Riepilogo sulla ricorsione

Un algoritmo o una funzione matematica sono definiti ricorsivamente quando possono essere definiti facendo riferimento a se stessi.

Si tratta cioè di un'applicazione del principio di induzione matematica:

Data una proprietà *P*, parametrica in *n*,

1. Si verifica che *P* è vera per $n=n_0$ (CASO BASE)

2. Fatta l'ipotesi che *P* sia valida per un *(n-1)* generico, se si può provare che *P* è vera anche per il caso *n*, allora *P* è vera per ogni $n>n_0$. (PASSO DI INDUZIONE)

Operativamente, risolvere un problema con un approccio ricorsivo comporta:

1. identificare un "caso base" la cui soluzione sia nota
2. e riuscire a esprimere la soluzione del caso generico (*n*) in termini dello stesso problema formulato su casi più semplici (*n-1, n-2,…*)

Le chiamate ricorsive decompongono via via il problema.

Un modo di procedere è quello di non calcolare nulla durante questa scomposizione, ma sintetizzare il risultato a partire dalla fine, dopo essere arrivati al caso "banale".
Il caso "banale" fornirà il valore di partenza con il quale sintetizzare (comporre), "a ritroso", i successivi risultati parziali.

ESERCIZIO 1

Scrivere un programma C che, dato un numero calcola la somma dei primi N numeri pari positivi in maniera ricorsiva.

Specifica Liv 1:

La somma dei primi N numeri pari è data dalla seguente,

$$S_N = 2*1 + 2*2 + 2*3 + \ldots + 2*i + \ldots + 2*(N-1) + 2*N.$$

Specifica Liv 2:

se N =1, $S_N = 2$, **(CASO BASE)**
se N >1, $S_N = 2 * N + S_{N-1}$ **(PASSO INDUTTIVO)**
(somma dell'N-esimo numero pari +
la somma dei primi N-1 numeri pari.)

```
int somma_pari(int N){
   if (N <= 1)
     return 2;
   else
     return 2*N + somma_pari(N-1);
}
```

ESERCIZIO 2

Scrivere una funzione che calcoli il valore M^N, dove M è un numero in virgola mobile e N un numero intero.

1mo caso:

Hp: M intero, N intero con N>=0

Specifica Liv 1:

M^N = M*M*M*…*M; N volte.

Specifica Liv 2:

Se N = 0, allora M^N = 1 (caso base)
Se N è pari, allora $M^N = (M^{N/2})^2$ (passo induttivo)
Se N è dispari, allora $M^N = (M^{N-1})*M$ (passo induttivo)

per considerare anche il caso di potenze negative cosa occorrerebbe fare?

Soluzione:

```
double myExp(double M, int N) {

   if (N < 0)
     return myExp(1.0/M, -N);

   if (N == 0)
     return 1;

   if (N % 2  == 0) {
     long t = myExp(M, N/2);
     return t*t;
   }

   if (N % 2  == 1)
     return M*myExp(M, N-1);

}
```

ESERCIZIO 3

Scrivere una funzione ricorsiva `void BoomBang(int k)` che stampa `k` volte la stringa "`Boom`" seguita dalla stampa della stringa "`Bang`" anche'essa `k` volte.

Esempio:

`BoomBang(3)`

output: `Boom Boom Boom Bang Bang Bang`

```
void BoomBang(int k) {

   if (k == 0) return;

   cout << "Boom ";
   BoomBang(k-1);
   cout << "Bang ";

}
```

ESERCIZIO 4

Scrivere un sottoprogramma che restituisca il valore massimo di un vettore di interi con procedimento ricorsivo.

Specifica Liv 1:
Si pensi di assegnare il primo elemento dell'*array* come massimo e confrontarlo con tutti gli altri cambiando il valore del massimo se questo è minore della cella corrente del vettore.

Specifica Liv 2:
Detta `N` la lunghezza del vettore `array`

```
Se N = 1, allora max = array[0]            (caso base)
```

Se `N > 1`, allora il massimo del vettore di `N` elementi (max) sarà uguale al risultato del confronto tra `array[0]` e il massimo degli elementi del sotto-vettore che che va da `array[1]` a `array[N]`. (N.B.: quindi è lungo `N-1`) (passo induttivo)

```
int myMaxArray(int v[], int N) {
   if (N <= 0) return 0; // valore massimo convenzionale!

   if (N == 1) return v[0];
   else {
     int t =  myMaxArray(&v[1], N-1);
     if (t > v[0]) return t;
     else return v[0];
   }
}
```

Chiamata nel `main()`:

```
...
int vett[3] = {18, 21, 27}, lung = 3, m;
m = myMaxArray(vett, lung);
...
```

Soluzione con prototipo alternative: `main()`:

```
int myMaxArray(int v[], int indiceInizio, int indiceFine) {
   if (indiceInizio > indiceFine) return 0; // valore massimo
                                            // convenzionale!

   if (indiceInizio == indiceFine) return v[indiceInizio];
   else {
     int t =  myMaxArray(v, 1+indiceInizio, indiceFine);
     if (t > v[indiceInizio]) return t;
     else return v[indiceInizio];
   }
}
```

Chiamata nel `main()`:

```
...
int vett[3] = {18, 21, 27}, lung = 3, m;
m = myMaxArray(vett, 0, lung-1);
...
```

ESERCIZIO 5.1

Scrivere un sottoprogramma che inverta un vettore di interi con procedimento ricorsivo.

```
void invertiVettore(int v[], int n) {
     if (n <= 1) return;
     int t = v[0];
         v[0] = v[n-1];
         v[n-1] = t;
         invertiVettore(&v[1], n-2);
}
```

Chiamata nel `main()`:

```
     ...
     int vett[3] = {18, 21, 27}, lung = 3, m;
     invertiVettore(vett, lung);
     ...
```

Soluzione con prototipo alternativo:

```
void invertiVettore(int v[], int indiceInizio, int indiceFine)
{
     if (indiceFine <= indiceInizio) return;
     int t = v[indiceInizio];
         v[indiceInizio] = v[IndiceFine];
         v[Indicefine] = t;
         invertiVettore(v, indiceInizio+1, indiceFine-1);
}
```

Chiamata nel `main()`:

```
     ...
     int vett[3] = {18, 21, 27}, lung = 3, m;
     invertiVettore(v, 0, lung-1);
     ...
```

ESERCIZIO 5.2

Scrivere un sottoprogramma ricorsivo che preso come parametro una stringa ritorni 1 se la stringa è palindroma e 0 altrimenti.

```
int palindromaRicorsiva(char s[], int idxInizio, int IdxFine) {
     if ( idxInizio >= IdxFine ) return 1;
     if ( s[idxInizio] != s[idxFine] ) return 0;
     return palindromaRicorsiva(s, idxInizio+1, idxFine-1)
}

int palindroma(char s[]) {

    return palindomaRicorsiva(s, 0, strlen(s)-1);
}
```

ESERCIZIO 6

Calcolo del coefficiente binomiale
Scrivere un programma C/C++ che stampi sullo standard output tutti i valori del triangolo di Tartaglia per un certo ordine N, utilizzando una funzione ricorsiva per il calcolo dei coefficienti binomiali, avente il seguente prototipo:

```
int cobin(int n, int k);
```

La costruzione del triangolo di Tartaglia è mostrata di seguito.
Si ricorda inoltre che ognuno dei coefficienti del triangolo prende il nome di coefficiente binomiale `cobin(n, k)`

```
k= 0  1  2   3   4   5
   1                        n = 0
   1  1                     n = 1
   1  2  1                  n = 2
   1  3   3   1             n = 3
   1  4   6   4   1         n = 4
   1  5   10  10  5   1     n = 5
.....................................
```

Soluzione:

Leggendo la figura del triangolo di Tartaglia riga per riga, è possibile dedurre come il calcolo di ognuna di esse sia funzione della riga precedente.
Il calcolo dei coefficienti binomiali segue dunque le seguenti regole:

- calcolo dei coefficienti agli estremi di una riga:
  con `k == 0` e `k == n`, `cobin(n,k) == 1` **(caso base)**

- calcolo dei coefficienti per valori non validi dei parametri:
  con `n < 0` e `k < 0` e `n < k`, `cobin(n,k) == 0` **(caso base)**

- Ogni coefficiente è la somma del suo "soprastante"
  e del predecessore di quest' ultimo. **(passo induttivo)**

```
/* Nota: #include <iomanip>
Larghezza campo di output (n colonne):
                cout << setw(n);
Numero di cifre decimali dopo la virgola (n cifre):
               cout << setprecision(n);
*/
```

```
#include <iostream>
#include <iomanip>
#define N 6

int cobin(int n, int k);

int main() {
   int n, k;
   for (n=0; n <= N; n++) {
      for (k=0; k <= n; k++)
         cout << setw(5) << cobin(n, k));
      cout << endl;
   }
   return 0;
} // end main


int cobin(int n, int k) {

     if  (n < k  || n < 0  || k < 0 ) return 0;
     if  (k == 0 || k == n || n == 0) return 1;

     return cobin(n-1, k-1) + cobin(n-1, k);

} // end cobin
```

ESERCIZIO 7 : Ricerca Dicotomica

Si scriva un sottoprogramma C/C++ che effettui la ricerca di un intero in un vettore **ordinato** (in senso crescente) in maniera ricorsiva, restituendo l' indice della prima occorrenza nel vettore del valore cercato.

```
// Restituisce l' indice della cella contenente un valore
// uguale a x; oppure -1

int dicotomica(int v[], int left, int right, int x) {
    int middle;

    if (left > right)
      return -1

    middle = (left+right) / 2;

    if (x < v[middle])
      return dicotomica(v, left, middle-1, x) ;

    if (x == v[middle]) return middle;

    return dicotomica(v, middle+1, right, x) ;
}

int cerca (int v[], int len, int x) {

    return dicotomica(v, 0, len-1, x);

} // end cerca
```