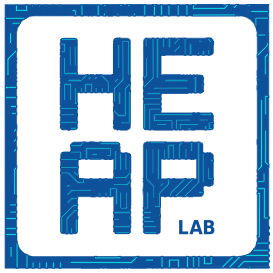
Esercizi su funzioni ricorsive in C

Stefano Cherubin*



28/11/2019

[**Informatica A**] Esercitazione #12

corso per Ing. Gestionale a.a. 2019/20

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

Indice

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

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 prodedimento ricorsivo.

2.1 Soluzione C

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

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 $M^N = \left(M^{M/2}\right)^2$ *passo indut.*
   - se N è dispari allora $M^N = \left(M^{N-1}\right) \cdot M$ *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 cresscenti 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  }
```

6 KITT

Negli anni '80 una serie TV americana introduceva uno dei primi concept di smart self-driving car con «Supercar». Il display montato sul muso dell'auto era composto da led rossi illuminati a diverse intensità. Il pattern di illuminazione del display era un segmento in movimento. Il segmento aveva maggiore intensità nella sua parte centrale e intensità decrescente verso i suoi estremi. Lo stesso segmento si spostava sul display avanti e indietro lungo la direzione orizzontale.

Si consideri di avere a disposizione un display simile. Per comandare l'illuminazione del display, esso riceve in input un vettore di N elementi interi. Ogni elemento del vettore rappresenta l'intensità di illuminazione di un elemento del display. Ogni elemento del display può illuminarsi a L livelli di intensità (da 0 a L-1).

Esempio di configurazione del vettore all'istante di tempo t_x.

```
t0: 1 2 3 2 1 0 0
t1: 0 1 2 3 2 1 0
t2: 0 0 1 2 3 2 1
t3: 0 0 0 1 2 3 2
t4: 0 0 0 0 1 2 3
t5: 0 0 0 1 2 3 2
t6: 0 0 1 2 3 2 1
t7: 0 1 2 3 2 1 0
t8: 1 2 3 2 1 0 0
```

Si scriva una funzione C ricorsiva che permetta di riempire il vettore display per ogni istante di tempo t_x. Si consiglia di utilizzare le seguenti funzioni per eseguire una più efficace implementazione:

```c
#include <stdio.h>
#include <unistd.h>  // sleep()

#define N 15

sleep(1); // stop the program for 1 second

void print(int v[]) {
  // print backspace: erase display
  for (int i = 0; i < N; i++) printf("\b\b\b");
  // print vector
  for (int i = 0; i < N; i++) printf("%d ", v[i]);
  // flush the buffer
  fflush(stdout);
  return;
}
```

6.1 Approccio alla soluzione

6.1.1 Dividi et impera

Il problema dato all'apparenza ha una complessità notevole. Ad una attenta anilisi si possono però riconoscere alcuni sottoproblemi più semplici che possono essere affrontati più facilmente. Scrivere una funzione C o anche solo un frammento di codice che risolva un sottoproblema sarà sicuramente di aiuto nella stesura della soluzione finale.

6.1.2 Disegna un segmento

Tra le altre cose, viene richiesto di disegnare un segmento in un vettore secondo un determinato pattern. Il pattern richiesto prevede una sequenza di numeri crescente seguita da una sequenza di numeri decrescente.

Esempio:

```
1 2 3 2 1
```

Una soluzione di getto... Un frammento di codice che produce questa soluzione è il seguente:

```
1  #define L 4
2  #define N 5
3
4  int i, k;
5  int v[N] = {0};
6  i = 0;
7  k = 1;
8  while (k < L) {
9    v[i] = k;
10   i++;
11   k++;
12 }
13 k -= 2;
14 while (k > 0) {
15   v[i] = k;
16   k--;
17   i++;
18 }
```

Facciamolo ricorsivo Il frammento di codice riportato qui sopra ha però un problema. Esso è scritto in forma iterativa. Il testo dell'esercizio esplicitamente chiedeva che venisse scritto in forma ricorsiva.

Riadattiamo quindi l'algoritmo in forma ricorsiva.

```
1  #define L 4
2  #define N 5
3
4  void fill_wrapper(int v[]) {
5    fill_up(v, 0, 1);
6    return;
7  }
8
9  void fill_up(int v[], int i, int k) {
10   if (k >= L) return fill_down(v, i+1, k-1);
11   v[i] = k;
12   fill_up(v, i+1, k+1);
13 }
14
15 void fill_down(int v[], int i, int k) {
16   if (k <= 0) return;
17   v[i] = k;
18   fill_down(v, i+1, k-1);
19 }
```

Si noti come in questo specifico caso le condizioni di terminazione dei cicli sono diventate casi base per terminare la ricorsione. La semantica è preservata.

Ottimizziamo Con l'aggiunta si un caso base nella ricorsione è possibile gestire anche la situazione di segmento più lungo del vettore.

Si noti che le due funzioni fill_up e fill_down sono molto simili tra di loro. Si propone di seguito una soluzione che le fonde assieme in una unica funzione con l'aggiunta di un parametro flag.

```
1  #define L 4
2  #define N 5
3
4  void fill_wrapper(int v[]) {
5    fill(v, 0, 1, 1);
6    return;
7  }
8
9  void fill(int v[], int i, int k, int direction) {
10   if (i >= N) // end of vector
11     return;
12   if (k >= L) // change direction
13     return fill(v, i+1, k-1, !direction);
14   v[i] = k;
15   // continue in the same direction
16   if (direction)
17     fill(v, i+1, k+1, direction);
18   else
19     fill (v, i+1, k-1, direction);
20   return;
21 }
```

6.1.3 Spostare un segmento

Il problema chiede di spostare il segmento avanti e indietro nel vettore.

Per fare ciò è sufficiente cambiare l'indice che stabilsce l'inizio del segmento.

```
1  int start;
2  for (start = 0; start < N; start ++) {
3    // initialize v to all 0
4    // ...
5    fill(v, start, 1, 1); // move ahead
6  }
7  for (start = N-2; start > 0; start --) {
8    // initialize v to all 0
9    // ...
10   fill(v, start, 1, 1); // move backward
11 }
```

Sarà sufficiente riadattare il codice scritto in precedenza per disegnare un segmento alla struttura qui illustrata per ottenere l'effetto desiderato.

6.1.4 Inserire una stampa a video

Si utilizzi la stampa a video fornita nel testo del problema. In quali punti è opportuno mostrare l'output? Al termine di ogni iterazione si avrà un risultato diverso. Si consiglia pertanto di osservare l'output di ogni iterazione.

```
1  int start;
2  for (start = 0; start < N; start ++) {
3    // initialize v to all 0
4    // ...
5    fill(v, start, 1, 1); // move ahead
6    print(v);
7  }
8  for (start = N-2; start > 0; start --) {
9    // initialize v to all 0
10   // ...
11   fill(v, start, 1, 1); // move backward
12   print(v);
13 }
```

6.1.5 Chi è più veloce: l'occhio o il computer?

Per ovvi motivi di velocità di esecuzione del computer, le iterazioni saranno sovrascritte con una velocità superiore alla umana possibilità di osservare l'output. Esiste però il modo di costringere il programma a fermarsi per un intervallo di tempo sufficiente a permettere l'osservazione del risultato.

```
1  int start;
2  for (start = 0; start < N; start ++) {
3    // initialize v to all 0
4    // ...
5    fill(v, start, 1, 1); // move ahead
6    sleep(1);
7  }
8  for (start = N-2; start > 0; start --) {
9    // initialize v to all 0
10   // ...
11   fill(v, start, 1, 1); // move backward
12   sleep(1);
13 }
```

6.2 Soluzione C

Listato 10: KITT

```c
#include <stdio.h>
#include <unistd.h>  // sleep()

#define N 15
#define L 5

void print(int v[]);
void fill(int v[], int start, int val, int raising);
void fill_wrap(int v[], int start);

int main(int argc, char*argv[]) {
  int display[N] = {0};
  for (int i = 0; i < N; i++) {
    fill_wrap(display, i);
    print(display);
    sleep(1); // stop the program for 1 second
  }
  for (int i = N-2; i > 0; i--) {
    fill_wrap(display, i);
    print(display);
    sleep(1); // stop the program for 1 second
  }
  return 0;
}

void fill_wrap(int v[], int start) {
  for (int i = 0; i < N; i++)
    v[i] = 0;  // initialize to 0 every time
  fill(v, start, 1, 1);
  return;
}

void fill(int v[], int start, int val, int raising) {
  if (start >= N) return;           // end of vector
  if (!raising && val <= 0) return; // end of segment
  if (val >= L) // change direction
    return fill(v, start+1, val-1, !raising);
  v[start] = val;
  if (raising) fill(v, start+1, val+1, raising);
  else fill(v, start+1, val-1, raising);
  return;
}
```

```
43
44 void print(int v[]) {
45   // print backspace: erase display
46   for (int i = 0; i < N; i++) printf("\b\b\b");
47   // print vector
48   for (int i = 0; i < N; i++) printf("%d ", v[i]);
49   // flush the buffer
50   fflush(stdout);
51   return;
52 }
```

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 beerware[1]

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/