Corso sul linguaggio C++

Modulo 5

2 – Utilizzo di classi

Prerequisiti

- Implementazione di classi
- Funzionamento di una classe
- Parametri formali e attuali di una funzione

Introduzione

In questa Unità vediamo alcuni concetti di **OOP** in C++, come la possibilità di utilizzare array di oggetti, e di avere metodi **statici**, ossia metodi che per essere istanziati, *non richiedono un oggetto*.

Vediamo alcuni concetti, già noti dalla programmazione procedurale, come la possibilità per una funzione (metodo) di avere oggetti come parametri e come valori di ritorno.

Array di oggetti

Un **vettore di oggetti** segue le stesse regole dei vettori di variabili semplici.

ATTIVITA': creare una classe *Video* con attributi intero *colori* e *tipo* del video (mono, cga, egavga, svga). Fornire i costruttori (senza parametri e con parametri) e i metodi modificatori *setXXX*() e accessori *getXXX*() per gli attributi.

Analisi del problema

Per la classe richiesta, poniamo come attributi *colori* (intero) e *tipo* che può essere definito come:

enum tipo {mono, cga, ega, vga, svga};

Forniamo un costruttore senza parametri *Libro()* e almeno i seguenti metodi:

- *setXXX*(…) metodo modificatore per impostare l'attributo XXX;
- *getXXX*() metodo accessore per estrarre il valore dell'attributo XXX.

Array di oggetti

ATTIVITA': implementare la classe *Video*

```
enum tipo {mono, cga, ega, vga, svga};
class video
{   private:
        int colori;
        enum tipo tv;
    public:
        void setColori (int num) {colori=num;};
        int  getColori() {return colori;};
        void setTipo (enum tipo t) {tv=t;};
        enum tipo getTipo() {return tv;};
};
```

Trattandosi di metodi molto semplici, l'implementazione viene fatta nella classe.

Array di oggetti

```
int main (int argc, char *argv[])
{   string nomi[5]={"mono", "cga", "ega", "vga", "svga"};
    video monitor[5];
    monitor[0].setColori(1);
    monitor[0].setTipo(mono);
        …
    monitor[4].setColori(512);
    monitor[4].setTipo(svga);
    for (int i=0;i<5; i++)
    {     cout<<"Tipo "<<nomi[i]<<": "<<monitor[i].getColori()<<"colori";
          cout<<endl;
    }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

ATTIVITA': implementare il **main()** in modo da creare un vettore di 5 oggetti di classe *Video*, di cui immettere le caratteristiche. Stampare i dati caricati.

Array di oggetti

```
int main (int argc, char *argv[])
{   string nomi[5]={"mono", "cga", "ega", "vga", "svga"};
    video monitor[5];
    monitor[0].setColori(1);
    monitor[0].setTipo(mono);
          …
    monitor[4].setColori(512);
    monitor[4].setTipo(svga);
    for (int i=0;i<5; i++)
    {      cout<<“Tipo "<<nomi[i]<<“: "<<monitor[i].getColori()<<“colori"
           cout<<endl;
    }
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

Creazione array di oggetti di classe *Libro*

Immette valori attributo

Stampa il vettore

Membri static

Può capitare il caso in cui:

- un *attributo debba sempre avere lo stesso valore* per tutte le istanze (oggetti) di una classe (ad es. il numero di ruote di una vettura);
- un metodo (o un attributo) *sia legato alla classe anziché agli oggetti* che essa origina (come le funzioni introdotte nel Modulo 2, o i *metodi della classe Math*).

In questo caso si parla di **membri static**.

Se un membro (**attributo** o **metodo**) è definito **static**, esso ha visibilità globale, ossia è *essere condiviso tra tutti gli oggetti della classe.*

Attributi static

Gli **attributi static**:

- hanno un *valore visibile da tutti gli oggetti della classe;*
- si dicono anche **variabili di classe**;
- si dichiarano come le *variabili istanza* con la sintassi

  static *tipo ident_ variabile;*
- in genere viene inizializzato come variabile globale, all'esterno del **main()**.

Attributi static

```
class Dati
{   private:
      int dato;
      static int statico;
    public:
      Dati (int a=0) {dato=a;}
      void setDato (int d) {dato=d;}
      int getDato() {return dato;}
      void setStatico (int s) {statico=s;}
      int getStatico() {return statico;}
};
```

Dichiarazione attributo **static**

Si definiscono i rispettivi metodi *set* e *get*.

Attributi static

```
int Dati::statico=5;
int main()
{   Dati dato1, dato2;
    dato1.setDato(2);
    cout<<"Campo dato: "<<dato1.getDato()<<endl;
    cout<<"Campo statico: "<<dato1.getStatico()<<endl;
    dato2.setDato(4);
    cout<<"Campo dato: "<<dato2.getDato()<<endl;
    cout<<"Campo statico: "<<dato2.getStatico()<<endl;
    dato1.setStatico(6);
    cout<<"Campo statico: "<<dato2.getStatico()<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Si imposta il valore dell'attributo statico. Ogni oggetto avrà questo valore.

Cambiando il valore dell'attributo statico, tutti gli oggetti vedranno il cambiamento.

Metodi static

I **metodi static**

- si dicono anche **metodi di classe**;
- *non richiedono un oggetto* per essere istanziati;
- hanno interfaccia con un *numero di parametri pari al numero di oggetti su cui il metodo agisce*;
- si istanziano con la sintassi

nomeclasse.nomemetodo (*listaparametri*);

Metodi static

```
class Conto
{  private:
     string titolare;
     int codice;
     double saldoprec;
     double saldoatt;
     static double perc;
   public:
     void setDati();              // lettura dati
     double Interesse()    {return (saldoprec*perc/100.0); }
     static double Interesse(Conto& persona)
     {   return (persona.saldoprec*perc/100.0);   }
     static double SaldoAttuale(Conto& persona)
     {   persona.saldoatt = persona.saldoprec + persona. Interesse(persona);
           return persona.saldoatt;
     }
};
```

I due metodi **static** *Interesse(Conto& persona)* e *SaldoAttuale(Conto& persona)* agiscono sul parametro *persona*.

Il metodo *Interesse()* agisce sull'oggetto corrente.

Nel metodo statico l'oggetto va *passato come parametro*

Metodi static

```
double Conto::perc=10.0;  /* globale */
int main()
{   Conto persona;
    persona.setDati();
    cout<<persona.Interesse()<<endl;
    cout<<Conto::Interesse(persona)<<endl;
    cout<<Conto::SaldoAttuale(persona)<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

I metodi statici si istanziano mediante il nome della classe

I metodi *Interesse(persona)* e *SaldoAttuale(persona)* sono di tipo **static** e quindi **non agiscono sull'oggetto corrente,** ma necessita un oggetto come parametro attuale

Il riferimento Oggetto corrente

Quando si istanzia più volte una classe, si ottengono oggetti indipendenti, ciascuno con i propri membri.

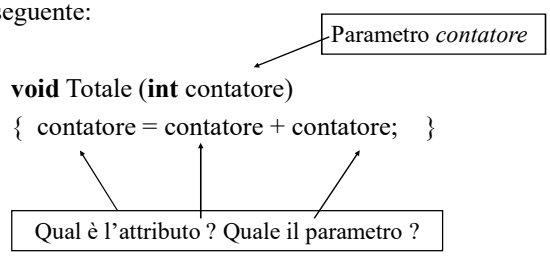
Per ciascun oggetto vengano fatte delle copie dei metodi ?

NO, in realtà il compilatore fa sì che i metodi vengano condivisi. Quando si istanzia un metodo su un oggetto viene creato un riferimento chiamato **riferimento oggetto corrente** e si indica con la parola chiave **this**.

Il riferimento Oggetto corrente

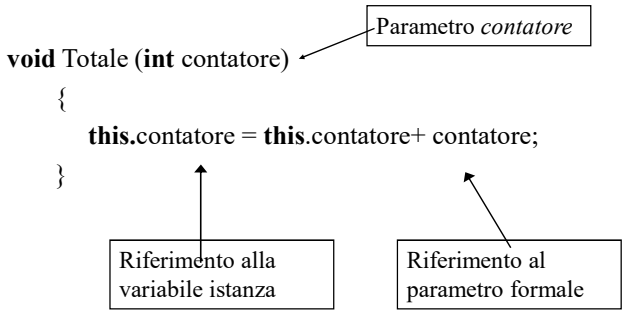
Questo riferimento è utile anche quando si vuole usare in un metodo (come parametro o come variabile locale) una variabile con lo stesso nome dell'attributo che si utilizza all'interno del metodo stesso.

Ad esempio, un metodo che debba incrementare un attributo *contatore* (variabile istanza) di un valore *contatore* (parametro), potrebbe essere il seguente:



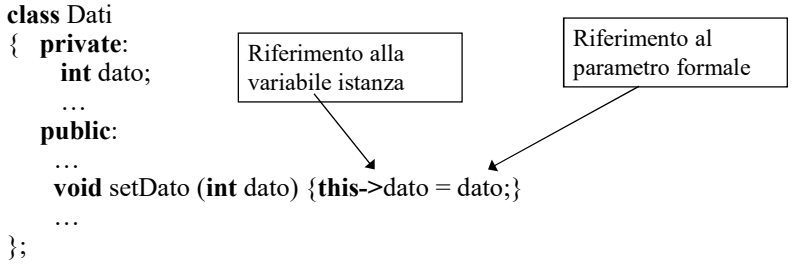
Il riferimento Oggetto corrente

La confusione tra il nome del parametro e quello della variabile istanza può essere evitata se si usa il **riferimento oggetto corrente** come segue:



Il riferimento Oggetto corrente

Ad esempio, nella classe *Dati* vista in precedenza, si poteva scrivere:



Oggetti come parametri

Come per i tipi predefiniti (**int**, **float**, ecc) anche un oggetto, può essere passato come parametro di un metodo. Nell'esempio precedente

cout<<Conto::Interesse(persona)<<**endl**;

il metodo *Interesse()* riceve come parametro un oggetto *persona* di classe *Conto*.

Consideriamo la seguente classe *Punto* e si osservi il metodo *Distanza*().

ATTIVITA': implementare la classe *Punto* con attributi *x* ed *y*, un costruttore con parametri, uno senza ed un metodo *Distanza*(Punto *p*) per calcolare la distanza tra due punti.

Oggetti come parametri

Si noti che il metodo *Distanza(Punto P)* riceve **come parametro un oggetto** di classe *Punto* e calcola la distanza tra due punti.(uno è il parametro, *l'altro è l'oggetto che istanzia il metodo)*

```
class Punto
{   private:  float x, y;
    public:
    Punto();
    Punto (float a, float b) {x=a; y=b;}
    void setX (float xf) {x=xf;}
    float getX() {return x;}
    …
    float Distanza(Punto p);
};
float Punto::Distanza(Punto p)
{   return (float) sqrt ((pow (p.x-x, 2)+pow (p.y-y, 2)));  }
```

Oggetto di classe *Punto* passato come **parametro formale**

ATTIVITA': scrivere il programma principale che dati i punti *p1(3, 4)* e *p2(0, 0)* ne calcoli e ne stampi la distanza.

Oggetti come parametri

```
int main()
{   Punto p1(3, 4);
    Punto p2(0, 0);
    cout<<p1.Distanza(p2);
    system("PAUSE");
   return EXIT_SUCCESS;
}
```

L'istanza del metodo *Distanza(....)* riceve *p2* come **parametro attuale**

Se nella classe si fosse dichiarato il metodo statico

```
static float Punto::Distanza(Punto p1, Punto p2)
   {return (float) sqrt ((pow (p1.x-p2.x, 2)+pow (p1.y-p2.y, 2)));  }
```

l'istanza sarebbe stata:

```
cout<<Punto::Distanza(p1, p2)<<endl;
```

ATTIVITA': aggiungere alla classe *Punto* il metodo *pMedio*(Punto *p*) per calcolare il punto medio tra due punti.

Oggetti come valori di ritorno

```
public Punto pMedio (Punto p)
{   Punto Ptemp;
    Ptemp.x = (p.x+x)/2;
    Ptemp.y = (p.y+y)/2;
    return Ptemp;
}
```

Come si vede *pMedio(Punto P)* riceve un **oggetto come parametro** e produce **un oggetto come valore di ritorno** che è il punto medio tra P e il punto (oggetto) che lo istanzierà nel programma principale)

In tal caso la stampa nel **main()** sarà:

```
cout<<"Il punto medio e' ";
Punto T = p1.pMedio(p2);
T.getP();              // stampa le coordinate di P
```

ATTIVITA': come si istanzia il metodo in questo caso?

Oggetti come valori di ritorno

Quindi:

- quando si implementa un metodo, il valore di ritorno può essere **void** o un **tipo qualunque** (*anche un oggetto della classe stessa* che potrebbe essere creato dal metodo);
- normalmente un metodo opera sull'**oggetto corrente**, individuato dal **puntatore "this**" Per questo motivo i metodi presentano un argomento in meno nella loro interfaccia.

Utilizzo di metodi **static**

ATTIVITA': aggiungere alla classe *Punto* il metodo

static *pMedio*(Punto *p1*, Punto *p2*) per calcolare il punto medio tra due punti.

```
static Punto pMedio (Punto p1, Punto p2)
{   Punto Ptemp;
    Ptemp.x = (p1.x + p2.x) / 2;
    Ptemp.y = (p1.y + p2.y)/2;
     return Ptemp;
}
```

Il metodo

Medio(pMedio(Punto *p1*, Punto *p2)*

può coesistere con quello non statico grazie al **polimorfismo**.

Nel programma main() si scriverà:

```
cout<<"Il punto medio e' ";
Punto p = Punto. pMedio(p1, p2);
p.getP();  // stampa coordinate del punto
```

Il metodo **static** non richiede un oggetto per essere istanziato, ma il nome della classe. *Si noti la presenza di 2 parametri.*

Argomenti

- Array di oggetti
- Membri **static**
- Attributi **static**
- Metodi **static**
- Il riferimento Oggetto corrente
- Oggetti come parametri
- Oggetti come valori di ritorno
- Utilizzo di metodi **static**

Altre fonti di informazione

- P.Gallo, F.Salerno – Informatica Generale 1, ed. Minerva Italica
- M.Romagnoli, P.Ventura – Linguaggio C/C++, ed. Petrini
- M. Bigatti – Il linguaggio Java, ed HOEPLI