

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 definire 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:

Nome del tipo

```
typedef int anno;
```

Struttura: è come il tipo int

# 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: lunedì, martedì, ...
  - 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 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

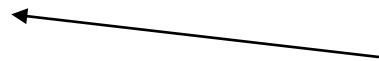
# Le strutture

- 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;
}
```

```
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;  
}  
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;  
}  
  
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 già 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 può 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 ;
```

# Scrittura/Lettura di strutture da file

- Possiamo trasformare la struttura in una stringa secondo una certa codifica
  - Lo vediamo in laboratorio
- Ma possiamo anche scrivere direttamente la sua rappresentazione binaria
  - Facciamo un esempio .....



# Scrittura su file (binario)

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

# Scrittura su file (binario)

```
#define MAXLEN 40
#define NSTUD 3
typedef struct studente { ... } studente_t ;

int main (void) {
    studente_t corsoA[NSTUD];
    int n; FILE * outf;
    /* parte che elabora gli studenti .... la vediamo in lab*/
    if ( ( outf = fopen("outfile","w") ) == NULL ) {
        perror("fopen: outfile"); exit(EXIT_FAILURE); }
    scrivi_stud(corsoA, n, outf);
    return 0 ;
}
```

# Scrittura su file (binario)

```
#define MAXLEN 40
typedef struct studente { ... } studente_t ;

/* scriviamo direttamente i byte della rappresentazione
   (attenzione a cambiare computer ...) */
void scrivi_stud (studente_t * a, int n, FILE* f) {
    int i;

    /* scrive n elementi a partire da a ognuno lundo
       sizeof(studente) byte nel file f */
    fwrite( a, sizeof(studente_t), n, f );
}
}
```

# Scrittura su file (testo)

```
#define MAXLEN 40

typedef struct studente { ... } studente_t ;

/* serializzazione: trasformazione in stringa e scrittura
   in file di testo
*/

void scrivi_stud (studente_t * a, int n, FILE* f) {
    int i;
    for (i=0; i<n; i++)
        fprintf(f, "%s:%s:%u\n", a[i].nome, a[i].cognome,
                a[i].matricola);
}
```

# Esempio: lettura (binario)

```
#include <stdio.h>
#define MAXLEN 40
#define NSTUD 3
typedef struct studente {...} studente_t;
/* legge da file binario nell'array x e stampa su
stdout */
int main (void) {
studente_t x[NSTUD]; int i; FILE* ifp;
ifp = fopen("studenti.out","r");
fread(x,sizeof(studente_t),NSTUD,ifp);
for (i=0;i<NSTUD;i++)
    printf("%d: %s %s %u\n",i, x[i].nome, x[i].cognome,
x[i].matricola);
return 0;
```

```
}
```