Scheda di lavoro N. 6

Questa è la *scheda dei caratteri.* Si tratta di scrivere alcune funzioni utili per la gestione delle stringhe di caratteri, corredate da programmi di prova. Le funzioni sono molto semplici; proprio per questo la scheda non contiene commenti e/o suggerimenti molto dettagliati.

Per tutti gli esercizi si applica uno schema comune. Il programma di prova dovrà:

(i) leggere una stringa di caratteri dalla tastiera;
(ii) chiamare una funzione che esegua un'operazione sulla stringa;
(iii) riscrivere su terminale il risultato.

In pratica, un solo programma potrà servire per tutti gli esempi, con poche modifiche.

La lettura da tastiera di una stringa di caratteri, oppure la scrittura della stringa sul terminale, si effettuano con le istruzioni

```
char str[100];
...
printf("Dimmi qualcosa:  ");
fgets(str,80,stdin);
printf("%s",str);
...
```

La prima linea alloca un vettore di 100 caratteri destinato a contenere la stringa. C'è posto per 99 caratteri oltre al `NUL` finale. La lettura viene effettuata dalla funzione

`fgets(str,100,stdin)`

`str` è il nome del vettore (che identifica l'indirizzo iniziale); 100 è la lunghezza massima della stringa, incluso il `NUL` finale; `stdin` identifica la tastiera (non puoi usare un nome diverso). Per la scrittura, il formato `%s` indica che il dato è una stringa di caratteri.

Per la lettura di un singolo carattere potrai usare le istruzioni

```
char a;
...
scanf("%c",&a);
```

Il formato `%c` specifica che il dato da leggere è un singolo carattere.

Ultima informazione: il carattere `NUL` ha il valore binario `0` (zero). Per mettere in evidenza che si tratta di un carattere sarà utile indicarlo con `'\000'` (gli apici fanno parte della sintassi).

1. Scrivi una funzione

`int lunghezza_stringa(char str[])`

che restituisca come valore intero la lunghezza della stringa `str` che ha ricevuto come argomento.

Il problema è solo come riconoscere la fine della stringa. Ricorda che per convenzione tutte le stringhe di caratteri in C sono terminate da un `NUL`. Ti basta quindi far scorrere il vettore partendo dal primo carattere, fin che trovi un `NUL`.

La differenza rispetto ai cicli che hai programmato fin qui è che non sai quale sia il valore massimo raggiungibile dall'indice corrente, perché non hai indicazioni a priori sulla lunghezza della stringa. Il test quindi deve coinvolgere il contenuto della stringa, e non l'indice. Quello che a prima vista ti sembrerà sorprendente è la semplicità dell'istruzione:

```
for(j=0; str[j] != '\000'; j++);
```

Nota che il ciclo non contiene istruzioni: è perfettamente lecito. Il contatore `j` viene incrementato continuamente, fin che non si arriva al `NUL` finale. La lunghezza della stringa coincide con l'indice che dà la posizione del carattere `NUL`. Quindi la funzione deve restituire il valore di `j` all'uscita dal ciclo.

A te completare il sorgente della funzione. Naturalmente, qui come per il seguito, dovrai scrivere anche un programma di prova per verificare che la funzione svolga correttamente il suo compito. Trovi un esempio nei files `funzioni_stringa.c` e `prova_stringhe.c`.

2. Scrivi una funzione

```
int cerca_carattere(char str[],char cc)
```

che restituisca la posizione del carattere `cc` dentro la stringa `str`. Se il carattere non c'è la funzione dovrà restituire la lunghezza della stringa.

La funzione è solo una piccola modifica di quella dell'esercizio 1. Anche in questo caso, prova a scriverla in forma compatta. Poi confronta la tua soluzione con quella che trovi nel file `funzioni_stringa.c`.

3. Scrivi una funzione

```
int copia_stringa(char str_in[],char str_out[])
```

che copia il contenuto della stringa `str_in` in `str_out`, incluso il `NUL` finale. La funzione deve restituire il numero di caratteri copiati.

I consigli sono gli stessi che ti ho dato per gli esercizi 1 e 2. Valgono anche per il seguito.

4. Scrivi una funzione

```
elimina_spazi(char str[])
```

che elimina dalla stringa `str` tutti i caratteri spazio (`'\040'`), ritorno a capo (`'\012'`) e `TAB` (`'\011'`). La lunghezza della stringa potrà diminuire, ovviamente. Non dimenticare di spostare anche il `NUL` finale. La funzione deve restituire il numero di caratteri rimasti nella stringa.

La parte interessante – che ti potrà creare qualche dubbio – sta nel fare questa operazione senza duplicare la stringa: la stringa in ingresso deve essere restituita modificata.

5. Scrivi due funzioni

```
int conv_maiuscolo(char str[])
int conv_minuscolo(char str[])
```

che convertono tutti i caratteri alfabetici della stringa `str` in maiuscolo o, rispettivamente, in minuscolo. Le funzioni devono restituire il numero di caratteri convertiti.

6. Scrivi una funzione

```
int inverti_stringa(char str[])
```

che inverte l'ordine dei caratteri nella stringa `str`, in modo che venga scritta a rovescio. Attento a non spostare ed a non dimenticare il `NUL` finale: deve restare sempre all'ultimo posto.

Questo esercizio è più difficile dei precedenti: dovrai immaginare un buon algoritmo, dal momento che l'inversione deve avvenire "sul posto", cioè senza copiare il contenuto della stringa in un'altra.

Suggerimento: dovrai scambiare tra loro il primo e l'ultimo carattere, poi il secondo ed il penultimo, il terzo ed il terzultimo, &c; ma attento a fermarti al momento giusto! Inoltre, tieni presente che il `NUL` finale deve restare al suo posto. Il tutto dovrà essere realizzato con un ciclo.

7. Scrivi una funzione

```
int confronta_stringhe(char str_1[],char str_2[])
```

che confronti le due stringhe in ingresso secondo l'ordine lessicografico. La funzione deve restituire `-1` se la prima stringa precede la seconda, `1` se la prima stringa segue la seconda, e `0` se le stringhe sono identiche.

Per ordine lessicografico si intende questo: stabilito un ordinamento per i singoli caratteri, si confrontano a coppie, in successione, i caratteri che occupano la stessa posizione entro le due stringhe; non appena si trova una coppia di caratteri diversi si dice che la stringa s_1 precede la stringa s_2 se il carattere della stringa s_1 precede quello della stringa s_2; se tutte le coppie che si possono costruire senza superare la lunghezza delle stringhe sono composte da caratteri identici, la stringa più corta precede l'altra; le stringhe sono uguali se hanno la stessa lunghezza e contengono, in ordine, gli stessi caratteri.

Se rifletti un momento sulle regole che ti ho enunciato, non si tratta di altro che dell'ordinamento alfabetico, almeno fin che ti limiti a considerare stringhe composte di soli caratteri alfabetici.

Nel nostro caso possiamo ammettere qualunque carattere del codice ASCII, ed assumere sui caratteri proprio l'ordinamento del codice ASCII. Questo semplifica sensibilmente la programmazione. Eccoti l'algoritmo, che potrai realizzare mediante un ciclo `for`

- (i) esegui un ciclo con un indice `j` che parte da zero e termina quando il `j`–esimo carattere di una qualunque delle due stringhe è un `NUL`;
  - (i.a) confronta il `j`–esimo carattere della prima stringa con il `j`–esimo carattere della seconda stringa;
  - (i.b) se il codice ASCII del carattere della prima stringa precede quello della seconda restituisci `-1`;
  - (i.c) se il codice ASCII del carattere della prima stringa segue quello della seconda restituisci `1`;
  - (i.d) incrementa `j` e torna a ripetere il ciclo.
- (ii) (a ciclo concluso) se il `j`–esimo carattere della prima stringa non è `NUL` restitusci `1`;
- (iii) altrimenti, se il `j`–esimo carattere della seconda stringa non è `NUL` restitusci `-1`;
- (iv) altrimenti restituisci `0`.