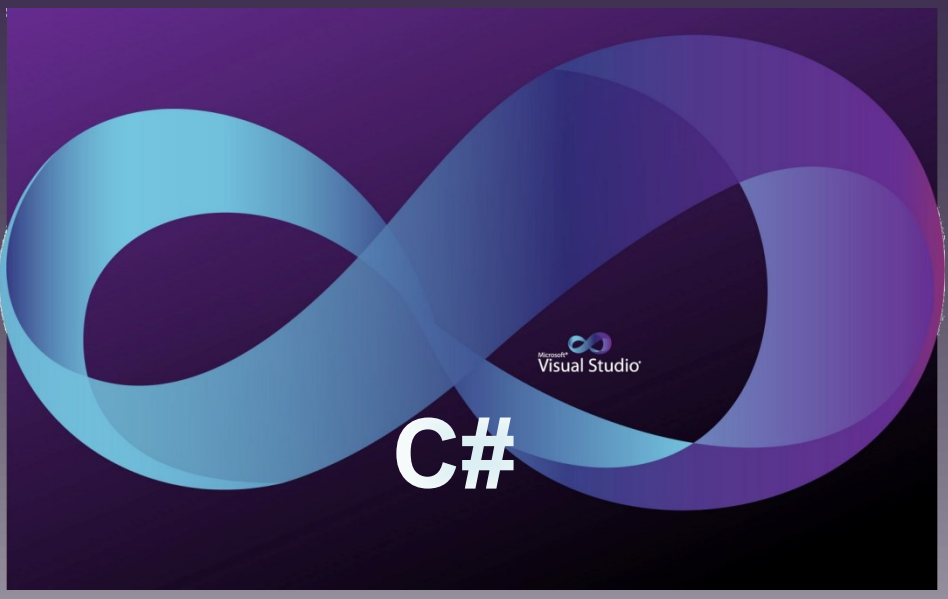
ISTITUTO TECNICO INDUSTRIALE
G. M. ANGIOY
SASSARI



CORSO DI PROGRAMMAZIONE

INTRODUZIONE ALLE ISTRUZIONI ITERATIVE

DISPENSA 03.01

03-01_Iterazioni_[ver_15]

Versione del: 03/11/2021
Revisione numero: 15

Prof. Andrea Zoccheddu
Dipartimento di Informatica

DIPARTIMENTO
INFORMATICA E TELECOMUNICAZIONI

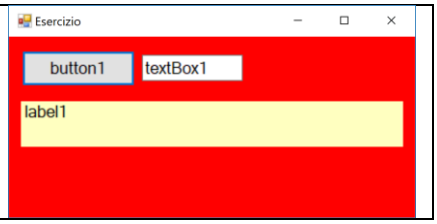
ISTRUZIONI ITERATIVE

ISTRUZIONI ITERATIVE

ISTRUZIONE WHILE (MENTRE . . .)

PROGETTO GUIDATO COL WHILE

- prepara un form1 simile alla figura
- associa al pulsante **button1** il seguente gestore di evento:



VC#

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";
    int num = Convert.ToInt32(textBox1.Text);
    int cont = 0;
    while (cont < num)
    {
        string s = Convert.ToString(cont);
        label1.Text = label1.Text + "; " + s;
        cont++;
    }
}
```

- prova a eseguire il progetto

ISTRUZIONI ITERATIVE (CHE RIPETONO . . .)

Talvolta occorre ripetere molte volte una operazione simile, modificando poche variabili della questione. Le istruzioni che «ripetono» sono dette iterative. Iterare è una parola italiana che significa ripetere. Le istruzioni iterative ripetono infatti più volte altre istruzioni. Spesso si desidera eseguire una stessa operazione più volte, per un numero precisato e costante di occorrenze oppure fino a quando non si raggiunge una situazione desiderata; per questo motivo le istruzioni iterative si chiamano anche cicli.

Visual C# mette a disposizione le seguenti 4 istruzioni iterative:

`while (condizione)` `  istruzione ;`	mentre (condizione) istruzione;

`do`
`  istruzione;`
`while (condizione);`
	esegui
	istruzione;
	mentre (condizione);

`for (inizio; condizione; passo)` `  istruzione ;`	per variabile da inizio a condizione, istruzione e passo;
`foreach ( var in collezione )` `  istruzione ;`	per ogni elemento della collezione istruzione;

Vedremo le varie istruzioni e le differenze e le similitudini tra di esse.

La istruzione classica, comune a gran parte dei linguaggi imperativi, è il while. L'istruzione while è la terza delle istruzioni fondamentali dei linguaggi imperativi, assieme all'assegnazione e all'if then else. Le altre sono istruzioni derivate dal while e utilizzate per comodità.

Le prime tre istruzioni, sono dette istruzioni iterative condizionali, perché la loro terminazione (o uscita dal ciclo) dipendono da **condizioni** dette guardie, ovvero da espressioni booleane di verità.

La prima (**while**) è detta anche istruzione iterativa condizionale con controllo in testa, perché prima di entrare nel ciclo controlla la condizione.

La seconda (**do while**) è detta anche istruzione iterativa condizionale con controllo in coda, perché controlla la condizione prima di uscire dal ciclo.

La terza (**for**) è una istruzione iterativa con passo, poiché specifica nella sintassi il «passo» che deve fare ad ogni giro.

La quarta (**foreach**) è una istruzione iterativa su collezioni, poiché specifica nella sintassi il «passo» che deve fare ad ogni giro.

ISTRUZIONE WHILE (MENTRE . . .)

La istruzione WHILE DO serve per eseguire un ciclo che deve proseguire mentre si verifica una condizione. La condizione è una espressione booleana, cioè può valere True o False. Il controllo viene effettuato subito e si può evitare di eseguire il ciclo se la condizione è subito False. È meglio tradurre while con la parola italiana mentre piuttosto che, come spesso accade, con finché; infatti la parola finché è invece la traduzione di until. Il senso di while do pertanto quello di esprimere in italiano mentre . . . esegui . . . .

Per esempio potresti considerare la istruzione seguente e la traduzione in italiano:

mentre (fa freddo)	mentre (non sono arrivato)
< tengo il fuoco acceso > ;	< vado avanti > ;

Sintassi e Semantica

La sintassi dell'istruzione è la seguente:

```
while (Condizione)
  Istruzione_1;
```

L'istruzione è composta della parola riservata **`while`**. Dopo la parola **`while`** occorre specificare una espressione booleana, detta Condizione o anche Guardia; la Guardia va necessariamente posta tra parentesi: le parentesi sono obbligatorie e fanno parte della sintassi.

La prima cosa che il programma esegue è la valutazione della Guardia: poiché questa espressione è booleana, essa può valere True oppure False.

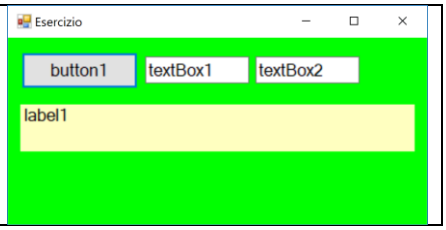
Se l'espressione vale True allora il programma esegue l'istruzione che segue e che si considera interna al ciclo. Dopo aver eseguito l'istruzione il programma torna su e valuta di nuovo la Condizione e, eventualmente, ripete il controllo.

Se l'espressione vale False invece il ciclo è terminato, e si prosegue ad eseguire il resto del programma, con l'istruzione successiva al **`while`**.

Prova a utilizzare il seguente esempio:

PROGETTO GUIDATO BREVE COL WHILE

- prepara un form1 simile alla figura
- associa al pulsante **button1** il seguente gestore di evento:



VC#

```
private void button1_Click(object sender, EventArgs e)
{
    int a = Convert.ToInt16(textBox1.Text);
    int b = Convert.ToInt16(textBox2.Text);
    while (a != b)
    if (a > b)
          a = a - b;
    else
          b = b - a;
    label1.Text = "" + a;
}
```

- prova il programma

RIFLESSIONE SUL PROGETTO GUIDATO

Nel programma proposto come esempio, supponiamo che nelle caselle di testo ci siano i valori 8 e 6.

a	B	Guardia	condizione	assegnazione
8	6	true **perché** 8 != 6	true **perché** 8 > 6	A = 2;
2	6	true **perché** 2 != 6	false **perché** NON è 2 > 6	B = 4;
2	4	true **perché** 2 != 4	false **perché** NON è 2 > 4	B = 2;
2	2	false **perché** NON 2 != 2	esce dal ciclo	

Osservazione: il controllo sulla guardia del **while** viene effettuato subito. Se essa risulta **False** il ciclo non viene eseguito **neppure una volta**.

Osservazione: dopo la condizione si deve porre **una sola** istruzione. Se si desidera eseguire più istruzioni DENTRO il ciclo occorre adoperare l'istruzione sequenza.

La condizione booleana di guardia può assumere tutte le forme desiderate. Per esempio è possibile usare come condizione una variabile booleana, come nell'esempio qui sotto, che usa una variabile Trovato:

```
while ( ! Trovato )
     istruzione_di_ricerca;
```

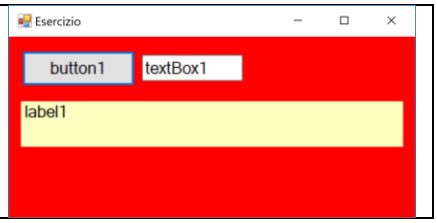
oppure è possibile usare un confronto come il seguente:

```
while ( ! ( (x > y) && (y > z) )
     istruzione_di_calcolo;
```

ISTRUZIONE DO . . . WHILE (. . .)

PROGETTO GUIDATO CON DO... WHILE

- prepara un form1 simile alla figura
- associa al pulsante **button1** il seguente gestore di evento:



VC#

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";
    int num = Convert.ToInt32(textBox1.Text);
    int cont = 0;
    do
    {
        string s = Convert.ToString(cont);
        label1.Text = label1.Text + "; " + s;
        cont++;
    } while (cont < num);
}
```

- prova a eseguire il progetto

ISTRUZIONE DO . . . WHILE (ESEGUI . . . MENTRE . . .)

L'istruzione DO WHILE serve per eseguire un ciclo che deve proseguire mentre si verifica una condizione. A differenza del while, prima si eseguono le istruzioni e dopo si verifica la condizione di permanenza nel ciclo. La condizione è una espressione booleana, cioè può valere True o False. Essa ha la forma:

esegui

< tengo il fuoco acceso > ;

mentre (fa freddo)

esegui

< vado avanti > ;

mentre (non sono arrivato)

Sintassi e Semantica

La sintassi dell'istruzione è la seguente:

```
do
  Istruzione_1;
while (Condizione)
```

L'istruzione è composta delle parole riservate **do** e **while**. Dopo il do segue l'istruzione (eventualmente un blocco) e il ciclo è concluso con la parola **while** che richiede una espressione booleana, (la Condizione o Guardia); la Guardia va necessariamente posta tra parentesi: le parentesi sono obbligatorie e fanno parte della sintassi.

La prima cosa che il programma esegue è l'istruzione; dopo averle eseguite segue la valutazione della Guardia: poiché questa espressione è booleana, essa può valere True oppure False.

Se l'espressione vale True allora il programma torna sopra per eseguire di nuovo le istruzioni.

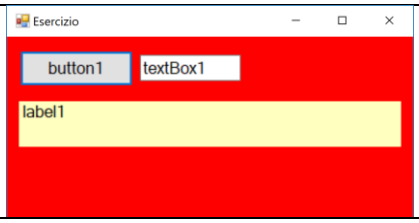
Se l'espressione vale False invece il ciclo è terminato, e si prosegue ad eseguire il resto del programma, con l'istruzione successiva al **do . . . while**. Prova a utilizzare il seguente esempio:

Osservazione: il controllo sulla guardia del **while** viene effettuato in seguito. In ogni caso il ciclo esegue il corpo **almeno una volta**.

ISTRUZIONE FOR (. . .)

PROGETTO GUIDATO

- prepara un form1 simile alla figura
- associa al pulsante **button1** il seguente gestore di evento:



VC#

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "";
    int num = Convert.ToInt32(textBox1.Text);
    for (int cont = 0; cont < num; cont++)
    {
        string s = Convert.ToString(cont);
        label1.Text = label1.Text + "; " + s;
    }
}
```

- prova a eseguire il progetto

ISTRUZIONE FOR (INIZIO; CONDIZIONE; PASSO)

L'istruzione for serve per eseguire un ciclo indicando con chiarezza quale (o quali) variabili intende usare per avanzare nel ciclo.

Sintassi e Semantica

La sintassi dell'istruzione è la seguente:

```
for («inzio»; «condizione»; «passo»)
{
    corpo
}
```

L'istruzione è introdotta dalla parola riservata **for** . Dopo il for ci sono tre opzioni dentro una parentesi tonda.

La prima opzione è l'inizio: serve per stabilire da quale valore debba iniziare la variabile; spesso si usa una dichiarazione con inizializzazione. Per esempio un inizio potrebbe essere:

```
int numero = 10;
```

che specifica che la variabile numero sarà usata come indice del ciclo e che inizia dal valore 10.

La seconda opzione è la condizione: serve per stabilire quando il ciclo si deve fermare; spesso si usa un operatore di confronto. Per esempio una condizione potrebbe essere:

```
numero <= 100;
```

che specifica che, se la variabile numero sarà superiore a 100, il ciclo finirà.

La terza opzione è il passo: serve per indicare come il ciclo deve proseguire; spesso si usa un contatore con un operazione di auto-incremento. Per esempio un passo potrebbe essere:

```
numero = numero * 2;
```

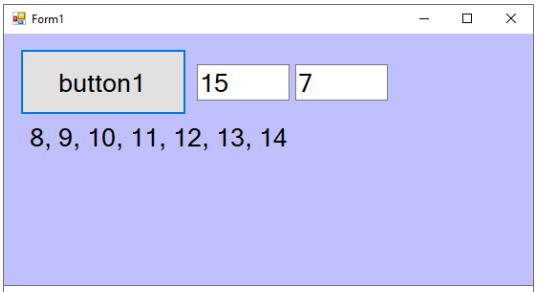
che significa che la variabile deve raddoppiare ad ogni giro.

ESERCIZI

ESERCIZI CON ISTRUZIONI ITERATIVE

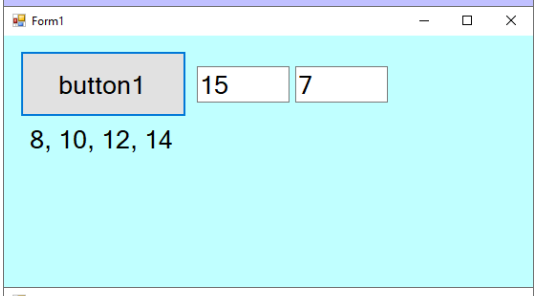
ESERCIZIO 1. INTERVALLO

- Si desidera realizzare un programma che, letti da due caselle di testo due numeri interi anche negativi, mostri in una etichetta la somma dei numeri compresi tra i due (estremi esclusi)



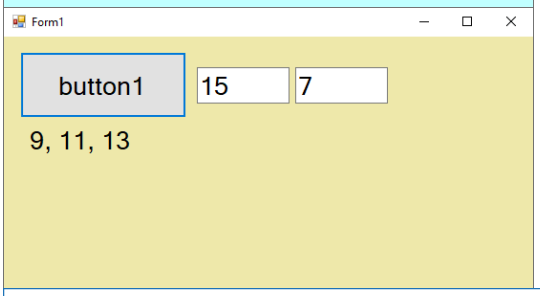
ESERCIZIO 2. INTERVALLO PARI

- Si desidera realizzare un programma che, letti da due caselle di testo due numeri interi anche negativi, mostri in una etichetta la somma dei numeri **pari** compresi tra i due (estremi esclusi)



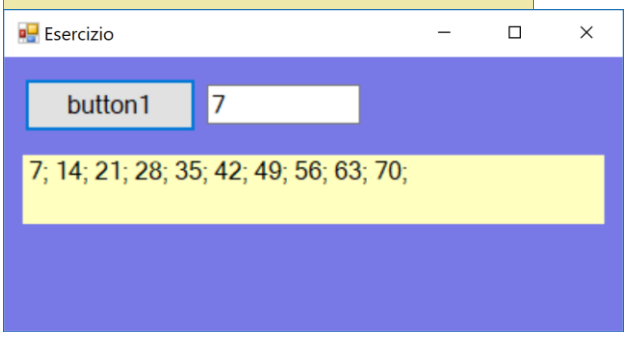
ESERCIZIO 3. INTERVALLO DISPARI

- Si desidera realizzare un programma che, letti da due caselle di testo due numeri interi anche negativi, mostri in una etichetta la somma dei numeri **dispari** compresi tra i due (estremi esclusi)



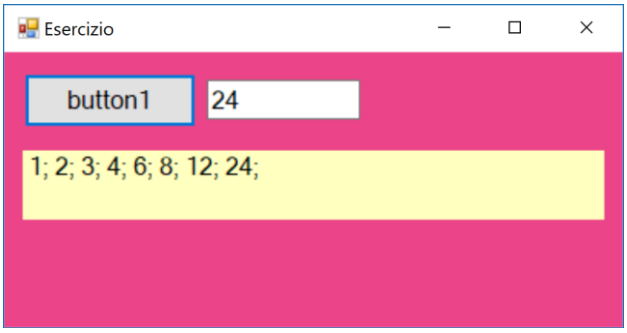
ESERCIZIO 4. TABELLINA

- Prepara un form1 simile alla figura
- Quella gialla è una label1 con Autosize a false
- Si desidera realizzare un programma che, letto un valore da una casella di testo, deve mostrare in una etichetta **la tabellina** del numero (da 1 a 10)



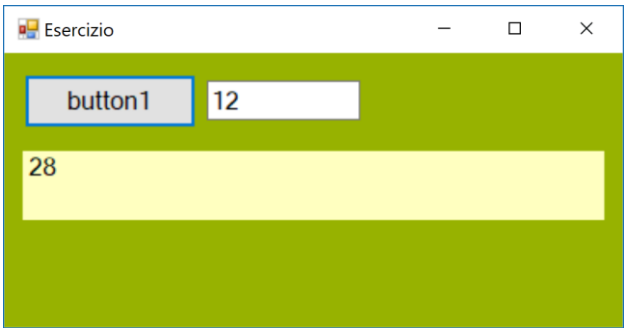
ESERCIZIO 5. DIVISORI

- Prepara un form1 simile alla figura
- Quella gialla è una label1 con Autosize a false
- Si desidera realizzare un programma che, letto un valore da una casella di testo, deve mostrare in una etichetta i divisori del numero dato
- è anche possibile usare una ListBox invece di una Label



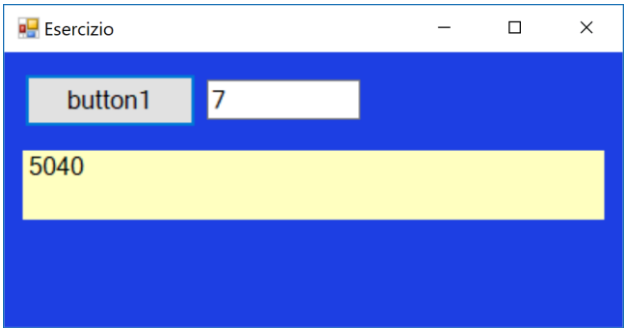
ESERCIZIO 6. SOMMA DIVISORI

- Prepara un form1 simile alla figura
- Quella gialla è una label1 con Autosize a false
- Si desidera realizzare un programma che, letto un valore da una casella di testo, deve mostrare in una etichetta la somma dei divisori del numero dato



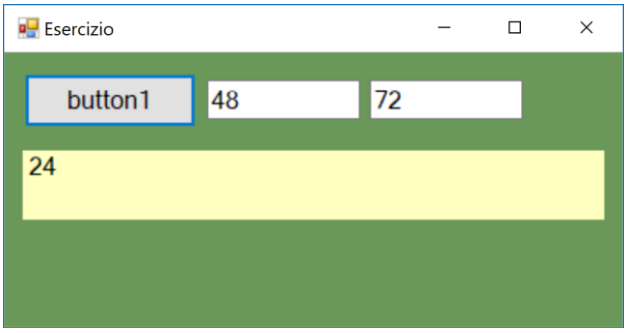
ESERCIZIO 7. FATTORIALE

- Prepara un form1 simile alla figura
- Quella gialla è una label1 con Autosize a false
- Si desidera realizzare un programma che, letto da una casella di testo un numero intero positivo, mostri in una etichetta il suo **fattoriale** (il prodotto del numero per tutti i suoi predecessori positivi)



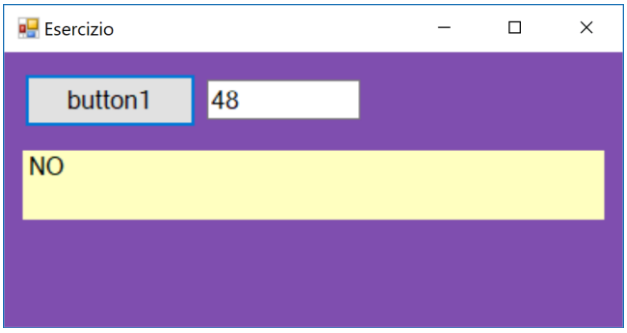
ESERCIZIO 8. M.C.D.

- Prepara un form1 simile alla figura
- Quella gialla è una label1 con Autosize a false
- Si desidera realizzare un programma che, letti da due caselle di testo due numeri interi positivi, mostri in una etichetta il loro **Massimo Comun Divisore**



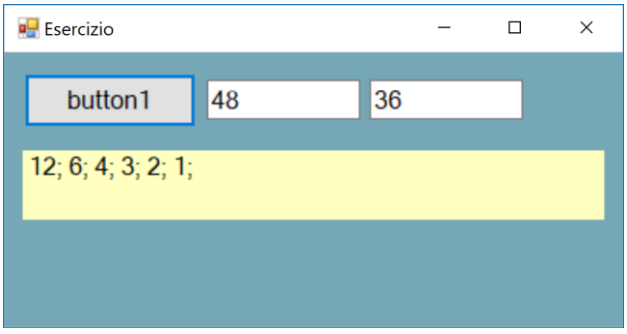
ESERCIZIO 9. NUMERO PRIMO

- Prepara un form1 simile alla figura
- Quella gialla è una label1 con Autosize a false
- Si desidera realizzare un programma che, letto un numero da una casella di testo, mostri in una etichetta se è un **numero primo** oppure no!



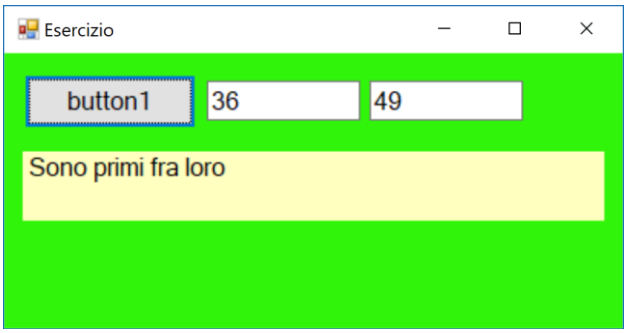
ESERCIZIO 10. DIVISORI COMUNI

- Prepara un form1 simile alla figura
- Quella gialla è una label1 con Autosize a false
- Si desidera realizzare un programma che, letti due numeri interi da due rispettive caselle di testo, deve mostrare i divisori comuni dei due numeri



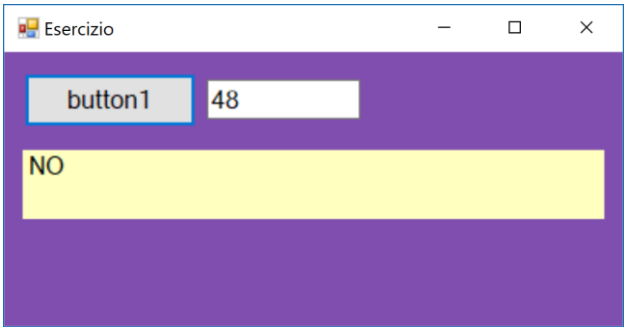
ESERCIZIO 11. NUMERI PRIMI TRA LORO

- Prepara un form1 simile alla figura
- Quella gialla è una label1 con Autosize a false
- Si desidera realizzare un programma che, letti due numeri interi da due rispettive caselle di testo, deve scrivere «**primi fra loro**» se il loro massimo comune divisore è 1, altrimenti scrive «**non primi**»



ESERCIZIO 12. NUMERO PERFETTO

- Prepara un form1 simile alla figura
- Quella gialla è una label1 con Autosize a false
- Si desidera realizzare un programma che, letto un numero intero da una casella di testo, deve scrivere «**perfetto**» se è uguale alla somma dei suoi divisori escluso sé stesso; altrimenti scrive «**non perfetto**»



ESERCIZIO 13. CONGETTURA DI COLLATZ

- Si legga un numero N da una textBox e si proceda con il seguente algoritmo
- Se n = 1, l'algoritmo termina.
- Altrimenti se n è pari, si divida per due;
- Altrimenti si moltiplichi per 3 e si aggiunga 1.
- Si ripeta l'algoritmo fino a raggiungere 1.
- Alla fine mostrare il numero dei passi eseguiti.

ESEMPIO

PER ESEMPIO, INIZIANDO CON N = 6, OTTENIAMO LA SUCCESSIONE 6 → 3, 10, 5, 16, 8, 4, 2, 1 CHE CONTA 8 PASSI.

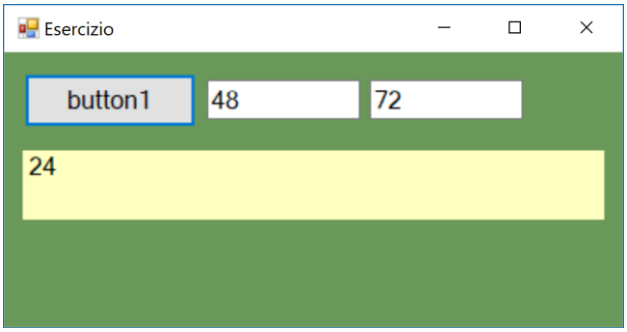
QUINDI 6 → 8.

ESERCIZIO 14. ALGORITMO DI EUCLIDE

- Si desidera realizzare un programma che, letti da due caselle di testo due numeri interi positivi svolge il seguente controllo:

I. Se i numeri sono uguali allora il programma termina mostrando il valore raggiunto

II. Se i numeri sono diversi, allora sottrae il minore dal maggiore e ripete l'operazione



SOMMARIO