Soluzioni degli esercizi sulla gestione della memoria

Maggio 2022

Esercizio 1

Saranno 0, 1 e 0 rispettivamente. Quando OGG1 è puntato da p, il suo contatore è incrementato e diventa 1. Quando OGG2 è puntato da p.next, il suo contatore è incrementato e diventa 1. Quando OGG3 è puntato da q, il suo contatore è incrementato e diventa 1. Quando OGG2 è puntato da q.next, il suo contatore è incrementato e diventa 2. Quando un puntatore a OGG2 è restituito come risultato (in seguito sarà puntato da r), il contatore di OGG2 è incrementato e diventa 3. Prima che p venga distrutto, il contatore di OGG2 (non più puntato da p.next) è decrementato e diventa 2. Quando p è distrutto, il contatore di OGG1 è decrementato e diventa 0. Prima che q venga distrutto, il contatore di OGG2 (non più puntato da q.next) è decrementato e diventa 1. Quando q è distrutto, il contatore di OGG3 è decrementato e diventa 0.

Esercizio 2

Domanda 1

A ciascuna costruzione di oggetto seguono degli inserimenti in coda: OGG1 inizialmente è [0, 1, 2, 3], OGG2 inizialmente è [100, 200] e OGG3 inizialmente è [10, 20]. L'invocazione alla sottoprocedura aggiorna OGG1 (sostituendo 3 con 40) e fa sì che h punti invece a OGG3, in cui 10 è sostituito con 30. Quindi la stampa produce:

```
[0, 1, 2, 40]
[30, 20]
```

Domanda 2

Durante ogni costruzione, il corrispettivo contatore è inizializzato a 0. Appena 0GG1 è puntato da l, il suo contatore è incrementato e diventa 1. Appena OGG2 è puntato da h, il suo contatore è incrementato e diventa 1. Siccome f è definito ma non invocato fino alla sua chiamata, il suo codice non viene eseguito e quindi non modifica alcun contatore.

Esercizio 3

```
C bar = new C(); // OGG2.ref = 1
{
  C foo = new C(); // OGG1.ref = 1, OGG2.ref = 1
  C fie = foo; // OGG1.ref = 2, OGG2.ref = 1
  bar = fie; // OGG1.ref = 3, OGG2.ref = 0
} // OGG1.ref = 1, OGG2.ref = 0
```

`OGG1.ref-c` vale 1, mentre `OGG2.ref-c` vale 0. Solo `OGG2.ref-c` può quindi essere restituito alla lista libera, perché solo il suo contatore è stato azzerato.

Esercizio 4

```
int *p = malloc(sizeof(int)), q = p;
*q = 0;
free(p);
printf("%d\n", *q); // whoops
```

Anche se dopo aver liberato la memoria puntata da p ci guardiamo bene dal dereferenziare p, erroneamente dereferenziamo q, che punta alla stessa zona di memoria deallocata.

Esercizio 5

```
class C {
  int n;
  C next;
}

C foo (){
  C p = new C(); // OGG1.ref = 1
  p.next = new C(); // OGG2.ref = 1
  C q = new C(); // OGG3.ref = 1
  q.next = p.next; // OGG2.ref = 2
  return p.next; // OGG2.ref = 3
} // OGG1.ref = 0, OGG2.ref = 1, OGG3.ref = 0

C r = foo();
```

OGG1.ref = 0, OGG2.ref = 1, OGG3.ref = 0