Informatica Generale
Homework P/2020:
The best of '10s Homeworks.

Ivano Salvo – Sapienza Università di Roma – 25/3/2020

Esercizio 1 (2017) 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 treXPiUno(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 2 (2019) Un numero è *altamente composto* se il numero dei suoi divisori distinti è strettamente maggiore di quello dei divisori 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
    * carica 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 (2018) Nello stato di Fibolandia hanno corso legale i Fiorinacci, moneta i cui tagli sono numeri di Fibonacci, e cioè 1, 2, 3, 5, 8, 13, 21, 34, 55, ... (osservate che a differenza della serie di Fibonacci, c'è ovviamente un'unica moneta di taglio 1). I commercianti di Fibolandia sono tutti matematici buontemponi e, dovendo dare di resto n Fiorinacci, consegnando le monete, pronunciano sempre il numero di tutti i possibili *insiemi* di tagli di monete che danno n come somma.

Esempio: Se il resto da dare fossero 9 Fiorinacci, viene pronunciato il numero 17 in quanto i possibili modi di dare il resto sono i seguenti (attenzione quindi che *non* vengono ricontati gruppi di monete che differiscono solo per l'ordine, ad esempio $\{5,2,1,1\}$ e $\{1,2,5,1\}$):

$$\{8,1\},\{5,3,1\},\{5,2,2\},\{5,2,1,1\},\{5,1,1,1,1\},\{3,3,3\},\{3,3,2,1\},\{3,3,1,1,1\},$$
$$\{3,2,2,2\},\{3,2,2,1,1\},\{3,2,1,1,1,1\},\{3,1,1,1,1,1,1\},\{2,2,2,2,1\},$$
$$\{2,2,2,1,1,1\},\{2,2,1,1,1,1,1\},\{2,1,1,1,1,1,1,1\},\{1,1,1,1,1,1,1,1,1\}$$

Scrivere una funzione C di prototipo:

```
int hereYouAre(int n)
   /* PREC: n>=0 */
```

che restituisce come risultato il numero pronunciato dai commercianti di Fibolandia quando danno resto n (se n fosse 0, il commerciante risponde 0).

Suggerimenti e Osservazioni: Questo esercizio non necessita di memorizzare i numeri di fibonacci: avendo una coppia di numeri consecutivi di Fibonacci, potete infatti calcolare alla bisogna sia i successivi che i precedenti.

Questo esercizio potrebbe apparire *molto* impegnativo, ma cercando con cura troverete problemi simili risolti.

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-2020-P-1.c`, `main-2020-P-2.c`, e `main-2020-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-2020-P-1.c LilyEvans.1.c
gcc -std=c99 main-2020-P-2.c LilyEvans.2.c
gcc -std=c99 main-2020-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-2020-P-1.c` dovrebbero generare con successo un file chiamato rispettivamente `LilyEvans.1.o` e `main-2020-P-1.o`.