Esame di Laboratorio del 12/01/2022

Esercizio 1 (5 Punti)

Scrivere un programma a linea di comando (è necessario scrivere il `main`) con la seguente sintassi:

```
head <filename>
```

Il programma prende in input da linea di comando un parametro di tipo stringa `filename`, e deve scrivere su `stdout` le prime 10 righe del file. La prima riga è composta da tutti i caratteri fino al primo a capo, la seconda dal carattere successivo alla prima riga fino al primo a capo, e così via. Se il file ha meno di 10 righe (termina prima di aver incontrato 10 a capo) si deve stampare un a capo aggiuntivo dopo l'ultimo carattere per terminare l'ultima riga.

Se il numero dei parametri passati al programma non è corretto o se non è possibile aprire il file, il programma termina con codice `1`, senza scrivere nulla su `stdout`, altrimenti termina con codice di uscita `0` dopo aver completato la scrittura.

Esercizio 2 (6 punti)

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

```
extern bool primigemelli(uint32_t start, uint32_t* x, uint32_t* y);
```

La funzione `primigemelli()` accetta come parametri un intero senza segno a 32 bit `start` e deve mettere nelle variabili puntate da `x` e `y` la prossima coppia di numeri primi gemelli maggiori o uguali a `start`. Se è possibile trovare una coppia di primi gemelli rappresentabile con `uint32_t` la funzione ritorna `true`, altrimenti ritorna `false` e non modifica le variabili puntate da `x` e `y`.

In matematica, si definiscono numeri primi gemelli due numeri primi che differiscono tra loro di due. Fatta eccezione per la coppia (2,3), questa è la più piccola differenza possibile fra due primi. Alcuni esempi di coppie di primi gemelli sono 3 e 5, 5 e 7, 11 e 13. Ricordo che un numero primo è un numero intero positivo che abbia esattamente due divisori distinti.

Ad esempio chiamando la funzione con `start=8`, questa ritornerà `true`, nella variabile puntata da `x` metterà 11 e nella variabile puntata da `y` metterà 13. Chiamando invece la funzione con `start=4294967295`, questa ritornerà `false`, dato che non è possibile rappresentare con 32 bit un numero maggiore di questo, primo o non primo.

Esercizio 3 (7 punti)

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

```
extern void paridispari(int *v, size_t n);
```

La funzione `paridispari()` accetta come parametri un puntatore a un vettore di `int` contenente n elementi.

La funzione deve riordinare il vettore fornito in modo che tutti i valori pari siano prima di quelli dispari. L'ordine in cui i valori compaiono non è importante, purché tutti i valori pari siano prima di quelli dispari.

Se `p=NULL`, oppure se `n=0`, la funzione non fa nulla.

Ad esempio, dato in input il vettore:

```
v = [ 1, 9, 8, 7, 2, 3, 5, 4, 6 ]
```

eseguendo `paridispari(v, 9)`, il vettore potrebbe diventare indifferentemene:

```
[ 4, 6, 8, 2, 7, 1, 5, 9, 3 ]
pari-------|  |-------dispari
```

oppure

```
[ 2, 4, 6, 8, 9, 7, 5, 3, 1 ]
pari-------|  |-------dispari
```

o qualsiasi altra combinazione dei numeri pari e dei numeri dispari. Questa funzione non invia nulla su `stdout`.

Esercizio 4 (6 punti)

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

```
extern void quadrati_progressivi(FILE *f, int n);
```

La funzione `quadrati_progressivi()` riceve in input un puntatore a `FILE` aperto in scrittura in modalità tradotta (testo) e un valore intero n, e scrive sul file un quadrato come da esempio, con `n=4`:

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

Per `n=4`, la funzione scrive su standard output un quadrato con un singolo `1` in alto a sinistra, circondato a destra e in basso dal valore`2`, circondato a destra e in basso dal valore`3`, circondato a destra e in basso dal valore `4`. Nelle righe i valori adiacenti sono separati da uno spazio.

La funzione scrive quindi progressivamente contorni quadrati di numeri crescenti fino a stampare il contorno esterno composto dal valore di `n`. Per numeri maggiori di `9`, la funzione stampa solo la cifra delle unità.

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

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

Scotland Yard è un gioco da tavolo cooperativo distribuito da Ravensburger, la cui prima edizione risale al 1983. Uno dei giocatori impersona il fuggitivo, "Mister X", mentre gli altri, nella parte dei poliziotti, tentano di acciuffare il malvivente attraverso le strade di Londra impiegando taxi, autobus e metropolitana.

La mappa del gioco originale è un grafo interamente connesso composto da 199 nodi connessi tra loro da 4 tipi di percorsi (percorsi gialli, verdi, rossi e neri).

Creare il file `map.h` e `map.c` che consentano di utilizzare la seguente struttura:

```
struct connections {
    size_t n;
    bool *data;
};
```

e la funzione:

```
extern struct connections *load_connections(const char *filename);
```

La struct consente di rappresentare una matrici di connessioni di dimensione qualsiasi, dove `n` è il numero di righe e colonne e `data` è un puntatore a `n×n` valori di tipo `bool` memorizzati per righe. La matrice contiene `1` in posizione `r,c` se il nodo numero `r+1` è connesso al nodo `c+1`, `0` altrimenti. Lo scarto di 1 è dovuto al fatto che i nodi sono numerati a partire da 1, mentre gli indici in C partono da 0.

Consideriamo ad esempio la mappa seguente:

```
1---2   3---4
|  / \     /
| /   \   /
|/     \ /
5---6   7---8
```

Questa corrisponde alla matrice

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

La matrice è sempre simmetrica e la diagonale principale è sempre nulla (un nodo non è mai connesso a se stesso). Questo corrisponderebbe ad una variabile `struct connections A`, con `A.n` = 8, e `A.data` che punta ad un'area di memoria contenente i valori `{ 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0 }`.

La funzione accetta come parametro il nome di un file di testo contenente una matrice di connessioni e deve restituire un puntatore a una nuova `struct connections` allocata dinamicamente. Nel file la prima riga contiene il numero di nodi della mappa, poi ogni riga successiva contiene (scritti come testo in ASCII in base 10 e separate da whitespace) le connessioni del nodo corrente con indice maggiore del suo, ovvero vengono memorizzati solo le posizioni degli 1 della parte triangolare superiore della matrice. La riga è terminata da un `.`.

Ad esempio, per la mappa precedente le connessioni verrebbero memorizzate come:

```
8
2 5 .
5 7 .
4 .
7 .
6 .
.
8 .
.
```

La funzione deve leggere il numero di nodi della matrice (chiamiamolo `n`), allocare una matrice di connessioni `n×n`, e per ogni connessione impostare un 1 nella posizione `r,c` e nella simmetrica `c,r`. Ad ogni `.` la riga è finita.

Notate che alla riga 7 del file, la riga è vuota, perché il nodo 6 non è connesso a nessun nodo di indice maggiore. La sua connessione al nodo 5 è riportata alla riga 5. L'ultima riga del file sarà sempre vuota, perché il nodo di indice massimo non può essere connesso a nodi di indice maggiore.

Se per qualsiasi motivo la funzione non riesce a leggere una matrice correttamente, ritorna `NULL`.