L'Allocazione Dinamica della Memoria

Maurizio Palesi
DIIT—Università di Catania
Viale Andrea Doria 6, 95125 Catania
mpalesi@diit.unict.it
http://www.diit.unict.it/users/mpalesi

Sommario

Questo documento tratta l'allocazione dinamica della memoria in C. Verrà evidenziato come, alcune volte, l'utilizzo di strutture dati di tipo statico non sono opportune in certi scenari applicativi. Saranno descritte in dettaglio le funzioni C per la gestione dinamica della memoria e presentati diversi esempi.

Indice

1 Introduzione

La scelta delle appropriate strutture dati è di fondamentale importanza per la risoluzione di un certo problema almeno tanto quanto un buon programma di manipolazione. Fin qui ci siamo occupati di strutture dati sia di tipo scalare (tipo intero, reale, carattere) sia di tipo strutturato (stringhe, vettori, e strutture). A ciascuna di esse abbiamo associato le relative rappresentazioni interne previste dal C. Le variabili corrispondenti ai tipi suddetti non possono mutare le loro caratteristiche in fase di esecuzione e pertanto sono dette *statiche*. Vogliamo occuparci ora di un altro tipo di variabili: quelle *dinamiche* cioè le variabili che possono essere create e/o accresciute in fase di esecuzione.

2 Perchè Allocare Dinamicamente?

Il lettore potrebbe chiedersi qual è la necessità di utilizzare l'allocazione statica piuttosto che quella dinamica. Si consideri per esempio il seguente frammento di codice per il calcolo della media degli elementi di un vettore. Una prima versione potrebbe essere:

```
#define N 10
main()
{
  int v[N], i, somma, media;

  /* inserimento dati */
  for (i=0; i<N; i++)
    {
      printf("Inserisci elemento %d-esimo: ");
      scanf("%d", &v[i]);
    }

  /* calcolo della media e visualizzazione */
  somma = 0;
  for (i=0; i<N; i++)
    somma = somma + v[i];

  media = somma/N;

  printf("La mediaè : %d", media);
}
```

Il problema in questo caso è che tale codice effettua la media esclusivamente di `N` valori. Se si volesse per esempio calcolare la media di 20 numeri piuttosto che di 10 occorrerebbe modificare `N`:

```
#define N 20
```

e ricompilare il programma. Una soluzione per per rendere il programma più flessibile sarebbe quella di chiedere all'utente di inserire il numero di elementi di cui calcolare la media e quindi utilizzare tale valore in luogo di `N`. Chiaramente una soluzione del tipo:

```
...
int num;
printf("Di quanti valori vuoi fare la media? ");
scanf("%d", &num);
int v[num];
...
```

non è corretta in C visto che la dichiarazione di un vettore richiede la conoscenza del numero di elementi a tempo di compilazione. Nel suddetto frammento di codice, infatti,

il valore della variabile `num` non è noto a tempo di compilazione ma soltanto a tempo di esecuzione (in questo caso dopo l'esecuzione della `scanf`). Nelle sottosezioni successive presentiamo due diverse soluzioni a questo problema. La prima sovradimensiona il vettore mentre la seconda fa uso dell'allocazione dinamica della memoria.

2.1 Sovradimensionamento

Una possibile soluzione potrebbe essere quella di sovradimensionare il vettore in questo modo:

```
#define MAXDIM 100
main()
{
  int v[MAXDIM], num, i, somma, media;

  printf("Di quanti valori vuoi fare la media? ");
  scanf("%d", &num);

  /* controllo se ho le risorse necessarie */
  /* per memorizzare tutti i dati */
  if (num > MAXDIM)
    printf("il vettore òpu contenere al massimo %d valori",
           MAXDIM);
  else
    {
      /* inserimento dati */
      for (i=0; i<num; i++)
        {
          printf("Inserisci elemento %d-esimo: ");
          scanf("%d", &v[i]);
        }

      /* calcolo della media e visualizzazione */
      somma = 0;
      for (i=0; i<num; i++)
        somma = somma + v[i];

      media = somma/num;

      printf("La mediaè : %d", media);
    }
}
```

Il problema di questo approccio è però lo spreco di memoria.

2.2 Allocazione Dinamica

Un approccio più efficiente dal punto di vista dell'utilizzazione della risorsa memoria sarebbe quello di utilizzare sempre un vettore formato da un numero di elementi esattamente uguale a quelli da elaborare.

```
main()
{
  int *v, num, i, somma, media;

  printf("Di quanti valori vuoi fare la media? ");
  scanf("%d", &num);

  /* alloco una àquantit di memoria necessaria */
  /* per memorizzare esattamente 'num' interi  */
  /* il puntatore v àpunter esattamente        */
  /* all'inizio di questa area                 */
  v = (int *)malloc(num * sizeof(int));

  /* inserimento dati */
  for (i=0; i<num; i++)
    {
      printf("Inserisci elemento %d–esimo: ");
      scanf("%d", &v[i]);
    }

  /* calcolo della media e visualizzazione */
  somma = 0;
  for (i=0; i<um; i++)
    somma = somma + v[i];

  media = somma/num;

  printf("La mediaè : %d", media);

  /* ora posso liberare l'area di memoria */
  /* precedentemente allocata             */
  free(v);
}
```

Le funzioni `malloc()`, `free()` e diverse altre saranno discusse nella sezione successiva.

3 Gestione Dinamica della Memoria

In questa sezione descriveremo in dettaglio le funzioni messe a disposizione dalla libreria standard C per la gestione dinamica della memoria. Le principali sono quattro: `malloc()`, `calloc()`, `free()`, `ralloc()` e sono accessibili includendo il file header `stdlib.h`:

```
#include <stdlib.h>
```

`malloc()` e `calloc()` allocano un blocco di memoria, `free()` libera un blocco di memoria precedentemente allocato e `realloc()` modifica le dimensioni di un blocco di memoria precedentemente allocato.

3.1 `calloc()` e `malloc()`

Il prototipo delle funzioni `malloc` e `calloc` è il seguente:

```
void *calloc(int num_elementi, int dim_elemento);
void *malloc(dim_totale);
```

`calloc()` alloca memoria per un vettore di `num_elementi` elementi di `dim_elemento` bytes ciascuno e restituisce un puntatore alla memoria allocata. Inoltre inizializza ogni byte di tale blocco di memoria a zero. `malloc()` alloca `dim_totale` bytes di memoria e restituisce un puntatore a tale blocco. In questo caso la memoria non è inizializzata a nessun valore.

Per determinare la dimensione in byte di un tipo di dato è possibile utilizzare la funzione `sizeof()`. Quindi, per esempio, per allocare un vettore di `n` elementi di tipo intero basta fare:

```
int *v;
v = (int *)calloc(n, sizeof(int));
```

Il puntatore `v` punterà al primo elemento di un vettore di `n` elementi interi. Si noti che è stato utilizzato il casting per assegnare il puntatore restituito da `calloc` a `v`. Infatti `calloc` restituisce un `void*` mentre `v` è un `int*`. Analogamente si può utilizzare la funzione `malloc` per ottenere lo stesso risultato:

```
int *v;
v = (int *)malloc(n * sizeof(int));
```

In questo caso occorre determinare il numero totale di bytes da allocare che è pari al numero di elementi del vettore per la dimensione in bytes del generico elemento. In questo caso poichè il vettore contiene interi, la dimensione del generico elemento è `sizeof(int)`.

Sia `calloc()` che `malloc()` restituiscono `NULL` se non riescono ad allocare la quantità di memoria richiesta.

3.2 `free()`

Il prototipo della funzione `free()` è:

```
void free(void *ptr);
```

Essa rilascia (libera o dealloca) lo spazio di memoria puntato da `ptr` il cui valore proveniva da una precedente `malloc()` o `calloc()` o `realloc()`. Se `ptr` è `NULL` nessuna operazione viene eseguita.

3.3 `realloc()`

Il prototipo della funzione `realloc()` è:

```
void *realloc(void *ptr, int dim_totale);
```

Essa modifica la dimensione di un blocco di memoria puntato da `ptr` a `dim_totale` bytes. Se `ptr` è `NULL`, l'invocazione è equivalente ad una `malloc(dim_totale)`. Se `dim_totale` è 0, l'invocazione è equivalente ad una `free(ptr)`. Naturalmente il valore di `ptr` deve provenire da una precedente invocazione di `malloc()` o `calloc()` o `realloc()`.

4 Matrici Dinamiche

Possiamo rappresentare una matrice come un vettore di vettori. Cioè l'elemento i-esimo di un vettore è un vettore che rappresenta le colonne della matrice. Per costruire una matrice dinamica di `r` righe e `c` colonne basterà eseguire le seguenti due operazioni:

1. Allocare dinamicamente un vettore di `r` puntatori (vettore delle righe).
2. Per ogni riga allocare un vettore di `c` elementi del tipo base considerato (vettore colonne).

La Figura 1 mostra la rappresentazione grafica di una matrice dinamica.

Per esempio il seguente frammento di codice definisce ed alloca una matrice dinamica di interi di `r` righe e `c` colonne.

```
{
  int **matrice, i;

  /* alloco il vettore delle righe. Ogni elemento */
  /* di questo vettoreè un puntatore              */
  matrice = (int **)malloc(r * sizeof(int *));
```

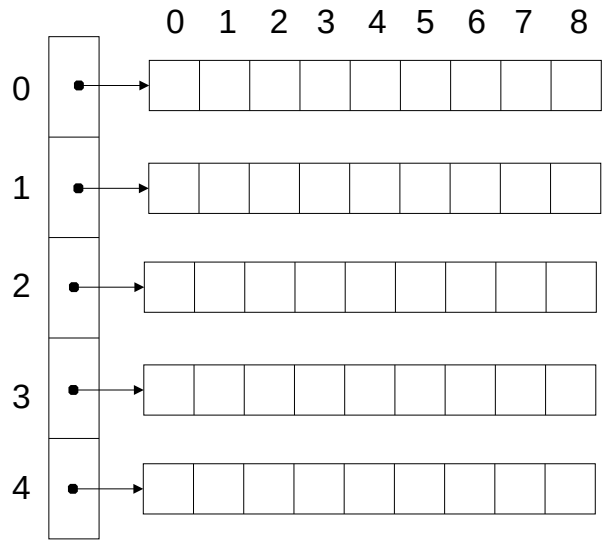

Figura 1: Rappresentazione di una matrice dinamica vista come un vettore di puntatori.

```
  /* per ogni riga alloco le colonne */
  for (i=0; i<r; i++)
    matrice[i] = (int *)malloc(c * sizeof(int));
}
```

Per disallocare una matrice dinamica, invece, occorre procedere alla rovescia: prima si disallocano tutte le colonne quindi si disalloca il vettore dei puntatori alle colonne.

```
{
  /* per ogni riga disalloco le colonne */
  for (i=0; i<r; i++)
    free(matrice[i]);

  /* disalloco il verrore delle righe */
  free(matrice);
}
```

5 Esercizi

Esercizio 5.1 ()

Calcolare la media degli elementi di una sequenza di valori double inseriti dall'utente. La fine della sequenza è identificata dal numero 0.0. Utilizzare vettori dinamici.

Risoluzione

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
  double *vettore, elemento, somma, media;
  int    i, j, fine;

  vettore = NULL;

  /* inserimento dati */
  i = 0;
  fine = 0;
  while (!fine)
    {
      printf("Inserisci un elemento (0 per finire): ");
      scanf("%lf", &elemento);

      if (elemento == 0.0)
        fine = 1;
      else
        {
          /* aumento di 1 la dimensione del vettore */
          vettore = (double *)realloc(vettore, (i+1) * sizeof(double));
          vettore[i] = elemento;
          i++;
        }
    }

  /* calcolo e stampa della media */
  somma = 0.0;
  for (j=0; j<i; j++)
    somma += vettore[j];

  media = somma / i;
  printf("La mediaè : %lf\n", media);

  /* dellocazione del vettore */
  free(vettore);
```

}

□

Esercizio 5.2 ()

Calcolare la trasposta di una matrice di interi allocata dinamicamente.

Risoluzione

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
  int **matrice, **trasposta;
  int righe, colonne, r, c;

  printf("Quante righe ha la matrice: ");
  scanf("%d", &righe);
  printf("Quante colonne ha la matrice: ");
  scanf("%d", &colonne);

  /* allocazione della matrice */
  matrice = (int **)malloc(righe * sizeof(int *));
  for (r=0; r<righe; r++)
    matrice[r] = (int *)malloc(colonne * sizeof(int));

  /* inserimento dati */
  for (r=0; r<righe; r++)
    for (c=0; c<colonne; c++)
      {
        printf("Inserisci elemento di riga %d e colonna %d: ", r, c);
        scanf("%d", &matrice[r][c]);
      }


  /* alloco la matrice trasposta */
  trasposta = (int **)malloc(colonne * sizeof(int *));
  for (c=0; c<colonne; c++)
    trasposta[c] = (int *)malloc(righe * sizeof(int));
```

```
  /* calcolo della trasposta */
  for (r=0; r<righe; r++)
    for (c=0; c<colonne; c++)
      trasposta[c][r] = matrice[r][c];

  /* stampo la trasposta */
  printf("La traspostaè :\n");
  for (c=0; c<colonne; c++)
    {
      for (r=0; r<righe; r++)
        printf("%d ", trasposta[c][r]);
      printf("\n");
    }

  /* dealloco la matrice */
  for (r=0; r<righe; r++)
    free(matrice[r]);
  free(matrice);

  /* dealloco la trasposta */
  for (c=0; c<colonne; c++)
    free(trasposta[c]);
  free(trasposta);
}
```

□

Esercizio 5.3 ()

Calcolare la trasposta di una matrice di interi allocata dinamicamente. Strutturare il programma a funzioni.

Risoluzione

```
#include <stdio.h>
#include <stdlib.h>

typedef int** TMatrice;

/*-----------------------------------------
  dichiarazione delle funzioni (prototipi)
 -----------------------------------------*/
```

```
*/
void LeggiRigheColonne(int *r, int *c);
TMatrice AllocaMatrice(int r, int c);
void Inserimento(TMatrice mat, int r, int c);
void CalcolaTrasposta(TMatrice mat, TMatrice trasp, int r, int c);
void VisualizzaMatrice(TMatrice mat, int r, int c);
void DeallocaMatrice(TMatrice *mat, int r);

/*-----------------------------------------
  programma principale
  -----------------------------------------
*/
void main()
{
  TMatrice matrice, trasposta;
  int righe, colonne, r, c;

  LeggiRigheColonne(&righe, &colonne);

  matrice = AllocaMatrice(righe, colonne);

  Inserimento(matrice, righe, colonne);

  printf("Hai inserito la matrice:\n");
  VisualizzaMatrice(matrice, righe, colonne);

  trasposta = AllocaMatrice(colonne, righe);

  CalcolaTrasposta(matrice, trasposta, righe, colonne);

  printf("La traspostaè :\n");
  VisualizzaMatrice(trasposta, colonne, righe);

  DeallocaMatrice(&matrice, righe);
  DeallocaMatrice(&trasposta, colonne);
}

/*-----------------------------------------
  definizione delle funzioni
  -----------------------------------------
*/
```

```
void LeggiRigheColonne(int *r, int *c)
{
  printf("Quante righe ha la matrice: ");
  scanf("%d", r);
  printf("Quante colonne ha la matrice: ");
  scanf("%d", c);
}

TMatrice AllocaMatrice(int r, int c)
{
  TMatrice mat;
  int i;

  mat = (TMatrice)malloc(r * sizeof(int *));
  for (i=0; i<r; i++)
    mat[i] = (int *)malloc(c * sizeof(int));

  return mat;
}

void Inserimento(TMatrice mat, int r, int c)
{
  int i, j;

  for (i=0; i<r; i++)
    for (j=0; j<c; j++)
      {
        printf("Inserisci elemento di riga %d e colonna %d: ", i, j);
        scanf("%d", &mat[i][j]);
      }
}

void CalcolaTrasposta(TMatrice mat, TMatrice trasp, int r, int c)
{
  int i, j;

  for (i=0; i<r; i++)
    for (j=0; j<c; j++)
      trasp[j][i] = mat[i][j];
}
```

```
void VisualizzaMatrice(TMatrice mat, int r, int c)
{
  int i, j;

  for (i=0; i<r; i++)
    {
      for (j=0; j<c; j++)
        printf("%d ", mat[i][j]);
      printf("\n");
    }
}

void DeallocaMatrice(TMatrice *mat, int r)
{
  int i;

  for (i=0; i<r; i++)
    free((*mat)[i]);
  free(*mat);
}
```

□