

A partire dai tipi fondamentali è possibile costruire nuove classi, dette *tipi derivati*. Abbiamo già incontrato alcuni dei principali tipi derivati. Gli array, le funzioni e i puntatori sono esempi di tipi derivati, nel senso che per essere specificati hanno bisogno di riferirsi a un tipo base. Se i tipi fondamentali rappresentano i più elementari costrutti trattati da un calcolatore, i tipi derivati possono essere usati per modellare oggetti più complessi e più vicini alle strutture del mondo reale. La capacità di poter derivare un tipo da altri è un meccanismo potente che il linguaggio C mette a disposizione del programmatore per risolvere una classe di problemi molto ampia.

Tra i tipi derivati che ancora non abbiamo preso in considerazione vi sono le strutture, le unioni e i campi. Di ognuno di essi indicheremo la sintassi e la modalità d'uso. Tratteremo poi un insieme di tipi derivati che nascono dalla composizione di altri tipi derivati. In particolare, parleremo di puntatori e funzioni, strutture e funzioni, puntatori e array e puntatori di puntatori.

## 12.2 Strutture

Per mezzo di un array è possibile individuare mediante un nome e un indice un insieme di elementi dello stesso tipo. Se per esempio volessimo rappresentare un'area di memoria di 200 byte su cui andare a scrivere o a leggere dei dati, potremmo definire una variabile `buf`:

```
int buf[100];
```

Ogni locazione di questa memoria verrebbe individuata attraverso un indice. Così

```
buf[10]
```

representerebbe l'undicesima locazione di memoria del buffer.

Ci sono però problemi in cui è necessario aggregare elementi di tipo diverso per formare una struttura. Per esempio, se si vuol rappresentare il concetto di data basta definire una *struttura* siffatta:

```
struct data {
    int giorno;
    char *mese;
    int anno;
};
```

Si osservi come per effetto di questa dichiarazione si sia introdotto nel programma un nuovo tipo, il tipo `data`. D'ora in poi sarà possibile definire variabili in cui tipo è `data`:

```
struct data oggi;
```

Come deve essere interpretata la variabile `oggi` di tipo `data`? Semplicemente come una variabile strutturata, composta di tre parti: due di tipo `int` – `giorno` e `anno` – e una di tipo `stringa` – `mese` –. La sintassi generale per la definizione di una struttura è:

```
struct nome_struttura {
    tipo_membro nome_membro1;
    tipo_membro nome_membro2;
    ...
    tipo_membro nome_membroN;
};
```

Gli elementi della struttura sono detti membri; essi sono identificati da un nome, *nome\_membro*, e da un tipo, *tipo\_membro*, che può essere sia fondamentale sia derivato. Nell'esempio della struttura `data` si ha che `giorno` e `anno` sono di tipo `int`, cioè un tipo fondamentale, mentre `mese` è di tipo `char *`, cioè di tipo derivato. Si osservi il punto e virgola posto in coda alla definizione di una struttura: è questo uno dei rari casi in cui in C occorre mettere un “;” dopo una parentesi graffa; il lettore presti la dovuta attenzione a questo particolare, spesso trascurato.

Una volta definita una struttura, *nome\_struttura* diviene un nuovo tipo a tutti gli effetti. Si possono allora definire variabili il cui tipo è *nome\_struttura*:

```
struct nome_struttura nome_variabile;
```

Per esempio si può scrivere:

```
struct data oggi, ieri, compleanno;
```

dove le variabili `oggi`, `ieri` e `compleanno` sono variabili di tipo `data`. Esiste anche un'altra sintassi per la definizione di variabili di tipo struttura. Per esempio:

```
struct automobile {
    char *marca;
    char *modello;
    int  venduto;
} a1, a2;
```

introduce la struttura `automobile`, e al contempo, anche due variabili di tipo `automobile`, `a1` e `a2`. In luogo di questa definizione di struttura e di variabili insieme, avremmo potuto usare la forma equivalente:

```
struct automobile {
    char *marca;
    char *modello;
    int  venduto;
};
struct automobile a1, a2;
```

È lasciato alla sensibilità del lettore stabilire quale delle due convenzioni usare per definire una variabile struttura.

Per poter accedere ai campi di una variabile di tipo struttura si fa uso dell'operatore punto (`.`):

```
oggi.giorno = 25;
oggi.mese = "Dicembre";
oggi.anno = 2001;
ieri.anno = oggi.anno;
```

Con le prime tre istruzioni si assegna la terna di valori `<25, "Dicembre", 2001>` alla variabile `oggi` e con la quarta si rendono uguali l'anno di `ieri` con quello di `oggi`.

La sintassi generale con cui si fa riferimento a un membro è

*nome\_variabile\_struttura.nome\_membro*

Per esempio, quando si vuole visualizzare il contenuto di una variabile struttura può essere necessario ricorrere all'operatore punto: si veda il Listato 12.1, la cui esecuzione produrrà il seguente risultato:

```
marca auto  = FERRARI
modello auto = F40
vendute     = 200
marca auto  = OPEL
modello auto = ASTRA
vendute     = 1200
```

```
/* Esempio di definizione di una struttura */
```

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

```
struct automobile {
    char *marca;
    char *modello;
    int  venduto;
};
```

```
main()
{
    struct automobile a1, a2;
```

```
    a1.marca = "FERRARI";
    a1.modello = "F40";
    a1.venduto = 200;
```

```
    a2.marca = "OPEL";
    a2.modello = "ASTRA";
```

```

a2.venduto = 1200;

printf("marca auto = %s\n", a1.marca);
printf("modello auto = %s\n", a1.modello);
printf("vendute = %d\n", a1.venduto);
printf("marca auto = %s\n", a2.marca);
printf("modello auto = %s\n", a2.modello);
printf("vendute = %d\n", a2.venduto);
}

```

#### Listato 12.1 La struttura automobile

Come si può osservare, le variabili `a1` e `a2` sono visualizzate “stampando” separatamente i membri `marca`, `modello` e `venduto` che le compongono.

Talvolta ci si può riferire a una variabile di tipo struttura nel suo insieme, attraverso il nome, senza dover far riferimento ai singoli membri. Per esempio, in C è possibile effettuare assegnazioni tra variabili struttura dello stesso tipo:

```

struct data compleanno, oggi;
    compleanno = oggi;

```

è una assegnazione consentita che effettua una copia di tutti i valori dei membri di `oggi` nei corrispondenti valori dei membri di `compleanno`. Due strutture corrispondono a due tipi differenti anche se hanno gli stessi membri. Per esempio:

```

struct s1 {
    int a;
};
struct s2 {
    int a;
};

```

sono due tipi struttura differenti:

```

struct s1 bob;
    struct s2 alex;
    ...
    alex = bob; /* errore: tipi dati discordi */

```

Quest’ultima assegnazione darebbe luogo a un errore, poiché si è tentato di assegnare una variabile di tipo `s1` a una di tipo `s2`. Inoltre si ricordi che i tipi struttura sono diversi anche dai tipi fondamentali. Per esempio, in

```

struct s1 bobo;
    int i;
    ...
    bobo = i; /* errore: tipi dati discordi */

```

l’ultima assegnazione provocherebbe un errore a causa della disomogeneità di tipo.

È possibile fare riferimento a una variabile struttura anche attraverso un puntatore, cioè tramite una variabile abilitata a contenere l’indirizzo di una variabile di tipo struttura:

```

struct data *pd, oggi, compleanno;
    pd = &oggi;
    (*pd).giorno = 31;
    (*pd).mese = "Gennaio";
    (*pd).anno = 2001;

```

Attraverso il puntatore `pd` si possono raggiungere i membri della variabile struttura `oggi`. Le parentesi tonde che circoscrivono `*pd` sono necessarie perché l’operatore “.” ha priorità maggiore rispetto all’operatore “\*”.

Poiché in C si accede frequentemente a una variabile struttura tramite puntatore per evitare costrutti sintattici laboriosi, è stato introdotto l’operatore freccia `->` per accedere direttamente ai membri di una variabile strutturata puntata da un puntatore. Nel frammento seguente le ultime tre istruzioni sono perfettamente equivalenti alle corrispondenti dell’esempio precedente:

```

struct data *pd, oggi, compleanno;

```

```

pd = &oggi;
pd->giorno = 31
pd->mese = "Gennaio";
pd->anno = 2001;

```

I puntatori a struttura sono molto usati specialmente per passare una struttura a una funzione, e per far sì che una funzione ritorni un risultato di tipo struttura:

```

int numero_mese(struct data *dt)
{
    if(dt->mese == "Gennaio")
        return(1);
    else
        if(dt->mese == "Febbraio")
            return(2);
        else
            ...
            if(dt->mese == "Dicembre")
                return(12)
            else
                return(0);
}

```

Alla funzione `numero_mese` viene passato un puntatore a variabile di tipo `data` e si ottiene il numero del mese relativo alla data puntata dal puntatore. Nel caso in cui il nome del mese non corrisponda ad alcuno di quelli conosciuti, la funzione ritorna un codice di errore pari a 0.

Infine, una ultima considerazione a proposito della inizializzazione di una variabile di tipo struttura. Abbiamo visto come per mezzo dell'operatore `punto` sia possibile assegnare dei valori a una variabile struttura. In alternativa si può usare una sintassi analoga a quella usata per inizializzare gli array:

```

struct automobile {
    char *marca;
    char *modello;
    int venduto;
};
struct automobile a = {"FERRARI", "F40", 200};

```

oppure

```

struct data {
    int giorno;
    char *mese;
    int anno;
} oggi = {25, "Dicembre", 2001};

```

Questo tipo di inizializzazione è ammesso solo se le corrispondenti variabili sono di tipo globale, cioè se sono definite all'esterno di un blocco ■.

## 12.3 Unioni

Le *unioni* (union) sono analoghe alle strutture: introducono nel programma una nuova definizione di tipo e sono costituite da un insieme di membri, che possono essere – in generale – di tipo e dimensione diversa. I membri di una unione, però – a differenza di una struttura –, coincidono, cioè hanno lo stesso indirizzo e vanno a occupare le medesime locazioni di memoria. Questo implica che l'occupazione di memoria di una unione coincide con quella del membro dell'unione di dimensione maggiore. La sintassi generale di una `union` è analoga a quella delle strutture:

```

union nome_unione {
    tipo_membro nome_membro1;
    tipo_membro nome_membro2;
    ...
    tipo_membro nome_membroN;
}

```