Le strutture

Una **struttura** C è una collezione di variabili di uno o più tipi, raggruppate sotto un nome comune.

- **Dichiarazione** di una struttura:

```
struct point {
        int x;
        int y;
};
```

  La dichiarazione di una struttura definisce un tipo.

- **Dichiarazione** di una **variabile** di tipo `struct point`:

```
struct point punto1;
```

- **Dichiarazione ed inizializzazione** di una **variabile** di tipo `struct point`:

```
struct point punto2 = { 15, 7 };
```

Accesso alle componenti, puntatori a strutture

- Alle **componenti** (o **membri**) della struttura si accede con l'operatore `.`:

```
punto1.x = 3;
punto1.y = 5;
```

- **Dichiarazione** di un **puntatore a struttura**:

```
struct point *pp;
```

- L'**accesso** alle componenti della struttura puntata da `pp` avviene mediante l'operatore `->`:

```
pp->x = 3;
```

Vettori di strutture

La dichiarazione

```
struct key {
     char *word;
     int count;
} keytab[NKEYS];
```

dichiara un tipo struttura, `key`, definisce un vettore `keytab` di strutture di questo tipo e riserva memoria per tali strutture.

Avremmo anche potuto scrivere:

```
struct key {
     char *word;
     int count;
};
struct key  keytab[NKEYS];
```

Strutture ricorsive

Esempio: definizione della struttura ricorsiva `tnode`

```
struct tnode {
     int date;
     struct tnode *left;
     struct tnode *right;
};
```

Per creare un nuovo nodo si usa la funzione della libreria std `malloc`, che alloca dinamicamente lo spazio in memoria necessario. La funzione

```
void *malloc(size_t n);
```

ritorna un puntatore a n byte di memoria non inizializzata, oppure `NULL` se la richiesta non può essere soddisfatta.

Esempio d'uso:

```
struct tnode *p;
p = (struct tnode *)malloc(sizeof(struct tnode));  \* si noti il casting
del valore restituito  */
p -> date = 1;
p -> left = p -> right = NULL;
```

Accesso a file

Un programma C può leggere l'input da file passati sulla linea di comando.

Prima di leggere/scrivere su un file, un programma C deve **aprire** il file tramite la funzione di libreria

```
FILE *fopen(char *name, char *mode);
```

La funzione `fopen` prende come parametro il nome del file, `name`, e una stringa, `mode`, che indica il **modo** di utilizzo del file, `"r"` (lettura), `"w"` (scrittura), `"a"` (append), e restituisce un **puntatore** (**file pointer**) da utilizzare per la lettura/scrittura del file. Il file pointer punta ad una struttura che contiene informazioni sul file (indirizzo di un buffer, posizione corrente nel buffer, etc.)

Dichiarazione di un file pointer e chiamata a `fopen`:

```
FILE *fp;
fp = fopen(name, mode);
```

Lettura, scrittura e chiusura di file

Una volta aperto, un file può essere letto/scritto mediante le funzioni:

- `int getc(FILE *fp)` che ritorna il prossimo carattere del file `fp`, `EOF`, in caso di errore o fine file;
- `int putc(int c, FILE *fp)` che scrive il carattere `c` sul file `fp` e ritorna il carattere scritto, oppure `EOF` in caso di errore;
- `char *fgets(char *line, int maxline, FILE *fp)` che legge dal file `fp` un numero di caratteri pari al minimo fra `maxline-1` e quelli compresi tra la posizione corrente ed il prossimo carattere di newline, memorizzandoli nell'array di caratteri puntato da `line` (in caso di errore o di end-of-file restituisce `NULL`, altrimenti `line`);
- `int fputs(char *line, FILE *fp)` che scrive nel file `fp` la stringa puntata da `line` (in caso di errore restituisce `EOF`, altrimenti `0`).

Al termine delle operazioni di lettura/scrittura di un file, è buona norma rilasciare il file pointer, utilizzando la funzione

`int fclose(FILE *fp)`

Standard input, standard output, standard error

Le costanti predefinite `stdin`, `stdout`, `stderr` sono file pointer, cioè oggetti di tipo `FILE` che si riferiscono rispettivamente a **standard input** (tastiera), **standard output** (video), **standard error** (video).

`stdin` e `stdout` possono essere rediretti su altri file o pipe, mediante i simboli di **ridirezione** e **pipeline**. E.g.: `prog < infile` , dove `prog` è un programma C e `infile` è un file.

Esercizi

- Scrivere un programma C, versione semplificata del comando Unix `cat`, per l'append di uno o più file su std output.

- Scrivere un programma C, versione semplificata del comando Unix `cmp` per il confronto di due file, che stampa la prima linea su cui i file differiscono.