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 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 il file `conversione.c` in cui deve essere definita la funzione corrispondente alla seguente dichiarazione:

```
extern char *converti(unsigned int n);
```

La funzione accetta come parametro un numero naturale o nullo `n` e deve restituire un puntatore ad un array di caratteri zero terminato allocato dinamicamente, contenente la rappresentazione in base 10 del numero intero, con le singole cifre rappresentate in ASCII. Ad esempio:
il numero 4355 genererà l'array di caratteri: "4355", ovvero { 52, 51, 53, 53, 0 }, o anche { 0x34, 0x33, 0x35, 0x35, 0x00 }.

Provate a fare questo esercizio prima senza e poi con la funzione `sprintf()`.

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

Esercizio 3

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

```
extern size_t conta_parole (const char *s);
```

La funzione accetta come parametro una stringa C e deve restituire in un dato di tipo `size_t` quante parole sono presenti all'interno della stringa, dove con "parola" intendiamo una sequenza di caratteri diversi da spazio. Ad esempio, colla stringa `"  Questa e' una stringa lunga 45 caratteri.  "` dovrebbe ritornare 7. Colla stringa `"1 2 3 a b c"` dovrebbe ritornare 6.

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

Esercizio 4

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

```
extern unsigned short bin2bcd(unsigned short val);
```

La funzione accetta come parametro un numero intero non negativo minore di 10000 e lo ritorna codificato in Binary Coded Decimal (BCD).

La codifica BCD è un modo comunemente utilizzato in informatica ed elettronica per rappresentare le cifre decimali in codice binario. In questo formato ogni cifra di un numero è rappresentata da un codice binario di quattro bit, il cui valore è compreso tra 0 (0000) e 9 (1001). Le restanti sei combinazioni non sono utilizzate. Per esempio il numero 127 è rappresentato in BCD come la sequenza di bit 0001, 0010, 0111.

Questa funzione impacchetta le 4 cifre di val dalla più significativa alla meno significativa in 16 bit (senza segno). Nel caso di 127 quindi produrrebbe 0000.0001.0010.0111, o in esadecimale (alla C) 0x0127. Cioè ogni cifra in base 10 viene convertita nella corrispondente cifra in base 16.
Il numero 0 diventerebbe 0x0000, il numero 9999 diventerebbe 0x9999 e così via.