Laboratorio di Fondamenti di Informatica Lab03_2015-10-22

Anno accademico 2015/2016

1. Controllo parentesi (basic version)

Si legga da stdin una sequenza *a priori illimitata* di caratteri terminata da '\n' e – senza memorizzarla, giacché è inutile – si controlli se nella sequenza le parentesi sono correttamente annidate e bilanciate (cioè se sono tutte ordinatamente richiuse). Per semplicità consideriamo solo le parentesi tonde. Esempi:

Sequenze corrette:

```
ciao
()
(ciao)belli
((ciao)(cia(o))(c(i)ao))(bell()i)
```

Sequenze non corrette:

```
cia)o
c(iao
ci(ao)be(lli
ci(ao)b(ell(i)!!
ciao(ci)ao) (belli)
```

Suggerimento: si tenga conto man mano di quante sono le parentesi già aperte, e si decida di conseguenza. **Si arresti la scansione nonappena si scopre che la sequenza è illegale**, anche prima di arrivare alla fine (come succede, ad esempio, nel primo e nell'ultimo dei casi scorretti mostrati sopra; negli altri tre casi occorre invece arrivare alla fine della sequenza per poterla dichiarare scorretta).

2. Triangolo di Floyd

Scrivere un programma che legge da stdin un numero intero n e stampa a video le prime n righe del triangolo di Floyd (un triangolo rettangolo che contiene tutti i numeri naturali disposti come segue). Ad esempio, per n=10:

```
 1
 2  3
 4  5  6
 7  8  9 10
11 12 13 14 15
16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54 55
```

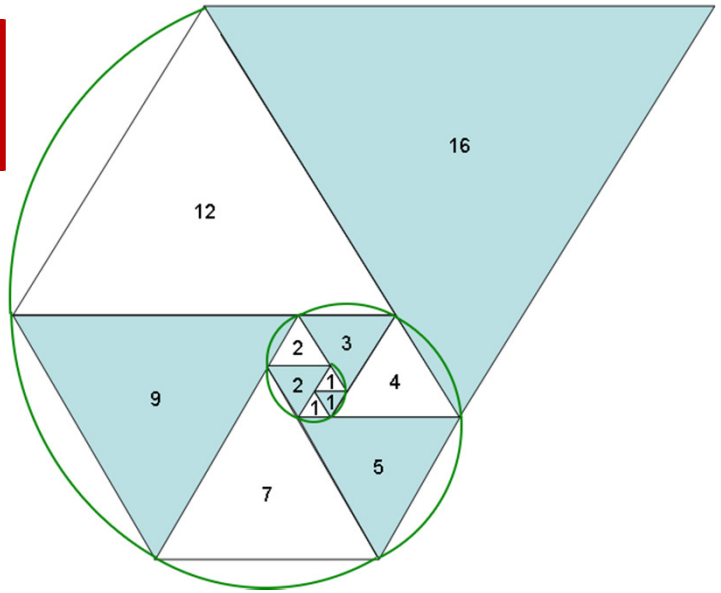
La soluzione più immediata utilizza **due cicli annidati**. Si progetti, come esercizio di "**elasticità espressiva**", anche una soluzione che utilizza **un solo ciclo**.

Variante *(probabilmente a prima vista non banale):*

- *Si stampi il triangolo "al contrario", cioè partendo dall'ultima riga (nell'esempio: partendo dal 46):*

```
46 47 48 49 50 51 52 53 54 55
37 38 39 40 41 42 43 44 45
29 30 31 32 33 34 35 36
22 23 24 25 26 27 28
...
```

3. Successione di Padovan



La successione di Padovan è la serie di numeri naturali P(n) definita dai valori iniziali:

P(0) = P(1) = P(2) = 1

E per tutti i valori di n > 3 dalla relazione:

P(n) = P(n-2) + P(n-3)

I primi valori della successione sono quindi:

1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, 21, …

Scrivere un programma che chiede all'utente un intero positivo n e stampa in ordine tutti i numeri della successione minori di n.

Variante: Scrivere un programma che **verifica se** il numero inserito (che deve essere positivo) è uno degli elementi della successione di Padovan oppure no.

Attenzione (aiutini):

- Si dichiarino e inizializzino opportune variabili dedicate a calcolare la sequenza di Padovan (tramite un ciclo che ad ogni iterazione calcoli l'elemento successivo)
- Si badi a gestire correttamente l'uscita dal ciclo!
- Non è agevole fare una verifica diretta sul numero inserito dall'utente. Conviene piuttosto calcolare a (partire da 1,1,1) tutti gli elementi della successione, in ordine crescente, fino a quando è possibile verificare l'appartenenza o meno del numero inserito da terminale

4. Indovina il numero

Si chieda all'utente di indovinare un numero compreso tra 0 e 100, generato casualmente dal programma. Si chieda ripetutamente all'utente di effettuare un tentativo, rispondendo di volta in volta se il numero proposto è minore o maggiore di quello «segreto».

```
Ho "pensato" un numero tra 0 e 100.
Indovinalo : 23
No, troppo piccolo. Riprova: 59
No, troppo grande. Riprova: 42
Bravo! Era il 42
```

Ma come si genera un numero casuale? È spiegato nella slide successiva, e inoltre… …trovate il programma guess.c ***su BeeP già parzialmente scritto (e funzionante)!***

Varianti e aggiunte (*per l'eventuale meditazione domestica*)

Si generi un numero compreso tra due costanti MIN e MAX, si facciano i complimenti all'utente se indovina sufficientemente (?) in fretta, e gli si dia apertamente dell'asino se invece impiega un numero di tentativi superiore al numero minimo che garantisce di indovinare sempre.

Gerazione di numeri casuali

In C è possibile ottenere numeri casuali[1] tramite la funzione **rand()**, funzione definita nella libreria **<stdlib.h>**. La funzione restituisce numeri interi compresi tra 0 e RAND_MAX (un valore molto alto, definito anch'esso nella libreria).

Per riportare il numero casuale all'interno di un intervallo voluto [0, N] si può usare una delle seguenti espressioni:

rand() % (N+1) oppure[2] **rand() / (RAND_MAX / N + 1)**

Per evitare di ottenere la stessa sequenza di numeri ad ogni esecuzione, inoltre, all'inizio del programma conviene inizializzare il generatore tramite la chiamata «**srand(time(NULL))**» La funzione **time()** è contenuta nella libreria **<time.h>**.

«time(NULL)» restituisce un numero intero che corrisponde alla data corrente rappresentata nel formato «UNIX epoc time» , che non è altro che il numero di secondi a partire dal 01/01/1970)

---

*1) In realtà sono numeri **pseudo**-casuali*

2) Attenzione: per i nostri scopi sono sostanzialmente equivalenti, ma è da preferire la seconda versione, che fa uso delle proporzioni, perché effettua una partizione dell'insieme dei risultati lavorando sui bit più "alti", che sembrano essere distribuiti molto più uniformemente. (fonte: http://c-faq.com/lib/randrange.html... Google is Your Friend!)