Informatica Generale
Homework 1: Naturalmente, naturali

Ivano Salvo – Sapienza Università di Roma – 27/3/2017

Esercizio 1 Scrivere una funzione C di prototipo:

```
int logIntero(int n, int b)
   /* PREC: n>0 & b>1 */
```

che presi in ingresso due parametri interi positivi n, b ($n > 0, b > 1$) calcola il logaritmo *intero* di n in base b. Più formalmente, la funzione deve restituire un numero intero k, per cui sia verificata la disuguaglianza $b^k \leq n < b^{k+1}$.

Esempi: Presi in input 63 e 2, il programma deve tornare 5, in quanto $2^5 = 32 \leq 63 < 64 = 2^6$. Presi in input 81 e 3, il programma deve stampare 4, in quanto $3^4 = 81 < 243 = 3^5$.

Esercizio 2 Dato un intero strettamente positivo n, definiamo il *successivo* di un naturale $n > 1$ come segue:

$$\begin{array}{rcl} n/2 & \text{se} & n \text{ è pari} \\ 3n+1 & \text{se} & n \text{ è dispari} > 1 \end{array}$$

1 non ha successivi.

Scrivere una funzione C di prototipo:

```
int treXPUno(int n, int *max)
  /* PREC: n>0 */
```

che calcola i successivi di un numero intero strettamente positivo n fino a che non viene raggiunto 1. La funzione dovrà tornare come risultato *il numero di passi* necessari affinché la sequenza che comincia con n raggiunga 1 e caricare la variabile *max* con il massimo numero incontrato durante la sequenza.

Esempio: La sequenza che comincia con 3, prosegue con 10, 5, 16, 8, 4, 2, 1: in questo caso la funzione dovrebbe restituire 7, e caricare 16 in *max*.

Gli arditi possono provare a dimostrare che la funzione termina sempre ☺.

Esercizio 3 Scrivere una funzione C di prototipo:

```
int kupleP(int n, int k)
  /* PREC: n,k>0 */
```

che presi in ingresso due parametri strettamente positivi n e k ($n, k > 0$) restituisca in output il numero delle k-uple *ordinate* $\langle p_1, \ldots, p_k \rangle$ di interi positivi ($1 \leq p_1 \leq p_2 \leq \ldots \leq p_k \leq n$) il cui prodotto è n.

ESEMPI: Per ogni k, se n è primo, il risultato deve sempre essere 1. Infatti, se n è primo, l'unica k-upla che dà come prodotto n è $\langle \underbrace{1, 1, 1, \ldots 1}_{k-1}, n \rangle$.

Anche se k è 1, il risultato è necessariamente 1. Infatti, in questo caso, l'unica k-upla che dà come prodotto n è $\langle n \rangle$.

Se n fosse 12 e k fosse 4, la funzione dovrebbe ritornare 4. Infatti ho che 12 può essere ottenuto come prodotto delle seguenti k-uple: $\langle 1, 1, 1, 12 \rangle$, $\langle 1, 1, 2, 6 \rangle$, $\langle 1, 1, 3, 4 \rangle$, $\langle 1, 2, 2, 3 \rangle$.

Note Pratiche

Dovete consegnare semplicemente la funzione richiesta, **rispettando il prototipo**. Il vostro file può contenere eventuali altre funzioni ausiliarie. Non dovete lasciare **nessuna** istruzione di input/output nel codice. Potete verificare se il vostro programma funziona, utilizzando i files `main.1.c`, `main.2.c`, e `main.3.c` forniti nella pagina degli Homework. I vostri file (contenenenti le funzioni richieste) dovranno chiamarsi `NomeCognome.#es.c`, ad esempio `LilyEvans.1.c`, `LilyEvans.2.c`, e `LilyEvans.3.c`, se il vostro nome fosse Lily Evans. Il comando utilizzato per la compilazione sarà:

```
gcc -std=c99 main.1.c LilyEvans.1.c
gcc -std=c99 main.2.c LilyEvans.2.c
gcc -std=c99 main.3.c LilyEvans.3.c
```

Osservate che i files dovrebbero poter essere compilati separatamente con l'opzione `-c` che traduce in linguaggio macchina, ma senza generare un eseguibile (`gcc -c -std=c99 LilyEvans.1.c` e `gcc -c -std=c99 main.1.c` dovrebbero generare con successo un file chiamato rispettivamente `LilyEvans.1.o` e `main.1.o`.