Laboratorio di programmazione

mercoledì 28 ottobre 2009

1 Esercizi introduttivi

1.1 Trasformazione orario in secondi

Scrivete una funzione con prototipo `void split_time ( long int tot_sec, int *h, int *m, int *s )` che, dato un orario fornito in numero di secondi dalla mezzanotte, calcoli l'orario equivalente in ore, minuti, secondi, e lo memorizzi nelle tre variabili puntate da (`h`), (`m`) e (`s`) rispettivamente.

1.2 I due valori più grandi

Scrivete una funzione con prototipo `void max_secondmax ( int a[], int n, int *max, int *second_max )` che, dato un array `a` di lunghezza `n` individui la posizione dell'elemento più grande e del secondo elemento per grandezza.

1.3 Puntatore al minimo

Scrivete una funzione con prototipo `int *smallest( int a[], int n )` che, dato un array `a` di lunghezza `n`, restituisca un puntatore all'elemento più piccolo dell'array.

1.4 Scambio di valori

Scrivete una funzione con prototipo `void scambia( int *p, int *q )` che scambi i valori delle due variabili puntate da `p` e `q`.

1.5 Da minuscolo a maiuscolo

Scrivete una funzione con prototipo `char *maiuscolo( char *stringa )` che trasformi da minuscolo in maiuscolo tutte le lettere del suo argomento `stringa` e ne restituisca un puntatore al primo carattere. Potete assumere che `stringa` sia dato da una stringa terminata da `'\0'` contenente caratteri ASCII (non solo lettere). Potete usare la funzione `toupper` della libreria `ctype.h`.

1.6 Lunghezza di una stringa

Scrivete una funzione con prototipo `int lung_stringa( char *s )` che, data una stringa `s`, ne calcoli la lunghezza. Provate a scrivere il programma usando un puntatore a carattere per scorrere la stringa.

2 Esercizi da svolgere in laboratorio

2.1 Indice della parola più piccola

Scrivete una funzione con prototipo `int smallest_word_index( char *s[], int n )` che, dato un array `s` lungo `n` di stringhe, restituisca l'indice della parola più piccola (secondo l'ordine alfabetico) contenuta nell'array. Per effettuare confronti tra stringhe, potete usare la funzione `strcmp` dal file di intestazione `string.h`.

2.2 Alfabeto farfallino

Quando la vostra docente di laboratorio di algoritmi era bambina, usava a volte, per comunicare con le sue amiche, uno speciale alfabeto, detto *alfabeto farfallino*. L'alfabeto farfallino consiste nel sostituire, a ciascuna vocale, una sequenza di tre lettere della forma vocale-f-vocale. Per esempio, alla lettera *a* viene sostituita la sequenza *afa*, alla lettera *e* la sequenza *efe* e così via.

Dovete scrivere un programma, di nome `farf` che, ricevendo come argomento (sulla riga di comando) una parola, ne stampi la traduzione in alfabeto farfallino. Potete assumere che la stringa in input non contenga lettere maiuscole.

Provate a modificare il programma in modo che accetti più parole sulla riga di comando.

Esempio di funzionamento

```
$./farf mamma
mafammafa
$./farf aiuola
afaifiufuofolafa
$./farf farfalla
fafarfafallafa
```

2.3 Rettangoli

Modificate il programma dell'esercizio "Figure geometriche" (lab 14 ottobre 2009) scrivendo delle funzioni che svolgano le seguenti operazioni:

- dati per argomenti due punti, creare e restituire una nuova struttura rettangolo;
- stampare i dati del rettangolo passato come argomento;
- calcolare l'area del rettangolo passato come argomento;
- calcolare il centro (l'intersezione delle diagonali) del rettangolo passato come argomento;
- traslare il rettangolo passato come argomento di `x` unità nella direzione x e `y` unità nella direzione y;
- stabilire se un punto `p` cade dentro il rettangolo passato come argomento oppure no, restituendo `VERO` o `FALSO`.

Per evitare che ad ogni chiamata venga copiata tutta la struttura, è utile passare come argomento un puntatore al rettangolo. In questo caso, sarà utile l'operatore `->`.

3 Altri esercizi

3.1 La strana sillabazione

Il professor Precisini, sostenendo che le regole di sillabazione della lingua italiana sono troppo complesse e piene di eccezioni, propone un nuovo e originale metodo di sillabazione. Il metodo consiste in questo: una sillaba è una sequenza

massimale di caratteri consecutivi che rispettano l'ordine alfabetico. Per esempio, la parola *ambire* viene sillabata come *am-bir-e*: infatti la lettera *a* precede la lettera *m*, e le lettere *b*, *i* e *r* rispettano anch'esse l'ordine. Analogamente, la parola *sotterfugio* viene sillabata come *s-ott-er-fu-gio*.

Dovete scrivere un programma, di nome `sillaba` che, ricevendo come argomento (sulla riga di comando) una parola, la sillabi. Potete assumere che la stringa in input sia costituita solo da lettere minuscole.

Esempio di funzionamento

```
$./sillaba amore
amor-e
$./sillaba scafroglia
s-c-afr-o-gl-i-a
```

3.2 Palindrome (con argomenti da linea di comando)

Scrivete una funzione che stabilisca se il suo argomento è una parola palindroma oppure no, usando due puntatori per scorrere la parola partendo dall'inizio e dalla fine. Quindi scrivete un programma che stabilisca, per ciascun argomento fornito da linea di comando, se si tratta di una parola palindroma oppure no.

3.3 La parola minima e la parola massima

Scrivete una funzione con prototipo

```
void smallest_largest( char *s[], int n, char **smallest, char **largest )
```

che, dato un array `s` lungo `n` di stringhe, trovi gli elementi minimo e massimo nell'array (secondo l'ordine alfabetico). Per effettuare confronti tra stringhe, potete usare la funzione `strcmp` dal file di intestazione `string.h`.

Inizializzare un array frastagliato da standard input può essere *doloroso*; consiglio quindi di testare la vostra funzione `smallest_largest` usando un `main` così strutturato:

```
int main( void ) {
  char *dict[] = { "ciao", "mondo", "come", "funziona", "bene", "questo", "programma" };
  int lun = 7;
  ... min;
  ... max;

  ...

  smallest_largest( dict, lun, &min, &max );
  printf( "La parola minima e' %s e la massima e' %s.\n", min, max );
  return 0;
}
```

Modificate le inizializzazioni di `dict` e `lun` in modo da testare la funzione con altri argomenti.

In alternativa, potete leggere da linea di comando una serie di parole e testare la funzione `smallest_largest` passando come argomenti `argv` e `argc` (o meglio, delle espressioni che coinvolgono `argv` e `argc`: ricordate che `argv[0]` contiene il nome del programma, che non va passato alla funzione `smallest_lasgest`!).