ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI

C#

CORSO DI PROGRAMMAZIONE

STRUTTURE DATI DINAMICHE: LISTE

DISPENSA 18.01

18-01_DatiDinamici_[07]

Versione del: 07/11/2022
Revisione numero: 31

Prof. Andrea Zoccheddu
Dipartimento di Informatica

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

LE STRUTTURE DINAMICHE

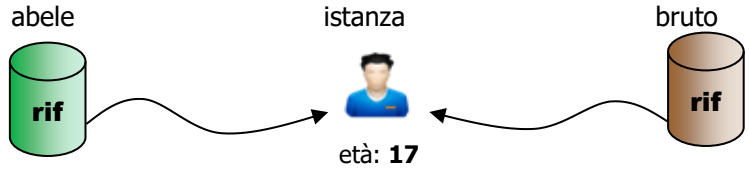
CLASSI E RIFERIMENTI

COSA SI INTENDE PER TIPO RIFERIMENTO?

Come si è osservato, discutendo degli oggetti, emerge che gli oggetti sono tipi riferimento, per cui la variabile non contiene direttamente il valore ma solo un puntatore verso l'istanza concreta. Questa situazione influenza il trasferimento dei dati tra locazioni di tipo oggetto. Esaminiamo il caso più semplice: l'assegnazione tra variabili di tipo oggetto:

VC#
```
Persona abele = new Persona ();     //abele punta a una persona appena creata
abele.età = 17;                     //la persona ha 17 anni
Persona bruto;                      //bruto non punta ad alcuna persona (nil)
```

VC#
```
bruto = abele;                      //bruto condivide la persona di abele
```



Nell'esempio qui sopra appare evidente che l'effetto dell'assegnazione è di far puntare le due variabili alla stessa istanza, cioè le variabili condividono lo stesso oggetto. Qualsiasi modifica avvenga sull'oggetto (per esempio la modifica del valore di un attributo) accedendo da una delle due variabili, influenza anche l'altra variabile, con quello che si definisce un «effetto collaterale» (side effect).

VC#
```
bruto.età = 19;                     //si modifica la persona di abele
```

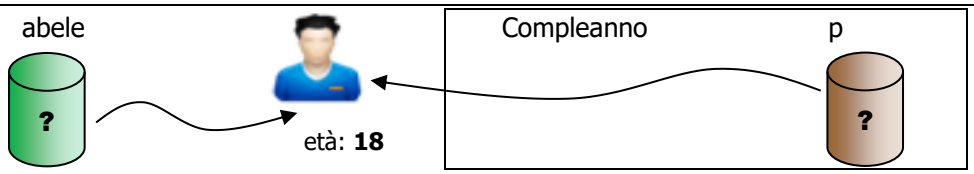
Per esempio, se attraverso la variabile bruto si modifica l'età, anche abele si troverà la istanza modificata.

Un secondo caso di condivisione è il passaggio di una variabile per parametro. Anche in questo caso se si dovesse modificare il parametro, la modifica si rifletterebbe sull'argomento. Consideriamo ad esempio la funzione seguente:

VC#
```
public void Compleanno ( Persona p )
{
      p.età++;
}
```

Consideriamo adesso una invocazione come la seguente:

VC#
```
Persona abele = new Persona ();     //abele punta a una persona appena creata
abele.età = 17;                     //la persona ha 17 anni
Compleanno ( abele );               //abele è l'argomento di Compleanno
```

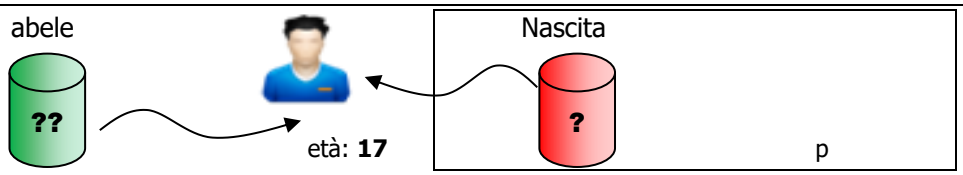


L'effetto dell'invocazione a Compleanno è di incrementare l'età dell'istanza di abele.

PASSAGGIO PER VALORE

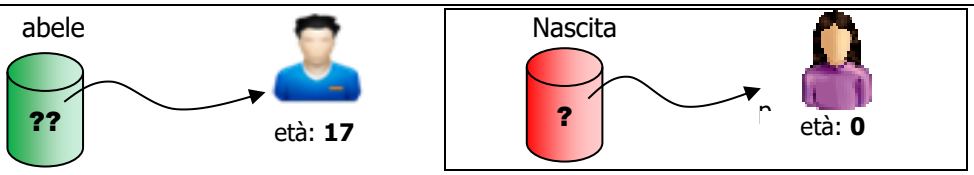
Una situazione diversa è però la creazione di una nuova istanza per il parametro. Consideriamo adesso un'invocazione come la seguente:

VC#
```
Persona abele = new Persona ();     //abele punta a una persona appena creata
abele.età = 17;                     //la persona ha 17 anni
Nascita ( abele );                  //abele è l'argomento di Nascita
```



Supponiamo che la funzione Nascita sia definita così:

VC#
```
public void Nascita ( Persona p)
{
      p = new Persona ();
      p.età = 0;
}
```



Quando la funzione nascita termina la sua esecuzione, l'istanza creata per il parametro formale p viene distrutto perché ha finito il suo scopo; la modifica dell'età con lo zero, però, resta confinato alla variabile p, pertanto la variabile abele non perde il suo valore.

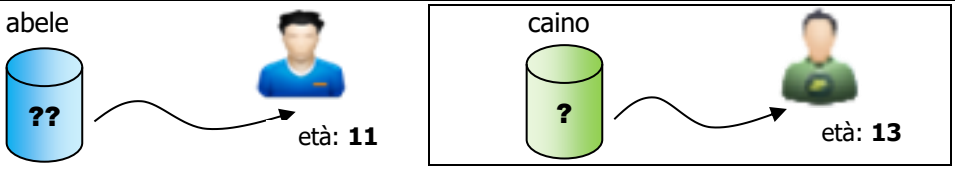
Questa situazione è valida anche per una normale assegnazione. Per esempio:

```VC#
Persona abele = new Persona ();     //abele punta a una persona appena creata
abele.età = 11;                     //abele ha 11 anni
caino = abele;
```

inizialmente abele e caino si riferiscono alla stessa Persona come nella figura seguente:

Ma quando caino deve far riferimento ad una nuova istanza, i due riferimenti differiscono come nella figura seguente:

```VC#
Persona caino = new Persona ();     //caino punta a una persona appena creata
caino.età = 13;                     //caino ha 13 anni
```



Si deve invece riflettere sui parametri passati per riferimento o per risultato; in questi casi il parametro condivide pienamente il destino dell'argomento. Questa situazione fa si che se il parametro dovesse istanziare un nuovo oggetto invocando un costruttore, anche l'argomento subirebbe lo stesso destino.

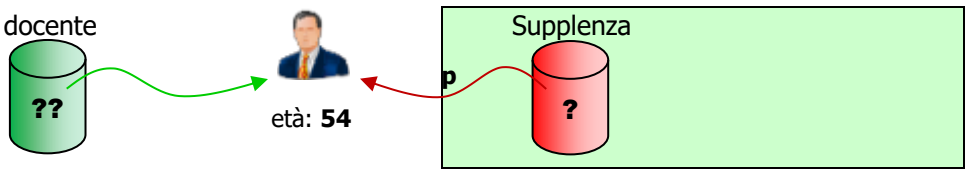
Consideriamo, per esempio, un'invocazione come la seguente:

```VC#
Persona docente = new Persona ();  //docente punta a una persona appena
creata
docente.età = 54;                  //docente ha 54 anni
Supplenza ( ref docente );         //docente è l'argomento di Supplenza
```

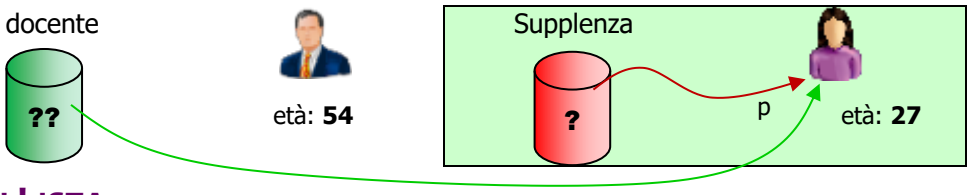
Supponiamo che la funzione Supplenza sia definita così:

```VC#
public void Supplenza ( ref Persona p)
{
      p = new Persona ();
      p.età = 27;
}
```

Al momento dell'invocazione il parametro **p** condivide l'oggetto puntato da docente:



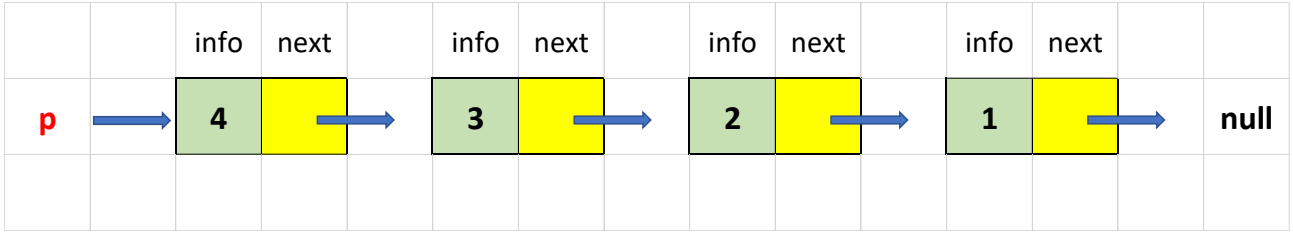
Nel momento in cui la funzione Supplenza invoca il costruttore Persona, questo nuovo oggetto sarà puntato anche da docente, come mostra la seguente figura:



CONCETTO DI LISTA

La lista è una struttura dati dinamica, ovvero che può modificare il numero dei suoi elementi (la dimensione) nel tempo. In questo si differenzia dal vettore che invece di solito conserva la stessa dimensione mentre lo si utilizza, salvo istanziarlo di nuovo (e perdere però tutti i dati).

La lista spesso viene rappresentata come una sequenza di coppie di caselle, dove ogni coppia si chiama nodo, agganciate con un riferimento (rappresentate con una freccia). La fine della lista è contraddistinta da un valore di terminazione (di solito null oppure nil). Le caselle accoppiate sono di diverso tipo: la prima casella è una informazione (spesso per semplicità è un numero intero), mentre la seconda casella è un riferimento ad un Nodo.



DICHIARAZIONE DI LISTA

In Visual C# è possibile dichiarare una lista con la seguente sintassi:

VC#
```
public class Lista
{
      public Tipo campo;
      public Lista prossimo;
}
```

Per esempio:

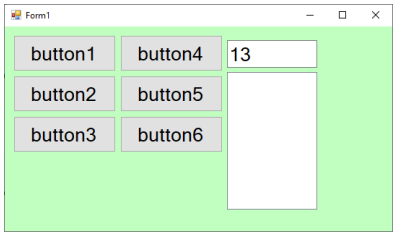
VC#
```
public class Nodo
{
      public int info;
      public Nodo next;
}
```

La definizione della lista è **intrinsecamente ricorsiva**, ovvero la struttura dati è dichiarata in termini di sé stessa.

In pratica la lista è un nodo, il quale contiene due celle, la prima cella contiene una informazione mentre la seconda cella è un altro nodo. Questa modalità prosegue fino a quando non si incontra un null.

PROGETTO GUIDATO: CREAZIONE

- Avviare Visual Studio e creare un nuovo progetto con un form simile al seguente:



- Selezionare il menu **Progetto | Aggiungi classe …** ed aggiungere una nuova classe come la seguente:

VC#
```
public class Nodo
{
    public int info;
    public Nodo next;
}
```

- Dichiarare una **lista globale** come la seguente:

VC#
```
// dichiarazione della lista
Nodo lista = null;
```

- Dichiarare il seguente metodo:

VC#
```
public Nodo Crea (int p)
{
    // creazione di una lista di p elementi
    Nodo rendi = null;
    Nodo nuovo;
    for (int k = 0; k < p; k++)
    {
        nuovo = new Nodo();
        nuovo.info = k;
        nuovo.next = rendi;
        rendi = nuovo;
    }
    return rendi;
}
```

- Associare al **button1** il seguente codice:

VC#
```
private void button1_Click(object sender, EventArgs e)
{
    // crea lista
    int n = Convert.ToInt16(textBox1.Text);
    lista = Crea(n);
    MessageBox.Show("Lista creata...");
}
```

- Dichiarare il seguente metodo:

VC#
```
public void Mostra (Nodo p)
{
    // visualizzazione in listBox1
    listBox1.Items.Clear();
    Nodo punta = p;
    while (punta != null)
    {
        listBox1.Items.Add(punta.info);
        punta = punta.next;
    }
}
```

- Associare al **button2** il seguente codice:

VC#
```
private void button2_Click(object sender, EventArgs e)
{
    // visualizzazione in listBox1
    Mostra(lista);
}
```

- Dichiarare il seguente metodo:

VC#
```
public int Conta (Nodo p)
{
    // conta il numero di elementi di p
    int cont = 0;
    Nodo punta = lista;
    while (punta != null)
    {
        cont++;
        punta = punta.next;
    }
    return cont;
}
```

- Associare al **button3** il seguente codice:

VC#
```
private void button3_Click(object sender, EventArgs e)
{
    int dim = Conta(lista);
    textBox1.Text = "" + dim;
}
```

- Dichiarare il seguente metodo:

VC#
```
public bool Cerca (int n, Nodo p)
{
    Nodo punta = lista;
    while (punta != null)
    {
        if (punta.info == n)
            return true;
        else
            punta = punta.next;
    }
    return false;
}
```

- Associare al **button4** il seguente codice:

VC#
```
private void button4_Click(object sender, EventArgs e)
{
    int checosa = Convert.ToInt32(textBox1.Text);
    bool trovato;
    trovato = Cerca (checosa, lista);
    if (trovato)
        MessageBox.Show("PRESENTE");
    else
        MessageBox.Show("ASSENTE");
}
```

- Dichiarare il seguente metodo:

VC#
```
public void AggiuntaInTesta (int n, ref Nodo p)
{
    Nodo nuovo = new Nodo();
    nuovo.info = n;
    nuovo.next = p;
    p = nuovo; // riferimento !!!
}
```

- Associare al **button5** il seguente codice:

VC#
```
private void button5_Click(object sender, EventArgs e)
{
    // AGGIUNTA IN TESTA
    int checosa = Convert.ToInt32(textBox1.Text);
    AggiuntaInTesta (checosa, ref lista);
}
```

- Dichiarare il seguente metodo:

VC#
```
public void AggiuntaInCoda (int n, ref Nodo p)
{
    // AGGIUNTA IN CODA
    Nodo nuovo = new Nodo();
    nuovo.info = n;
    nuovo.next = null;
    //----------------------
    if (p == null)
    {
        p = nuovo;
    }
    else //-----------------
    {
        Nodo temp;
        temp = p;
        while (temp.next != null)
        {
            temp = temp.next;
        }
        temp.next = nuovo;
    }
}
```

- Associare al **button6** il seguente codice:

VC#
```
private void button6_Click(object sender, EventArgs e)
{
    // AGGIUNTA IN CODA
    int checosa = Convert.ToInt32(textBox1.Text);
    AggiuntaInCoda (checosa, ref lista);
}
```

- Provare il progetto ed i vari algoritmi

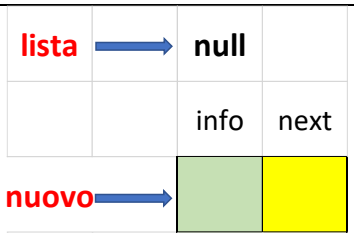
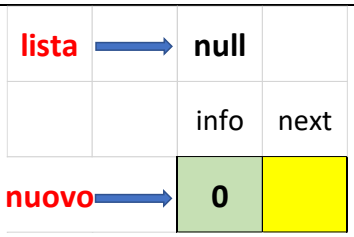
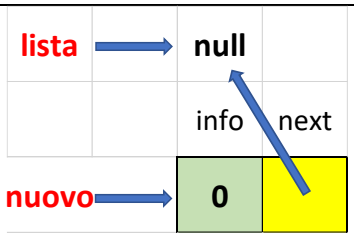
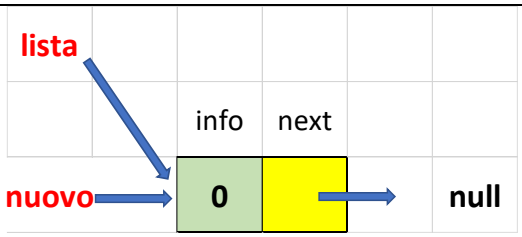
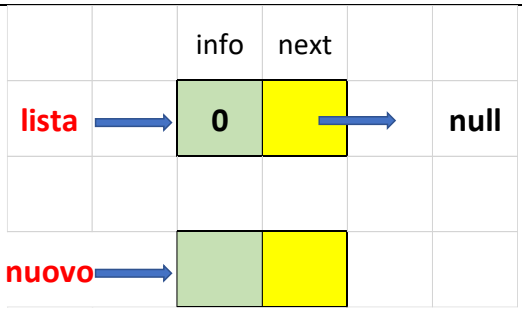
COMMENTO AL PROGETTO GUIDATO

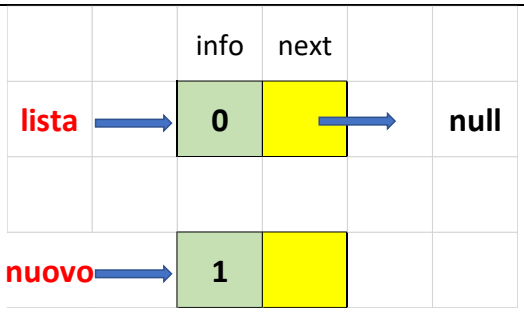
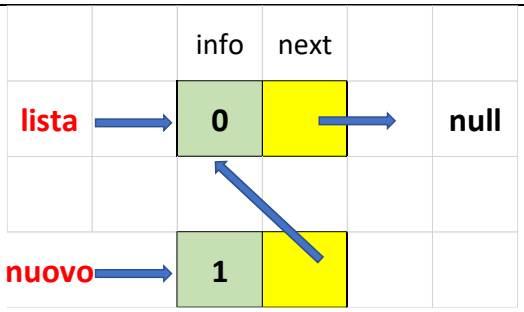
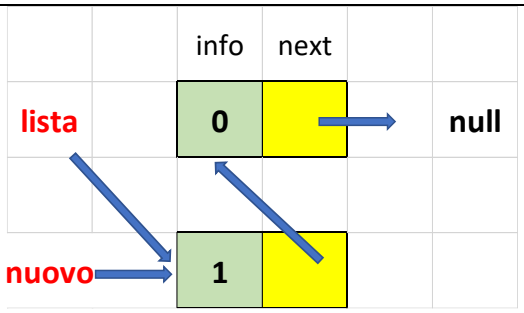
È stata dichiarata una lista globale chiamata lista che riceverà il riferimento al primo nodo della lista.

Button1 – spiegazione

Il metodo invocato da button1 crea una nuova lista di un numero prefissato di elementi. Vediamo il suo funzionamento. Ad ogni iterazione del ciclo istanza un nodo nuovo e lo prepara con un valore intero (che coincide con la variabile di iterazione, arbitrariamente); poi il campo next del nuovo nodo fa riferimento alla lista finora creata e quindi la lista si aggiorna puntando alla testa della lista.

Con una rappresentazione grafica potremmo illustrare il procedimento come segue:

prima del ciclo		La lista è inizialmente vuota
Dentro il ciclo Prima istruzione		Viene creato un nuovo Nodo
Dentro il ciclo Seconda istruzione		Il campo info del nuovo Nodo riceve il valore k
Dentro il ciclo Terza istruzione		Il campo next del nuovo Nodo riceve il valore della lista
Dentro il ciclo Quarta istruzione		La lista punta allo stesso Nodo di nuovo cioè alla lista appena creata
Dentro il ciclo Si deve iterare con k = 1 Prima istruzione		La lista resta uguale al precedente ciclo ma nuovo riceve un nodo nuovo

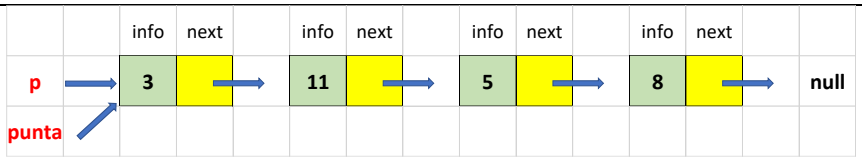
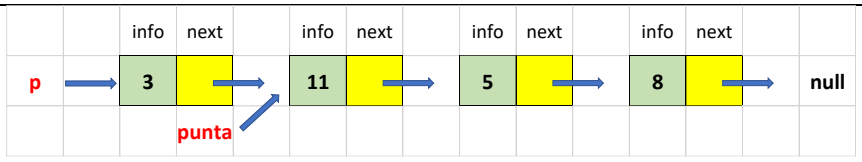
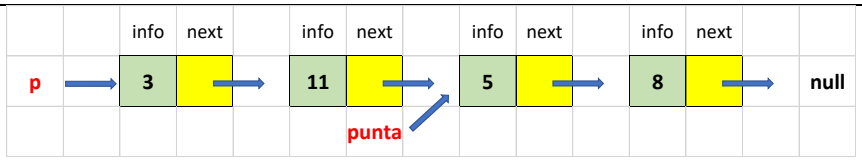
Dentro il ciclo Iterazione con k = 1 Seconda istruzione		La lista resta uguale al precedente ciclo ma nuovo ha informazione k
Dentro il ciclo Iterazione con k = 1 Terza istruzione		La lista resta uguale al precedente ciclo ma nuovo si aggancia alla lista (modifica del campo Next)
Dentro il ciclo Iterazione con k = 1 Quarta istruzione		La lista si sposta e punta il nodo nuovo

Il programma procede analogamente e crea una lista di k elementi

Button2 – spiegazione

Il metodo invocato dal button2 visualizza le informazioni di una lista. Vediamo il suo funzionamento. Ad ogni iterazione del ciclo esplora un nodo della lista.

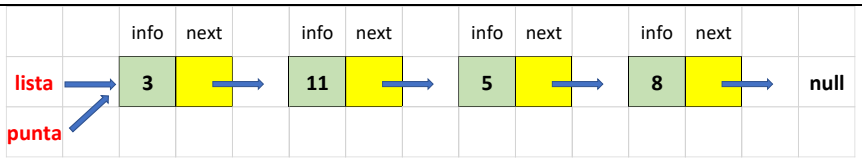
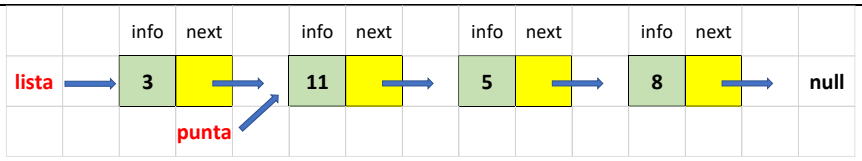
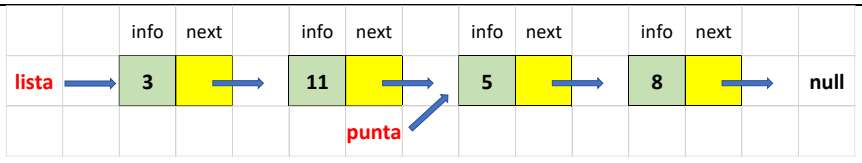
Con una rappresentazione grafica potremmo illustrare il procedimento come segue:

prima del ciclo		La lista è passata per parametro e non possiamo sapere come sia Punta fa riferimento al primo nodo di lista
Guardia del ciclo	VERA	Punta non è null
Prima iterazione *Prima istruzione*	Aggiunge al listBox1 il numero 3	
Prima iterazione *Seconda istruzione*		Punta prende il suo stesso next Cioè avanza …
Guardia del ciclo	VERA	Punta non è null
Seconda iterazione *Prima istruzione*	Aggiunge al listBox1 il numero 11	
Seconda iterazione *Seconda istruzione*		Punta prende il suo stesso next Cioè avanza …

Il programma procede analogamente e visualizza l'intera lista

Button3 – spiegazione

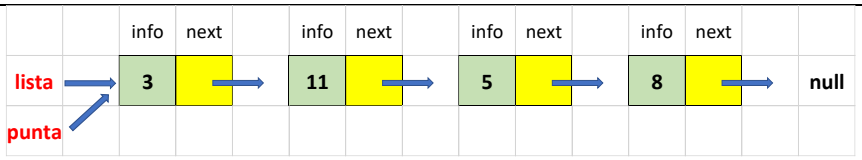
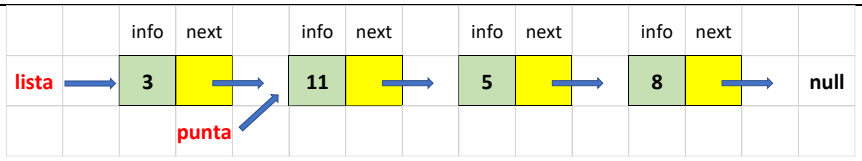
Il metodo invocato da button3 conta il numero di Nodi di una lista. Vediamo il suo funzionamento. Ad ogni iterazione del ciclo conta un nodo della lista.

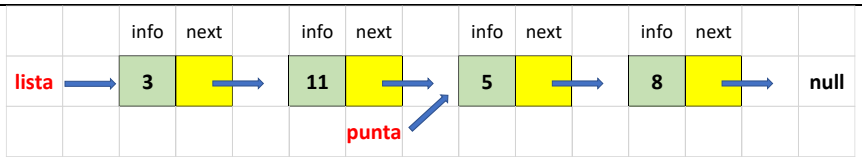
prima del ciclo	 conta ← **0**	La lista potrebbe essere vuota oppure puntare un Nodo Punta fa riferimento al primo nodo di lista
Guardia del ciclo	VERO perché punta non fa riferimento a null ma esiste almeno un Nodo	
cont++	conta ← **1**	
Avanti		punta si sposta sul nodo successivo
Guardia del ciclo	VERO perché punta non fa riferimento a null ma esiste almeno un Nodo	
cont++	conta ← **2**	
Avanti		punta si sposta sul nodo successivo

Il programma procede analogamente e conta il numero di nodi dell'intera lista

Button4 – spiegazione

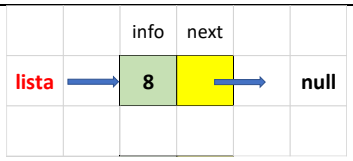
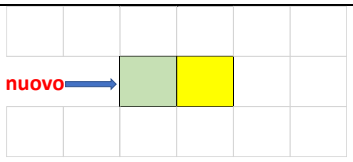
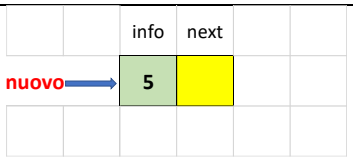
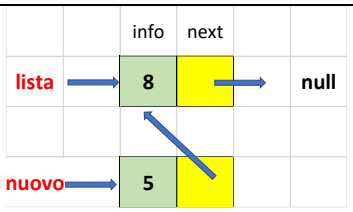
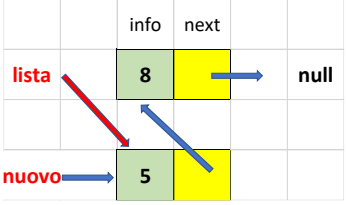
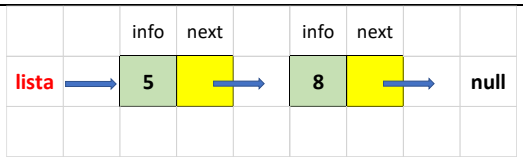
Il metodo invocato da button4 cerca se è presente un valore (intero) in qualche campo informazione dei Nodi di una lista. Vediamo il suo funzionamento. Ad ogni iterazione del ciclo controlla un nodo della lista. Supponiamo che si stia cercando il valore **5**.

prima del ciclo	 trovato ← **false** checosa ← **5**	La lista potrebbe essere vuota oppure puntare un Nodo Punta fa riferimento al primo nodo di lista
Guardia del ciclo	VERA	
Ricerca	3 non è 5 quindi non l'ho trovato	
Avanti		punta si sposta sul nodo successivo
Guardia del ciclo	VERA	
Ricerca	11 non è 5 quindi non l'ho trovato	

Avanti		punta si sposta sul nodo successivo
Guardia del ciclo	VERA	
Ricerca	**5 è uguale a 5 quindi l'ho trovato**	Esco dal ciclo

Button5 – spiegazione

Il metodo invocato da button5 aggiunge un Nodo in testa alla lista. Supponiamo per svolgere l'esempio che la lista non sia vuota, ma l'algoritmo funziona anche se la lista è vuota.

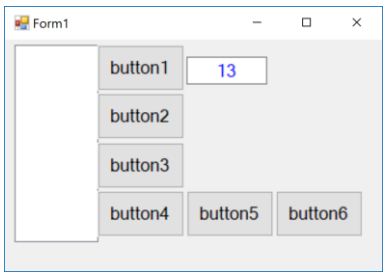
Prima istruzione	 checosa ← **5**	La lista potrebbe essere vuota oppure puntare a un Nodo
Seconda istruzione		Creazione di un nuovo nodo
Terza istruzione		Inserimento della informazione
Quarta istruzione		Aggancio il nuovo Nodo in testa alla lista
Quinta istruzione		La variabile lista adesso punta alla nuova testa della lista
Cioè		È il precedente passaggio con una grafica diversa

METODI RICORSIVI SULLE LISTE

Abbiamo visto nel paragrafo precedente che, poiché la lista in effetti è una classe, è possibile dichiarare dei metodi che accettano liste come parametri e anche metodi che restituiscono liste come tipo. I metodi visti prima erano iterativi, ovvero basati su dei cicli. Tuttavia, poiché la lista è un tipo di dato intrinsecamente ricorsivo, è possibile definire numerosi metodi di tipo ricorsivo. Vediamo alcuni esempi.

PROGETTO GUIDATO: CREAZIONE

- Avviare Visual Studio e creare un nuovo progetto con un form simile al seguente:



- Selezionare il menu **Progetto | Aggiungi classe …** ed aggiungere una nuova classe come la seguente:

VC#
```
public class Nodo
{
    public int info;
    public Nodo next;
}
```

- Dichiarare una **lista globale** come la seguente:

VC#
```
// dichiarazione della lista
Nodo lista = null;
```

- Sempre nell'ambiente globale preparare un metodo come il seguente:

VC#
```
public Nodo CreaLista (int n)
{
    if (n <= 0)
        return null;
    else
    {
        Nodo tmp = new Nodo();
        tmp.info = n;
        tmp.next = CreaLista(n - 1);
        return tmp;
    }
}
```

- Sempre nell'ambiente globale preparare un metodo come il seguente:

VC#
```
public void MostraLista (Nodo px, ListBox ax)
{
    ax.Items.Clear();
    MostraListaRicorsiva (px, ax);
}

// ----------------------------------------------

public void MostraListaRicorsiva (Nodo px, ListBox ax)
{
    if (px != null)
    {
        ax.Items.Add(px.info);
        MostraListaRicorsiva (px.next, ax);
    }
}
```

- Associa al **button1** il codice seguente:

VC#
```
private void button1_Click(object sender, EventArgs e)
{
    // creazione di una lista di 10 elementi
    Nodo lista;
    lista = CreaLista(10);
    MostraLista(lista, listBox1);
}
```

COMMENTO ALL'ESERCIZIO

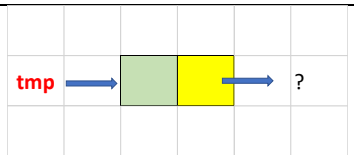
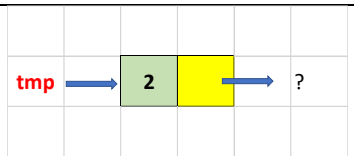
Il metodo CreaLista è un metodo ricorsivo che costruisce una lista nuova costituita da n elementi dove n è il valore intero passato come parametro.

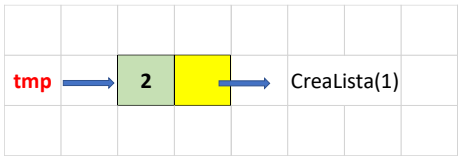
Per esempio l'invocazione

VC#
```
private void button1_Click(object sender, EventArgs e)
{
    Nodo lista = CreaLista(2);
}
```

Avrebbe il funzionamento seguente:

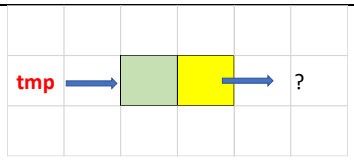
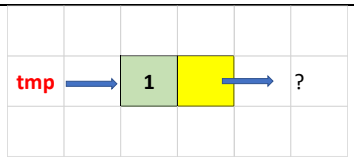
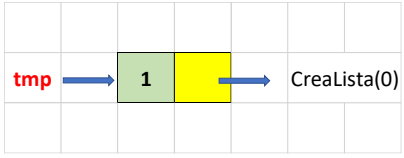
invocazione `CreaLista(2);`

condizione if	il parametro n vale 2 che non è zero quindi si esegue il ramo else
`Nodo tmp = new Nodo();`	
`tmp.info = n;`	

`tmp.next = CreaLista(n - 1);`	ricorsione: 

L'esecuzione della funzione viene congelata e si aspetta il risultato della nuova invocazione `CreaLista(1);`

invocazione `CreaLista(1);`

condizione if	il parametro n vale 1 che non è zero quindi si esegue il ramo else
`Nodo tmp = new Nodo();`	
`tmp.info = 1;`	
`tmp.next = CreaLista(n - 1);`	ricorsione: 

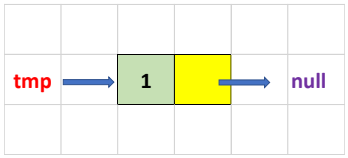
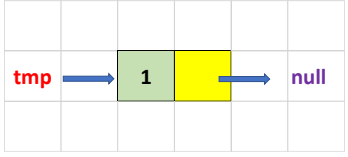
L'esecuzione della funzione viene di nuovo congelata e si aspetta il risultato della nuova invocazione `CreaLista(0);`

invocazione `CreaLista(0);`

condizione if	il parametro n vale 0 che **è zero** quindi si esegue il primo ramo
`return null;`	

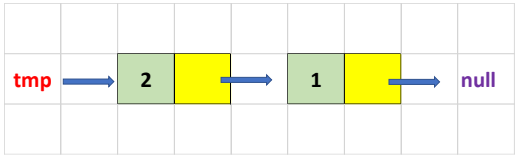
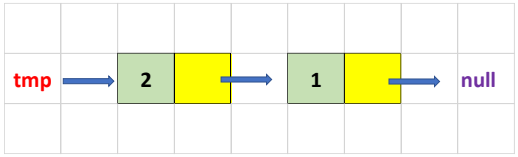
Il risultato `CreaLista(0);` che vale null, è subito restituito alla precedente invocazione, la ricorsione di `CreaLista(1);` quindi l'ambiente di questo metodo viene distrutto e si scongela l'ambiente di `CreaLista(1).`

invocazione `CreaLista(1);`

`tmp.next = CreaLista(0);`	risultato della ricorsione: 
`return tmp;`	risultato di CreaLista(1): 

Il risultato `CreaLista(1);` è una lista di un unico nodo, che viene restituito alla precedente invocazione, la ricorsione di `CreaLista(2);` quindi l'ambiente di questo metodo viene distrutto e si scongela l'ambiente di `CreaLista(2).`

invocazione `CreaLista(2);`

`tmp.next = CreaLista(1);`	risultato della ricorsione: 
`return tmp;`	risultato di CreaLista(2): 

Il risultato `CreaLista(2);` è una lista di due nodi, che viene restituito al chiamante, cioè nell'invocazione di button1, se ci fosse.

ALTRI METODI RICORSIVI

Vediamo altri metodi ricorsivi, che potete provare aggiungendo il codice al progetto guidato precedente.

CALCOLO RICORSIVO DELLA LUNGHEZZA DI UNA LISTA

VC#

```
public int ContaLista (Nodo p)
{
    if (p == null)        // se la lista è nulla ha zero elementi
        return 0;
    else
        return 1 + ContaLista (p.next);
}
```

SOMMA RICORSIVA DEGLI ELEMENTI DI UNA LISTA

VC#

```
public int SommaLista (Nodo p)
{
    if (p == null)        // se la lista è nulla ha somma zero
        return 0;
    else
        return p.info + SommaLista (p.next);
}
```

COPIA RICORSIVA DI UNA LISTA

VC#

```
public Nodo CopiaLista (Nodo p)
{
    if (p == null)        // se la lista è nulla la sua copia è nulla
        return null;
    else
    {
        Nodo tmp = new Nodo();
        tmp.info = p.info;
        tmp.next = CopiaLista(p.next);
        return tmp;
    }
}
```

ESERCIZI

ESERCIZI

ESERCIZI DI BASE

ESERCIZIO 1. CREAZIONE ITERATIVA DI LISTA CON VALORI CASUALI

Dichiarare un metodo iterativo che crea una lista di k elementi e che pone in ogni nodo della lista un valore random compreso tra 100 e 999. Un button invoca prima la creazione e poi la visualizza in una listBox1.

ESERCIZIO 2. CREAZIONE RICORSIVA DI LISTA CON VALORI CASUALI

Dichiarare un metodo ricorsivo che crea una lista di k elementi e che pone in ogni nodo della lista un valore random compreso tra 100 e 999. Un button invoca prima la creazione e poi la visualizza in una listBox1.

ESERCIZIO 3. RICERCA RICORSIVA DI LISTA CON VALORI CASUALI

Dichiarare un metodo ricorsivo che cerca un valore intero k in una lista qualsiasi. Se il valore è presente rende true, altrimenti rende false. Un button invoca prima la creazione, poi la visualizza in una listBox1, poi legge un valore da textBox1, poi mostra un messaggio «trovato» o «non trovato».

ESERCIZIO 4. DICHIARAZIONE DI LISTA CON DUE INFORMAZIONI

Dichiarare una lista che invece di un solo campo informazione intero, ne possiede due.
Rifare gli esercizi precedenti per questo tipo di lista.

ESERCIZIO 5. DICHIARAZIONE DI LISTA DI CARTE DA GIOCO

Dichiarare una lista che invece di un solo campo informazione, ne possiede due di tipo stringa.
Definire un metodo CreaMazzo che rende una lista con 40 carte da gioco, con valori da 1 a 10 e con semi ♥♦♣♠.

ESERCIZIO 6. CREAZIONE RICORSIVA DI LISTA CON VALORI CRESCENTI

Dichiarare un metodo ricorsivo che crea una lista di k elementi e che pone in ogni nodo della lista un valore maggiore del precedente. Un button invoca prima la creazione e poi la visualizza in una listBox1.

ESERCIZIO 7. CREAZIONE RICORSIVA DI LISTA CON VALORI DECRESCENTI

Dichiarare un metodo ricorsivo che crea una lista di k elementi e che pone in ogni nodo della lista un valore maggiore del precedente. Un button invoca prima la creazione e poi la visualizza in una listBox1.

ESERCIZIO 8. CONFRONTO DI LISTE

Dichiarare un metodo ricorsivo che ricevute per parametro due liste verifica se sono uguali (stessa lunghezza e stessi valori nella stessa sequenza).
Un button invoca prima la creazione e poi la visualizza in una listBox1.

ESERCIZIO 9. COPIA DI LISTA

Dichiarare un metodo ricorsivo che ricevuta per parametro una lista rende una sua copia.
Un button invoca prima la creazione e poi la visualizza in una listBox1.

ESERCIZIO 10. INVERSIONE DI LISTA

Dichiarare un metodo ricorsivo che ricevuta per parametro una lista rende una nuova lista con i nodi invertiti per sequenza (il primo diventa l'ultimo e così via).
Un button invoca prima la creazione e poi la visualizza in una listBox1.

SOMMARIO