System call per l'accesso a file

Nome	Significato
`open`	apre un file in lettura e/o scrittura o crea un nuovo file
`creat`	crea un file nuovo
`close`	chiude un file precedentemente aperto
`read`	legge da un file
`write`	scrive su un file
`lseek`	sposta il puntatore di lettura/scrittura ad un byte specificato
`unlink`	rimuove un file
`remove`	rimuove un file
`fcntl`	controlla gli attributi associati ad un file

La system call `lseek`

La system call `lseek` permette l'**accesso random** ad un file, cambiando il numero del prossimo byte da leggere/scrivere.

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int filedes, off_t offset, int start_flag);
```

Il parametro `filedes` è un descrittore di file.
Il parametro `offset` determina la nuova posizione del puntatore in lettura/scrittura.
Il parametro `start_flag` specifica da dove deve essere calcolato l'`offset`. `startflag` può assumere uno dei seguenti valori simbolici:

`SEEK_SET` (0)	:	`offset` è misurato dall'inizio del file
`SEEK_CUR` (1)	:	`offset` è misurato dalla posizione corrente del puntatore
`SEEK_END` (2)	:	`offset` è misurato dalla fine del file

`lseek` ritorna la nuova posizione del puntatore.

Offset validi e non validi

Il parametro `offset` può essere **negativo**, cioè sono ammessi **spostamenti all'indietro** a partire dalla posizione indicata da `start_flag`, purchè però non si vada oltre l'inizio del file.
Tentativi di spostamento prima dell'inizio del file generano un errore.

È possibile spostarsi **oltre la fine del file**.
Ovviamente non ci saranno dati da leggere in tale posizione.
Futuri accessi tramite la `read` ai byte compresi tra la vecchia fine del file e la nuova posizione danno come risultato il carattere ASCII null.

Esempio:

```
off_t newpos;
⋮
newpos = lseek(fd, (off_t)-16, SEEK_END);
```

Esempio: gestione di un hotel

Sia `residents` un file contenente la lista dei residenti di un hotel.

La linea 1 di tale file contiene il nome della persona che occupa la camera 1, la linea 2 contiene il nome della persona che occupa la camera 2, etc.

Ogni linea è lunga 41 caratteri, i primi 40 contengono il nome dell'occupante, l'ultimo è un `newline`.

La funzione `getoccupier`

```
/* getoccupier -- restituisce il nome dell'occupante la camera passata come
parametro */

#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

#define NAMELENGTH 41

char namebuf[NAMELENGTH];   /* buffer per contenere il nome */
int infile = -1;   /* conterra' il file descriptor di residents;
          l'inizializzazione serve affinche' venga aperto una sola volta */

char *getoccupier(int roomno)
{
  off_t offset;
  ssize_t nread;
  if (infile == -1 && (infile = open("residents", O_RDONLY)) == -1)
  {
     return (NULL);
  }
```

. . . codice di `getoccupier`

```
  offset = (roomno -1) * NAMELENGTH;

/* cerca la linea relativa alla camera e legge il nome dell'occupante */
  if (lseek(infile, offset, SEEK_SET) == -1)
     return (NULL);

  if ( (nread = read(infile, namebuf, NAMELENGTH)) <= 0)
     return (NULL);

/* crea una stringa rimpiazzando il newline con un terminatore null */
  namebuf[nread - 1] = '\0';
  return (namebuf);
}
```

Programma per la stampa dei nomi degli occupanti

```
/* stampa i nomi dei residenti in un albergo di 10 camere */

#define NROOMS   10

main()
{
  int j;
  char *getoccupier(int), *p;

  for (j=1; j <= NROOMS; j++)
  {
     if (p = getoccupier(j))
        printf("Room %2d, %s\n", j, p);
     else
        printf("Error on room %d\n", j);
  }
}
```

Esercizio

- Implementare un meccanismo per decidere se una camera è vuota, modificando eventualmente la funzione `getoccupier` e il file `residents`.
  Scrivere una procedura `findfree` per trovare la prima camera libera.

- Scrivere le procedure
  - `freeroom` per rimuovere un occupante da una camera;
  - `addguest` per assegnare una camera ad un ospite, controllando che questa sia libera.

- Utilizzando le funzioni `getoccupier`, `freeroom`, `addguest`, e `findfree`, scrivere un programma `frontdesk` per gestire il file `residents`.