Laboratorio di Fondamenti di Informatica 1/12/2015

Anno accademico 2015/2016

1. Mettiamo i puntini

Si scriva una funzione **ricorsiva** che, ricevuto in ingresso un numero intero, lo stampi a schermo aggiungendo il separatore delle migliaia.

Ad esempio:

1234	=>	**1.234**
12021	=>	**12.021**
37	=>	**37**
12376428	=>	**12.376.428**
1376428	=>	**1.376.428**
12006428	=>	**12.006.428**
310000008	=>	**310.000.008**

Suggerimenti:

- Il caso base si ha quando il numero è minore di 1000
- Si ricordi, inoltre, che l'operatore % calcola il resto di una divisione tra interi, e che "modulando" per 1000 si possono estrarre le ultime tre cifre di un qualsiasi numero maggiore di 1000

2. strcmp(strcmp(strcmp(strcmp(…))))

Si scriva una funzione **ricorsiva** che confronta due stringhe e, così come la funzione di libreria strcmp()), le confronta:

int strecursivecmp(**char** * s1, **char** * s2);

restituendo **0** : se le due stringhe sono uguali
-1 : se s1 precede alfabeticamente s2
1 : se s1 segue alfabeticamente s2

strecursivecmp("zucca", "zucca") → **0**
strecursivecmp("zucca", "cenerentola") → **1**
strecursivecmp("cenerentola", "zucca") → **-1**
strecursivecmp("**Z**ucca", "**c**enerentola") → **-1**
strecursivecmp("cenere", " cenerentola") → **-1**
strecursivecmp("cenerentola", " cenere") → **1**

Suggerimenti:

- Il caso base si ha quando …. ? ☺
- il passo induttivo consiste nel verificare separatamente il primo carattere di s1 ed s2 e "rimandare" la verifica di uguaglianza sulla stringhe che iniziano dal secondo carattere di s1 ed s2.

3. Riscende, risale, …

Si codifichino **in modo ricorsivo** le seguenti funzioni

```
void effe1(int n), void effe2(int n), …, void effe5(int n)
```

già codificate iterativamente, (qui esemplificate per n=3) :

$F_0(3)$ = 333221.122333
$F_1(3)$ = 111223.322111
$F_2(3)$ = 33322**1**22333
$F_3(3)$ = 11122**3**22111
$F_4(3)$ = 122**333**221
$F_5(3)$ = 322**111**223

In particolare, si raccomanda di «cimentarsi» con F_0, F_3 ed F_5

4. Aritmetica ricorsiva (1/3)

Supponiamo, limitatamente alla soluzione di questo esercizio, che il linguaggio C NON possegga né i costrutti per esprimere iterazione (while, for, do, goto, …) né gli operatori aritmetici `+` , `*` , `-` e `/` (in buona sostanza il loro uso qui "non è permesso" in alcun modo).

Si considerino **invece** le seguenti funzioni (precedente e successivo di un numero intero), che, appunto, non usano gli operatori succitati:

```
int pre(int x) { return --x; }
int suc(int x) { return ++x; }
```

Si considerino poi le seguenti (informali e incomplete) bozze di **formulazione ricorsiva** delle funzioni di ***addizione***, ***sottrazione***, ***moltiplicazione***, ***divisione***, ***sommatoria*** e ***produttoria***, applicate a numeri *interi* ***non negativi***:

4. Aritmetica ricorsiva (2/3)

(1) $a + b = pre(a) + suc(b)$ $\quad a, b \in N + \text{‘0’}$

(2) $a - b = pre(a - pre(b))$ $\quad a, b \in N + \text{‘0’}, \mathbf{b \leq a}$

(3) $a * b = a + (a * pre(b))$ $\quad a, b \in N + \text{‘0’}$

(4) $a / b = suc((a - b) / b)$ $\quad (\text{se } a \geq b)\ a, b \in N + \text{‘0’}$

(5) $\sum(n) = \sum_{i=1\ldots n} i = n + \sum_{i=1\ldots pre(n)} i$ $\quad n \in N, \text{con } \sum(0) = 0$

(6) $\prod(n) = \prod_{i=1\ldots n} i = n * \prod_{i=1\ldots pre(n)} i$ $\quad n \in N, \text{con } \prod(0) = 0 \text{ (non 1!)}$

4. Aritmetica ricorsiva (3/3)

Si implementino tramite funzioni ***RICORSIVE*** le funzioni relative alle suddette operazioni, espresse in forma induttiva (add() sott() molt() div() sommatoria() produttoria()) ***nel campo dei numeri interi non negativi***, **identificando opportunamente i casi base e prestando attenzione alla terminazione**.

ATTENZIONE: le definizioni proposte e le soluzioni conseguenti sono particolarmente inefficienti – ma non è l'ottimizzazione lo spirito dell'esercizio!

Estensione: se ne scrivano versioni **adatte a ricevere come parametri interi *qualsiasi* (cioè anche negativi)**, eventualmente riformulando i problemi e/o le "definizioni ricorsive".

Suggerimento: Prestate particolare attenzione alla terminazione.

5. Vettore Ciclico (reloaded)

Si codifichi una funzione **ricorsiva**

```
int ciclico( int v[], int d )
```

che stabilisce (restituiendo 0 o 1) se il vettore di interi v di dimensione d è ciclico.

Come ci si muove nel vettore: il valore contenuto in ogni elemento, se compreso tra 0 e N-1, rappresenta l'indice del prossimo elemento da visitare; se invece è esterno a tale intervallo rappresenta un riferimento "errato", che indica una posizione esterna al vettore, e comporta la fine del processo di esplorazione.

Nell'esempio $\mathbf{v_1}$ è ciclico, $\mathbf{v_2}$ non lo è.