   typedef struct studente {
     char nome[MAXLEN+1];
     char cognome[MAXLEN+1];
     unsigned matricola;
   } studente_t;
```



Nome del tipo

Le strutture

- Aggregati di variabili
  - dichiarazione di una nuova struttura:

```
#define MAXLEN 40
   typedef struct studente {
     char nome[MAXLEN+1];
     char cognome[MAXLEN+1];
     unsigned matricola;
   } studente_t;
```

  - Nota sulle stringhe: In questo modo la macro definisce la lunghezza massima di nome e cognome mentre lo spazio per il carattere terminatore è garantito dal +1

Le strutture

- Aggregati di variabili
  - dichiarazione di una nuova struttura:

```
#define MAXLEN 40
   typedef struct studente {
     char nome[MAXLEN+1];
     char cognome[MAXLEN+1];
     unsigned matricola;
   } studente_t ;

   /* dichiarazioni di variabili e array */
   studente_t x,  corsoA[NSTUD], *pstud;
```

Le strutture

- Esempio con variabili dello stesso tipo

```
typedef struct complessi {
   double real;
   double img;
} complessi_t ;

/* dichiarazioni di variabili e array */
complessi_t x, y, , z[N], *p;
```

Operazioni sulle strutture

- Assegnamento fra strutture

```
typedef struct complessi {
   double real; double img;
} complessi_t ;

complessi_t x, y, z[N], *p;
x = y; /* legale copia tutti i valori dei
  campi di y nei corrispondenti campi di x
  */
```

Operazioni sulle strutture

- E i confronti ?

```
typedef struct complessi {
   double real; double img;
} complessi_t ;

complessi_t x, y, z[N], *p;
x = y; /* legale copia tutti i valori dei
  campi di y nei corrispondenti campi di x
  */

if ( x == y ) /* sbagliato, non è possibile
  effettuare confronti fra strutture */
```

Operazioni sulle strutture

- Accesso ai campi (operatore punto (.))

```
typedef struct complessi {
   double real; double img;
} complessi_t ;

complessi_t x, y, z[N], *p;

x.real=2.0;
x.img=1.0;
y.real=x.real + 0.5;
y.img=y.real + 0.1;
```

Operazioni sulle strutture

- Inizializzazione con {....}

```
typedef struct complessi {
   double real; double img;
 } complessi_t ;

 complessi_t x = {2.0,1.0}, y, z[N], *p;

 y.real=x.real + 0.5;
 y.img=y.real + 0.1;
```

Operazioni sulle strutture

- Puntatore alla struttura (&) e operatore freccia (->)

```
typedef struct complessi {
   double real; double img;
} complessi_t ;

complessi_t x, y, z[N], *p, *q;
p = &x;
q = &y;
p->real=2.0; /* equivale a (*p).real */
p->img=1.0;
q->real=p->real + 0.5;
q->img=q->real + 0.1;
```

Operazioni sulle strutture

- Puntatore alla struttura (&) e operatore freccia (->)

```
typedef struct complessi {
   double real; double img;
} complessi_t ;

complessi_t x, y, z[N], *p, *q;
p = &x;
q = &y;
p->real=2.0; /* equivale a (*p).real */
p->img=1.0;
q->real=p->real + 0.5;
q->img=q->real + 0.1;
```

Le parentesi sono essenziali se no viene applicato prima il punto

Operazioni sulle strutture

- Dimensione di una struttura(sizeof)

```
typedef struct complessi {
   double real; double img;
} complessi_t ;

complessi_t x, y, z[N], *p, *q;
printf("%lu",sizeof(x));
printf("%lu",sizeof(complessi_t));
/* sono equivalenti */

/* non è detto che la lunghezza di una
 struttura sia = alla somma della lunghezza
 dei suoi campi! (allineamento .....) */
```

Operazioni sulle strutture

- Passaggio a funzione (per valore)
  - L'intera struttura viene copiata sullo stack come per i tipi base

```
typedef struct complessi { double real; double img;
 } complessi_t ;

 complessi_t somma (complessi_t x, complessi_t y) {
   complessi_t r;
   r.real=x.real+y.real;
   r.img=x.img+y.img;
   return r;
 }
 int main (void) {
   complessi_t A, B, C; .....
   A = somma(B,C); .... }
```

Operazioni sulle strutture

- Passaggio a funzione (per valore)
  - Attenzione se la struttura contiene un campo che è un array viene copiato integralmente sullo stack quando la passiamo alla funzione
  - Se vogliamo un passaggio più efficiente possiamo usare i puntatori

```
complessi_t somma (complessi_t* x, complessi_t* y) {
  complessi_t r;
  r.real=x->real+y->real;
  r.img=x->img+y->img;
  return r;
}
int main (void) {
  complessi_t A, B, C; .....
  A = somma(&B,&C); .... }
```

Operazioni sulle strutture

- Passaggio a funzione (per valore)
  - Ovviamente anche il risultato può essere assegnato ad una struttura passata per puntatore

```
void somma (complessi_t* x, complessi_t* y,
  complessi_t* r) {
   r->real=x->real+y->real;
   r->img=x->img+y->img;
   return;
}

int main (void) {
   complessi_t A, B, C; .....
   somma(&B, &C, &A); ....
}
```

Campi di una struttura

- devono avere nomi univoci all'interno di una struttura
- strutture diverse possono avere campi con lo stesso nome
- i nomi dei campi possono coincidere con altri nomi gia utilizzati (es. per variabili o funzioni)
- Esempio:

```
int x;
typedef struct a { char x; int y; } a_t;
typedef struct b { int w; float x; } b_t;
```

Campi di una struttura

- Possono essere tipi semplici o altre strutture definite precedentemente
- un campo di una struttura non puo essere del tipo struttura che si sta definendo,
- un campo può però essere di tipo puntatore alla struttura

```
/* vediamo un esempio, il classico tipo lista
  su cui ci concentreremo più avanti */
typedef struct lista {
   int a;
   struct lista *p;
} lista_t ;
```