Informatica Generale
Homework 4: Finchè c'è memoria c'è speranza

docente: Ivano Salvo – Sapienza Università di Roma

pubblicazione: 13.IV.2012 - consegna 3.V.2012

Finalmente questo homework vi darà la possibilità di implementare un tipo di dato di cui avete tanto sentito parlare: i numeri naturali. Non gli insipidi `unsigned int` del C, ma i Naturali veri con la 'N' maiuscola, quelli di Peano, per intenderci. O quantomeno quel sottoinsieme finito che la vostra macchina vi permette di rappresentare ☹.

Dovrete definire il tipo `naturale`, dimodochè possiate rappresentare numeri naturali comunque grandi, con l'unica limitazione della memoria concessa dal sistema all'esecuzione del vostro programma.

Avete, in linea di principio, libertà di rappresentazione. Tuttavia, dovendo produrre stampe in decimale, forse vi converrà ragionare in decimale. Potete rappresentare un intero con una sequenza (o lista) di cifre decimali, implementando le usuali operazioni come vi ha insegnato la maestra elementare, cifra a cifra (sconsiglio vivamente, per ragioni di efficienza di implementare tutto a partire dal successore!). Per ragioni di efficienza, siccome un calcolatore moderno in un ciclo macchina (o circa) usualmente fa operazioni su dati di 32 bit, è una buona idea rappresentare un naturale come una sequenza di "blocchetti di cifre" rappresentabili su 32 bit.

Ad esempio, scegliendo come dato per un singolo blocchetto gli `unsigned int` a 32 bit, idealmente potete rappresentare in un blocchetto i numeri naturali compresi nell'intervallo $[0, 2^{32} - 1]$, cioè $U = [0, 4.294.967.295]$[1]. Dovendo ragionare in decimale potete pensare che un blocchetto rappresenti numeri di 9 cifre decimali, quindi nell'intervallo $I = [0, 999.999.999]$: sommando due numeri che stanno in I, ottenete ancora un `unsigned int` e dovete, eventualmente considerare un riporto implementando l'algoritmo della maestra per la somma. Mi pare che il riporto non possa mai essere maggiore di 1.

Diverso il caso se nei vostri algoritmi decidete di implementare il prodotto tra `naturali` (non è strettamente necessario). In tal caso, per implementare il classico algoritmo per fare le moltiplicazioni con carta e penna, vi serve fare il prodotto blocchetto per blocchetto, ma $999.999.999 \times 999.999.999 = 999.999.998.000.000.001$ che è un numero di 18 cifre e non è un `unsigned int`. In tal caso, dovete o ridurre la dimensione del numero rappresentato ciascun blocchetto, oppure....

[1] a scanso di equivoci, i punti per separare blocchi di 3 cifre decimali sono qui messi solo per facilitare la lettura. Non dovranno apparire negli output dei vostri programmi.

Attenzione anche alle stampe. Se state lavorando con blocchetti di k cifre e il numero memorizzato ha meno di k cifre, ma rappresenta un "pezzo" interno di un numero naturale, nelle stampe dovete aggiungere degli 0 (significativi!) davanti al vostro numero, fino a completare le k cifre. Ad esempio, se state lavorando con blocchetti di 3 cifre, il numero 12032551 verrà rappresentato dai tre blocchetti 12, 32, 551: nella stampa dovete anteporre uno 0 al 32 (ma non al 12, visto che è il blocchetto con cui il numero inizia).

Per rinfrancare lo spirito prima di mettersi al lavoro, una facezia matematica: un "interessante" teorema sui numeri naturali, che qualcuno di voi forse già conosce.

Teorema 1 *Tutti i numeri naturali sono "interessanti".*
Dim: Supponete per assurdo che esistano numeri naturali non interessanti. Consideriamo l'insieme N di tutti i numeri naturali non interessanti. Date le nostre assunzioni, $N \neq \varnothing$. Quindi N, essendo un sottoinsieme non vuoto dei naturali, ammette minimo (l'ordine sui naturali è totale e *ben fondato*, non ammette cioè catene infinite decrescenti). Sia ora n il minimo di N, cioè il minimo dei numeri non interessanti. Godendo di questa proprietà però, n è chiaramente "interessante", il che contraddice il fatto che $n \in N$. Assurdo ☺. ✓

Esercizio 1: (FATTORIALE) Scrivere un programma che legge in input un numero intero n (che potete memorizzare in un usuale `int`) e restituisce in output una sequenza di cifre decimali che rappresenta il numero naturale che rappresenta $n!$. Tenete conto che ad esempio 95! ha già 141 cifre.

OSSERVAZIONE: questo esercizio non vi obbliga a definire il prodotto tra due valori di tipo `naturale`. Siccome l'input è un intero, lo potete memorizzare in una variabile di tipo `int` e dopodichè vi sarà sufficiente una funzione di prototipo `naturale prod(naturale, int);`, che potreste implementare con somme successive, magari sfruttando l'algoritmo della moltiplicazione egiziana, oppure altri algoritmi che la vostra fantasia vi suggerisce.

Esercizio 2: Scrivere un programma che legge due sequenze di cifre decimali ciascuna chiusa da un `*`, le interpreta come due numeri naturali (la prima cifra letta è la più significativa), verifica quale dei due è maggiore, e stampa in output la differenza ottenunta sottraendo il minore dal maggiore. Ad esempio, dato l'input[2]: `1 5 6 * 2 1 6 *`, il vostro programma dovrà rispondere `60`.

Esercizio 3: (CODIFICHE) Leggere una sequenza di interi $a_1, a_2, \ldots, a_n$ positivi chiusa da un -1. Restituire in output la codifica della sequenza nel numero naturale $\sum_{i \in [n]} p_i^{a_i}$ dove p_i è l'i-esimo numero primo. Supporre che la sequenza non sia più lunga di 150 valori (vi bastano quindi i numeri primi minori di 1000, per tenersi larghi).

[2] le cifre saranno in realtà separate da un carattere `\n` come nell'esercizio del lucchetto.