title: Esame di Laboratorio del 14/07/2021 tags: fdi1, esame

Esercizio 1 (5 Punti)

Nel file `minmax.c` inserire la definizione della funzione:

```
extern void minmax(const int *v, size_t n, int *min, int *max);
```

Dato un vettore di numeri interi `v` e la sua lunghezza `n`, la funzione deve trovare i valori minimo e massimo contenuti nel vettore e inserirli nelle variabili puntate da `min` e `max`.

Se `n=0` o `v=NULL` la funzione non modifica le variabili puntate da `min` e `max`.

Esercizio 2 (6 punti)

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

```
extern int scrivi_conta_caratteri(FILE *f, const char* s);
```

La funzione `scrivi_conta_caratteri()` accetta come parametri un puntatore a file `f` già aperto in scrittura in modalità tradotta (testo) ed una stringa C `s`.

La funzione scrive sul file `f` il contenuto della stringa `s`, seguito da un ritorno a capo, seguito dalla sequenza periodica `1234567890` ripetuta per tutti i caratteri di `s`, in modo da facilitare l'utente nell'identificare la posizione dei caratteri all'interno della stringa. La funzione restituisce il numero di caratteri della stringa `s` correttamente scritti su file. Se, per qualsivoglia motivo, la funzione non scrive nessun elemento sul file, essa quindi restituisce `0`.

Ad esempio, data in input la stringa `scrivimi su file per completare l'esercizio.`:

la funzione scriverà su file:

```
scrivimi su file per completare l'esercizio.↵
12345678901234567890123456789012345678901234↵
```

e restituirà 44. Con il simbolo ↵ indichiamo il ritorno a capo.

Esercizio 3 (7 punti)

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

```
struct punto {
    double x, y;
```

```
};
```

e la funzione:

```
extern struct punto *punti_in_cerchio(const struct punto *p, size_t n, double r, s
```

La `struct point` consente di rappresentare un punto nel piano cartesiano, caratterizzato dalle sue coordinate sull'asse x e sull'asse y.

La funzione `punti_in_cerchio()` accetta come parametri un puntatore a vettore di `struct punto` contenente `n` elementi, un raggio `r`, e un puntatore a `size_t count`.

La funzione restituisce un vettore che contiene tutti i punti contenuti nel vettore `p`, che sono all'interno di un cerchio di raggio `r` con centro nell'origine.

La funzione inserisce la dimensione del vettore di ritorno nella zona di memoria a cui punta `count`. Se `p=NULL`, oppure se `n=0`, oppure se `r` non è positivo, oppure se nessun punto è all'interno del cerchio, la funzione restituisce `NULL` e imposta il valore puntato da `count` a `0`.

Ad esempio, dato in input il vettore di 3 punti:

```
v = { (5, 1)  (4, 3.7)  (6.3, 8) }
```

eseguendo `punti_in_cerchio(v, 3, 6.0, count)`, la funzione restituisce il vettore:

```
{ (5, 1)  (4, 3.7) }
```

e imposta il valore puntato da `count` a 2. Eseguendo invece `punti_in_cerchio(v, 3, 2.0, count)`, la funzione restituisce `NULL` e imposta il valore puntato da `count` a `0`.

Esercizio 4 (7 punti)

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

```
extern void stampa_quadrati_concentrici(int n);
```

La funzione `stampa_quadrati_concentrici` riceve un singolo valore in input `n`, e scrive su standard output un quadrato come da esempio, con `n=4`:

```
4 4 4 4 4 4 4
4 3 3 3 3 3 4
4 3 2 2 2 3 4
4 3 2 1 2 3 4
4 3 2 2 2 3 4
```

```
4 3 3 3 3 3 4
4 4 4 4 4 4 4
```

Per n=4, la funzione scrive su standard output un quadrato con un singolo `1` al centro, circondato dal valore2 ripetuto in un quadrato di 3×3, circondato dal valore3 ripetuto in un quadrato di 5×5 , circondato dal valore 4 ripetuto in un quadrato di 7×7. Nelle righe i valori adiacenti sono separati da uno spazio.

La funzione scrive quindi, uno dentro l'altro, contorni quadrati di numeri crescenti fino a stampare il contorno esterno composto dal valore di n. Per numeri maggiori di `10`, la funzione stampa l'ultima cifra del numero.

Ad esempio, per n=11 l'output corretto è:

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 1
1 0 9 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 0 1
1 0 9 8 7 7 7 7 7 7 7 7 7 7 7 7 8 9 0 1
1 0 9 8 7 6 6 6 6 6 6 6 6 6 6 7 8 9 0 1
1 0 9 8 7 6 5 5 5 5 5 5 5 5 6 7 8 9 0 1
1 0 9 8 7 6 5 4 4 4 4 4 4 4 5 6 7 8 9 0 1
1 0 9 8 7 6 5 4 3 3 3 3 3 4 5 6 7 8 9 0 1
1 0 9 8 7 6 5 4 3 2 2 2 3 4 5 6 7 8 9 0 1
1 0 9 8 7 6 5 4 3 2 1 2 3 4 5 6 7 8 9 0 1
1 0 9 8 7 6 5 4 3 2 2 2 3 4 5 6 7 8 9 0 1
1 0 9 8 7 6 5 4 3 3 3 3 3 4 5 6 7 8 9 0 1
1 0 9 8 7 6 5 4 4 4 4 4 4 4 5 6 7 8 9 0 1
1 0 9 8 7 6 5 5 5 5 5 5 5 5 5 6 7 8 9 0 1
1 0 9 8 7 6 6 6 6 6 6 6 6 6 6 7 8 9 0 1
1 0 9 8 7 7 7 7 7 7 7 7 7 7 7 7 8 9 0 1
1 0 9 8 8 8 8 8 8 8 8 8 8 8 8 8 8 9 0 1
1 0 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Qualora la funzione riceva un valore di n non positivo, non invia nulla in output.

Esercizio 5 (8 Punti)

Creare il file `utf8.h` e `utf8.c` che consentano di utilizzare la seguente funzione:

```c
extern size_t utf8_encode(uint32_t codepoint, uint8_t seq[4]);
```

La funzione riceve un `codepoint` Unicode (un valore a 32 bit compreso tra 0 e 10FFFF) e lo converte in una sequenza da 1 a 4 byte secondo lo standard UTF-8, inserendo ogni byte nello spazio puntato da `seq`. La funzione ritorna il numero di byte prodotti in output, o `0` se il codice è maggiore di 10FFFF.

La conversione avviene secondo la tabella seguente:

Numero di byte	Primo codepoint	Ultimo codepoint	Byte 1	Byte 2	Byte 3	Byte 4
1	U+0000	U+007F	0xxxxxxx			
2	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

I caratteri indicati con x vengono sostituiti dai bit del codice, inserendo nel primo byte i bit dal più significativo al meno significativo, continuando poi nei byte successivi, sempre dal più significativo al meno significativo.

Consideriamo la codifica del segno dell'Euro, €:

- Il codice Unicode per "€" è U+20AC.
- Siccome questo codice si trova tra U+0800 e U+FFFF, serviranno tre byte per la codifica.
- Il valore esadecimale 20AC è 0010.0000.1010.1100 in binario. I due bit a zero a sinistra vengono aggiunti perché una codifica a tre byte ha bisogno di 16 bit del codice.
- Dalla tabella vediamo che un codice a tre byte comincia con 1110...
- I quattro bit più significativi del codice vengono inseriti al posto delle quattro x del primo byte (1110.0010), lasciando 12 bits da codificare (0000.1010.1100).
- Tutti i bit di continuazione possono contenere esattamente 6 bit. Quindi i sei bit successivi vengono messi nel secondo byte e 10 viene memorizzato nei due bit più significativi (ovvero 1000.0010).
- Infine gli ultimi 6 bit del codice vengono messi nel terzo byte e di nuovo 10 nei due bit più significativi (1010.1100).

I tre byte così ottenuti 1110.0010 1000.0010 1010.1100 possono essere scritti in esadecimale come E2 82 AC.

Chiamando quindi la funzione `utf8_encode()` con `codepoint` pari a 20AC, troveremo E2 in `seq[0]`, 82 in `seq[1]`, AC in `seq[2]` e la funzione ritorna 3. In questo caso `seq[4]` non viene utilizzato dalla funzione.