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

Nel file `file.c` implementare la definizione della funzione:

```
extern bool scrivi_intero(const char *filename, int i);
```

La funzione accetta come parametro una stringa C che contiene un nome di file e un numero intero a 32 bit con segno e deve:

1. aprire il file in modalità scrittura non tradotta (testo)
2. scrivere sul file in formato testo il valore di i in base 10
3. chiudere il file

La funzione ritorna false se `filename` è `NULL` o se non riesce ad aprire correttamente il file.

In questo esercizio non create un file `stringa.h`, e nel file `main.c` mettete la funzione `main` che utilizza la funzione `scrivi_intero`. Il `main` non deve generare warning.

Esercizio 2

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 `leggininteri()` 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 3

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 `leggininteri2()` 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 4

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 5

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.

Esercizio 6

Nel file `cornicetta.c` implementare la definizione della funzione:

```
extern void stampa_cornicetta (const char *s);
```

La funzione deve inviare a `stdout` la stringa passata come parametro circondandola con una cornicetta composta dei caratteri ‘\’ e ‘/’ agli spigoli e di ‘-’ e ‘|’ sui lati. Prima e dopo la stringa bisogna inserire uno spazio. Ad esempio chiamando la funzione con `s=“ciao”`, la funzione deve inviare su `stdout`:

```
/------\
| ciao |
\------/
```

Ovvero (visualizzando ogni carattere in una cella della seguente tabella):

/	-	-	-	-	-	-	\	<a capo>
|	<sp.>	c	i	a	o	<sp.>	|	<a capo>
\	-	-	-	-	-	-	/	<a capo>

Si ricorda che in C il carattere ‘\’ deve essere inserito come ‘\\’. Gli <a capo> a fine riga sono obbligatori per una soluzione corretta.

Esercizio 7

Creare il file `trim.c` che consenta di utilizzare la seguente funzione:

```
extern char *trim(const char *s);
```

La funzione accetta una stringa zero terminata e ritorna un’altra stringa zero terminata, allocata dinamicamente nell’heap, contenente i caratteri della stringa in ingresso, senza tutti gli spazi iniziali e finali. La funzione deve ritornare `NULL` (e quindi non allocare memoria) se `s` è `NULL`.
Ad esempio:

```
“senza spazi”     → “senza spazi”
“ prima”          → “prima”
“dopo ”           → “dopo”
“ a b ”           → “a b”
“ ”               → “”
```

Esercizio 8

Nel file `felici.c` implementare la definizione della funzione:

```
extern int felice(unsigned int num);
```

La funzione prende come input il valore `num` e ritorna 1 se il numero è felice, 0 se è infelice.

Un numero felice è definito tramite il seguente processo: partendo con un qualsiasi numero intero positivo, si sostituisca il numero con la somma dei quadrati delle sue cifre, e si ripeta il processo fino a quando si ottiene 1 (dove ulteriori iterazioni porteranno sempre 1), oppure si entra in un

ciclo che non include mai 1. I numeri per cui tale processo dà 1 sono numeri felici, mentre quelli che non danno mai 1 sono numeri infelici. È possibile dimostrare che se nella sequenza si raggiunge il 4, il numero è infelice. Possiamo estendere il concetto allo 0, che ovviamente genera la sequenza composta solo di 0 e quindi possiamo considerarlo infelice.

Ad esempio, 7 è felice e la sequenza ad esso associata è:

$$7 \to 7^2 = 49 \to 4^2 + 9^2 = 97 \to 9^2 + 7^2 = 130 \to 1^2 + 3^2 + 0^2 = 10 \to 1^2 + 0^2 = 1$$

mentre 8 è infelice e la sequenza ad esso associata è:

$$8 \to 8^2 = 64 \to 6^2 + 4^2 = 52 \to 5^2 + 2^2 = 29 \to 2^2 + 9^2 = 85 \to 8^2 + 5^2 = 89 \to$$
$$8^2 + 9^2 = 145 \to 1^2 + 4^2 + 5^2 = 42 \to 4^2 + 2^2 = 20 \to 2^2 + 0^2 = 4$$