Esercizio 1

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

```
extern char *concatena (const char *s1, const char *s2);
```

La funzione accetta come parametri due stringhe zero terminate e deve restituire un puntatore ad una stringa (allocata dinamicamente nell'heap) formata dalla prima seguita dalla seconda. Ricordarsi il terminatore.
Ad esempio, date le stringhe "prova" e "test" la funzione deve ritornare la stringa "provatest".

Esercizio 2

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

```
extern void stampa_quadrato (unsigned int lato);
```

La funzione deve inviare a `stdout` un quadrato composto da un contorno di asterischi e riempito di spazi di lato `lato`. Ad esempio chiamando la funzione con `lato=5`, la funzione deve inviare su stdout il seguente output:

```
*****
*   *
*   *
*   *
*****
```

Esercizio 3

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

```
extern double calcola_seno(double x);
```

La funzione deve calcolare il valore di $\operatorname{sen} x$ utilizzando la seguente equazione:

$$\operatorname{sen} x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Il parametro x è un angolo espresso in radianti. Si ottiene una precisione sufficiente con almeno 20 iterazioni. Un esempio di chiamata è il seguente:

```
int main (void) {
	double x = 3.14159265/4.0;
	double y;

	y = calcola_seno(x);
}
```

In questo caso y vale (circa) 0.707107

Esercizio 4

Creare i file `dati.h` e `dati.c` che consentano di utilizzare la seguente struttura:

```
struct dato {
	double valore;
	unsigned classe;
};
```

e la funzione:

```
extern struct dato *read_dati (const char *filename, unsigned int *pn);
```

La struttura contiene il campo `valore` che descrive un certo valore numerico e il campo `classe` che descrive la tipologia del dato. La classe sarà un valore tra 0 e 10.
La funzione accetta come parametro un nome di file che deve essere aperto in lettura in modalità tradotta (testo) e un puntatore ad una variabile di tipo `unsigned int` in cui si dovrà inserire il numero di dati presenti in un file così strutturato:

```
<valore><whitespace><classe><a capo>
<valore><whitespace><classe><a capo>
<valore><whitespace><classe><a capo>
...
```

La funzione deve ritornare un puntatore ad una zona di memoria (allocata dinamicamente nell'heap) contenente tutti i dati letti dal file.

Ad esempio, un file valido è:

```
0.0961		2
0.4929		0
0.9252		0
0.2186		1
```

In questo caso la variabile puntata da `pn` varrà 4. Per testare la funzione, utilizzare i file `dati1.txt`, `dati2.txt`, `dati3.txt` disponibili nella pagina dell'esame.

Esercizio 5

Estendere l'esercizio precedente aggiungendo al file `dati.h` la dichiarazione e al file `dati.c` la definizione della seguente funzione:

```
extern int salva_conteggio_bin (const char *filename, struct dato *pdati, unsigned int n);
```

La funzione accetta come parametro un nome di file che deve essere aperto in scrittura in modalità non tradotta (binario), un puntatore alla zona di memoria contenente variabili di tipo `struct dato` e il numero di dati contenuti in quella zona di memoria. La funzione deve scrivere sul file 11 interi senza segno a 32 bit in little endian corrispondenti al numero di dati di ogni possibile classe (da 0 a 10).
Ad esempio, chiamando la funzione in questo modo:

```
#include "dati.h"
int main (void) {
	struct dato x[] = { {0.1,0}, {1.45,2}, {0.81,0} };
	salva_conteggio_bin("file.bin",x,3);
}
```

il file `file.bin`, visto in un editor esadecimale conterrebbe:

```
02 00 00 00 00 00 00 00 01 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
```