Università della Calabria

Corso di Laurea in Ingegneria Elettronica

Programmazione Orientata agli Oggetti

A.A. 2006/2007

Classi e oggetti – Terza parte

Paolo Trunfio *

* DEIS, Università della Calabria – http://si.deis.unical.it/~trunfio

Costruttore di copia

```
#include <iostream>
using namespace std;

class MyClass {
  private:
    int *p;
  public:
    MyClass(int v);
    MyClass(const MyClass &m);
    ~MyClass();
    int getVal() const {
      return *p;
    }
};

MyClass::MyClass(int val) {
  p = new int(val);
}

MyClass::MyClass(const MyClass
                 &m) {
  int val = *(m.p);
  p = new int(val);
}

MyClass::~MyClass() {
  delete p;
}

int main() {
  MyClass a(3);
  cout << a.getVal() << endl;  // 3
  MyClass b(a);
  cout << b.getVal() << endl;  // 3
  system("pause");
}
```

Funzioni friend

```
#include <iostream>
using namespace std;

class MyClass {
  private:
    int x;
  public:
    MyClass(int x) {
      this->x = x;
    }
    friend double avg(MyClass a,
                      MyClass b);
};

double avg(MyClass a, MyClass b) {
  double somma = a.x + b.x;
  return somma/2;
}

int main() {
  MyClass a(3);
  MyClass b(4);
  cout << avg(a, b) << endl;  // 3.5
  system("pause");
}
```

Nota: La funzione *avg* non è membro della classe *MyClass* ma ha accesso ai suoi membri privati.

Overload degli operatori

Il meccanismo dell'overload degli operatori consente di definire il significato assunto da un operatore (per esempio: **+**, **-**, **>>**) sugli oggetti di una particolare classe.

In tal modo è possibile utilizzare con gli oggetti gli operatori che sono normalmente utilizzati per i tipi predefiniti.

Per sottoporre ad overload un operatore è necessario definire il suo funzionamento in relazione alla classe a cui è applicato.

A tale scopo è necessario creare una funzione *operator*. Le funzioni *operator* possono essere definite come funzioni membro della classe, oppure come funzioni *friend*.

La forma generale di una funzione *operator* è la seguente:

```
tipo operator#(argomenti) {
  …
}
```

Esempio: una classe con l'operatore somma (1)

```
#include <iostream>
using namespace std;

class Complesso {  // Numeri complessi della forma a + jb
  private:
    double a;  // coefficiente della parte reale
    double b;  // coefficiente della parte immaginaria
  public:
    Complesso(double a, double b);
    double getCoeffReale() const { return a; }
    double getCoeffImmag() const { return b; }
    Complesso operator+(const Complesso &n) const;
};

Complesso::Complesso(double a, double b) {
  this->a = a;
  this->b = b;
}
```


Esempio: una classe con l'operatore somma (2)

```
// definizione dell'operatore somma
Complesso Complesso::operator+(const Complesso &n) const {
  return Complesso (a+n.a, b+n.b);
}

int main () {
  Complesso x(3,5);
  Complesso y(1,4);
  Complesso z = x + y;
  double a = z.getCoeffReale();
  double b = z.getCoeffImmag();
  cout << a << " + j" << b << endl;  // 4 + j9
  system("pause");
}
```

Una classe Complesso (1)

```
class Complesso {
   double a;
   double b;
 public:
   Complesso(double a = 0, double b = 0);
   double getCoeffReale() const;
   double getCoeffImmag() const;

   Complesso operator+(const Complesso &n) const;
   Complesso operator-(const Complesso &n) const;
   Complesso operator*(const Complesso &n) const;
   Complesso &operator=(const Complesso &n);
   bool operator==(const Complesso &n) const;
   bool operator!=(const Complesso &n) const;

   friend ostream &operator<<(ostream &output, const Complesso &n);
   friend istream &operator>>(istream &input, Complesso &n);
};
```


Una classe Complesso (2)

```
// costruttore
Complesso::Complesso(double a, double b) {
  this->a = a;
  this->b = b;
}

double Complesso::getCoeffReale() const {
  return a;
}

double Complesso::getCoeffImmag() const {
  return b;
}
```

Una classe Complesso (3)

```
// operatore somma
Complesso Complesso::operator+(const Complesso &n) const {
  return Complesso (a+n.a, b+n.b);
}

// operatore differenza
Complesso Complesso::operator-(const Complesso &n) const {
  return Complesso (a-n.a, b-n.b);
}

// operatore moltiplicazione
Complesso Complesso::operator*(const Complesso &n) const {
   return Complesso(a*n.a-b*n.b, b*n.a+a*n.b);
}
```


Una classe Complesso (4)

```
// operatore di assegnamento
Complesso& Complesso::operator=(const Complesso &n) {
  a = n.a;
  b = n.b;
  return *this;
}

// operatore di uguaglianza
bool Complesso::operator==(const Complesso &n) const {
   return a == n.a && b == n.b;
}

// operatore di disuguaglianza
bool Complesso::operator!=(const Complesso &n) const {
   return a != n.a || b != n.b;
}
```

Una classe Complesso (5)

```
// operatore di inserimento
ostream &operator<<(ostream &output, const Complesso &n) {
  output << n.a;
  if (n.b >= 0)
     output << "+j" << n.b;
  else
     output << "-j" << -1*n.b;
  return output;
}

// operatore di estrazione
istream &operator>>(istream &input, Complesso &n) {
  input >> n.a >> n.b;
  return input;
}
```


Una classe Complesso (6)

```
int main() {
 Complesso x;
 cout << "Valore iniziale di x = " << x << endl; // 0+j0
 cout << "Immetti i nuovi coefficienti di x: ";
 cin >> x;
 cout << "Nuovo valore di x = " << x << endl;
 Complesso y(3, 5);
 cout << "Valore di y = " << y << endl; // 3+j5
 cout << "x + y = " << x + y << endl;
 cout << "x * y = " << x * y << endl;
 Complesso z = (x + y) * (x - y);
 cout << "Valore di z = " << z << endl;
 if (x == z)
   cout << "x e' uguale a z" << endl;
 else
   cout << "x e' diverso da z" << endl;
 system ("pause");
}
```