Strutture e File

Strutture

Una **struttura** (o **record**) serve per **aggregare** elementi (anche di tipo diverso) sotto un unico nome.

Sono un tipo di dato **derivato**, ovvero costruito a partire da dati di altri tipi.

Definizione di una struttura

Esempio:
```
struct data {
  int giorno;
  int mese;
  int anno;
};
```

- parola chiave `struct` introduce la definizione della struttura
- `data` è l'**etichetta** della struttura, che attribuisce un nome alla definizione della struttura
- `giorno`, `mese`, `anno` sono i **campi** (o **membri**) della struttura

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 nomi di variabili o funzioni

  Esempio:
  ```
  int x;
  struct a { char x; int y; };
  struct b { int w; float x; };
  ```
- possono essere di tipo diverso (semplice o altre strutture)
- un campo di una struttura non può essere del tipo struttura che si sta definendo

  Esempio:
  ```
  struct s { int a;
             struct s next;  };
  ```
- un campo può però essere di tipo puntatore alla struttura

  Esempio:
  ```
  struct s { int a;
             struct s *next;  };
  ```
- la definizione della struttura non provoca allocazione di memoria, ma introduce un nuovo tipo di dato

Dichiarazione di variabili di tipo struttura

Esempio: `struct data oggi, appelli[10], *pd;`

- `oggi` è una variabile di tipo `struct data`
- `appelli` è un vettore di 10 elementi di tipo `struct data`
- `pd` è un puntatore a una `struct data`

Una variabile di tipo struttura può essere dichiarata contestualmente alla definizione della struttura.

Esempio:
```
struct studente {
  char nome[20];
  long matricola;
  struct data ddn;
} s1, s2;
```

In questo caso si può anche **omettere l'etichetta** di struttura.

Operazioni sulle strutture

Si possono assegnare variabili di tipo struttura a variabili **dello stesso tipo** struttura.

Esempio:
```
struct data d1, d2;
...
d1 = d2;
```

Nota: questo permette di assegnare interi vettori.

Esempio:
```
struct matrice { int elementi[10][10]; };
struct matrice a, b;
...
a = b;
```

Non è possibile effettuare il confronto tra due variabili di tipo struttura.

Esempio:
```
struct data d1, d2;
if (d1 == d2) ...
```
Errore!

Motivo: una struttura può contenere dei "**buchi**" dovuti alla necessità di allineare i campi con le parole di memoria.

L'equivalenza di tipo tra strutture è **per nome**.

Esempio:
```
struct s1 { int i; };
struct s2 { int i; };
struct s1 a, b;
struct s2 c;

a = b;      OK
a = c;      Errore, perché a e c non sono dello stesso tipo.
```

Si può ottenere l'indirizzo di una variabile di tipo struttura tramite l'operatore `&`.

Si può rilevare la dimensione di una struttura con `sizeof`.

Esempio:
```
sizeof(struct data)
```

Attenzione: **non** è detto che la dimensione di una struttura sia pari alla somma delle dimensioni dei singoli campi (ci possono essere **buchi**).

Accesso ai campi della struttura

Avviene tramite l'**operatore punto**

Esempio:
```
struct data oggi;
oggi.giorno = 8;    oggi.mese = 5;   oggi.anno = 2002;
printf("%d %d %d", oggi.giorno, oggi.mese, oggi.anno);
```

Accesso tramite un puntatore alla struttura.

Esempio:
```
struct data *pd;
pd = malloc(sizeof(struct data));
(*pd).giorno = 8;   (*pd).mese = 5;   (*pd).anno = 2002;
```

N.B. Ci vogliono le `()` perché "`.`" ha priorità più alta di "`*`".

Operatore freccia: combina il dereferenziamento e l'accesso al campo della struttura.

Esempio:
```
struct data *pd;
pd = malloc(sizeof(struct data));
pd->giorno = 8;   pd->mese = 5;   pd->anno = 2002;
```

Inizializzazione di strutture

Può avvenire, come per i vettori, con un elenco di inizialzzatori.

Esempio: `struct data oggi = { 8, 5, 2002 };`

Se ci sono meno inizializzatori di campi della struttura, i campi rimanenti vengono inizializzati a 0 (o `NULL`, se il campo è un puntatore).

Variabili di tipo struttura dichiarate esternamente alle definizioni di funzione vengono inizializzate a 0 per default.

Passaggio di parametri di tipo struttura

È come per i parametri di tipo semplice:

- il passaggio è per valore $\Longrightarrow$ viene fatta una **copia dell'intera struttura** dal parametro attuale a quello formale
- è possibile effettuare anche il passaggio per indirizzo

Per passare per valore ad una funzione un vettore (il vettore, non il puntatore iniziale) è sufficiente racchiuderlo in una struttura.

Una funzione può restituire un valore di tipo struttura.

Esempio: file `tipi/structvet.c`

`typedef`

Attraverso `typedef` il C permette di creare dei sinonimi di tipi definiti in precedenza.

Esempio:
```
struct data { int giorno, mese, anno; };
typedef struct data Data;
Data d1, d2;
Data appelli[10];
```

`Data` è un sinonimo di `struct data`.

N.B. `typedef` non crea un nuovo tipo, ma un **nuovo nome di tipo**, che può essere usato come sinonimo.

Esempio: file `tipi/complex.c`

In generale, una `typedef` ha la forma di una dichiarazione di variabile preceduta dalla parola chiave `typedef`, e con il nuovo nome di tipo al posto del nome della variabile.

Esempio:
```
typedef Data Appelli[10];
typedef int *PuntInt;
```

Elaborazione dei file

Un file è una **sequenza di byte** (o caratteri) memorizzata su disco (memoria di massa), alla quale si accede tramite un **nome**.

I byte corrispondono a dei dati di tipo `char`, `int`, `double`, ecc. (esattamente come la sequenza di byte che immettiamo da input o leggiamo da tastiera).

I file vengono gestiti dal sistema operativo, e il programma deve invocare le funzioni del SO per accedere ai file: viene fatto dalla libreria standard di input/output.

Principali operazioni sui file:

- lettura da file
- scrittura su file
- apertura di file: comunica al SO che il programma sta accedendo al file
- chiusura di file: comunica al SO che il programma rilascia il file

Per ogni programma vengono aperti automaticamente tre file:

- `stdin` (sola lettura): lettura è da tastiera
- `stdout` (sola scrittura): scrittura è su video
- `stderr` (sola scrittura): per messaggi di errore (scritti su video)

Apertura di un file: tramite la funzione `fopen`

Deve essere effettuata prima di poter operare su un qualsiasi file.

Esempio:
```
FILE *fp;
fp = fopen("pippo.dat", "r");
```

`FILE` è un tipo struttura definito in `stdio.h`.

Dichiarazione di `fopen`: `FILE * fopen(char *nomefile, char *modo);`

Parametri di `fopen`:

- il nome del file
- la modalità di apertura:
  - `"r"` (read) . . . sola lettura
  - `"w"` (write) . . . crea un nuovo file per la scrittura; se il file esiste già ne elimina il contenuto corrente
  - `"a"` (append) . . . crea un nuovo file, o accoda ad uno esistente per la scrittura
  - `"r+"` . . . apre un file per l'aggiornamento (lettura e scrittura), a partire dall'inizio del file
  - `"w+"` . . . crea un nuovo file per l'aggiornamento; se il file esiste già ne elimina il contenuto corrente
  - `"a+"` . . . crea un nuovo file, o accoda ad uno esistente per l'aggiornamento

`fopen` restituisce un puntatore ad una struttura di tipo `FILE`

- la struttura mantiene le informazioni necessarie alla gestione del file
- il puntatore (detto **file pointer**) viene utilizzato per accedere al file
- se si verifica un errore in fase di apertura, `fopen` restituisce `NULL`

Esempio:
```
FILE *fp;
if ((fp = fopen("pippo.dat", "r")) == NULL) {
  printf("Errore!\n");
  exit(1);
} else ...
```

Chiusura di un file: tramite `fclose`, passando il file pointer

Esempio: `fclose(fp);`

Dichiarazione di `fclose`: `int fclose(FILE *fp);`

Se si è verificato un errore, `fclose` restituisce `EOF` (-1).

La chiusura va **sempre** effettuata, appena sono terminate le operazioni di lettura/scrittura da effettuare sul file.

Scrittura su e lettura da file

Consideriamo solo file ad **accesso sequenziale**: si legge e si scrive in sequenza, senza poter accedere agli elementi precedenti o successivi.

Scrittura su file: tramite `fprintf`

Dichiarazione di `fprintf`: `int fprintf(FILE *fp, char *formato, ...);`

- come `printf`, tranne che per `fp` (`printf(...);` è equivalente a `fprintf(stdout, ...);`)
- scrive sul file identificato da `fp` a partire dalla posizione corrente
- il file deve essere stato aperto in scrittura, append, o aggiornamento
- in caso di errore restituisce `EOF`, altrimenti il numero di byte scritti

Esempio:

```
int ris1, ris2;
FILE *fp;
if ((fp = fopen("risultati.dat", "w")) != NULL) {
  ris1 = ...;   ris2 = ...;
  fprintf(fp, "%d %d\n", ris1, ris2);
  fclose(fp);
}
```


Lettura da file: tramite fscanf

Dichiarazione di `fscanf`: `int fscanf(FILE *fp, char *formato, ...);`

- come `scanf`, tranne che per `fp` (`scanf(...);` è equivalente a `fscanf(stdin, ...);`)
- legge dal file identificato da `fp` a partire dalla posizione corrente
- il file deve essere stato aperto in lettura o aggiornamento
- restituisce il numero di assegnamenti fatti agli argomenti specificati nell'attivazione dopo la stringa di formato
- se il file termina o si ha un errore prima del primo assegnamento, restituisce `EOF`

Verifica di fine file: si può usare `feof`

Dichiarazione di `feof`: `int feof(FILE *fp);`

Restituisce vero (1) se per il file è stato impostato l'**indicatore di end of file**, ovvero:

- quando si tenta di leggere oltre la fine del file
- per lo standard input, quando viene digitata la combinazione di tasti
  - `ctrl-z` (per Windows)
  - `return ctrl-d` (per Unix)

Esempio: Conteggio del numero di caratteri e di linee in un file.

Implementazione: file `file/contaclf.c`

Esercizio: Conteggio del numero di parole in un file (uno o più spazi bianchi e/o `'\n'` sono equivalenti ad uno spazio bianco).

Soluzione: file `file/contapar.c`

Esempio: Calcolare la somma degli interi in un file.

```
int somma = 0, n;
FILE *fp;

if ((fp = fopen("pippo.txt", "r")) != NULL) {
  while (fscanf(fp, "%d", &n) == 1)
    somma += n;
  fclose(fp);
}
```

Esempio: Creazione di una copia di un file.

```
char ch;
FILE * fpr, *fpw;

if (((fpr = fopen("in.txt", "r")) != NULL) &&
    ((fpw = fopen("out.txt", "w")) != NULL))  {
  while (fscanf(fpr, "%c", &ch) == 1)
    fprintf(fpw, "%c", ch);
  fclose(fpr);
  fclose(fpw);
}
```

Esempio: Simulare 100 lanci di due dadi, memorizzando i risultati in un file. Leggere poi i risultati dal file, calcolarne e stamparne la media, e stampare sul file `frequenze.txt` le frequenze dei risultati dei lanci.

Implementazione: file `file/duedadif.c`

Esercizio: Calcolare le frequenze dei caratteri alfabetici in un file il cui nome è letto da tastiera.