Le strutture

Informatica
Prof. Viglietti Francesco
a.s. 2012-13

introduzione

Una struttura è un tipo di dato costruito dal programmatore, che serve per aggregare dati di tipo diverso.

Supponiamo per es. di voler definire un nuovo tipo di dato Autovettura che possa conservare dati quali: Targa, Marca e Modello, Cilindrata, Colore, Posti ed Esente dal bollo. Le informazioni sono diverse tra loro; per es. Targa, Marca e Modello saranno di tipo stringa; Cilindrata e Posti possono essere interi; Colore potrebbe essere stringa (o tipo Color); infine Esente dal bollo potrebbe essere un bool (vero o falso).

Una singola autovettura sarà una variabile di tipo Autovettura e conterrà tutte queste informazioni insieme, come un pacchetto.

Poiché questo tipo è formato da tanti dati aggregati si dice anche dato aggregato o strutturato.

Anche gli array sono dati aggregati (o strutturati)

Targa	ZY 057 FV
Marca	Ferrari
Modello	F1-2013
Cilindrata	2400
Colore	Rosso
Posti	2
Esente dal bollo	false

In c#

Per costruire una struttura in C# si fanno 2 passi distinti: il primo definisce un nuovo tipo di dato, l'autovettura in generale. Il secondo è quello di dichiarare una (o molte) variabile di tipo Autovettura.

- *Primo Passo: costruire (dichiarare) un nuovo tipo di dato.* Nel seguente es. è mostrata una semplice struttura che descrive una Autovettura:

```
public struct Autovettura
{
  public string Targa, Marca, Modello, Colore;
  public int Cilindrata, Posti;
  public bool Esente;
}
```

Come si nota il modificatore **public** rende accessibile da qualsiasi punto del programma. Poi c'è la parola chiave **struct** che indica che voglio costruire una struttura, seguita dal nome che avrà il mio nuovo tipo di dato.

Tra le graffe si elencano i campi, ovvero le caratteristiche della struttura. Anche queste **public**. Per ogni campo è necessario indicare il tipo.

- Secondo Passo: dichiarare variabili del nuovo tipo. Nel seguente es. è mostrato come dichiarare delle variabili di tipo Autovettura:

```
Autovettura miaAuto;
Autovettura rottame = new Autovettura();
miaAuto = new Autovettura();
```

la variabile miaAuto è solo dichiarata; la variabile rottame è anche istanziata. È possibile anche istanziare la variabile separatamente dalla dichiarazione, come nel caso di miaAuto.

Accesso ai campi

Una struttura ha dei campi, ovvero le informazioni contenute in essa. (Per es. un campo Targa). In questi campi normalmente ci si può scrivere e leggere (come per le celle di un array).

- Scrittura nei campi. Nel seguente es. è mostrato come accedere in scrittura ai campi delle variabili di tipo Autovettura (NB la variabile deve essere già dichiarata e istanziata):

```
miaAuto.Targa = "XX087KB";
miaAuto.Marca = "Ford";
miaAuto.Modello = "Focus";
miaAuto.Cilindrata = 1200;
miaAuto.Colore = "Ciano";
miaAuto.Posti = 3;
miaAuto.Esente = false;
```

i valori assegnati devono essere compatibili coi tipi dei campi.

Si osserva anche che occorre prima indicare il nome della variabile e dopo il nome del campo.

Accesso ai campi 2

- Lettura dei campi. Nel seguente es. è mostrato come accedere in lettura ai campi delle variabili di tipo Autovettura :

int cil = miaAuto.Cilindrata;

string marc = miaAuto. Marca;

autoMamma.Marca = miaAuto.Marca;

rottame.Esente != miaAuto.Esente;

miaAuto.Cilindrata += 100;

miaAuto.Posti++;

anche in lettura occorre indicare prima il nome della variabile e dopo il nome del campo.

È possibile copiare dati da un campo all'altro, purché siano rispettate le regole di compatibilità.

È possibile eseguire operazioni sui dati.

È possibile eseguire assegnazioni speciali (con operazione) che leggono e scrivono insieme; per

esempio miaAuto.Cilindrata += 100; che porta la cilindrata a 1300.

Strutture annidate

È possibile comunque dichiarare campi di tipo struttura. Per esempio supponiamo di voler dichiarare una struttura Studente che disponga dei campi Nome, Classe e VotiFinali; i voti finali sono i voti nelle diverse materie di fine anno. Quindi uno studente ha tanti voti (diciamo 12) ciascuno compreso tra 1 e 10. Per es:

- Nome Giulio
- Classe 4 I INF
- Voti 7 8 5 7 6 7 6 6 7 4 7 9

Come si può osservare il campo Voti deve contenere tanti voti. Per questo potrei pensarlo come un array di interi.

In c#

- 1: dichiarare un nuovo tipo di dato. Es. come dichiarare il tipo Studente:

```
public struct Studente
{
  public string nome, classe;
  public int[] voti;
}
```

- 2: dichiarare variabili del nuovo tipo. Es. come dichiarare delle variabili di tipo Studente:

```
Studente candidato;
```

- 3: Usare variabili del nuovo tipo. Per prima cosa occorre allocare memoria:

```
candidato = new Studente();
```

a questo punto c'è memoria per i campi Nome e Classe MA NON PER I VOTI. Infatti i voti sono un array e richiedono una separata new per allocare memoria per esso. Quindi:

```
candidato.nome = "Giulio" ;
candidato.classe = "4 I INF" ;
candidato.voti = new[12] ;
candidato.voti[0] = 7 ;
candidato.voti[11] = 9 ;
```

come si vede si sono allocate 12 celle di memoria per l'array Voti candidato; poi si sono assegnati i voti solo alla prima ed all'ultima cella.

errore

Potrei pensare di risolvere subito la questione dichiarando Studente in modo da allocare 12 celle all'array Voti, così:

```
public struct Studente
{
  public string nome;
  public string classe;
  public int[] voti = new int[12];
}
```

Purtroppo NON SI PUO', e il tentativo solleva un errore. Vedremo in seguito un modo per risolvere questa situazione, ma per ora ci accontentiamo di comprendere solo come sono fatte le strutture in memoria.

Campi di tipo struttura

Se è possibile dichiarare campi di tipo struttura, come un array, allora è possibile anche annidare strutture. Per es. supponiamo che lo Studente abbia una moto; allora è sufficiente dichiarare il campo Moto dentro Studente. Ma poniamo che Moto sia una struttura che contiene le

informazioni Marca, Modello e Cilindrata. Allora potrei fare:

```
public struct Moto
{
  public string marca, modello;
  public int cilindrata;
}
public struct Studente
{
  public string nome, classe;
  public int[] voti;
  public Moto moto; //uso minuscole/maiuscole risolve conflitto di nomi.
}
```

accesso

Nel caso di campi di tipo array occorre porre le parentesi quadre dopo il nome del campo e specificare l'indice di accesso.

Nel caso di campi di tipo struttura occorre porre il punto dopo il nome del campo e specificare il nome di accesso. Per es:

Studente candidato;

candidato = new Studente();

- per gli array fare:

candidato.voti = new[12] ;

candidato.voti[0] = 7 ;

candidato.voti[11] = 9 ;

- per le strutture fare:

candidato.moto = new Moto();

candidato.moto.marca = "Piaggio";

candidato.moto.modello = "Fuoco";

candidato.moto.cilindrata = 124;

Array di strutture

Per es. se ho dichiarato un Brano come una struttura che aggrega i dati Titolo e Durata, potrei adesso pensare ad una sequenza come ad un array di Brani. Allora dovrei dichiarare prima le strutture e dopo l'array di strutture; così:

```
public struct Brano
{
  public string titolo;
  public int durata;
}
Brano[] sequenza;
```

dove ogni cella di sequenza è un Brano.

Ora proviamo ad allocare memoria per i dati aggregati. Prima alloco memoria per l'array, così:

```
sequenza = Brano[100];
```

ma ogni cella adesso non contiene nulla, fino a quando non richiedo memoria per ciascuna struttura ivi contenuta. Cioè il tentativo di accedere ai campi della struttura brano interna darebbe errore poiché non esiste ancora. Per creare memoria dovrei fare così:

```
for (int i = 0; i<sequenza.Lenght; i++)
  sequenza[i] = new Brano();
```

accesso

Nel caso di campi in celle dell'array occorre porre le parentesi quadre dopo il nome dell'array e poi specificare il nome del campo preceduto dal punto.

```
sequenza[0].Titolo = "Ave Maria";
sequenza[1].Titolo = "Dont't cry";
for (int i = 0; i<sequenza.Lenght; i++)
  sequenza[i].durata = 10*(i+1);
```

posso anche fare assegnazioni tra tipi compatibili, ma occorre molta attenzione perché OCCORRE RIFLETTERE SULLA DIFFERENZA TRA TIPI VALORE E TIPI RIFIERIMENTO. Esempio:

```
public struct Brano
{
  public string titolo;
  public int durata;
}
Brano mioBrano = new Brano();   //questo è un singolo brano
Brano[] sequenza = new Brano[99];   //questi sono tanti brani
 . . .
Sequenza[3] = mioBrano;   //assegnazione tra strutture
```

In effetti non si solleva alcun errore: mioBrano è una struttura di tipo Brano e dello stesso tipo è anche la terza cella di sequenza; quindi l'assegnazione è corretta e produce anche effetto.

esercizi

Es. 1

- Dichiarare una struttura Animale con Nome, Specie, Razza, Sesso, Età e Peso.
- Dichiarare due gatti soriani, ed assegnare opportuni valori per rappresentarli con diverso nome e sesso, ma stesso peso.

Es. 2

- Dichiarare una struttura CD con Titolo, Durata, Anno, Artista.
- Dichiarare due CD musicali, ed assegnare opportuni valori per rappresentarli di stesso autore ma diversi titoli, anni e durate.

Es. 3

- Dichiarare una struttura Film con Titolo, Durata, Anno, Regista.
- Dichiarare due Film, ed assegnare opportuni valori per rappresentare due film di quest'anno.

Es. 4

- Dichiarare una struttura Brano con Titolo, Durata.
- Dichiarare due Brani, ed assegnare opportuni valori per rappresentare due canzoni che ti piacciono.

esercizi

Es.5

- Dichiarare una struttura Brano con titolo, durata.
- Dichiarare una struttura CD con titolo, durata, anno, artista.
- Dentro CD dichiarare anche un vettore di Brani.
- Dichiarare una variabile di tipo Brano, una variabile di tipo CD ed infine allocare memoria per
- entrambe. Allocare poi memoria per ciascun Brano del CD. Scrivere i seguenti dati:
- - Il CD è «The Wall», 4712 secondi, 1979, «Pink Floyd»,
- - Il terzo brano del CD si intitola «Another Brick in the Wall» e dura 190 secondi

Es.6

- Dichiarare una struttura Cavallo con nome, sesso, età, peso, punteggio.
- Dichiarare una struttura Gara con data, partecipanti (array di cavalli).
- Dichiarare una variabile di tipo Gara ed allocarle memoria. Allocare poi memoria per ciascun
- Cavallo della Gara. Preparare il punteggio di ciascun cavallo in
- gara assegnandogli 100 punti.