

**LEGGETE LA GUIDA PER LA CREAZIONE DEI PROGETTI E PER IL DEBUGGING!**

*Gli esercizi seguenti devono essere risolti, compilati e testati utilizzando il debugger. Per ognuno si deve realizzare una funzione main() che ne testi il funzionamento. **Fate progetti diversi per ogni esercizio.***

Attenzione! La Microsoft ha definito funzioni non standard più sicure di quelle definite nello standard e segnala l'uso di funzioni considerate pericolose con questo warning (o errore):

warning C4996: '<nome funzione>': This function or variable may be unsafe. Consider using <nome funzione sicura> instead. To disable deprecation, use \_CRT\_SECURE\_NO\_WARNINGS. See online help for details.

In questo corso utilizzeremo solo le versioni standard e quindi per convincere Visual Studio a fare il suo dovere aggiungete **come prima riga di ogni file .c che produce l'errore** la seguente definizione

```
#define _CRT_SECURE_NO_WARNINGS
```

## Esercizio iniziale da fare con calma

Creare il file main.c contenente:

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int x = 266;
    char s[] = "prova";
    FILE *f;

    f = fopen("out1", "wt");
    fprintf(f, "%i\n%s", x, s);
    fclose(f);

    f = fopen("out2", "wb");
    fprintf(f, "%i\n%s", x, s);
    fclose(f);

    f = fopen("out3", "wt");
    fwrite(&x, sizeof x, 1, f);
    fwrite(&s, sizeof s, 1, f);
    fclose(f);

    f = fopen("out4", "wb");
    fwrite(&x, sizeof x, 1, f);
    fwrite(&s, sizeof s, 1, f);
    fclose(f);

    return 0;
}
```

Questo programma genera 4 file. Facendo molta attenzione alla modalità di apertura dei file, determinate **prima di eseguirlo** quanti byte saranno grandi i file e quale sarà il loro contenuto. Poi verificate la dimensione e riflettete nuovamente in caso di errore. Infine usate HxD per verificare che cosa contengono. Vi ricordo che lo eseguite sotto Windows su una macchina little endian.

## Esercizio 1

Creare i file `lettura.h` e `lettura.c` che consentano di utilizzare la seguente funzione:

```
extern int *leggiinteri(const char *filename, size_t *size);
```

È stato definito un formato binario di dati per memorizzare numeri interi a 32 bit, che consiste in un numero intero a 32 bit che indica quanti numeri sono presenti nel file, seguito da altrettanti numeri interi a 32 bit. Tutti i numeri sono in little endian.

Ad esempio un file contenente i valori 7, 8 e 9 verrebbe codificato con i seguenti byte (rappresentati in esadecimale nel seguito):

```
03 00 00 00 07 00 00 00 08 00 00 00 09 00 00 00
```

La funzione `leggiinteri()` riceve in input il nome di un file come stringa C e deve: aprire il file in lettura in modalità non tradotta (binaria), leggere il primo intero (chiamiamolo  $n$ ), allocare dinamicamente memoria per  $n$  interi, leggere gli  $n$  interi, impostare `size` al valore di  $n$ , e ritornare un puntatore alla memoria allocata. Se l'apertura fallisce o se non è possibile leggere tutti gli  $n$  interi, ritornare `NULL`. Ricordarsi comunque di chiudere il file se l'apertura è andata a buon fine.

## Esercizio 2

Creare i file `lettura.h` e `lettura.c` che consentano di utilizzare la seguente funzione:

```
extern int *leggiinteri2(const char *filename, size_t *size);
```

È stato definito un formato binario (veramente banale) di dati per memorizzare numeri interi a 32 bit, che consiste nel memorizzare i numeri interi a 32 bit in little endian tutti in sequenza. Ad esempio un file contenente i valori 7, 8 e 9 verrebbe codificato con i seguenti byte (rappresentati in esadecimale nel seguito):

```
07 00 00 00 08 00 00 00 09 00 00 00
```

La funzione `leggiinteri2()` riceve in input il nome di un file come stringa C e deve: aprire il file in lettura in modalità non tradotta (binaria), leggere gli interi allocando dinamicamente la memoria, impostare `size` al numero di interi trovati nel file, e ritornare un puntatore alla memoria allocata. Se l'apertura fallisce o se non è possibile leggere nemmeno un intero, ritornare `NULL`. Ricordarsi comunque di chiudere il file se l'apertura è andata a buon fine.

## Esercizio 3

Creare i file `lettura.h` e `lettura.c` che consentano di utilizzare la seguente funzione:

```
extern double *leggidouble(const char *filename, size_t *size);
```

È stato definito un formato testo (veramente banale) di dati per memorizzare numeri con la virgola, che consiste nel memorizzare i numeri in formato testo in base 10 separandoli con whitespace. Ad esempio un file contenente i valori 7.2, 8.19 e 9.0 verrebbe codificato come:

```
7.2 8.19 9.0
```

o anche

```
7.2↵  
8.19↵  
9.0↵
```

o anche

```
7.2↵  
8.19 ↵  
→ 9.0↵
```

(Con ↵ indichiamo l'<a capo> e con → indichiamo il tab)

La funzione `leggidouble()` riceve in input il nome di un file come stringa C e deve: aprire il file in lettura in modalità tradotta (testo), leggere i numeri in variabili di tipo `double` allocando dinamicamente la memoria, impostare `size` al numero di numeri trovati nel file, e ritornare un puntatore alla memoria allocata. Se l'apertura fallisce o se non è possibile leggere nemmeno un numero, ritornare `NULL`. Ricordarsi comunque di chiudere il file se l'apertura è andata a buon fine.

## Esercizio 4

Creare i file `scrittura.h` e `scrittura.c` che consentano di utilizzare la seguente funzione:

```
extern int scrivimaiuscolo(const char *filename);
```

La funzione riceve in input il nome di un file come stringa C e deve: aprire il file in lettura in modalità tradotta (testo), se l'apertura fallisce ritornare 0, altrimenti leggere tutti i caratteri dal file e scriverli su `stdout` convertendoli in maiuscolo se sono lettere minuscole, infine ritornare 1. Ricordarsi di chiudere il file se l'apertura è andata a buon fine.