# RICORSIONE

- La ricorsione è la capacità di un metodo di *richiamare se stesso*.

- Una definizione ricorsiva consiste in due parti:
  - *Caso base*: vengono elencati gli elementi di base che sono i blocchi costitutivi dell'insieme;
  - *Caso induttivo/ricorsivo*: vengono fornite le regole per costruire nuovi elementi a partire dagli elementi di base o da elementi che sono già stati costruiti.



# RICORSIONE

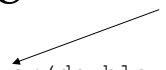
- La funzione che definisce l'elevamento di un numero $x$ ad una potenza non negativa $n$ è un esempio di funzione ricorsiva:

$$x^n = \begin{cases} 1 & \text{se } n = 0 \text{ (caso base)} \\ x \cdot x^{n-1} & \text{se } n > 0 \text{ (caso ricorsivo)} \end{cases}$$

- Implementiamo questa funzione in C# e analizziamo come avviene un'invocazione ricorsiva.



# ESEMPIO



```
/*102*/ public static double Power(double x, int n){
/*103*/           if (n==0)
/*104*/                 return 1;
/*105*/           return x*Power(x,n-1);
          }

           static void Main(string[] args) {
              ...
/*136*/         y = Power(5.6,2);
              ...
```

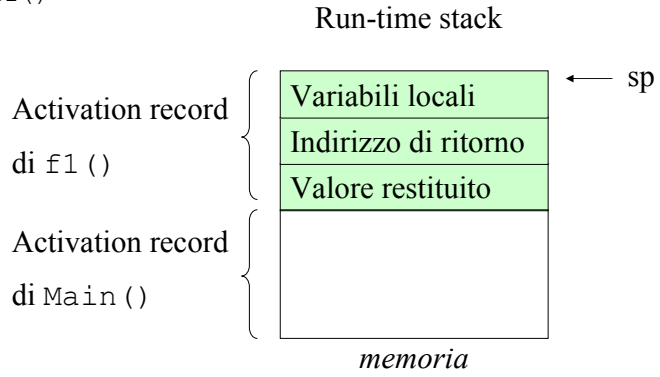


# INVOCAZIONE DI UN METODO

- Cosa succede quando un metodo viene invocato?
- Il *sistema* deve *memorizzare* lo stato del chiamante, creare lo spazio per lo stato del chiamato e sapere dove riprendere l'esecuzione del programma dopo che il metodo è terminato.
- Lo stato del metodo, cioè la documentazione di attivazione (*activation record*), viene allocato sulla *pila* di esecuzione (*run-time stack*) e un puntatore (*stack pointer*) tiene traccia della posizione nella *pila*.

# INVOCAZIONE DI UN METODO

Per esempio se il `Main()` chiama il metodo `f1()`






# CHIAMATA RICORSIVA

Una traccia delle chiamate ricorsive dell'esempio:

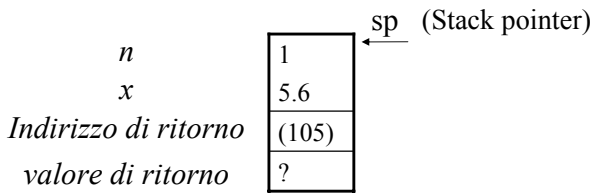
| | | |
|---|---|---|
| Invocazione 1 | `Power(5.6,2)` | |
| Invocazione 2 | `Power(5.6,1)` | |
| Invocazione 3 | `Power(5.6,0)` | |
| *Invocazione 3* | `1` | |
| *Invocazione 2* | `5.6` | |
| *Invocazione 1* | `31.36` | |

Il caso base deve essere sempre presente per interrompere la ricorsione.



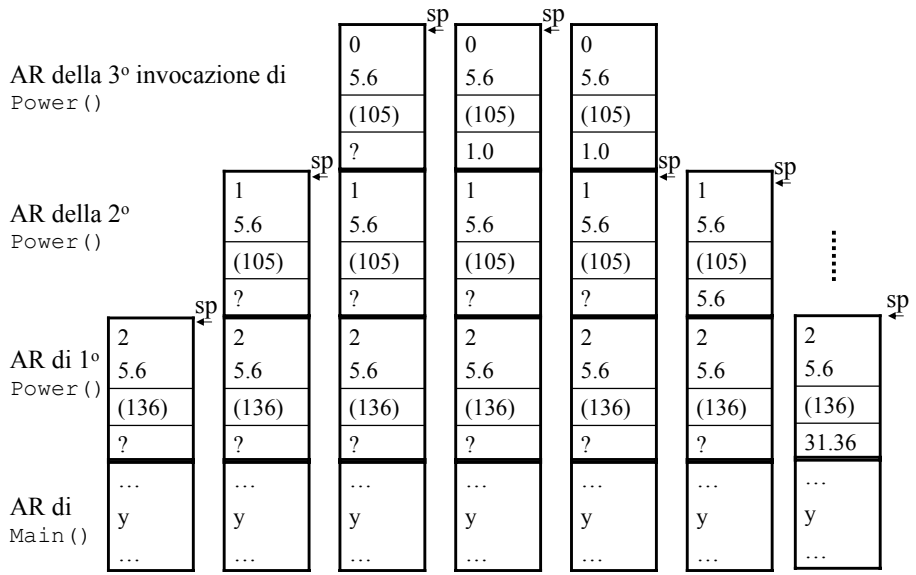
# CHIAMATA RICORSIVA

- Vediamo una possibile rappresentazione grafica della pila di esecuzione dell'esempio precedente:
  - I numeri nei commenti (per es. /*136*/) rappresentano l'indirizzo dato dal sistema a quella "linea di codice";
  - Il ? indica il valore di ritorno non ancora calcolato;






# CHIAMATA RICORSIVA



# RICORSIONE IN CODA

- L'esempio visto rappresenta un tipo di ricorsione chiamato *ricorsione in coda*.
- La ricorsione in coda è caratterizzata dall'uso di una sola invocazione ricorsiva al termine del metodo.
- Tale *ricorsione* può essere trasformata in *iterazione* attraverso l'uso di un ciclo.
- Quale vantaggio ad usare la ricorsione? La ricorsione sembra essere più intuitiva, perchè più simile alla definizione originale, e permette di scrivere codice coinciso.



# RICORSIONE IN CODA

- Tuttavia bisogna porre attenzione alle *risorse* che vengono utilizzate per produrre le chiamate ricorsive: lo spazio di memoria utilizzato e il tempo necessario ad attivare nuove chiamate.
- In generale la ricorsione in coda non è una caratteristica consigliabile.
- La potenza della ricorsione è legata a particolari strutture dati o ad algoritmi specifici.



# RICORSIONE NON IN CODA

- Un esempio di tale ricorsione e la visualizzazione di una stringa di ingresso in ordine inverso:

```
static void Main(string[] args){
    Console.Write("Inserisci una stringa: ");
    Reverse();
}
```

Una possibile uscita

Inserisci una stringa: 123456
654321

Una possibile uscita

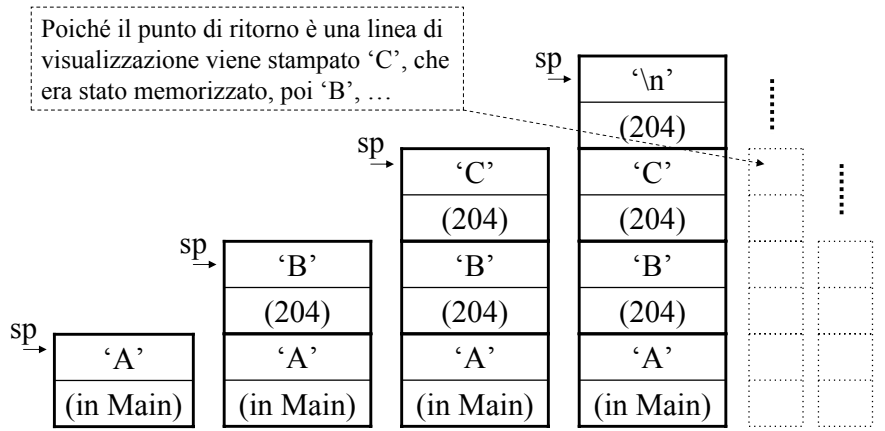
Inserisci una stringa: qwert
trewq

```
/*200*/  public static void Reverse(){
/*201*/       char ch=Convert.ToChar(Console.Read());
/*202*/       if (ch!='\n'){
/*203*/            Reverse();
/*204*/            Console.Write("{0}", ch);}
        }
```



# RICORSIONE NON IN CODA

Vediamo la pila di esecuzione con la stringa "ABC".



# RICORSIONE NON IN CODA

- La trasformazione della ricorsione in iterazione di solito richiede la *gestione esplicita* di una *pila*. Vediamo un esempio che usa un *array* per memorizzare i caratteri:

```
public static void ReverseIter()
{
    char[] stack = new char[80];
    int top = 0;
    stack[top] = Convert.ToChar(Console.Read());
    while (stack[top] != '\n')
        stack[++top] = Convert.ToChar(Console.Read());
    for (top -= 1; top >= 0; top--)
        Console.Write("{0}", stack[top]);
}
```



# RICORSIONE ECCESSIVA

- Se una funzione ricorsiva *ripete* il calcolo di alcuni parametri, il tempo di esecuzione può diventare elevato anche per casi molto semplici.
- Consideriamo i numeri di Fibonacci:

$$F(n) = \begin{cases} n & \text{se } n < 2 \\ F(n-2) + F(n-1) & \text{altrimenti} \end{cases}$$

- Una possibile implementazione ricorsiva è la seguente.



# RICORSIONE ECCESSIVA

```
public static int Fib(int n){
    if (n<=0) return 0;
    if (n==1) return 1;
    return Fib(n-1)+Fib(n-2);
}
```

- Il metodo è semplice e di facile comprensione ma estremamente inefficiente. Per esempio, il calcolo di `Fib(31)` richiede 2178308 addizioni e 4356617 chiamate.



# RICORSIONE ECCESSIVA

- L'origine di questa inefficienza risiede nella *ripetizione degli stessi calcoli*, come può essere evidenziato dall'albero delle invocazioni: