Università di Bergamo

Reti di Telecomunicazione

Cenni sulla programmazione a oggetti con il C++

Fabio Martignon

Quando il C classico può creare problemi

- programmatori diversi che lavorano allo stesso progetto possono dare gli stessi nomi a funzioni diverse
- è necessario concordare su strutture dati comuni
- se qualche programmatore modifica anche leggermente una struttura questo può avere impatto su tutti gli altri
- il livello di astrazione dei pezzi di software rimane sempre lo stesso

Vantaggi della programmazione ad oggetti

- è possibile separare il lavoro dei programmatori dividendo il progetto mediante un'architettura logica ad alto livello:
  - insieme di oggetti logici
  - caratteristiche degli oggetti
  - attributi degli oggetti visibili dall'esterno
  - metodi di manipolazione degli oggetti

Vantaggi della programmazione ad oggetti

- una volta definiti i nomi degli oggetti non vi è più possibilità di ambiguità
- le funzioni che manipolano gli oggetti (funzioni membro) possono avere anche gli stessi nomi basta che siano riferite ad oggetti diversi
  - ad esempio la funzione `move(x,y)` può servire per spostare un oggetto grafico ed è diversa da oggetto ad oggetto pur avendo lo stesso nome

Vantaggi della programmazione ad oggetti

- le variabili che compongono un oggetto possono essere di tipo
  - private: accessibili solo alle funzioni membro
  - public: accessibili a tutti
- anche gli operatori possono essere "overloaded", ovvero possono essere associati a procedure di elaborazione diverse a seconda del tipo di oggetto (ad esempio: oggetto1+oggetto2=oggetto3)

Vantaggi della programmazione ad oggetti

- **ereditarietà: quando un oggetto viene creato può ereditare le caratteristiche di un altro oggetto e a queste aggiungerne di proprie**
- **polimorfismo: funzioni membro differenti possono avere lo stesso nome, ma essere distinte sulla base del modo con cui vengono usate o in base all'oggetto dell'albero di ereditarietà cui si riferiscono**

Dichiarazione di una classe

```
class point {
      private:
            double x; // ascissa
            double y; // ordinata
      public:
            int id;   // identificativo
      };
```

- la parola “class” definisce un tipo analogamente al tipo “struct” del C

Dichiarazione di una classe

- l'elenco delle variabili e funzioni membro è preceduto dalla parole:
  - `private`:
    - precede le variabili o funzioni membro che possono essere usate solo da funzioni membro (può essere sottinteso)
  - `public`:
    - precede le variabili o funzioni membro usabili da tutti
  - `protected`:
    - precede le variabili o funzioni membro che possono essere usate solo dalle funzioni membro di quella classe o da quelle derivate per ereditarietà

Funzioni membro

```
class point {
      private:
            double x; // ascissa
            double y; // ordinata
      public:
            int id;   // identificativo
            void move(double xx, double yy);
      };
void point::move(double xx, double yy) {
      x=xx;
      y=yy;
      }
```

prototipo della funzione

corpo della funzione

Membri static

- la parola riservata `static`, quando applicata ad un membro, indica che quel membro è condiviso da tutti gli oggetti di quella classe
- in altre parole se si usa una variabile static il suo valore può essere modificato da uno degli oggetti di quella classe e risultare modificato anche per gli altri

Funzioni e classi friend

- **anche se delle variabili o funzioni sono private è possibile consentirne l'accesso ad alcune classi o funzioni esterne mediante la dichiarazione di `friend`**

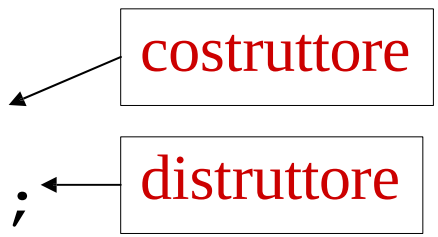
```
class point {
      private:
            double x; // ascissa
            double y; // ordinata
      public:
            friend class topology;
            int id;   // identificativo
            void move(double xx, double yy)
      };
```

Costruttori e distruttori

- costruttori e distruttori sono delle speciali funzioni, sempre public, che il programma crea automaticamente quando alloca o dealloca memoria per un oggetto
- il C++ consente di utilizzarle per eseguire delle operazioni volute dal programmatore al momento delle creazione o distruzione di un oggetto

Costruttori e distruttori: esempio

```
class triangle {
      protected:
            point* v1;// 1 vertice
            point* v2;// 2 vertice
            point* v3;// 3 vertice
      public:
            triangle();
            ~triangle();
            void set_v1(double x, double y);
            void set_v2(double x, double y);
            void set_v3(double x, double y);
      };
```



Costruttori e distruttori: esempio

```
triangle::triangle() {
          v1=new point(0,0);
          v2=new point(0,0);
          v3=new point(0,0);
          }
triangle::~triangle();
          delete v1;
          delete v2;
          delete v3;
          }
void triangle::set_v1(double x, double y) {
          v1->move(x,y);
          }
```

Ereditarietà

- dopo aver definito una classe è possibile definire nuove classi che abbiano le stesse caratteristiche della prima e che in più aggiungano altre caratteristiche
- la prima classe è detta classe base
- le classi derivate sono dette classi figlio
- passando da classe base a classi figlie, le funzioni o variabili:
  - public restano public
  - protected diventano private
  - private non vengono ereditate

Ereditarietà

```
class colored_triangle: public triangle {
     private:
          char* color;
     public:
          colored_triangle();
          ~colored_triangle();
          void set_color(char* c);
          char* get_color();
          };
void colored_triangle::set_color(char* c){
     color=c;
     }
char* colored_triangle::get_color(){
     return color;
     }
```

Funzioni con lo stesso nome

```
class point {
      private:
            double x;   // ascissa
            double y;   // ordinata
      public:
            point(double X, double Y);
            point();
            ~point(){};
      };
point::point(double X, double Y) {
      x=X;
      y=Y;
      }
point::point() {
      x=0.0;
      y=0.0;
      }
```

Funzioni membro virtual

- funzioni che nella classe base hanno la dichiarazione preceduta dalla parola chiave `virtual`
- possono essere ridefinite (nuovo corpo) nelle classi derivate mantenendo lo stesso nome:
  - `virtual void move(double x, double y);`
- una classe si definisce virtuale se ha membri virtuali
- una funzione si definisce virtuale pura o astratta se gli si assegna il valore zero:
  - `virtual void move(double x, double y)=0;`
- una classe che contiene almeno una funzione virtuale pura è detta essa stessa astratta
- si possono definire puntatori ad una classe astratta ma non possono essere creati oggetti di quel tipo

Funzioni membro virtual

```
class object {
      …
      public:
            virtual double area()=0;
       }
class triangle : public object {
      …
      public:
             virtual double area();
      };
class square : public object{
      …
      public:
             virtual double area();
      };
```

Funzioni membro virtual

- se viene usato un puntatore alla classe base astratta è poi possibile usare il puntatore per puntare ad un oggetto di una classe figlio
- se viene usata una funzione virtual viene eseguita quella della funzione figlio

```
void print_area(object *o) {
      double aa=o->area();
      printf("area oggetto = %f\n", aa);
      return;
      }
```