**16° lezione prog1**

**Percorso relativo e percorso assoluto**

**I percorsi  assoluti(absolute path)**  su Linux partono tutti alla stessa radice / invece su Windows si distinguono le varie unità (C: D:). Ogni device su Windows identifica una radice come ad esempio un hard disk.

**Il percorso relativo(relative path)** parte con ./ che si riferisce alla directory  corrente su Linux.

**Operazioni sui files**

 Un file e una sequenza di bit il linguaggio C mette a disposizione delle funzioni che si utilizzano per manipolare i file(), operano sugli stream.

L'apertura di un file si esegue con la seguente  istruzione:

FILE \*fopen(char \* filename, const char mode);

 Questa funzione ha svariati parametri come:

1. **argomento**: una stringa che rappresenta il nome del file  e il  percorso (pathname) che può essere relativo o assoluto.Solitamente si utilizza quello relativo. **Es FILE \*fp=fopen(“input.txt”)In questo modo si sta indicando la  directory corrente**.Si deduce pertanto,che si tratta di un percorso relativo. Se invece si volesse specificare un percorso assoluto (./home..)
2. **argomento: è** una stringa che  specifica la modalità di apertura.

Ci sono diverse modalità di apertura dei file:

* **r:** read  legge il file (il file deve esistere altrimenti si ha un errore)
* **w:** scrive nel file e se non è stato creato lo crea. Sè il file è stato scritto e poi  lo si riapre in modalità w  il contenuto di questo viene perso.
* **a**: scrive nel file senza cancellarne il contenuto
* **w+** sia read che write con l’aggiunta della creazione del file se ancora non e’ stato creato
* **r+** sia read che write

**Questa funzione ritorna il puntatore al file(è il puntatore alla struttura file).Se si verifica qualche errore restituisce NULL, pertanto bisogna verificare se il file è stato aperto correttamente. Il puntatore al file va conservato per poter accedere al file.**Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

Per scrivere  nel file bisogna  ci hanno delle funzioni:

* **fprinf();** che vuole come prima parametro lo stream in questo caso il puntatore al file, si utilizza come la funzione printf(); Per capire se ci sono stati problemi in fase di scrittura questa funzione restituisce un intero il numero di caratteri stampati(numero positivo se la stampa è andata a buon fine). Con questa istruzione bisogna utilizzare fflush() passandogli come parametro il puntatore al file
* **fgets():** serve a leggere le stringhe in un file, ha due parametri una stringa e un puntatore al file. La memorizza nella locazione di memoria puntata da stringa. Con lun intendiamo il numero massimo di caratteri della stringa che può essere letto. Ha il seguente prototipo **char \*fgets**(**char** \*stringa, **int** lun, **FILE** \*fp);, memorizza il carattere \n
* **fscanf():** si utlizza per leggere token che vanno memorizzate in opportune variabili. **esempio** ret=fscanf(fp, ”%s %s %u”, s1, s2, x);. Per capire se e’ avvenuta una lettura corretta si va testare il valore dei token acquisiti se è uguale a NUL no ha letto valori.
* **fpunts():** si utilizza per scrive sul file vuole come parametri la stringa da scrivere e il puntatore al file. restituisce valore positivo se la scrittura è andata a buon fine e un numero <1 o EOF se la scrittura non è andata bene

**La chiusura del file**

Si ha un'indiscrezione per la chiusura del file **fclose()**. Questa funzione provoca l'immediata invocazione della funzione fflash, pertanto tutti i byte che stanno nel buffer saranno “scaricati” nel file, inoltre la risorsa **file descriptor** viene liberata. Il file descriptor si utilizzano per I/O sul S.O. sono limitati, vengo gestiti dal S.O. **Per evitare una saturazione di questi bisogna chiudere il file quando non lo si sta più utilizzando(v**errà approfondita maggiormente nella materia sistemi operativi[...]). Di default **i file verranno chiusi alla fine dell’esecuzione del programma**. Per verificare che il file si è chiuso correttamente bisogna testare che il valore restituito alla funzione non sia uguale a EOF. Se non si ha nessun errore verrà restituito 0. Ha il seguente prototipo: **int fclose(FILE \*stream).**

**Funzione EOF (End Of File)**

**La funzione ci dice se ci troviamo alla fine del file (se si è verificato un evento EOF)**

**Funzione perror()**

Si utilizza per customizzare un messaggio di stampa che spiega la natura dell’errore con la precedente istruzione

**La scrittura e la lettura di byte all’interno di un file(blocchi di byte)**

Con i file binari (.bin) si ha la possibilità di leggere e scrivere dei record all’interno del file.

**La funzione fwrite()Immagine che contiene testo, schermata, Carattere, bianco

Descrizione generata automaticamente**

Ha il seguente prototipo: **size t fwrite(const void \*restrict ptr, size t size, size t nmemb, FILE \*restrict stream)**

1. **parametro formale** è un puntatore ad un blocco di memoria
2. **parametro formale** indica il numero di blocchi ad esempio se si definisce una struttura chiamata record formata da due stringhe ed un numero scriverò **sizeof(struct record)**
3. **parametro formale (n\_memb)** numero di blocchi da scrivere (size\_t) 1. In questo modo sto dicendo che si deve allocare un blocco di **sizeof(struct record).**
4. **Parametro formale (stream):** puntatore al file.

**Restituisce il numero di elementi scritti.** Questa informazione può essere utilizzata per verificare la corretta scrittura sul file**.**

if(fwrite(&record\_to\_write, sizeof(struct record), (size\_t) 1, fp)<1){  
perror("\n Error writing record ");   
onemore = 0;

**La funzione fread()**

Ha la seguente sintassi: **size t fwrite(const void \*restrict ptr, size t size, size t nmemb, FILE \*restrict stream);**

1. **parametro formale** è un puntatore ad un blocco di memoria
2. **parametro formale** indica il numero di blocchi ad esempio se si definisce una struttura chiamata record formata da due stringhe ed un numero scriverò **sizeof(struct record)**
3. **parametro formale (n\_memb)** numero di blocchi da scrivere (size\_t) 1. In questo modo sto dicendo che si deve allocare un blocco di **sizeof(struct record).**
4. **Parametro formale (stream):** puntatore al file.

**Restituisce il numero di elementi letti.** Questa informazione può essere utilizzata per verificare la corretta lettura del file**.**

if((ret=fread(&my\_record, sizeof(struct record), 1, fp))<1){ *//may be EOF* if(feof(fp))  
 printf("\n EOF!");  
 else  
 printf("\n problema in lettura! ");   
}

**La funzione fseek()**

Ha il seguente prototipo: **int fseek(FILE \*stream, long offset,int whance);**

Quando si esegue una lettura o una scrittura la testina (l’indicatore di posizione del file si sposta) in avanti. Ha 3 paramntri:

1. **File \*stream**: il puntatore al file
2. **offset**: per far spostare la testina avanti con offset positivo e indietro con offset negativo.
3. **whance**: è il punta dal quale si inzia a contare per poszionare la testina.
   * **SEEK\_CUR** posizione corrente
   * **SEEK\_END** fine del file
   * **SEEK\_SET** inzio del file

if(fseek(fp, -sizeof(struct record), SEEK\_CUR)<0)  
 perror("\n Error in fseek() call");  
else if(fwrite(&my\_record, sizeof(struct record), 1, fp)<1)  
 fprintf(stderr, "\n Error writing the updated record..");  
else  
 nwrite = 1;