File

Linguaggio C

Dichiarazione e apertura di un file

- **Dichiarazione**
  - FILE* mioFile; //mioFile è un puntatore a un FILE
- **Apertura**
- Sintassi
  - FILE* fopen("stringa_con_il_nome_del_file",metodo);
- Esempio
  - mioFile = fopen("fileSuDisco.dat","r") //esempio apertura in lettura
- **I vari metodi di apertura**:
  - "w" =Crea il file(se esiste lo sovrascrive), il puntatore si posiziona all'inizio
  - del file, è possibile solo scrivere nel file, non leggere
  - "r" =Apre il file solo in lettura,il puntatore si posiziona all' inizio
  - "w+" =Crea il file (se esiste lo sovrascrive), il puntatore si posiziona all'inizio
  - del file, è possibile sia scrivere che leggere
  - "r+" =Apre un file in lettura e scrittura, il puntatore si posiziona all' inizio
  - "a" =Apre il file in append.
  - "a+" =Apre il file in append e in lettura.
  - (Se ai metodi sopra elencati si aggiunge "b" (es. "rb+" o "wb") --> il file è aperto in binario)

Chiusura

- Sintassi:
  - int fclose(FILE*);
- Esempio
  - fclose(mioFile);

Posizionamento

- Per ottenere la posizione attuale all'interno di un file si utilizza:
  - long ftell(FILE*)
- Esempio:
  - long pos:
  - pos=ftell(mioFile)
- Spostamento:
  - int fseek(FILE*,long int offset,int tipo)
  - offset è un valore (con segno) che rappresenta i byte di spostamento
  - tipo è un codice che può assumere i valori:
    - 0 per spostarsi partendo dall' inizio del file
    - 1 per spostarsi partendo dalla posizione corrente
    - 2 per spostarsi partendo dalla fine del file

Lettura e scrittura

- fread viene utilizzata per leggere n byte da un file
- int fread (void *punt, dim_elemento, num_elementi, FILE *nomefile)
  - Legge un blocco di dati binari o testuali dal e li memorizza in un vettore identificato da punt
  - Restituisce il numero di elementi effettivamente letti.
- fwrite viene utilizzata per scrivere n byte su un file
- int fwrite (void *punt, dim_elemento, num_elementi, FILE *nomefile)
  - Scrive un blocco di dati binari sul file prelevandoli dal vettore identificato da punt
  - Restituisce il numero di elementi effettivamente scritti.
- E' utile utilizzare l'operatore sizeof() che per determinare il numero di byte da leggere o scrivere.

Scrittura e lettura formattata

- fprintf e fscanf funzionano esattamente come printf e scanf ma operano sul file che ricevono come primo parametro.
- fprintf(file_destinazione, stringa_di_controllo, elementi);
  - Restituisce il numero di elementi effettivamente scritti o un numero negativo in caso di errore.
- fscanf(file_sorgente, stringa_di_controllo, indirizzo_elementi);
  - Restituisce il numero di elementi effettivamente letti o un numero negativo in caso di errore.
- Notare che fprintf(stdout,""); ed fscanf(stdin,""); sono l'equivalente di printf e scanf.

Lettura e scrittura di caratteri

- fgetc,fgets,fputc,fputs funzionano in modo simile a getc,gets,putc,puts;
- int fputc(int c,*FILE);
  - Analogo a "putchar"; scrive un carattere alla volta (come intero). Restituisce il carattere in ingresso come intero o EOF in caso di errore.
- int fgetc(*FILE);
  - Analogo a "getchar"; legge carattere per carattere il contenuto del file puntato. Restituisce il carattere come intero o EOF in caso di errore

Lettura e scrittura di stringhe

- char *fgets (char *s, int n, FILE *nomefile)
  - Legge caratteri dal file fino a quando:
    - ha letto n-1 caratteri
    - ha raggiunto un carattere di newline
    - ha raggiunto la fine del file
    - Restituisce l'indirizzo della stringa ricevuta come parametro, oppure NULL in caso di errore.
- int *fputs (char *s, FILE *nomefile)
  - Scrive la stringa "s" sul file
    - restituisce 0 o un valore diverso in caso di errore.

Fine file

- feof(*FILE);
- feof restituisce *vero,true,1* dopo che viene effettuata la lettura oltre al file,quindi dopo che una lettura ha dato esito negativo.