Tipi di Dati Derivati

Un tipo di dati è detto derivato se è costruito a partire dai tipi di dati fondamentali (**int**, **char**, **float**, **double** con o senza specificatori).

Gli array sono tipi di dati derivati omogenei perché tutti gli elementi devono essere dello stesso tipo.

Per definire un tipo di dati derivato non omogeneo bisogna definire una struttura mediante la parola chiave **struct**.

Una struttura è composta da vari campi o membri, ciascuno dei quali può essere di tipo fondamentale o derivato.

Definizione di Strutture

Un tipo struttura può essere definito in diversi modi sintatticamente equivalenti.

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

```
struct {
  int giorno, mese, anno;
  char *nome_giorno;
} ieri, oggi, domani;
/* dichiara 3 variabili senza definire un tipo utilizzabile */
```

```
struct data {
  int giorno, mese, anno;
  char *nome_giorno;
} giorno;
struct data giorni[30];
```

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

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

Accesso ai Campi di una Struttura

I campi delle strutture sono acceduti per mezzo dell'operatore punto.

Come esempio ecco il programma **data** che stampa la data di oggi.

```
*/ File sorgente data.c */
#include "data.h"
#include "in_out.h"
main()
{
  data oggi;
  oggi.giorno = 28;
  oggi.mese = 4;
  oggi.anno = 2003;
  oggi.nome_giorno = "Lunedì";
  stampa_data(oggi);
}
```

La definizione della struttura di tipo **data** è nel file **data.h** e il prototipo della funzione **stampa_data** è nel file **in_out.h**

Gli Altri File per data

```
*/ File sorgente in_out.c */
#include <stdio.h>
#include "data.h"
#include "in_out.h"
void stampa_data(data d)
{
  printf("Oggi è il %d/%d/%d ed è %s\n" , d.giorno, d.mese, d.anno, d.nome_giorno);
}
```

```
*/ File sorgente data.h */
typedef struct {
  int giorno, mese, anno;
  char *nome_giorno;
} data;
```

```
*/ File sorgente in_out.h */
void stampa_data(data d);
```

Strutture e Funzioni: Argomenti

La funzione **stampa_data** del programma **data** prende in input una struttura di tipo **data**.

```
void stampa_data(data d);
```

In questo modo il passaggio della variabile *oggi*, avviene per valore, cioè la funzione **stampa_data** ottiene una copia, campo per campo, della struttura *oggi*.

Spesso le strutture sono grandi, per cui il passaggio per valore è lento e pesante ed è preferibile usare i puntatori.

Strutture e Funzioni: Ritorno

Se una funzione restituisce una struttura, il valore è restituito per copia campo a campo nella struttura del chiamante.

Anche in questo caso, se la struttura è grande, questa copia è lenta pesante.

Ancora è preferibile che la funzione lavori sul puntatore alla struttura piuttosto che su una sua struttura locale che deve essere poi ricopiata.

Gli Operatori "->" e " . "

Quando si lavora con un puntatore ad una struttura, i campi devono essere acceduti tramite l'operatore -> (meno e maggiore).

Quando si lavora direttamente con una struttura invece si usa l'operatore punto.

Entrambi gli operatori di accesso ai membri di una struttura (. e ->), sono valutati da sinistra a destra ed entrambi sono a priorità massima.

Il Programma complex

Scriviamo il programma **complex** che fa la somma di due numeri complessi.

Tutte le funzioni di **complex** hanno strutture passate per riferimento e nessuna restituisce strutture.

*/ File sorgente **complex.h** */

```
typedef struct {
   double re;
   double im;
} complex;
```

*/ File sorgente **funz.h** */

```
void leggi_complesso(char **, complex *);
void stampa_somma(complex *);
void somma(complex *, complex *, complex *);
```

Le Funzioni di complex

*/ File sorgente **funz.c** (continua....) */

```
#include <stdio.h>
#include <stdlib.h>
#include "complex.h"
#include "funz.h"

void leggi_complesso(char *str[2], complex *c)
{
   if((sscanf(str[0], "%lf", &c-> re))!=1) {
      printf("Input error\n");
      exit(2);
   }
   if((sscanf(str[1], "%lf", &c-> im))!=1) {
      printf("Input error\n");
      exit(2);
   }
}
```

*/ (.... Segue) File sorgente **funz.c** */

```
void somma(complex *a, complex *b, complex *c)
{
   a -> re = b -> re + c -> re;
   a -> im = b -> im + c -> im;
}

void stampa_somma(complex *c)
{
   printf("La somma è %f + i%f\n", c->re, c->im);
}
```

Il main di complex

*/ File sorgente **complex.c** */

```
#include <stdio.h>
#include <stdlib.h>
#include "complex.h"
#include "funz.h"

main(int argc, char **argv)
{
   complex a, b, sum;
   if(argc != 5) {
      printf("Input error\n");
      exit(2);
   }
   leggi_complesso(&argv[1], &a);
   leggi_complesso(&argv[3], &b);
   somma(&sum, &a, &b);
   stampa_somma(&sum);
}
```

Esempi di Accesso ai Campi

Definiamo una struttura **studente** e scriviamo un piccolo programma che ne fa uso.

*/ File sorgente **student.h** */

```
typedef struct {
  char *cognome, *nome;
  unsigned long matricola;
} studente;
```

*/ File sorgente **student.c** */

```
#include <stdio.h>
#include "student.h"
main()
{
   studente s, *p = &s;
   s.cognome = "Rossi";       /* p->cognome = "Rossi"; */
   s.nome = "Mario";          /* p->nome = "Mario"; */
   s.matricola = 531002;      /* p->matricola = 531002; */
   printf("Matricola: %ld\n",  (*p).matricola);
   printf("Terza lettera nome: %c\n",  *(p->nome + 2));
   printf("Quarta lettera nome: %c\n",  (p->nome) [3]);
   printf("Prima lettera nome: %c\n",  (s.nome) [0]);
   printf("Quinta lettera nome: %c\n",  *((*p).nome + 4));
   printf("????: %c\n", * p->nome + 1);
}
```

Inizializzare Strutture

Le strutture possono essere inizializzate in vari modi

*/ File sorgente **strutture.h** */

```
typedef struct {
  double re;
  double im;
} complex;

struct indirizzo {
  char *via;
  int civico;
  char *citta;
  char *cap;
} ind = {"Tiburtina", 622, "Roma", "00159"};
typedef struct indirizzo indirizzo;
```

*/ File sorgente **strutture.c** */

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

main()
{
   complex vet[3][3] = {
      { {1.0, -0.1}, {2.0, 0.2}, {3.0, 0.3} },
      { {4.0, -0.1}, {5.0, 0.2}, {6.0, 0.3} },
   }; /* gli elementi in vet[2][] valgono (0.0, 0.0) */
   indirizzo old_ind = {0};
   /* tutti i campi valgono 0 o NULL */
   printf("Il cap di ind è %s\n", ind.cap);
   printf("vet[2][0] = (%f, %f)\n", vet[2][0].re, vet[2][0].im);
   printf("La via di old_ind Š %s\n", old_ind.via);
}
```

Strutture con Campi Struttura

Una struttura può avere campi che sono a loro volta strutture. Scriviamo il programma **studente2** che gestisce una classe di 100 studenti e stampa tutti gli studenti che sono nati nel 1980.

```
*/ File sorgente def.h */
#define CLASSE 100
typedef struct {
   int giorno, mese, anno;
} data;
typedef struct {
  char cognome[20], nome[20];
  unsigned long matricola;
  data data_di_nascita;
} studente;
```

```
*/ File sorgente in_out.h */
void inserisci_studenti(studente *);
void stampa_per_anno(studente *, int);
```

Gli Altri File per studente2

```
*/ File sorgente in_out.c (continua....)*/
#include <stdio.h>
#include "def.h"
#include "in_out.h"
void inserisci_studenti(studente *s) {
  int i=0;
  do {
     printf("\nCognome: ");
     scanf("%s", (s+i)->cognome);
     printf("\nNome: ");
     scanf("%s", (s+i)->nome);
     printf("\nMatricola: ");
     scanf("%ld", &(s+i)->matricola);
     printf("\nData di Nascita (g m a): ");
     scanf("%d", &(s+i)->data_di_nascita.giorno);
     scanf("%d", &(s+i)->data_di_nascita.mese);
     scanf("%d", &(s+i)->data_di_nascita.anno);
     printf("\nAltro studente? (s/n) ");
     getchar();
  } while(++i < CLASSE && getchar() !='n');
}
```

```
*/ (Segue... File sorgente in_out.c */
void stampa_per_anno(studente *s, int a) {
  int i;
  for(i=0;i<CLASSE;i++)
    if((s+i)->data_di_nascita.anno == a) {
      printf("%s ", (s+i)->cognome);
      printf("%s\n", (s+i)->nome);
    }
}
```

```
*/ File sorgente studente2.c */
#include "def.h"
#include "in_out.h"
main()
{
  studente classe[CLASSE];
  inserisci_studenti(classe);
  stampa_per_anno(classe, 1980);
}
```

Un Altro Esempio: Carte da Gioco

Scriviamo il programma **carte** che usa un array di strutture per rappresentare un mazzo di carte napoletane.

Le carte sono inizializzate dalla funzione **inizializza_carte** e poi vengono stampate dalla funzione **stampa_carte**.

```
*/ File sorgente carte.h */
typedef struct {
  int valore;
  char seme; /* 'B', 'D', 'C', 'S' */
} carta;
```

```
*/ File sorgente funz.h */
#define NUM_CARTE 40
void inizializza_carte(carta *);
void stampa_carte(carta *);
```

Gli Altri File per carte: funz.c

```
*/ File sorgente funz.c (continua....) */

#include <stdio.h>
#include "carte.h"
#include "funz.h"

void inizializza_carte(carta *m)
{
  int i;
  char seme[] = "BDCS";
  for(i=0;i<NUM_CARTE;i++) {
     m[i].valore = (i % 10) + 1;
     m[i].seme = seme[i/10];
  }
}
```

```
*/ (Segue ...) File sorgente funz.c */
void stampa_carte(carta *c) {
  printf("%d di ", c->valore);
  switch(c->seme) {
    case 'B':
      printf("Bastoni\n");
      break;
    case 'D':
      printf("Denari\n");
      break;
    case 'C':
      printf("Coppe\n");
      break;
    case 'S':
      printf("Spade\n");
      break;
  }
}
```

Gli Altri File per carte: carte.c

```
*/ File sorgente carte.c */

#include <stdio.h>
#include "carte.h"
#include "funz.h"

main() {
   carta mazzo_di_carte[NUM_CARTE];
   int i;
   inizializza_carte(mazzo_di_carte);
   for(i=0;i<NUM_CARTE;i++) {
       printf("Carta n. %d: ", i);
       stampa_carte(&mazzo_di_carte[i]);
   }
}
```

Campi di Bit

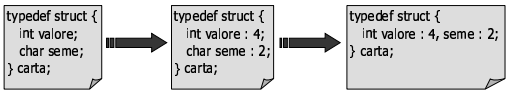
In una struttura un campo di tipo **int** o **char** (spesso **unsigned**) può essere dichiarato come un campo di bit.

Un campo di bit è composto da un certo numero di bit, indicato nella dichiarazione (al più pari alla dimensione del tipo utilizzato).

In genere tutti i campi di bit in una struttura vengono dichiarati come campi consecutivi per permettere al compilatore di compattare i dati nel minor numero possibile di parole macchina.

Campi di Bit: un Esempio

La struttura **carta** che abbiamo definito prima in realtà ha bisogno di 4 bit per il campo valore e 2 bit per il campo seme.



In questo modo una struttura di tipo **carta** sembrerebbe occupare solo 6 bit. In realtà occupa 1 parola macchina, cioè 4 byte, mentre la definizione originale ne occupa 2, cioè 8 byte.

Limitazioni sui Campi di Bit

I campi di bit presentano le seguenti limitazioni

- Non si possono definire array di campi di bit
- Non si possono definire puntatori ai campi di bit
- Non si può usare l'operatore di indirizzo **&** sui campi di bit

Si può però accedere a un campo di bit tramite l'operatore **->**

```
*/ File sorgente carte2.h */
typedef struct {
   int valore : 4, seme : 2;
} carta;
```

```
*/ File sorgente carte2.c */
#include <stdio.h>
#include "carte2.h"
main()
{
  carta c, *p=&c;
  c.valore=2;
  c.seme=0;
  printf("La carta è il %d di %d\n", p->valore, p->seme);
}
```

Esercizio

Scrivere un programma che gestisce informazioni relative a 10 studenti. Per ogni studente siamo interessati a

- Nome
- Cognome
- Matricola
- Data di nascita
- Giorno di iscrizione
- Media esami

Il programma deve prevedere le seguenti operazioni selezionabili da un menù

- Inserimento dei dati relativi a uno studente
- Stampa dei dati relativi allo studente i (i parametro della funzione)
- Ricerca di uno studente per nome (nome parametro della funzione)
- Stampa dei dati relativi a tutti gli studenti
- Stampa dei dati relativi a tutti gli studenti con media > 27
- Stampa dei dati relativi a tutti gli studenti immatricolati in un dato giorno (data parametro della funzione)