31. Le Classi (parte 2)

Corso di Informatica

Corso di Laurea in Matematica (D.M. 270/04) - A.A. 2020/2021
Angelo Cardellicchio
angelo.cardellicchio@uniba.it
16/12/2020

Outline

- Copiare un oggetto
- Allocazione dinamica di nuovi oggetti

Copiare un oggetto (1)

- La copia di un oggetto avviene mediante un apposito **costruttore di copia**
- Questo viene normalmente generato in maniera automatica dal compilatore, ma può (e deve, quando ci sono variabili allocate dinamicamente) essere gestito dallo sviluppatore
- Un costruttore di copia accetta necessariamente almeno un parametro, ovvero una reference ad un'istanza della classe

```
// vettore.h
Vettore(const Vettore& altro);
// vettore.cpp
Vettore::Vettore(const Vettore& altro)
{
    setModulo(altro.modulo);
    setAngolo(altro.angolo);
}
```

Copiare un oggetto (2)

- La parola chiave **const** viene specificata per diversi motivi: ad esempio, vogliamo essere in grado di copiare anche istanze costanti della nostra classe, e, nei fatti, non ah molto senso modificare l'istanza che stiamo copiando
  - *Esistono anche motivi legati all'ambito di una costante, il cui approfondimento non sarà trattato. Per chi volesse, però, sulle dispense è disponibile un riferimento ad un articolo di blog apposito*
- La copia precedente è una **shallow copy** (copia superficiale). Ciò significa che ci stiamo limitando ad effettuare una copia per valore di ogni proprietà della nostra istanza
- La shallow copy va bene soltanto nel caso non ci siano variabili allocate dinamicamente; in caso contrario, è necessario usare una **deep copy**

Copiare un oggetto (3)

- Ad esempio:

```
class Vettore {
    // ...
    private:
        double *puntatoreModulo;
}

Vettore(const Vettore& altro)
{
    // shallow copy
    modulo = altro.modulo;
    angolo = altro.angolo;
    // deep copy
    puntatoreModulo = new double;
    if (altro.puntatoreModulo != nullptr) {
        *puntatoreModulo = *(altro.puntatoreModulo);
    }
}
```

Copiare un oggetto (4)

- Possiamo anche ridefinire l'operatore di assegnazione **=**
- Per farlo, usiamo un opportuno overload
- La definizione è analoga a quella del costruttore di copia, ad eccezione del fatto che viene restituito un riferimento all'istanza specifica (***this**)

```
Vettore v1(10.0, 45.0);
Vettore v2(15.0, 60.0);
v2 = v1;
```

```
// geometria.h
Vettore& operator=(const Vettore& altro);
// geometria.cpp
Vettore& operator=(const Vettore& altro) {
    modulo = altro.modulo;
    angolo = altro.angolo;
    return *this;
}
```

Esercizio

- Modifichiamo la classe **`Vettore`** andando ad aggiungere un costruttore di copia ed effettuando l'overloading dell'operatore di assegnazione. Verifichiamone inoltre il funzionamento.

10 minuti

Allocazione dinamica di nuovi oggetti

- Anche gli oggetti di tipo non primitivo possono essere allocati in maniera dinamica
- Per farlo, si utilizzano le parole chiave **`new`** e **`delete`**, esattamente come per i tipi primitivi
- È però importante usare uno dei costruttori della classe!

```
Vettore *v = new Vettore(10.0, 45.0);
//...
delete v;
```

- Ricordiamo che abbiamo adesso a che fare con un puntatore, per cui usiamo l'operatore freccia per accedere ai suoi membri pubblici

Domande?

42