

Esercizi su funzioni ricorsive in C

Stefano Cherubin*

15/12/2016

[**Informatica A**] Esercitazione #12

corso per Ing. Gestionale a.a. 2016/17

*<nome>.<cognome>@polimi.it

Indice

1	Ricerca binaria	3
1.1	Soluzione	4
1.1.1	Ricerca binaria (ricorsiva)	4
1.1.2	Programma	4
1.1.3	Programma e funzione (modificati)	5
1.1.4	Versione iterativa	6
2	Inversione array ricorsiva	7
2.1	Soluzione C	7
3	Boom Bang	8
3.1	Soluzione C	8
4	Potenza ricorsiva	9
4.1	Soluzione C	9
5	Triangolo di Tartaglia	10
5.1	Coefficienti Binomiali	10
5.1.1	Approccio alla soluzione	10
5.1.2	Soluzione C	11
5.2	Stampa il tringolo	11
5.2.1	Soluzione C - Stampa del triangolo	11
5.2.2	Soluzione C - Stampa del triangolo al contrario	12

1 Ricerca binaria

Si consideri già disponibile il seguente frammento di codice

```
1 #include <stdio.h>
2 #define MAXLEN 15
3 void loadVector(int v[], int *n) {
4     int i;
5     do {
6         printf("\nQuanti elementi nel vettore? (0
          - %d):", MAXLEN);
7         scanf("%d", n); /*scanf("%d", &n)*/
8     } while (*n < 0 || *n > MAXLEN);
9     printf("\nInserisci %d elementi del vettore,
          separa con spazi ", *n);
10    for (i = 0; i < *n; ++i) {
11        scanf("%d", &v[i]);
12    }
13 }
14
15 void sortVector(int v[], int n) {
16     int i, j, tmp;
17     for (i = 0; i < n - 1; i++) {
18         for (j = i; j < n; j++) {
19             if (v[i] > v[j]) {
20                 tmp = v[i];
21                 v[i] = v[j];
22                 v[j] = tmp;
23             }
24         }
25     }
26 }
```

1. Scrivere una funzione ricorsiva che esegua la ricerca di un elemento all'interno del vettore secondo l'algoritmo di ricerca binaria (aka ricerca dicotomica).
2. Scrivere un programma che acquisisca in input un vettore di numeri interi e una chiave di ricerca (anch'essa come numero intero) e attraverso la funzione di ricerca scritta al punto precedente stabilisca se l'elemento chiave è contenuto nel vettore oppure no.
3. Se necessario, modificare il codice finora scritto per fare in modo che la funzione di ricerca abbia il seguente prototipo `int binarySearch(int key, int v[], int n);`.
4. Scrivere la funzione di ricerca binaria in versione iterativa (senza l'utilizzo di ricorsione).

1.1 Soluzione

1.1.1 Ricerca binaria (ricorsiva)

Listato 1: Ricerca binaria ricorsiva

```
1 int binarySearch(int key, int v[], int start, int end)
  {
2   int middle = (end + start) / 2;
3   if (start > end) {
4     return -1;
5   }
6   if (v[middle] == key) {
7     return middle;
8   }
9   if (key > v[middle]) {
10    return binarySearch(key, v, middle+1, end);
11  }
12  return binarySearch(key, v, start, middle-1);
13 }
```

1.1.2 Programma

Listato 2: Programma per la ricerca

```
1 int main() {
2   int v[MAXLEN];
3   int n, key, ret;
4   loadVector(v, &n);
5   sortVector(v, n);
6   printf("\nInsert an element as search key:");
7   scanf("%d", &key);
8   ret = binarySearch(key, v, 0, n);
9   if (ret == -1)
10    printf("\nElemento NON trovato.");
11  else
12    printf("\nElemento trovato.");
13  return 0;
14 }
```

1.1.3 Programma e funzione (modificati)

Listato 3: Ricerca binaria (con wrapper)

```
1  /* La medesima funzione di prima, rinominata */
2  int _binarySearch(int key, int v[], int start, int end
3  ) {
4      int middle = (end + start) / 2;
5      if (start > end) {
6          return -1;
7      }
8      if (v[middle] == key) {
9          return middle;
10     }
11     if (key > v[middle]) {
12         return _binarySearch(key, v, middle + 1, end);
13     }
14     return _binarySearch(key, v, start, middle - 1);
15 }
16 /* Funzione "wrapper" che si occupa solo di eseguire l'
17   'inizializzazione del parametro mancante per la
18   funzione _binarySearch */
19 int binarySearch(int key, int v[], int n) {
20     return _binarySearch(key, v, 0, n);
21 }
22 int main() {
23     int v[MAXLEN];
24     int n, key, ret;
25     loadVector(v, &n);
26     sortVector(v, n);
27     printf("\nInserisci una chiave da cercare:");
28     scanf("%d", &key);
29     ret = binarySearch(key, v, n);
30     if (ret == -1)
31         printf("\nElemento NON trovato.");
32     else
33         printf("\nElemento trovato.");
34     return 0;
35 }
```

1.1.4 Versione iterativa

Listato 4: Ricerca binaria iterativa

```
1 int binarySearchIter(int key, int v[], int n) {
2     int start, end, middle;
3     start = 0;
4     end = n;
5     while (start < end) {
6         middle = (start + end) / 2;
7         if (v[middle] == key) {
8             return middle;
9         }
10        if (key < v[middle]) {
11            end = middle - 1;
12        } else {
13            start = middle + 1;
14        }
15    }
16    return -1;
17 }
```

2 Inversione array ricorsiva

Scrivere una funzione che inverta un vettore di interi con procedimento ricorsivo.

2.1 Soluzione C

```
1 void vectorInverterRecursive(int v[], int n) {
2     if (n <= 1) return;
3     int t = v[0];
4     v[0] = v[n - 1];
5     v[n - 1] = t;
6     vectorInverterRecursive(&v[1], n - 2);
7 }
```

3 Boom Bang

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

3.1 Soluzione C

Listato 5: Boom Bang

```
1 void BoomBang(int k) {  
2     if (k == 0)  
3         return;  
4     printf("Boom ");  
5     BoomBang(k - 1);  
6     printf("Bang ");  
7 }
```


4 Potenza ricorsiva

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

1. Si implementi la funzione rispecchiando la seguente specifica:

- se $N = 0$ allora $M^N = 1$ *caso base*
- se N è pari allora $M^N = (M^{N/2})^2$ *passo indut.*
- se N è dispari allora $M^N = (M^{N-1}) \cdot M$ *passo indut.*

2. Si trovi il modo di gestire anche il caso di potenze negative ($N < 0$).

4.1 Soluzione C

Listato 6: Potenza ricorsiva

```
1 double myExp(double M, int N) {
2     if (N < 0) {
3         return myExp(1.0 / M, -N);
4     }
5     if (N == 0) {
6         return 1;
7     }
8     if (N % 2 == 0) {
9         long t = myExp(M, N / 2);
10        return t*t;
11    }
12    if (N % 2 == 1) {
13        return M * myExp(M, N - 1);
14    }
15 }
```

5 Triangolo di Tartaglia

Un coefficiente binomiale $\binom{n}{k}$ è il coefficiente che ha il k -esimo termine dello sviluppo della potenza n di un binomio, $(a + b)^n$, ordinando per potenze decrescenti di a .

Il triangolo di tartaglia è formato dai coefficienti binomiali aventi n crescenti dall'alto verso il basso e k crescenti da sinistra verso destra.

$\downarrow n \backslash k \rightarrow$	0	1	2	3	4	0	1	2	3	4
0	$\binom{0}{0}$					1				
1	$\binom{1}{0}$	$\binom{1}{1}$				1	1			
2	$\binom{2}{0}$	$\binom{2}{1}$	$\binom{2}{2}$			1	2	1		
3	$\binom{3}{0}$	$\binom{3}{1}$	$\binom{3}{2}$	$\binom{3}{3}$		1	3	3	1	
4	$\binom{4}{0}$	$\binom{4}{1}$	$\binom{4}{2}$	$\binom{4}{3}$	$\binom{4}{4}$	1	4	6	4	1

5.1 Coefficienti Binomiali

Sulla base del triangolo di Tartaglia sopra indicato, si scriva una funzione per il calcolo del coefficiente binomiale `int cobin(int n, int k)`

5.1.1 Approccio alla soluzione

Come si può notare dall'inizio del triangolo di Tartaglia tracciato poco sopra, ogni riga può essere ottenuta a partire dalla riga precedente, secondo le seguenti regole:

- con $k = 0$ e con $k = n$ si ha che $\binom{n}{k} = 1$ *caso base*
- i coefficienti binomiali con $(n < k, n < 0, k < 0)$ non sono validi e si assumono avere valore 0 *caso base*
- ogni coefficiente è la somma del soprastante e del predecessore di quest'ultimo *passo indut.*

5.1.2 Soluzione C

Listato 7: Coefficiente binomiale

```
1 int cobin(int n, int k) {
2     if (n < k || n < 0 || k < 0)
3         return 0;
4     if (k == 0 || k == n || n == 0)
5         return 1;
6     return cobin(n - 1, k - 1) + cobin(n - 1, k);
7 }
```

5.2 Stampa il tringolo

1. Stampare il triangolo di Tartaglia fino alla riga n , con n inserito dall'utente.
2. Stampare il triangolo di Tartaglia al contrario, partendo dalla riga n , con n inserito dall'utente.

5.2.1 Soluzione C - Stampa del triangolo

Listato 8: Triangolo di Tartaglia

```
1 int cobin(int n, int k) {
2     if (n < k || n < 0 || k < 0)
3         return 0;
4     if (k == 0 || k == n || n == 0)
5         return 1;
6     return cobin(n - 1, k - 1) + cobin(n - 1, k);
7 }
8
9 void triangoloTartaglia(int maxRiga) {
10     int n, k, c;
11     for (n = 0; n <= maxRiga; ++n) {
12         for (k = 0; k <= n; ++k) {
13             c = cobin(n, k);
14             printf ("%d ", c);
15         }
16         printf ("\n");
17     }
18 }
19
20 int main() {
21     int r;
22     do {
23         printf ("Triangolo di Tartaglia, fino a riga ");
24         scanf("%d", &r);
```

```

25     } while (r < 0);
26     triangoloTartaglia(r);
27     return 0;
28 }

```

5.2.2 Soluzione C - Stampa del triangolo al contrario

Listato 9: Triangolo di Tartaglia invertito

```

1  int cobin(int n, int k) {
2      if (n < k || n < 0 || k < 0)
3          return 0;
4      if (k == 0 || k == n || n == 0)
5          return 1;
6      return cobin(n - 1, k - 1) + cobin(n - 1, k);
7  }
8
9  void triangoloTartagliaReverse(int maxRiga) {
10     int n, k, c;
11     for (n = 0; n <= maxRiga; ++n) {
12         for (k = n; k >= 0; --k) {
13             c = cobin(n, k);
14             printf ("%d ", c);
15         }
16         printf ("\n");
17     }
18 }
19
20 int main() {
21     int r;
22     do {
23         printf("Triangolo di Tartaglia invertito, dalla a
                riga ");
24         scanf("%d", &r);
25     } while (r < 0);
26     triangoloTartagliaReverse(r);
27     return 0;
28 }

```

Licenza e crediti

Crediti

Quest'opera contiene elementi tratti da materiale di Gerardo Pelosi redatto per il corso di Fondamenti di Informatica per Ingegneria dell'Automazione a.a. 2014/15.

Licenza beerware¹

Quest'opera è stata redatta da Stefano Cherubin. Mantenendo questa nota, puoi fare quello che vuoi con quest'opera. Se ci dovessimo incontrare e tu ritenessi che quest'opera lo valga, in cambio puoi offrirmi una birra.

¹<http://people.freebsd.org/~phk/>