Esercitazione di Laboratorio: Esercizi di Ripasso

Esercizio 1

Scrivere un programma a linea di comando con la seguente sintassi:

```
occorrenze <s> <c>
```

I parametri sono nell'ordine una stringa, `s`, e un carattere, `c`. Il programma deve contare le occorrenze di `c` in `s` e stamparle su `stdout`.

Se il numero di parametri passati al programma non è corretto, questo deve stampare su `stderr` la stringa `Il numero di parametri non e' corretto. Sintassi del programma: "occorenze <s> <c>"` e terminare con codice di errore `1`. In tutti gli altri casi il programma termina con codice di uscita `0`.

Ad esempio `occorrenze mamma m` dovrebbe stampare il valore 3.

Esercizio 2

Scrivere un programma a linea di comando con la seguente sintassi:

```
mul <a> <b>
```

Dati due numeri interi passati come argomenti da linea di comando, `a` e `b`, il programma deve stampare su `stdout` il risultato del prodotto.

Se il numero di parametri passati al programma non è corretto, questo deve stampare su `stderr` la stringa `Il numero di parametri non e' corretto. Sintassi del programma: "mul <a> <b>"` e terminare con codice di errore `1`. In tutti gli altri casi il programma termina con codice di uscita `0`.

N.B.: per convertire una stringa in un numero è possibile utilizzare la funzione di libreria `int atoi(const char *str);` o `long int strtol (const char* str, char** endptr, int base);`.

Esercizio 3

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

```
extern void ToUpper(char *str);
```

Data una stringa C, la procedura deve convertire le lettere minuscole in maiuscola senza utilizzare funzioni o procedure. Si utilizzi unicamente l'aritmetica dei puntatori per la risoluzione dell'esercizio, ovvero non si acceda mai agli elementi della stringa con l'operatore `[]`.

Se `str` è `NULL`, la procedura non effettua nessuna operazione.

Esercizio 4

Creare i file `vettore.c` e `vettore.h` che consentano di utilizzare la `struct`:

```
struct vettore{
    int *data;
    size_t size;
};
```

e le seguenti funzioni:

```
extern void Push(struct vettore *v, int d);
extern int Pop(struct vettore *v);
```

La `struct` rappresenta un vettore di dati di tipo `int`. Nello specifico, `data` è l'indirizzo di memoria a partire dal quale i numeri sono memorizzati, mentre `size` è la dimensione di `data` in numero di elementi.

La procedura `Push()` prende in un input una `struct vettore` e un intero `d` e deve, riallocando opportunamente la memoria, aggiungere `d` in coda al vettore.

La funzione `Pop()` prende in input una `struct vettore` e ne rimuove il primo elemento, spostando tutti gli altri a sinistra e riducendo opportunamente la memoria allocata. Infine, la funzione ritorna l'elemento rimosso. La `Pop()` non sarà mai invocata su un vettore vuoto.

Esercizio 5

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

```
extern int ContaOccorrenze(const char *filename, const char *str);
```

La funzione riceve come parametro due stringhe C (puntatori a vettori di `char` zero terminati), `filename` e `str`. La stringa `filename` è il nome di un file di testo che deve essere aperto in lettura in modalità tradotta, mentre `str` contiene una parola, ovvero una sequenza di caratteri priva di *whitespace*, lunga al massimo 99 caratteri. La funzione `ContaOccorrenze()` ritorna il numero di occorrenze di `str` all'interno del file `filename`. Se `filename` non esiste, o se non è possibile aprirlo, la funzione ritorna `0`.

Se `str` è `NULL` la funzione ritorna `0`.

`filename` non contiene mai parole più lunghe di 99 caratteri.