```
int fclose(FILE* stream); //Chiude il file o lo stream specificato, eventualmente dopo aver svuotato i buffer ad esso associati (vedi fflush()).
int feof(FILE* stream); //Controlla l'indicatore di fine file per un dato stream.
int ferror(FILE* stream); //Controlla l'indicatore di errore per un dato stream. Se La funzione ritorna un valore diverso da zero significa che è avvenuto un errore durante una precedente operazione di input/output.
int fflush(FILE* stream); //Svuota i buffer per un dato stream trasferendo i dati dalla memoria al supporto fisico.
int fgetpos(FILE* stream, fpos_t* pos); //Riporta la posizione corrente di lettura o scrittura nel file identificato dal primo parametro e la salva nella variabile specificata nel secondo. In caso di errore, la variabile errno viene impostata a un codice identificativo dell'errore.
FILE* fopen(const char* path, const char* mode); //Apre un file identificato dal parametro path con la modalità descritta da mode e vi associa una struttura descrittiva di tipo FILE. In caso di errore, la variabile errno viene impostata a un codice identificativo dell'errore.
      "r":  sola lettura
      "r+": lettura/scrittura
      "w":  scrittura. Se il file non esiste, viene creato; se invece il file esiste, ne viene cancellato il contenuto.
      "w+": lettura/scrittura. Se il file non esiste, viene creato; se invece il file esiste, ne viene cancellato il contenuto.
      "a":  scrittura append. Se il file non esiste, viene creato; se invece il file esiste, il contenuto viene conservato inalterato ed i nuovi dati scritti vengono aggiunti dopo quelli preesistenti.
      "a+": lettura/scrittura append. la scrittura è permessa solo alla fine del file. Se il file non esiste, viene creato; se invece il file esiste[vedi sopra]
size_t fread(void* data, size_t size, size_t count, FILE* stream); //Legge dallo stream stream un numero di elementi specificato da count, ognuno della dimensione in byte size. Il numero totale di byte che la funzione tenta di leggere dallo stream è size*count. I dati letti vengono memorizzati nell'array data. La dimensione size è normalmente specificata come sizeof(tipoDati).
FILE* freopen(const char* path, const char* mode, FILE* stream); //Chiude lo stream stream e associa allo stesso un nuovo file. È grossomodo equivalente a fclose() seguito da fopen(), salvo per il fatto che il parametro stream è garantito rimanere inalterato tramite la funzione freopen(). Un comune utilizzo di questa funzione è il riassegnamento degli stream standard stdin, stdout e stderr a file arbitrari.
int fseek(FILE* stream, long int offset, int partenza); //Sposta la posizione attuale nello stream: la seguente operazione di lettura o scrittura avverrà alla posizione indicata. In caso di corretta esecuzione, il flag di end of file viene resettato e vengono cancellati tutti i caratteri inseriti nel buffer di lettura tramite la funzione ungetc().
int fsetpos(FILE* stream, const fpos_t* pos); //Sposta la posizione attuale nello stream ad una posizione precedentemente memorizzata attraverso la funzione fgetpos(). In caso di corretta esecuzione, il flag di end of file viene resettato e vengono cancellati tutti i caratteri inseriti nel buffer di lettura tramite la funzione ungetc(). In caso di errore, la variabile errno viene impostata a un codice identificativo dell'errore.
long int ftell(FILE* stream); //Riporta la posizione corrente del puntatore di lettura/scrittura del file stream. In caso di errore, la variabile errno viene impostata a un codice identificativo dell'errore.
size_t fwrite(const void * data, size_t size, size_t count, FILE* stream); //Scrive un numero count di elementi di dimensione size dall'array data nello stream stream.
char* gets(char* s); // "Never use gets"
void rewind(FILE* stream); //Reimposta la posizione del puntatore di lettura/scrittura del file stream all'inizio del file. Gli indicatori di errore e di fine file sono reimpostati a zero.
FILE* tmpfile(); //Crea un file temporaneo e lo apre in modalità "wb+" (vedi fopen()). Alla chiusura dello stream o al termine del programma, il file viene cancellato.
```

<stdio.h>

```
int fprintf ( FILE * stream, const char * format, ... ); //Writes the C string pointed by format to the stream. If format includes format specifiers (subsequences beginning with %), the additional arguments following format are formatted and inserted in the resulting string replacing their respective specifiers.
int fscanf ( FILE * stream, const char * format, ... ) //Reads data from the stream and stores them according to the parameter format into the locations pointed by the additional arguments.
int printf ( const char * format, ... ) //Writes the C string pointed by format to the standard output (stdout). If format includes format specifiers (subsequences beginning with %), the additional arguments following format are formatted and inserted in the resulting string replacing their respective specifiers.
int scanf ( const char * format, ... ) //Reads data from stdin and stores them according to the parameter format into the locations pointed by the additional arguments.
int snprintf ( char * s, size_t n, const char * format, ... ) //Composes a string with the same text that would be printed if format was used on printf, but instead of being printed, the content is stored as a C string in the buffer pointed by s (taking n as the maximum buffer capacity to fill). If the resulting string would be longer than n-1 characters, the remaining characters are discarded and not stored, but counted for the value returned by the function.
int sprintf ( char * str, const char * format, ... ) //Composes a string with the same text that would be printed if format was used on printf, but instead of being printed, the content is stored as a C string in the buffer pointed by str.
int sscanf ( const char * s, const char * format, ...) //Reads data from s and stores them according to parameter format into the locations given by the additional arguments, as if scanf was used, but reading from s instead of the standard input (stdin).
```

Varie ed eventuali

```
void node_insert(struct node* parent, struct node* nd){
    if ( parent->value < nd->value ){
        if ( parent->left ){
            node_insert(parent->left,nd);
            return;
        }
        parent->left = nd;
        return;
    }
    if ( parent->right ){
        node_insert(parent->right, nd);
        return;
    }
    node->right = nd;
}

struct node* node_new(int value){
    struct node* nd = NEW(struct node);
    nd->value = value;
    nd->left = NULL;
    nd->right = NULL;
    return nd;
}
```