Laboratorio di Algoritmi e Strutture Dati

Docente: Camillo Fiorentini

27 novembre 2007

In C una *stringa* è rappresentata da una sequenza di caratteri terminante con il carattere nullo `'\0'` il cui valore è 0. Una stringa va scritta in un array di `char` di dimensione adeguata (ad esempio, la stringa `"cane"` richiede un array di `char` di *almeno* 5 elementi).

Con l'istruzione

```
scanf("%s", w);
```

viene letta da standard input una stringa e scritta nell'array `w`. Si ricordi che vengono saltati tutti i caratteri di spaziatura che precedono la stringa e che la lettura della stringa termina quando si incontra un carattere di spaziatura oppure "end-of-file" (quindi non è possibile leggere stringhe contenenti caratteri di spaziatura). È compito di chi chiama `scanf()` assicurarsi che `w` sia l'indirizzo di un array di lunghezza sufficiente per contenere la stringa letta (oppure, dichiarare nella documentazione del programma qual è la massima lunghezza di una stringa per cui il programma funziona correttamente).

Usare compilatore `gcc`. Si consiglia di compilare con opzione `-Wall` che segnale (come warning) le anomalie presenti nel programma rilevate dal compilatore.

Esercizio 1: funzioni su stringhe

Si assume che le stringhe abbiano lunghezza massima `MAX`, dove `MAX` è una macro definita nel programma. Per ogni punto, scrivere una funzione `main()` che permetta di verificare la correttezza della funzione scritta. Per l'Esercizo 2 occorre svolgere i punti a, b, c e d.

a. Scrivere il codice di una funzione

```
int strLength(char *w)
```

che restituisce il numero di caratteri della parola `w` (senza considerare il carattere di fine stringa `'\0'` usato nella rappresentazione C). Ad esempio, se `w` è la stringa `"cane"` la funzione deve restituire 4, se `w` è la stringa nulla `""` la funzione deve restituire 0.

b. Scrivere il codice di una funzione

```
void strCopy(char *t, char *s)
```

che copia una stringa da `s` (source) a `t` (target) (incluso il carattere di fine stringa `'\0'`). Si assume che `t` sia l'indirizzo di un array di `char` sufficientemente grande per poter contenere la stringa da copiare.

c. Scrivere il codice di una funzione

```
void addChar(char *word, char *source, char c)
```

che pone in `word` la stringa ottenuta aggiungendo alla stringa `source` il carattere `c`. Ad esempio, se `source` è la stringa `"cane"` e `c` il carattere `'a'`, `word` deve essere la stringa `"canea"`. Si assume che `word` sia l'indirizzo a un array di `char` sufficientemente grande per poter contenere la stringa risultante.
Usare le funzioni `strLength()` e `strCopy()`.

d. Scrivere il codice di una funzione

```
void deleteChar(char *word, char *source, int k)
```

che pone in `word` la stringa ottenuta togliendo dalla stringa `source` il carattere `source[k]`. Ad esempio, se `source` è la stringa `"cane"` e `k` il vale 1, `word` deve essere la stringa `"cne"`. Si assume che $0 \leq k < strLength(source)$ e che `word` sia l'indirizzo di un array di `char` sufficientemente grande per poter contenere la stringa risultante. Usare la funzione `strCopy()`.

e. Scrivere il codice di una funzione

```
int strUguali(char *w1, char *w2)
```

che restituisce 1 se le stringhe `w1` e `w2` sono uguali, 0 altrimenti (occorre confrontare le stringhe carattere per carattere, *non* i valori di `w1` e `w2`).

Provare la funzione con le seguenti coppie di stringhe:

```
cane canzone
cane cane
cane canea
gattone gatto
```

f. Scrivere il codice di una funzione ricorsiva

```
void strInv(char *t, char *s)
```

che pone in `t` la stringa ottenuta invertendo la stringa `s`. Ad esempio, se `s` è la stringa `"cane"`, `t` deve essere la stringa `"enac"`. Si assume che `t` sia l'indirizzo di un array di `char` sufficientemente grande per poter contenere la stringa risultante.

Occorre distinguere tra il caso in cui `s` è la stringa nulla e il caso in cui `s` contiene almeno un carattere (richiede una chiamata ricorsiva). Usare la funzione `strLength()`.

Esercizio 2: generazione di anagrammi

L'obiettivo dell'esercizio è quello di scrivere una funzione di intestazione

```
void anagrammi(char *word)
```

che stampa tutti gli anagrammi della parola `word`. Si assume che la parola `word` abbia lunghezza al massimo `MAX`, dove `MAX` è una macro definita nel programma, e che `word` contenga solo lettere minuscole. Se gli anagrammi sono tanti, conviene redirigere lo standard output in un file `out.txt`:

```
a > out.txt
```

La soluzione più naturale è di usare una funzione ricorsiva, infatti:

- Gli anagrammi di una parola *word* sono ottenibili concatenando a ogni carattere c di *word* gli anagrammi della parola ottenuta togliendo c da *word*.

Per evitare ripetizioni, ogni carattere c di *word* va considerato una sola volta.

Esempio
Gli anagrammi della parola `cava` hanno la forma:

(1). `c` più gli anagrammi della parola `ava`

`cava` `caav` `cvaa`

(2). `a` più gli anagrammi della parola `cva`

`acva` `acav` `avca` `avac` `aacv` `aavc`

(3). `v` più gli anagrammi della parola `caa`

`vcaa` `vaca` `vaac`

Notare che le parole in (1) hanno la forma

(1.1). `ca` più gli anagrammi della parola `va`

`cava` `caav`

(1.2). `cv` più gli anagrammi della parola `aa`

`cvaa`

Questo suggerisce che per impostare correttamente il ragionamento induttivo occorre definire una funzione Anagr che stampa tutte le parole ottenute concatenando una parola $pref$ con gli anagrammi della parola *word*. Ad esempio, se $pref$ è la parola `ca` e *word* è la parola `va`, la funzione deve stampare:

```
cava
caav
```

Ad alto livello la definizione di ANAGR è:

ANAGR (*pref*, *word*)
1 **if** *word* e' la stringa nulla
2 **then** stampa *pref*
3 **else**
4 **for each** carattere c in *word*
5 **do if** il carattere c non e' ancora stato usato
6 **then** dichiara che c e' gia' stato usato
7 $newPref \leftarrow$ stringa ottenuta aggiungendo a $pref$ il carattere c
8 $newWord \leftarrow$ stringa ottenuta eliminando da *word* il carattere c
9 ANAGR $(newPref, newWord)$

La correttezza e terminazione della funzione si dimostra per induzione sulla lunghezza di *word* (si noti che $newWord$ ha un carattere in meno di *word*, quindi *per ipotesi di induzione* la chiamata ricorsiva svolge correttamente il suo compito).

a. Scrivere il codice della funzione

```
/* Stampa su linee distinte tutte  le parole ottenute concatenando la stringa pref
   con gli anagrammi della stringa word */

 void anagr(char *pref, char *word)
```

che traduce l'algoritmo di alto livello sopra riportato. Per tradurre il ciclo alla linea 4 occorre considerare come in C sono rappresentate le stringhe (è necessario determinare la lunghezza di `word` usando `strLength()`). Nelle linee 7 e 8 usare le funzioni `addChar()` e `deleteChar()`.

Per stabilire se un carattere è già stato usato si può dichiarare un array

```
 char used[N];
```

dove `N` è il numero delle lettere dell'alfateto, tale che, per $k \geq 0$, `used[k]` vale 1 se la k-esima lettera dell'alfabeto è stata usata, `used[k]` vale 0 altrimenti (si ricordi che per ipotesi le parole contengono solo lettere minuscole). Ad esempio, se si ha

```
used[0] == 0        used[1] == 1       used[2] == 1         used[3] == 0     ....
```

significa che il carattere `'a'` non è ancora stato usato, i caratteri `'b'` e `'c'` sono stati usati, `'d'` non è stato usato. Ricordarsi di inizializzare l'array nel modo corretto.

b. Scrivere il codice della funzione

```
void anagrammi(char *word)
```

che stampa tutti gli anagrammi di `word` (è sufficiente una chiamata ad `anagr()` con i parametri giusti).

c. Supponiamo ora di voler stampare gli anagrammi numerandoli. Ad esempio:

```
1 cava
2 caav
3 cvaa
 ....
12 vaac
```

Occorre modificare la funzione `anagr()` nel seguente modo:

- La funzione deve avere un parametro `n` che rappresenta il numero degli anagrammi che sono già stati stampati.
- La funzione deve restituire il numero complessivo degli anagrammi stampati.

Il tipo di `n` e il tipo della funzione deve essere un tipo che permette di rappresentare numeri interi positivi grandi. Conviene usare il tipo `unsigned long int` e porre all'inizio del programma la definizione

```
typedef unsigned long int    bigInt;
```

che definisce un nuovo tipo `bigInt` che è sinonimo di `unsigned long int`. Per stampare una variabile di tipo `unsigned long int` con `printf()` occorre usare il carattere di conversione `%lu`.

Per provare il programma, occorre modificare opportunamente la funzione `anagrammi()`. Fare in modo che restituisca il numero di anagrammi trovati.

d. Quanti sono gli anagrammi di una parola contenente n caratteri distinti?
Quanti sono gli anagrammi di una parola in cui possono esserci ripetizione di caratteri?
Qual è lo spazio di memoria richiesto dalla funzione `anagrammi()`?

e. Provare a risolvere l'esercizio senza usare la ricorsione.