TECNICHE DI PROGRAMMAZIONE

LAB 1: *INPUT - OUTPUT*

GESTIONE INPUT E OUTPUT – ESERCIZIO 0

Definizioni e controlli

```c
#include <stdio.h>
#define PIGRECO 3.14159

int main(void) {
    FILE *fp_read, *fp_write;
    int num_int;
    double result;
    char letter, string[100+1];

    if ((fp_read = fopen("../Input.txt", "r")) == NULL)
{
        printf("Error opening file\n");
        return 1;
    }

    if ((fp_write = fopen("../Output.txt", "w")) ==
NULL) {
        printf("Error opening file\n");
        return 2;
    }
```

I/O da tastiera

```c
// scanf integer
    printf("Enter a number:");
    scanf("%d", &num_int);
    printf("The number is: %d \n ", num_int);

    // casting
    result = (double)(num_int) / 3 * PIGRECO;
    printf("The result is: %f \n", result);

    // scanf and printf char
    printf("Enter character one:");
    scanf("%c", &letter);
    printf("The character is: %c \n", letter);

    // getchar and putchar character
    printf("Enter a character:");
    letter = getchar();
    printf("The character is: ");
    putchar(letter);
    putchar('\n');

    // gets and puts string
    printf("Enter a string:-");
    gets(string);
    printf("The string is: ");
    puts(string);
```

I/O da file

```c
// fgetc and fputc string
    while (!feof(fp_read)) {
        letter = fgetc(fp_read);
        if (!feof(fp_read)) {
            fputc(letter, fp_write);
        }
    }

    rewind(fp_read);

    // fgets and fputs string
    while (!feof(fp_read)) {
        fgets(string, 20, fp_read);
        if (!feof(fp_read)) {
            fputs(string, fp_write);
        }
    }

    rewind(fp_read);

    // fscanf and fprintf string
    while (!feof(fp_read)) {
        fscanf(fp_read,"%s", string);
        if (!feof(fp_read)) {
            fprintf(fp_write, "%s", string);
        }
    }
```

Chiusura file

```c
    fclose(fp_read);
    fclose(fp_write);

    return 0;
}
```

DEFINIZIONI E CONTROLLI

```c
#include <stdio.h>
#define PIGRECO 3.14159

int main(void) {
    FILE *fp_read, *fp_write;
    int num_int = 0;
    double result;
    char letter, string[100+1];

    if ((fp_read = fopen("../Input.txt", "r")) == NULL) {
        printf("Error opening file\n");
        return 1;
    }

    if ((fp_write = fopen("../Output.txt", "w")) == NULL) {
        printf("Error opening file\n");
        return 2;
    }
```

Includere le librerie necessarie

Associa un identificatore ad una costante letterale (`define` non richiede = o `;`)

Creare il `main` del programma (senza non è possibile eseguire il codice)

Definire le variabili, i file richiedono il `*` davanti al nome (poiché sono puntatori), un valore iniziale può essere assegnato ad una variabile durante la definizione, le stringhe di caratteri devono essere create specificando il numero di caratteri +1 (non dimenticare lo `\0`) oppure assegnando direttamente la stringa. Ogni variabile ha bisogno del suo formato che va dichiarato all'inizio.

Controllare i file quando vengono aperti, se il puntatore al file restituisce `NULL` c'è stato un errore ed il programma viene terminato.

I/O DA TASTIERA

```
// scanf integer
    printf("Enter a number:");
    scanf("%d", &num_int);
    printf("The number is: %d \n ", num_int);

    // casting
    result = (double)(num_int) / 3 *
PIGRECO;
    printf("The result is: %f \n", result);

    // scanf and printf char
    printf("Enter character one:");
    scanf("%c", &letter);
    printf("The character is: %c \n", letter);

    // getchar and putchar character
    printf("Enter a character:");
    letter = getchar();
    printf("The character is: ");
    putchar(letter);
    putchar('\n');

    // gets and puts string
    printf("Enter a string:");
//    getchar();
    gets(string);
    printf("The string is: ");
    puts(string);
```

scanf permette di leggere i valori inseriti da tastiera, è necessario specificare il formato (`%d`, `%f`, `%c`, `%s`, etc.) e la variabile dove salvare il valore (se è un valore numerico o un carattere bisogna precedere la variabile con **&**)

printf permette di stampare a schermo variabili, è necessario specificare il formato ma la variabile **NON** deve essere preceduta da **&**

Il casting avviene mettendo tra parantesi la variabile da trasformare preceduta dal formato di conversione scelto tra parentesi.

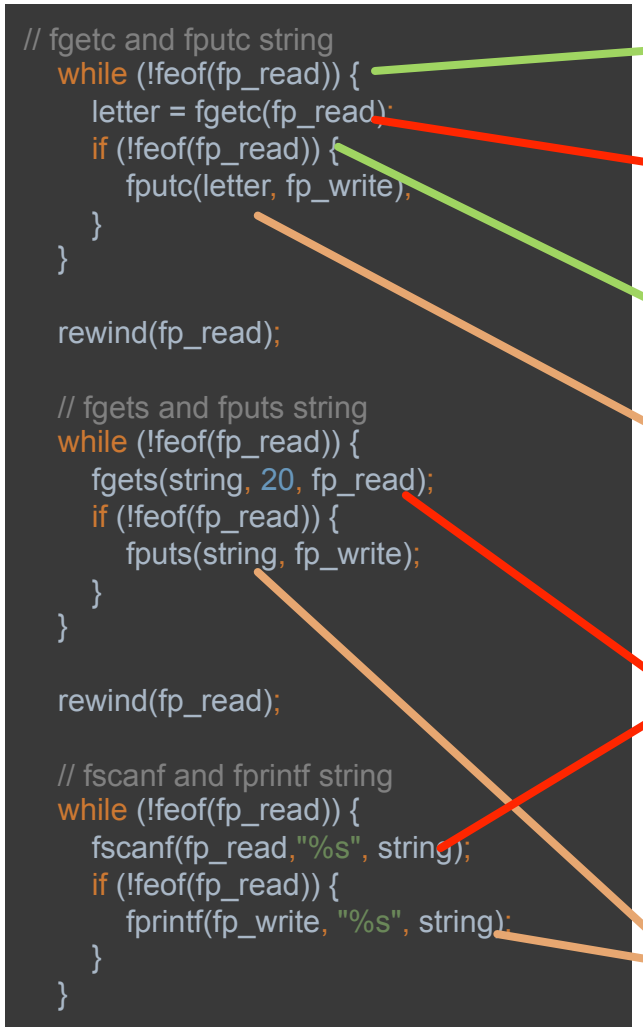
`letter=getchar()` e *`scanf(formato, &variabile)`* sono due metodi per leggere singoli caratteri da tastiera

`putchar(letter)` e *`printf(formato, variabile)`* sono due metodi per scrivere singoli caratteri a schermo

`gets(string)` permette di leggere una stringa da tastiera (fino ad a capo o `EOF`)

`puts(string)` permette di scrivere una stringa a schermo (`stdout`)

I/O DA FILE



```
// fgetc and fputc string
    while (!feof(fp_read)) {
        letter = fgetc(fp_read);
        if (!feof(fp_read)) {
            fputc(letter, fp_write);
        }
    }

    rewind(fp_read);

    // fgets and fputs string
    while (!feof(fp_read)) {
        fgets(string, 20, fp_read);
        if (!feof(fp_read)) {
            fputs(string, fp_write);
        }
    }

    rewind(fp_read);

    // fscanf and fprintf string
    while (!feof(fp_read)) {
        fscanf(fp_read,"%s", string);
        if (!feof(fp_read)) {
            fprintf(fp_write, "%s", string);
        }
    }
```

Per leggere tutto il file è necessario effettuare un ciclo finchè non si raggiungere la fine (EOF).

In maniera analoga esistono delle funzioni per leggere da file,
`letter=getc(puntatore_file)` permette di leggere un singolo carattere dal file

La funzione `!feof(`*`puntatore_file`*`)` é vera soltanto **DOPO** aver tentato di leggere oltre EOF.

`fputc(carattere, puntatore_file)` permette di scrivere un singolo carattere su file

Esistono due metodi per leggere stringhe da file:
`fgets(stringa, len, puntatore_file)` permette di leggere una `stringa` di lunghezza massima `len-1` da un file `puntatore_file`.
`fscanf(puntatore_file, formato, stringa)` permette di leggere una stringa nel `formato` specificato ma se la stringa supera la lunghezza dichiarata nella fase di definizioni delle variabili, il programma da errore durante l'esecuzione.

`fputs(stringa, puntatore_file)` e *`fprintf(puntatore_file, formato, stringa)`* sono due metodi per scrivere stringhe su file

CHIUSURA FILE E PROGRAMMA

```
    fclose(fp_read);
    fclose(fp_write);

    return 0;
}
```

`fclose(puntatore_file)` è necessario ricordarsi di chiudere il file una volta che non è più necessario usarlo

Il codice di ritorno del programma, se tutto è andato a buon fine il programma ritorna 0.