Liste Code e Pile

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

introduzione

In C# le Liste sono degli oggetti che permettono di collezionare insieme tanti elementi e di eseguire operazioni sia sulla collezione che sui singoli elementi. Puoi pensare ad una lista come un vettore, che però può cambiare dimensione a seconda delle esigenze e permette di aggiungere o togliere elementi.

In C# ci sono tanti oggetti lista. Qui ne vedremo qualcuno.

LIBRERIA

Per usare una collezione occorre aggiungere una direttiva **using** di inclusione dello spazio **System.Collections**.

Innanzitutto perciò dovrai scrivere:

using System.Collections;

dopo le altre direttive using del tuo programma.

ArrayList (elenco)

È una classe semplice con cui collezionare elementi.

La dichiarazione di un oggetto ArrayList è:

ArrayList numeri;

Senza istanza, si può anche istanziarlo:

ArrayList numeri = new ArrayList();

Operazioni 1

OPERAZIONI SULLA LISTA

Per cancellare tutti gli elementi dalla lista si usa il metodo **Clear():**

numeri.Clear();

Per inserire un elemento alla fine della lista si usa il metodo **Add():**

numeri.Add(13);//inserisce 13 nella lista

numeri.Add("ciao");//inserisce la stringa ciao

Poiché la lista accetta oggetti si possono archiviare elementi diversi tra loro. Tuttavia di solito conviene usare una lista di tipi omogenei. Per esempio:

```
numeri.Clear();
for (int i = 0; i < 50; i++)
  numeri.Add(i);  // inserisce 50 numeri nella lista.
```

Per inserire un elemento in mezzo alla lista puoi usare il metodo **Insert,** che richiede di specificare il dato da inserire e la posizione; per esempio:

numeri.Insert(100, 25); //inserisce il 100 in posizione 25

Quando una lista ha degli elementi puoi esplorarli con la istruzione **foreach**, così:

```
foreach (int num in numeri)
  Console.WriteLine(num);
```

Con foreach non si possono aggiungere o rimuovere elementi dalla lista.

Foreach

È un costrutto con lo scopo di semplificare l'accesso agli elementi di una collezione. Esso fornisce un meccanismo estremamente semplice e compatto.

foreach (tipo iteratore in collezione)

istruzione;

Il costrutto funziona nel seguente modo:

1) viene dichiarata e inizializzata una variabile indice nascosta, la cui funzione è quella di puntare agli elementi della collezione;

2) per ogni iterazione, ad iteratore viene assegnato l'iesimo elemento della collezione, puntato dalla variabile indice nascosta;

3) il ciclo termina dopo che alla variabile iteratore è stato assegnato l'ultimo elemento della collezione.

La variabile iteratore assume il valore di ogni elemento della collezione.

Vi sono 3 requisiti importanti da considerare riguardo la variabile iteratore:

1) dev'essere dello stesso tipo degli elementi della collezione;

2) è proibito modificare il suo valore;

3) essendo dichiarata localmente al ciclo, il suo campo d'azione è ristretto al ciclo.

Foreach 2

Il ciclo foreach() è l'ideale quando si deve accedere a tutti gli elementi di una collezione senza la necessità di modificarli. Si consideri ad es. di voler visualizzare i nomi dei dipendenti di un'azienda memorizzati in un vettore:

```
string [] dipendenti =  new string [10];
// ... qui i nomi dei dipendenti vengono memorizzati nel vettore
foreach  (string  nome  in  dipendenti)
  Console.WriteLine(nome);
```

analogamente usando for() si avrebbe:

```
string [] dipendenti =  new   string  [10];
// ... qui i nomi dipendenti vengono memorizzati nel vettore
for  ( int  i = 0; i < 10; i++)
  Console.WriteLine(dipendenti[i]);
```

Come si vede, l'implementazione tramite foreach() risulta molto più semplice, poiché non c'è alcun bisogno di considerare indici, inizializzazioni, incrementi, condizioni e accesso a elementi; tutto il lavoro viene svolto in modo trasparente dal linguaggio.

Foreach 3

Il ciclo foreach() è in grado di trattare una matrice come se fosse un vettore, scorrendo prima tutti gli elementi della riga 0, poi quelli della riga 1, e così via, fino all'ultimo elemento dell'ultima riga.

Ad es, si consideri il problema di calcolare la somma di tutti gli elementi di una matrice:

```
double [,] m = new double [10, 5]; // ... qui la matrice riceve i propri valori
double  somma = 0;
foreach  ( double  valore  in m)
  somma = somma + valore;
Console.WriteLine("La somma è: {0}", somma);
```

La soluzione di un simile problema attraverso un ciclo for() risulta meno compatta:

```
double [,] m = new double [10, 5];// ... qui la matrice riceve i propri valori
double  somma = 0;
for (int  r = 0; r < 10; r++)
  for (int  c = 0; c < 5; c++)
    somma = somma + m[r,c];
Console.WriteLine("La somma è: {0}", somma);
```

Operazioni 2

Per accedere ad un singolo elemento si usa la notazione vettoriale con un indice tra parentesi quadre; ad esempio, le seguenti istruzioni:

```
int i = numeri[1]; //assegna ad i il secondo elemento della lista
numeri[0] = 17; //assegna 17 al primo elemento della lista
//non si può accedere ad un elemento che non esiste neanche in scrittura
numeri[10] = 23; //errore: l'elemento 10 non esiste
int j = numeri[11]; //errore: l'elemento 11 non esiste
```

Per sapere quanti elementi ci sono nella lista si usa la proprietà **Count:**

```
static ArrayList colori = new ArrayList(); //dichiarazione e istanziazione
colori.Clear();  //cancellazione lista
colori.Add("rosso");  // aggiunta colori
colori.Add("verde");
colori.Add("azzurro");
int quanti = colori.Count; //assegna 3 alla variabile quanti
Console.WriteLine(quanti);
Console.ReadLine();
```

Operazioni 3

Per rimuovere un elemento dalla lista si usa **Remove**. Per es.:

```
static ArrayList colori = new ArrayList();
colori.Clear();
colori.Add("rosso");
colori.Add("verde");
colori.Add("azzurro"); //qui hai tre elementi in lista
colori.Remove("verde");  //qui ne restano due
foreach (string x in colori)
    Console.WriteLine(x);
Console.ReadLine();
```

il programma prepara una lista con tre elementi; poi però ne rimuove uno. Alla fine stampa a video solo rosso e azzurro, mentre il verde è stato rimosso.

Operazioni 4

Per rimuovere un elemento dalla lista si può anche usare **RemoveAt**, che rimuove un elemento in base al suo indice. Es.:

```
static ArrayList colori = new ArrayList();
colori.Clear();
colori.Add("rosso");
colori.Add("verde");
colori.Add("azzurro"); //qui hai tre elementi in lista
colori.RemoveAt(1);  //qui ne restano due
foreach (string x in colori)
    Console.WriteLine(x);
Console.ReadLine();
```

Alla fine stampa a video solo rosso e azzurro, mentre il verde (che era l'elemento numero 1) è stato rimosso. Osserva che il rosso è l'elemento numero zero.

Esercizi

1. Prepara un programma visuale con un pulsante che memorizza in una lista dei nomi presi da una textBox; con un altro pulsante visualizza a video gli elementi nella lista; un terzo pulsante elimina dalla lista un nome digitato nella textBox; infine un quarto pulsante permette di eliminare un nome in base alla sua posizione
2. Prepara un programma visuale; dichiara tre ArrayList A, B e C. Il primo pulsante genera casualmente numeri positivi nella lista A, negativi nella B e ripulisce la lista C. Il secondo pulsante copia tutti gli elementi di A e B nella lista C; il secondo pulsante cambia di segno i numeri pari delle liste A e B; il terzo rimuove dalla lista C tutti gli elementi negativi
3. Di sicuro ti sei chiesto se sia possibile costruire una lista di strutture; prova a dichiarare una struttura Studente con i campi Nome, Età, Classe; prova poi a costruire un programma (possibilmente visuale) che prende in ingresso i dati di uno studente, prepara una struttura adatta e lo inserisce nella lista; prova anche a visualizzare l'elenco degli studenti esistenti

La coda (queue)

Una coda è una struttura che gestisce i dati con politica FIFO. È la tipica coda che si fa in ordine di arrivo, come per esempio le code coi numeretti della posta o del supermercato, per essere serviti nell'ordine di arrivo. Anche Queue è una collezione di dati.

La dichiarazione di un oggetto Queue è:

```
Queue pazienti = new Queue();
Queue numeri = new Queue();
```

Operazioni 1

Per cancellare un elemento dalla coda si usa il metodo **Clear():**

numeri.Clear();

Per inserire un elemento alla fine della coda si usa il metodo **Enqueue():**

numeri.Enqueue(23); //inserisce 23 in coda

Per sapere quanti elementi ci sono in coda si usa la proprietà **Count:**

```
int quanti = numeri.Count;
Console.WriteLine(quanti);
```

Per esplorare la coda usa l'istruzione foreach:

```
foreach (int e in numeri)
  Console.WriteLine(e);
```

Per eliminare un elemento dall'inizio della coda si usa il metodo **Dequeue();** Dequeue restituisce il dato estratto dalla coda (che diventa più piccola) ma il dato restituito può essere sprecato (se non serve):

```
numeri.Dequeue();//lo rimuove ma perdi il dato
int a = (int)numeri.Dequeue();//lo rimuove e il dato va in a
```

Attenzione: è un ERRORE tentare di accedere direttamente ad un elemento in coda

```
numeri[0] = 7; //solleva un errore !!!
int z = numeri[0]; //solleva un errore !!!
```

Esercizi

1. Prepara un programma visuale con un pulsante che memorizza in una coda dei nomi presi da una textBox; un pulsante mostra il numero di elementi in coda; un terzo pulsante elimina dalla coda il prossimo nome e lo mostra in una etichetta
2. Di sicuro ti sei chiesto se sia possibile costruire una coda di strutture; prova a dichiarare una struttura Cliente con i campi Nome, Età, Richiesta; il programma deve prendere in ingresso i dati di un cliente, costruire una struttura e metterlo in coda (ad es. "Gigi", 23, "Informazioni"); un altro pulsante invece estrae dalla coda il cliente e lo visualizza in una casella di testo
3. Forse sai che al Pronto Soccorso i pazienti sono classificati con un colore: rosso (urgentissimo), giallo (grave), verde (moderato), bianco (lieve). Funziona così: quando una persona arriva al pronto soccorso l'infermiere lo registra (nome e età) e decide anche il colore di priorità; appena un medico è libero deve gestire prima quelli con codice rosso, poi giallo, poi verde e poi bianco. Dichiara una struttura Paziente con i campi Nome, Età, Colore; il programma deve gestire 4 code diverse e inserire il paziente nella lista corretta; quando un medico è libero preme il pulsante e il programma fornisce subito il paziente da servire (quindi sceglie il primo dei pazienti codice verde solo se non ci sono codici rosso e giallo). Le informazioni vanno visualizzate in una etichetta; i dati prelevati da caselle di testo

Pila (Stack)

Una pila è una struttura che gestisce i dati con politica LIFO. Pensa ad una pila di DVD (in una campana col perno) dove inserire e togliere i DVD. Quando ne inserisci uno, inevitabilmente lo poni sopra e copri quelli sotto; quando devi estrarne uno, prelevi sempre quello di sopra. Non è possibile inserirlo o prelevarlo dal mezzo. La dichiarazione di un oggetto Stack è:

Stack dvd = new Stack();

Stack numeri = new Stack();

Operazioni

Per cancellare qualsiasi elemento dalla pila siusa il metodo **Clear**():

numeri.Clear();

Per inserire un elemento alla fine della pila si usa il metodo **Push**():

numeri.Push(13); //inserisce sempre in testa

Per sapere quanti elementi ci sono in pila si usa la proprietà **Count**:

int quanti = numeri.Count;

Console.WriteLine(quanti);

Per esplorare la pila si usa l'istruzione foreach:

```
foreach (int e in numeri)
  Console.WriteLine(e);
```

Per eliminare un elemento dall'inizio della pila usa il metodo **Pop**(); Pop restituisce il dato estratto dalla pila (che diventa più piccola) ma il dato restituito può essere sprecato (se non serve):

numeri.Pop();//lo rimuove ma perdi il dato

int a = (int)numeri.Pop();//lo rimuove e il dato va in a

Attenzione: è un ERRORE tentare di accedere direttamente ad un elemento in pila

numeri[0] = 7; //solleva un errore !!!

int z = numeri[0]; //solleva un errore !!!

esercizi

1. Prepara un programma visuale con un pulsante che memorizza in una pila dei nomi presi da una textBox; un pulsante mostra il numero di elementi in pila; un terzo pulsante elimina dalla pila il prossimo nome e lo mostra in una etichetta
2. Sono ormai certo che ti sei chiesto se sia possibile costruire una pila di strutture; prova a dichiarare una struttura FILM con i campi Titolo, Anno, Genere; il programma deve prendere in ingresso i dati di un film, costruire una struttura e metterla in pila (ad es. "Leon", 1994, "Azione"); un altro pulsante invece estrae dalla pila il film e lo visualizza in una casella di testo
3. Forse sai che alle banchine del Porto i Container sono dei grandi contenitori di merci. La gru solleva i Container e li mette in pila uno sull'altro. Poi dalla banchina li carica nelle stive delle navi. Dichiara una struttura Container coi campi Codice, Peso e Tara; poi prepara un programma che carica in una pila i dati dei Container prelevati da caselle di testo; un altro pulsante invece toglie dalla pila i Container e mostra a video i dati

La lista tipizzata

La lista tipizzata rappresenta un elenco di oggetti fortemente tipizzato accessibile per indice. Fornisce metodi per la ricerca, l'ordinamento e la modifica degli elenchi. La classe List<T> è l'equivalente generico della classe ArrayList.

Per utilizzare la classe List<T> o la classe ArrayList, che dispongono di funzionalità simili, ricorda che List<T> è una classe indipendente dai tipi con prestazioni migliori nella maggior parte dei casi. Se per il tipo T della classe List<T> si utilizza un tipo di riferimento (per es. oggetti o componenti), il comportamento delle due classi è identico. Tuttavia, se per il tipo T si utilizza un tipo di valore (per es. un tipo elementare come int o bool, o una struttura), conviene prendere in considerazione i problemi relativi all'implementazione e alla conversione boxing dell'ArrayList (in pratica la lista tipizzata costringe a un uso corretto dei tipi, evita di mischiare tipi diversi, evita di convertire gli elementi nel tipo desiderato e migliora le prestazioni in memoria). Si osservi che le stringhe, pur essendo un tipo riferimento, vanno considerate tipo valore ai fini della conversione e dell'accesso al List<>.

La dichiarazione di un oggetto List<T> è simile alla seguente:

```
List<string> studenti = new List<string>();//lista di stringhe
List<double> pagamenti = new List<double>();//lista di decimali
//dichiarazione di struttura
public struct Paziente {
        public string nome;
        public int età;
        public string codice;
    }
List<Paziente> listaAttesa = new List<Paziente>(); //dichiarazione di lista di strutture
```

Operazioni

Per cancellare qualsiasi elemento dalla lista usa il metodo **Clear():**

```
listaAttesa.Clear();
```

Per inserire un elemento alla fine della lista usa il metodo **Add():**

```
Paziente x ;
x.nome="gigi";
x.età= 17;
x.codice= "rosso
listaAttesa.Add(x); //inserisce un paziente in listaAttesa
for (int i = 0; i < 21; i++)
{
  x.età++;
  listaAttesa.Add(x); //inserisce pazienti in listaAttesa
}
```

Per inserire un elemento in mezzo alla lista puoi usare il metodo **Insert**, che richiede di specificare il dato da inserire e la posizione; per esempio:

```
listaAttesa.Insert(0, x); //inserisce paziente all'inizio
```

Quando una lista ha degli elementi puoi esplorarli con la istruzione foreach, così:

```
foreach (Paziente p in listaAttesa)
  Console.WriteLine(p.nome + " " + p.età + " " + p.codice);
```

Se usi foreach non puoi aggiungere o rimuovere elementi dalla lista, o avrai un errore.

Operazioni 2

Per accedere ad un singolo elemento è possibile usare la notazione vettoriale con un indice tra parentesi quadre; ricorda che però è un trucco, reso possibile da un metodo nascosto e non è un accesso in memoria. Comunque puoi usare, ad esempio, le seguenti istruzioni:

Paziente prossimo = listaAttesa[0];

listaAttesa[1] = prossimo;

Per sapere quanti elementi ci sono nella lista puoi usare la proprietà **Count**:

int quanti = listaAttesa.Count; // ottiene il numero di elementi

Console.WriteLine(quanti);

Se vuoi rimuovere un elemento dalla lista puoi usare **Remove.** Per esempio:

listaAttesa.Remove(prossimo); //rimuove il primo Paziente concordante

Il metodo cerca il primo elemento che è uguale a quello proposto; se lo trova lo rimuove dalla lista. Gli altri elementi vengono scalati nella lista, cioè non ci sono "buchi" ma sempre elementi consecutivi, a partire dallo zero.

Per rimuovere un elemento dalla lista puoi anche usare **RemoveAt**, che rimuove un elemento in base al suo indice. Per esempio:

listaAttesa.RemoveAt(2); //rimuove il paziente 2 (il terzo)

Il primo paziente è sempre il numero zero. Anche in questo caso gli elementi sono scalati.

esercizi

1. Prepara un programma visuale per memorizzare le interrogazioni, ciascuna delle quali è composta da data, materia e voto; materia e voto sono scelte da rispettivi combo Box; un pulsante memorizza in una lista una interrogazione; un pulsante calcola la media dei voti in un materia scelta in una comboBox; un pulsante elimina dalla lista tutte le interrogazioni insufficienti; infine un pulsante elimina l'interrogazione zero
2. Prepara un programma per memorizzare i dati di partite di calcio; ogni partita è formata da due squadre (stringhe) e dai gol di ciascuna squadra; un pulsante aggiunge una partita (dati prelevati da 4 comboBox); un pulsante calcola i gol segnati dalla squadra del comboBox1; un pulsante elimina tutte le partite pareggiate; un pulsante calcola e mostra in una listBox la classifica delle squadre; Nota: nei primi due comboBox metti le stesse 6 squadre; quando aggiungi una partita verifica che una squadra non giochi contro sé stessa; nei due combo estanti proponi gol in numero da 0 a 10 (massimo);