Esercizi su funzioni e matrici in C

Stefano Cherubin*

17/11/2016

[**Informatica A**] Esercitazione #10

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

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

Indice

1 Tappeto elastico 2.0

Sulla stessa struttura definita per il problema presentato nell'esercitazione #8, si vuole apportare una modifica alle richieste del problema per venire incontro a un raffinamento dei requisiti del software. Viene di seguito riportato il testo originale del problema con le richieste aggiornate.

```
#define DIM 10
typedef struct Intv { unsigned int r, c; }
   Casella;
typedef Casella TappetoElastico[DIM][DIM];
```

Pare che tra le discipline degli ottocenteschi "circhi delle pulci" non ci fosse il tappeto elastico (non in quello del Prof. Heckler, almeno). Avrebbe potuto funzionare così:

> la pulce salta sulla prima cella (0,0) del tappeto elastico (quadrato), atterrandovi legge le coordinate (riga e colonna) della prossima cella su cui saltare, e da lì continua a saltare, ogni volta leggendo le coordinate della cella successiva verso cui saltare. Se/quando le coordinate lette indicano un punto esterno al tappeto, la pulce scende (tra gli applausi del pubblico).
>
> Si implementi una funzione C `ciclico` che riceve in input un `TappetoElastico` e restituisce 1 se esso obbliga a saltare indefinitivamente, 0 se invece a un certo punto la pulce potrà scendere e, in quel caso, fornisca anche indicazione di quanti salti è stato necessario effettuare prima di saltare fuori dal tappeto.

1.1 Approccio alla soluzione

1.1.1 Restituire più di un dato al chiamante

In questo esercizio viene richiesto di fornire indicazioni relative alla ciclicità del tappeto nei valori di ritorno ma viene anche chiesto di fornire indicazioni riguardo al numero di salti compiuti. Vediamo come è possibile fornire queste indicazioni da una funzione.

Output tramite printf Una prima idea molto rude può essere quella di eseguire una printf per comunicare il risultato della computazione prima di eseguire l'istruzione di return.

Sebbene letteralmente questo approccio rispecchi i requisiti del problema, non è usanza comune nè tantomeno si può definire elegante lasciare alla funzione il compito di comunicare il risultato della sua computazione all'utente. Il risultato del calcolo dovrebbe prima passare dal chiamante perché possa valutare cosa farne. Se chi ha chiamato una tale funzione lo riterrà necessario, allora quel dato verrà mostrato all'utente.

Restituire tipi strutturati È possibile definire una struttura di comodo in modo che possa essere usata come tipo restituito.

```
typedef struct { int haCicli; int nSalti; } S;

S ciclico(TappetoElastico te) { ... }
```

Tuttavia il testo del problema richiedeva esplicitamente che il valore di ritorno fosse solamente il dato relativo alla ciclicità del tappeto. Questo non è l'approccio che la soluzione richiedeva.

Passaggio di parametri per indirizzo Il passaggio di parametri per indirizzo consente di passare ad una funzione un puntatore ad una variabile definita nel chiamante (o a un livello superiore) concedendo quindi alla funzione la possibilità di modificare un valore che resterà tale anche dopo l'istruzione di return.

Questo significa che modificando il valore puntato da un parametro puntatore, esso non svanirà al termine della funzione e può essere usato all'occorrenza per fornire un ulteriore output. La soluzione è presentata più nel dettaglio di seguito.

1.2 Soluzione C

Questa versione prende spunto dalla versione 3 della funzione `contasalti` presentata nell'esercitazione #8.

Viene aggiunto un parametro, di tipo puntatore a intero, che sarà utilizzato in caso di tappeto aciclico per memorizzare il numero di salti compiuti per uscire dal tappeto.

Listato 1: Tappeto ciclico

```
1  int ciclico(TappetoElastico te, int *nSalti)  {
2    int tec[DIM][DIM] = { 0 };
3    int jmpOff, jmpCount = 0, r = 0, c = 0;
4    do {
5      tec[r][c] = 1;
6      jmpCount++;
7      r = te[r][c].r;
8      c = te[r][c].c;
9      jmpOff = !(r >= 0 && r < DIM && c >= 0 && c < DIM)
           ;
10   } while (jmpOff == 0 && tec[r][c] == 0);
11   if (jmpOff == 1) {
12     *nSalti = 1 + jmpCount;
13     return 0;
14   }
15   return 1;
16 }
```

2 ASCII Art: il bersaglio

Scrivere una funzione che prenda in input come parametro una matrice quadrata di numeri interi e un numero intero `n`. Si riempia la matrice secondo lo schema «bersaglio», come in esempio.

Si assuma che la matrice sia stata dichiarata di dimensione `NxN`, con `#define N 50` in precedenti istruzioni.

Esempio: `n = 5`

1	1	1	1	1
1	2	2	2	1
1	2	3	2	1
1	2	2	2	1
1	1	1	1	1

2.1 Soluzione per colorazioni successive

Questa soluzione propone di riempire tutta la matrice quadrata di 1

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

...per poi restringere i limiti del quadrato e riempire con i 2...e infine con i 3

1	1	1	1	1
1	2	2	2	1
1	2	2	2	1
1	2	2	2	1
1	1	1	1	1

1	1	1	1	1
1	2	2	2	1
1	2	3	2	1
1	2	2	2	1
1	1	1	1	1

Listato 2: Bersaglio (colorazioni successive)

```
1 void bersaglio(int m[][N], int n) {
2   int i, j, k;
3   for (k = 0; k < (n + 1) / 2; k++)
4     for (i = k; i < n - k; i++)
5       for (j = k; j < n - k; j++)
6         m[i][j] = k + 1;
7   return;
8 }
```

2.2 Soluzione per simmetria

Questa soluzione propone di sfruttare la simmetria del pattern per risparmiare calcoli.

1	1	1	0	0
1	2	2	0	0
1	2	3	0	0
0	0	0	0	0
0	0	0	0	0

Questa sotto-matrice (si consideri la parte non nulla) si può riempire con la regola definita nella versione precedente limitata al solo quadrante in alto a sinistra.

```
m[i][j] = k
```

Successivamente, la parte mascherata da 0 può essere ottenuta per simmetria

```
m[i][n-1-j] = m[i][j]
m[n-1-i][j] = m[i][j]
m[n-1-i][n-1-j] = m[i][j]
```

Listato 3: Bersaglio (per simmetria)

```
1  void bersaglio(int m[][N], int n) {
2    int i, j, k;
3    for (k = 0; k < (n + 1) / 2; k++)
4      for (i = 0; i < (n + 1) / 2; i++) {
5        for (j = 0; j < (n + 1) / 2; j++) {
6          m[i][j] = k + 1
7          /* per simmetria, completo la riga */
8          m[i][n-j-1] = m[i][j];
9          /* per simmetria, completo la colonna */
10         m[n-1-i][j] = m[i][j];
11         /* completo l'ultimo quadrante */
12         m[n-1-i][n-1-j] = m[i][j];
13       }
14     }
15   return;
16 }
```

3 ASCII Art: la farfalla

Scrivere una funzione che prenda in input come parametro una matrice quadrata di numeri interi e un numero intero `n`. Si riempia la matrice secondo lo schema «farfalla», come in esempio.

Si assuma che la matrice sia stata dichiarata di dimensione `NxN`, con `#define N 50` in precedenti istruzioni.

Esempio: `n = 6`

1	0	0	0	0	1
1	1	0	0	1	1
1	1	1	1	1	1
1	1	1	1	1	1
1	1	0	0	1	1
1	0	0	0	0	1

3.1 Soluzione con condizione esatta

La soluzione consiste nello scandire la matrice e riconoscere il settore in cui ci si trova in base agli indici.

Si ricorda che i settori sono delimitati dalla diagonale principale (equazione $i = j$) e dalla diagonale secondaria (equazione $i = n - 1 - j$).

Listato 4: Farfalla

```
1  void farfalla(int m[][N], int n) {
2    int i, j;
3    for (i = 0; i < n; i++)
4      for (j = 0; j < n; j++)
5        if ((i > j && i < n - 1 - j) || (i < j && i > n
             - 1 - j))
6          m[i][j] = 0;
7        else
8          m[i][j] = 1;
9    return;
10 }
```

3.2 Soluzione per simmetria

Similmente all'esercizio precedente, è possibile definire solo un quadrante ed applicare per simmetria il pattern al resto della matrice.

Listato 5: Farfalla (per simmetria)

```
1  void farfalla(int m[][N], int n) {
2    int i, j;
3    for (i = 0; i < (n + 1) / 2; i++)
4      for (j = 0; j < (n + 1) / 2; j++)
5        if (i < j) {
6          m[i][j] = 0;
7          m[i][n-1-j] = 0;
8          m[n-1-i][j] = 0;
9          m[n-1-i][n-1-j] = 0;
10       }
11       else {
12         m[i][j] = 1;
13         m[i][n-1-j] = 1;
14         m[n-1-i][j] = 1;
15         m[n-1-i][n-1-j] = 1;
16       }
17   return;
18 }
```

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/