Informatica Generale
Homework P/2019: Primi e Composti

Ivano Salvo – Sapienza Università di Roma – 21/3/2018

Esercizio 1. Per ogni $n \geq 1$ il *primoriale* Π_k è il prodotto $p_1 p_2 \dots p_k$ dei primi k numeri primi (conveniamo $p_1 = 2$). La successione dei primoriali è 2, 6, 30, 210, 2310, 30030, 510510,.... Scrivere una funzione C di prototipo:

```
int primoriale(int k)
   /* PREC: k>=0, POST: torna P_k per k>0, 1 per k=0
    */
```

che restituisce il k-esimo primoriale e 1 se $k = 0$.

Esercizio 2 Un numero è *altamente composto* se il numero dei suoi divisori distinti è strettamente maggiore di quello di tutti i naturali che lo precedono. La successione dei numeri altamente composti è 1, 2, 4, 6, 12, 24, 36, 48, 60, 120, 180, 240, 360, 720, 840, 1260, ... Scrivere una funzione C di prototipo:

```
int altamenteComposto(int n, int *d)
   /* PREC: n>0
    * POST: torna 1 se n altamente composto, 0 altrimenti
    * arica in *d il numero dei divisori di n
    */
```

che preso in ingresso un numero n restituisce 1 se n è altamente composto e 0 altrimenti. Carica **sempre** nella variabile `*d` il numero dei divisori di n.

Esempi: Preso in input 3 la funzione torna 0 e carica in `*d` il valore 2 in quanto 3 è divisibile per 1 e per sè stesso (la funzione carica in `*d` sempre 2 sui numeri primi). Preso in input 2 tuttavia restituisce 1, perché 1 ha un solo divisore e invece 2 è il primo ad averne 2. Preso in input 12 restituisce 1 e carica in `*d` il valore 6.

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-P-1.c`, `main-P-2.c`, e `main-P-3.c` forniti nella pagina degli Homework.

Potete chiamare come volete il file `.c` contenenti la funzione richiesta (ed eventuali funzioni ausiliarie): il sistema li rinominerà automaticamente usando il vostro twiki name. Se il vostro twiki name fosse `LilyEvans`, essi verranno chiamati `LilyEvans.1.c`, `LilyEvans.2.c`, e `LilyEvans.3.c`. Il comando utilizzato per la compilazione sarà:

```
gcc -std=c99 main-P-1.c LilyEvans.1.c
gcc -std=c99 main-P-2.c LilyEvans.2.c
gcc -std=c99 main-P-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-P-1.c` dovrebbero generare con successo un file chiamato rispettivamente `LilyEvans.1.o` e `main-P-1.o`.