Capitolo 19

Ricorsione

La ricorsione

- Una funzione matematica è definita ***ricorsivamente*** quando nella sua definizione compare un riferimento a sé stessa
- **Esempio:**
  - Funzione fattoriale su interi non negativi:

    f(n) = n!

    è definita ricorsivamente come segue:
    - f(n)=1 se n=0 (caso ***base***)
    - f(n)=n*f(n-1) se n>0 (caso ***generico***)
  - Usando il metodo induttivo si specifica come tale funzione si comporta nel caso ***base*** e nel caso ***generico***

Induzione matematica

- Immaginando di avere x_k, costruisci x_{k+1}.
  - Induttivamente, il calcolo del fattoriale di un numero n viene ricondotto al calcolo del fattoriale di n-1, il calcolo del fattoriale di n-1 a quello di n-2, etc., fino a raggiungere un caso base (fattoriale di 0), a risultato noto.
- Metodo particolarmente utile per alcuni problemi (intrinsecamente ricorsivi) o che lavorano su strutture dati ricorsive (liste, alberi).

Esempi di problemi ricorsivi

- **Somma** dei primi n numeri naturali:
  - somma(n) =
    - 0 se n=0
    - n+somma(n-1) altrimenti
- Generare l'n-esimo **numero di Fibonacci**:
  - fib(n) =
    - 0 se n=0
    - 1 se n=1
    - fib(n-1)+fib(n-2) altrimenti

Esempi di problemi ricorsivi

- Calcolo del **minimo** di una sequenza di elementi:
  - [a1, a2, a3, ...] = [a1|[a2, a3, ...]]
    - min([a1]) = a1
    - min([a1,a2]) = a1 se a1<a2, altrimenti a2
    - min([a1|Z]) = min([a1,min(Z)])
  - sviluppando:
    - min([a1|Z]) = min([a1,min([a2, min([a3,...])])])
- Calcolo della **lunghezza** di una sequenza:
    - lung([]) = 0
    - lung([a1|Z]) = 1+ lung(Z)

Programmazione ricorsiva

- Molti linguaggi di programmazione offrono la possibilità di definire funzioni/procedure ricorsive
- Esempio: Calcolo del fattoriale di un numero

```
public static void main(String[] args) {
     Scanner scanner=new Scanner(System.in);
     System.out.println("Introdurre N: ");
     int n=scanner.nextInt();
     System.out.println("Fattoriale di "+n+"="+fattoriale(n));
}
public static int fattoriale(int n) {
     if (n==0) return 1;
     else return n*fattoriale(n-1);
}
```

  - Non tutti i linguaggi di alto livello supportano procedure ricorsive (ad esempio il FORTRAN non consente di scrivere sottoprogrammi ricorsivi)

Calcolo della somma dei primi N naturali


```
public static void main(String[] args) {
     Scanner scanner=new Scanner(System.in);
     System.out.println("Introdurre N: ");
     int n=scanner.nextInt();
     System.out.println("Somma fino a"+n+"="+sum(n));
}

public static int sum(int n) {
     if (n==0) return 0;
     else return n+sum(n-1);
}
```

- Sono esempi di ***ricorsione lineare*** (una sola chiamata ricorsiva nel corpo della funzione)

N-esimo numero di Fibonacci

- fib(n) =
  - 0 se n=0
  - 1 se n=1
  - fib(n-1)+fib(n-2) altrimenti

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

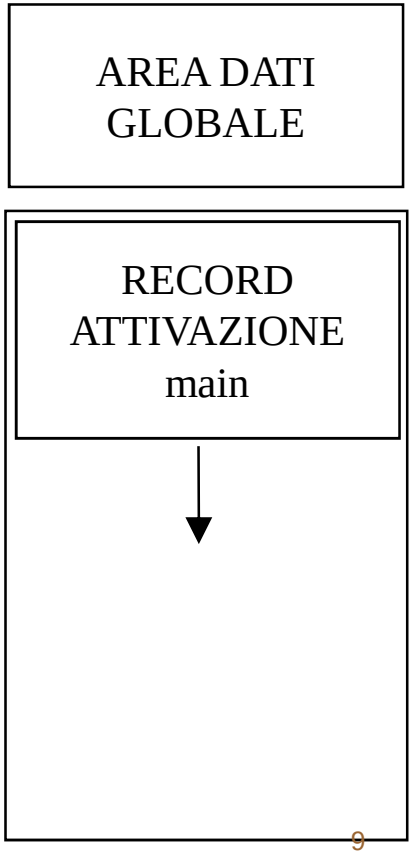
- È un esempio di ***ricorsione non lineare*** (più chiamate ricorsive nel corpo della funzione per determinare il valore restituito dalla funzione)

Stato della memoria (1)

- Vediamo cosa succede all'interno della memoria quando viene eseguito un metodo ricorsivo
- Pila di attivazioni per la chiamata

```
public static void main(String[] args) {
     System.out.println("Fattoriale di 2“
             +fattoriale(2));
}
```

- All'inizio dell'esecuzione:



Stato della memoria (2)

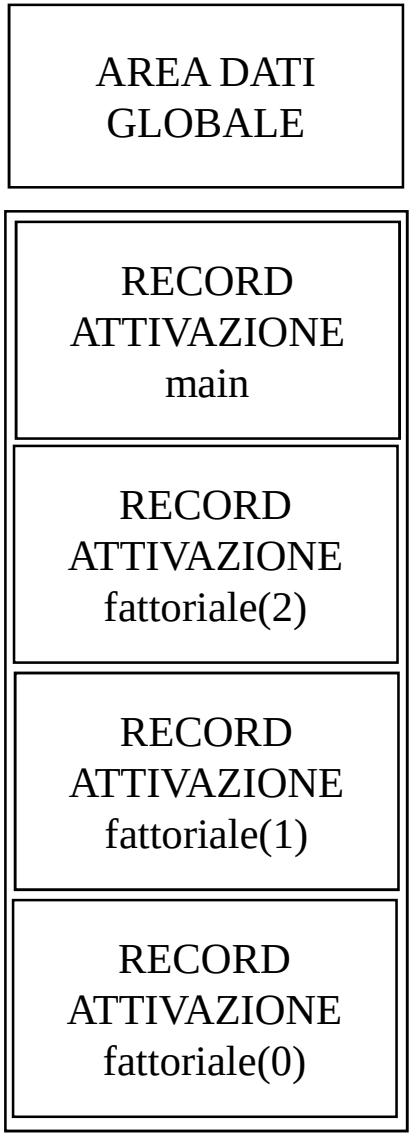
- Dopo la prima attivazione della funzione fattoriale (fattoriale(2))

Stato della memoria (3)

- Dopo la seconda attivazione (fattoriale(1))

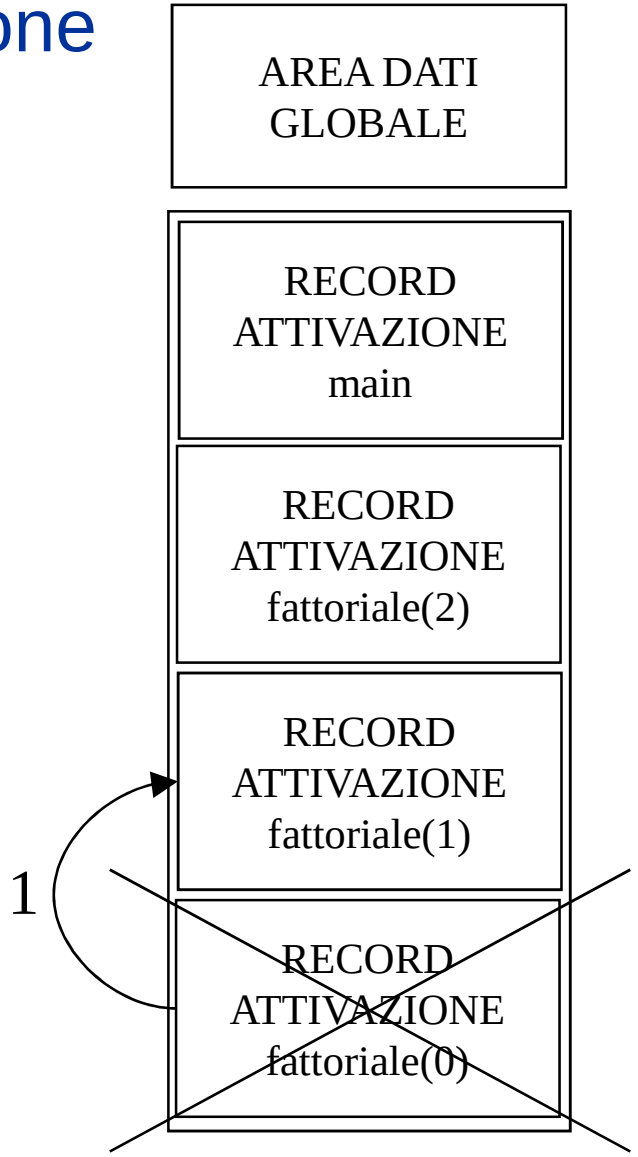
Stato della memoria (4)

- Dopo la terza attivazione (fattoriale(0))



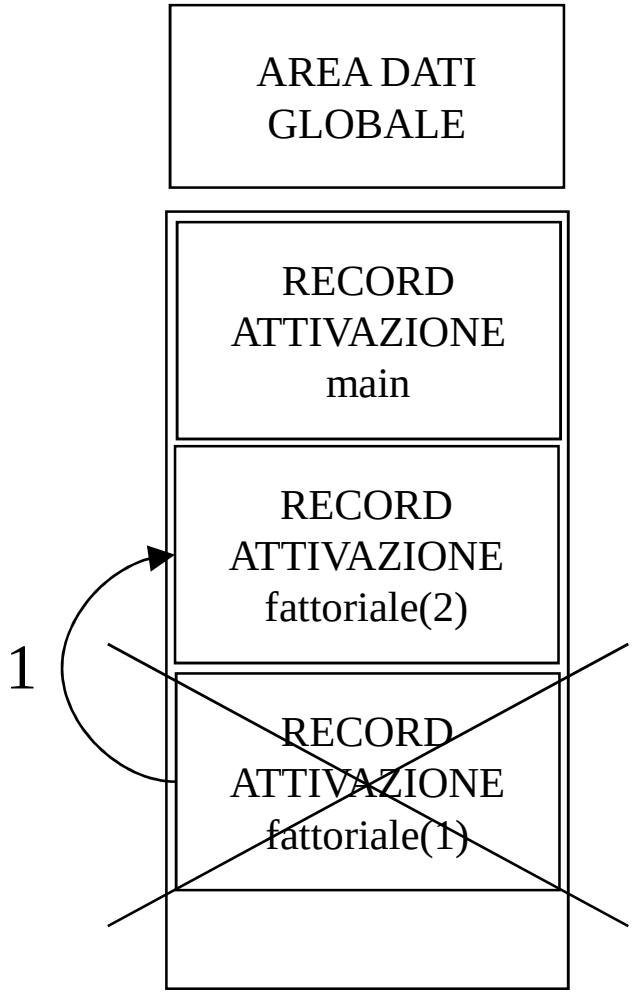
Stato della memoria (5)

- Termine della terza attivazione (return)



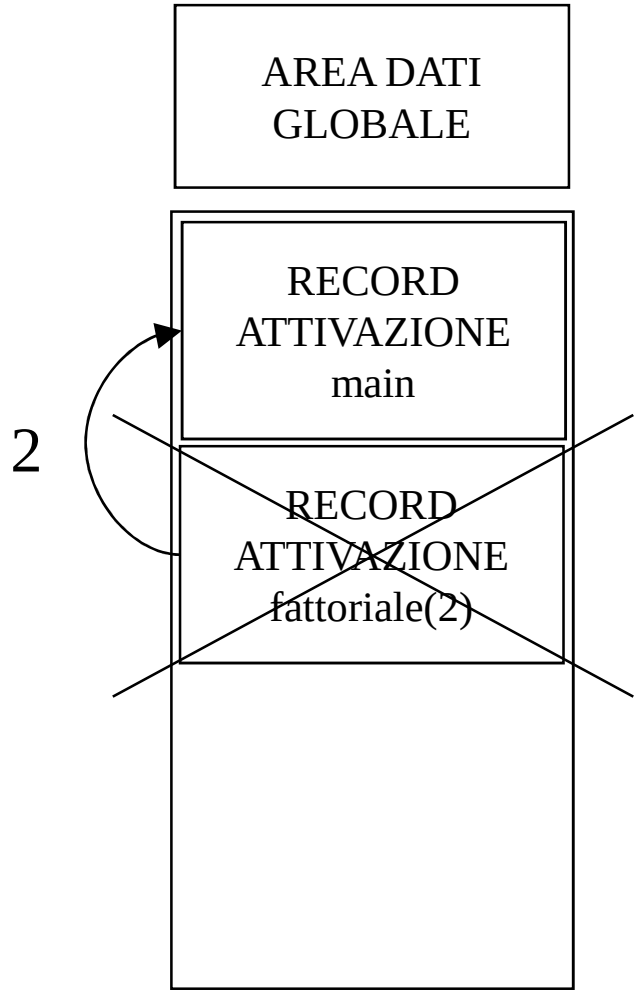
Stato della memoria (6)

- Termine della seconda attivazione (return)



Stato della memoria (7)

- Termine della prima attivazione (return)
- Viene restituito il valore 2 al main, e questo stampa il risultato



Ricorsione ed iterazione

- La ricorsione è sempre realizzabile mediante **iterazione**
- Ad esempio:
  - Realizzazione iterativa del fattoriale:

```
public static void main(String[] args) {
     Scanner scanner=new Scanner(System.in);
     System.out.println("Introdurre N: ");
     int n=scanner.nextInt();
     System.out.println("Fattoriale di "+n+"="+itFattoriale(n));
}
public static int itFattoriale(int n) {
     int f=1;
     for (int i=1; i<=n; i++)
            f *= i;
     return f;
}
```

Ricorsione e Iterazione

- Quando conviene utilizzare la ricorsione?
  - Soluzioni ricorsive sono spesso più vicine alla definizione matematica di certe funzioni
  - Versioni iterative sono generalmente ***più efficienti*** di una soluzione ricorsiva (sia in termini di memoria che di tempo di esecuzione)

Esercizi

- Scrivere la versione iterativa della procedura per il calcolo dell'n-esimo numero di Fibonacci.
- Scrivere una procedura PrintRev che legge in ingresso una sequenza di caratteri (terminata da '.') e stampa la sequenza al contrario:
  - **ROMA.**
  - **AMOR**

  Definirne una versione ricorsiva senza utilizzare il tipo stringa.

Ricorsione *Tail*

- Quando la chiamata ricorsiva di una funzione/procedura F è l'ultima istruzione del codice di F, si dice che F è tail-ricorsiva.
- Ad esempio:

```
public static void main(String[] args) {
     Scanner scanner=new Scanner(System.in);
     System.out.println("Introdurre due numeri: ");
     int n=scanner.nextInt();
     int m=scanner.nextInt();
     System.out.println("Somma di "+n+" e "+m+"="+f(n,m));
}
public static int f(int x, int y) {
     if (x==0) return y;
     else if (x>0) return f(x-1,y+1);
     else return f(x+1,y-1);
}
```

(in pratica, f somma x ad y)

Ricorsione Tail

- La computazione che si origina tramite l'invocazione di una funzione tail-ricorsiva corrisponde ad un ***processo computazionale iterativo***
- Un processo computazionale è ***ricorsivo*** quando è caratterizzato da una catena di operazioni posticipate, il cui risultato è disponibile solo dopo che l'ultimo anello della catena si è concluso
- In un processo computazionale ***iterativo***, ad ogni passo è disponibile una frazione del risultato

Nell'esempio

- Supponiamo: n=2, m=3

```
System.out.println("Somma di "+n+" e "+m+"="+f(n,m));
```

- f(2,3) => f(1,4) => f(0,5) => return 5
  - Il risultato non viene ri-elaborato dalle attivazioni intermedie, ma passato semplicemente da ciascuna al chiamante.
  - Il compilatore, per ottimizzare l'occupazione dello stack potrebbe utilizzare il medesimo record di attivazione per tutte le attivazioni successive della funzione tail ricorsiva.
- Consente di ottimizzare lo spazio di memoria allocato sullo stack

Versione tail-ricorsiva del fattoriale


```
public static int fattoriale(int n) {
     if (n==0) return 1;
     else return n*fattoriale(n-1);
}
```

- Versione tail-ricorsiva:

```
public static int fattoriale(int n) {
     return tailFattoriale(1,n,1);
}

private static int tailFattoriale(int i, int n, int f) {
     if (i<=n) return fatt_tail(++i,n,f*i);
     else return f;
}
```

Esercizio

- Scrivere una versione ricorsiva dell'algoritmo che calcola il prodotto come sequenza di somme

```
public static void main(String[] args) {
     Scanner scanner=new Scanner(System.in);
     System.out.println("Introdurre due numeri: ");
     int n=scanner.nextInt();
     int m=scanner.nextInt();
     System.out.println(“Prodotto di "+n+" e "+m+"="+prod(n,m));
}

public static int prod(int x, int y) {
     if(y==0) return 0;
     else return x+prodotto(x,y-1);
}
```

Esercizio

- Scrivere una versione ricorsiva dell'algoritmo di ordinamento di un vettore per massimi successivi

```
public static void main(String[] args) {
     Scanner scanner=new Scanner(System.in);
     System.out.println("Quanti valori? ");
     int n=scanner.nextInt();
     int[] V=new int[n];
     for(int i=0; i<n; i++) {
            System.out.println((i+1)+"-esimo valore: ");
            V[i]=scanner.nextInt();
     }
     ordina(V,n);
     for(int i=0; i<n; i++) {
            System.out.println((i+1)+"-esimo valore: "+V[i]);
}
```

Ordinamento del vettore


```
public static void ordina(int[] V, int N) {
   if (N==1) return;
   for(int max=N-1, int j=0; j<N; j++)
        if (V[j]>V[max]) max=j;
        if (max<N-1) { // scambio
                tmp=V[N-1];
                V[N-1]=V[max];
                V[max]=tmp;
        }
   ordina(V, N-1);     // ricorsione
}
```

Complessità di Algoritmi Ricorsivi

- Sia $S(n)$ il costo incognito dell'algoritmo per un input di dimensione n
- Dal codice dell'algoritmo, occorre ricavare l'equazione ricorsiva di costo, espanderla e cercare di riconoscerla (ad esempio può risultare la somma parziale di una serie aritmetica o geometrica, ecc.)

Esempio


```
public void scrivi(int n) {
   if(n>1) scrivi(n-1);
   System.out.print(" "+n);
}

public static void main(String[] args) {
   Scanner scanner=new Scanner(System.in);
   int n=scanner.nextInt();
   scrivi(n);
}
```

- Equazione Ricorsiva:
  - Caso base: $S(1) = 1+1 = 2$
  - Caso n: $S(n) = 1+S(n-1)+1 = 2+S(n-1)$
- Complessità asintotica: n

Calcolo del fattoriale


```
public static void main(String[] args) {
     Scanner scanner=new Scanner(System.in);
     System.out.println("Introdurre N: ");
     int n=scanner.nextInt();
     System.out.println("Fattoriale di "+n+"="+fattoriale(n));
}
public static int fattoriale(int n) {
     if (n==0) return 1;
     else return n*fattoriale(n-1);
}
```

- Equazione Ricorsiva:
  - Caso base: $S(1) = 1+1= 2$
  - Caso n: $S(n) = 1+S(n-1)+1 = 2+S(n-1)$
- Complessità asintotica: n