FUNZIONI RICORSIVE

Una funzione matematica è ***definita ricorsivamente*** quando nella sua definizione compare un riferimento (***chiamata***) ***a se stessa***.

Esempio:

Funzione fattoriale su interi non negativi:

f(n) = n!

definita ricorsivamente come segue:

$$f(n) = \begin{cases} 1 & \text{se } n=0 \\ n*f(n-1) & \text{se } n>0 \end{cases}$$

Usando il metodo induttivo si specifica come tale funzione si comporta nel caso base e nel passo generico.

Induzione matematica:

immaginando di avere x_k, costruisci x_{k+1}.

Informalmente, il calcolo del fattoriale di un numero n viene ricondotto al calcolo del fattoriale di n-1, fino a raggiungere un caso base (fattoriale di 0), noto.

Metodo particolarmente utile per alcuni problemi (intrinsecamente ricorsivi) o che lavorano su strutture dati ricorsive (liste, alberi).

Esempi di problemi ricorsivi:

1) **Somma** dei primi *n* numeri naturali:

$$\text{somma}(n) = \begin{cases} 0 & \text{se } n=0 \\ n+\text{somma}(n-1) & \text{altrimenti} \end{cases}$$

2) Generare l'*n*-esimo **numero di Fibonacci**:

$$\text{fib}(n) = \begin{cases} 0 & \text{se } n=0 \\ 1 & \text{se } n=1 \\ \text{fib}(n-1)+\text{fib}(n-2) & \text{altrimenti} \end{cases}$$

3) **Ricerca** di un elemento *el* in una sequenza di interi:

$$\text{ricerca}(el,sequenza) = \begin{cases} \text{falso} & \text{se } sequenza \text{ terminata, altrimenti} \\ \text{vero} & \text{se } el = \underline{\text{primo}}(sequenza), \text{ altrimenti} \\ & \text{ricerca}(el, \underline{\text{resto}}(sequenza)) = \end{cases}$$

PROGRAMMAZIONE RICORSIVA:

Molti linguaggi di programmazione offrono la possibilità di definire funzioni/procedure ricorsive.

Calcolo del fattoriale di un numero:

```
#include <stdio.h>

int fattoriale(unsigned int n);

main(void)
{  int n;

   printf("\nIntrodurre N:\t");
   scanf("%d",&n);

   printf("\nFattoriale di %d:\t%d\n",
            n, fattoriale(n));
}

int fattoriale(unsigned int n)
{
   if (n==0) return 1;
   else
      if (n==1) return 1
      else return n*fattoriale(n-1);
                  /* ricorsione */
}
```


Calcolo della somma dei primi N naturali:

```
#include <stdio.h>

int sum_to(unsigned int n);

main(void)
{
   int n;

   printf("\nIntrodurre N:\t");
   scanf("%d",&n);

   printf("\nSomma fino a %d:\t%d\n",
                           n, sum_to(n));
}

int sum_to(unsigned int n)
{
   if (n==0) return 0;
   else return n+sum_to(n-1);
                  /* ricorsione */
}
```

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

N-esimo numero di Fibonacci:

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

```
int fib(unsigned int n)
{
   if (n==0) return 0;
   else
      if (n==1) return 1
      else return fib(n-1)+fib(n-2);
            /* ricorsione non lineare*/
}
```

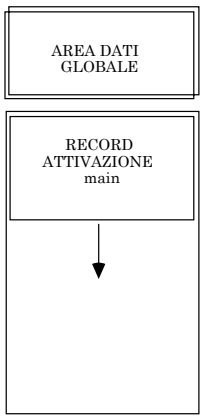
E` un esempio di ***ricorsione non lineare*** (più chiamate ricorsive nel corpo della funzione per determinare il valore restituito dalla funzione)

Ricorsione e Modello Run Time: ESEMPIO

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

```
main(){
  int x = 2, y;
  y = fattoriale(x);
}
```

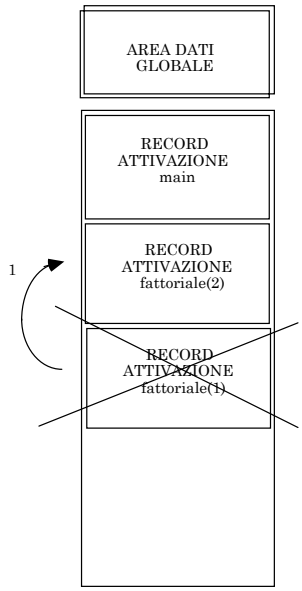
All'inizio dell'esecuzione:



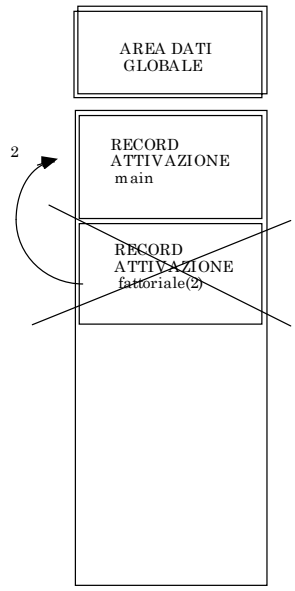
Dopo la prima attivazione della funzione fattoriale (fattoriale(2)):

Dopo la seconda attivazione (fattoriale(1)):

Termine della seconda attivazione (return):



Termine della prima attivazione (return):



Al termine della prima attivazione di fattoriale, viene restituito al main il valore 2, e questo stampa il risultato.

Ricorsione ed iterazione:

Versioni ricorsive

sono generalmente più vicine alla definizione matematica di certe funzioni:

- serie ricorsive, quali ad esempio i numeri di Fibonacci;
- funzioni matematiche ricorsive (fattoriale, etc.).

Versioni iterative

sono generalmente ***più efficienti*** di una soluzione ricorsiva (sia in termini di memoria che di tempo di esecuzione).

Schemi di (sotto)programmi ricorsivi del tipo:

$P(x_k)$	+	**if** caso base (x_0) **return** $expr_0$
		else expr($P(x_{k-1})$)

sono esprimibili con costrutti iterativi come segue:

$P(x_k)$	+	inizializzazione (caso base, x_0)
		while B $expr(x_{k-1})$

Calcolo del fattoriale (versione iterativa):

```
#include <stdio.h>

int fatt_it(unsigned int n);

main(void)
{   int n;

    printf("\nIntrodurre N:\t");
    scanf("%d",&n);

    printf("\nFattoriale di %d:\t%d\n",
              n, fatt_it(n));
}

int fatt_it(unsigned int n)
{
    int naux, f;
    f = 1;
    for (naux=1; naux<=n; naux++)
        f *= naux;
    return f;

}
```

Esercizio 2.1:

Scrivere la versione iterativa della procedura per il calcolo dell'n-esimo numero di Fibonacci.

```
int fib_it(unsigned int n)
{   unsigned int i,x=1,y=0,z;
    for(i=1;i<=n;i++)
       {   z = x;
           x += y;
           y=z;}
    return x;
}
```


Esercizio 2.2:

Scrivere una procedura PrintRev che legge in ingresso una sequenza di caratteri (terminata da '.') e stampa la sequenza al contrario:

AUTOMA.

AMOTUA

a) Definirne una versione ricorsiva senza utilizzare il tipo stringa.

```
#include <stdio.h>
#include <string.h>
void print_rev(char car);
main(void)
{   printf("\nIntrodurre una sequenza
           terminata da .:\t");
    print_rev(getchar());
}
void print_rev(char car);
{   if (car != '.')
       { print_rev(getchar());
         putchar(car);}
    else return;
}
```


b) PrintRev: versione ***iterativa*** con stringa (al max 30 caratteri).

```
#include <stdio.h>
#include <string.h>
#define MAXLEN 30

void print_rev_it(char word[])
{
    int i;
    for (i=strlen(word)-1; i>=0; i--)
    putchar(word[i]);
    return;
}

main()
{   int n;
    char parola[MAXLEN];
    printf("\nIntrodurre una parola:\t");
    scanf("%s",&parola);
    print_rev_it(parola);
}
```


c) PrintRev: versione ***ricorsiva*** con stringa.

```
#include <stdio.h>
#include <string.h>
#define MAXLEN 30

void print_rev(char word[], int i)
{
    if(strlen(word)-i>1) print_rev(word,i+1);
    putchar(word[i]);
    return;
}

main()
{
    int n;
    char parola[MAXLEN];
    printf("\nIntrodurre una parola:\t");
    scanf("%s",&parola);
    print_rev(parola,0);
}
```

Ricorsione Tail:

Si è in presenza di ***Tail Recursion*** quando la chiamata ricorsiva di una funzione/procedura *F* è l'ultima istruzione del codice di *F*.

Consente di ottimizzare lo spazio di memoria allocato sullo stack.

Esempio:

```
#include <stdio.h>
int f (int x, int y);
main()
{int n,m;
   printf("\nIntrodurre due numeri:\t");
   scanf("%d%d",&n,&m);
   printf("Somma di %d e %d:
                \t %d",n,m,f(n,m));
}

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)

La computazione che si origina tramite l'invocazione di una funzione tail-ricorsiva corrisponde ad un ***processo computazionale iterativo***.

Processo computazionale *ricorsivo*

- è caratterizzato da una catena di operazioni posticipate , il cui risultato è disponibile solo dopo che l'ultimo anello della catena si è concluso.

Processo computazionale *iterativo*

- ad ogni passo è disponibile una frazione del risultato.

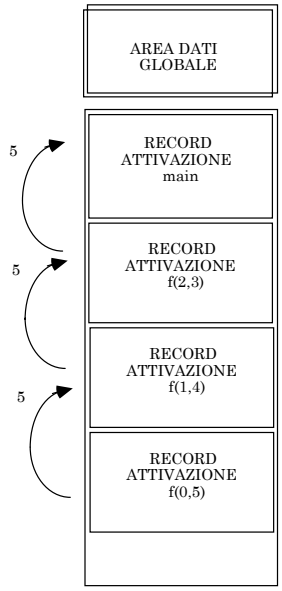
Siano `n=2; m=3`

Nel main: `printf(...,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.

In pratica, un compilatore in grado di ottimizzare l'occupazione dello stack potrebbe utilizzare il medesimo record di attivazione per tutte le attivazioni successive della funzione tail ricorsiva.



In alcuni casi è possibile scrivere versioni tail-ricorsive di funzioni e procedure.

Esempio fattoriale:

```
int fattoriale(unsigned int n)
{
   if (n==0) return 1;
   else
      if (n==1) return 1
      else return n*fattoriale(n-1);
                  /* ricorsione */
}
```

Versione tail-ricorsiva:

```
int fattoriale(unsigned int n)
{  return fatt_tail(1,n,1);}

int fatt_tail(unsigned int i,unsigned int n,
                                  long int f)
{  if (i<=n)  return fatt_tail(++i,n,f*i);
   else   return f;    }
```