Ricorsione

Prof. Orazio Mirabella

La ricorsione

- La ricorsione consiste nella possibilità di *definire una funzione mediante se stessa.*
- È basata sul ***principio di induzione matematica***:
  - ➔ se una proprietà P vale per $n=n_0$ → *CASO BASE*
  - ➔ e si può provare che, *assumendola valida per n*, allora vale per n+1, allora P vale per ogni $n >= n_0$
- Operativamente, risolvere un problema con un approccio ricorsivo comporta:
  - ➔ l'identificazione di un "caso base" ($n=n_0$) in cui la soluzione sia nota
  - ➔ la capacità di esprimere il caso generico n in termini dello stesso problema ma in uno o più casi più semplici (n-1, n-2, ecc).

La ricorsione: esempio

Esempio: il fattoriale di un numero naturale fact(n) = n!

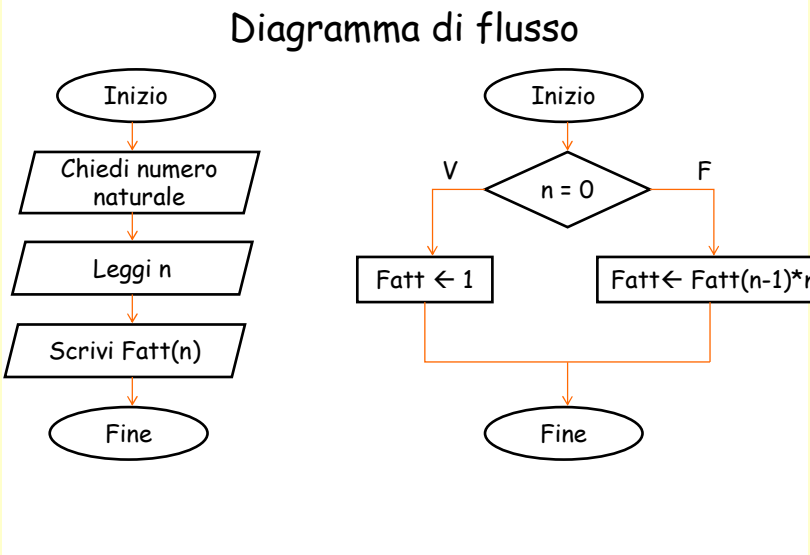
fact(n) : N → N

fact(n) = 1 se n = 0

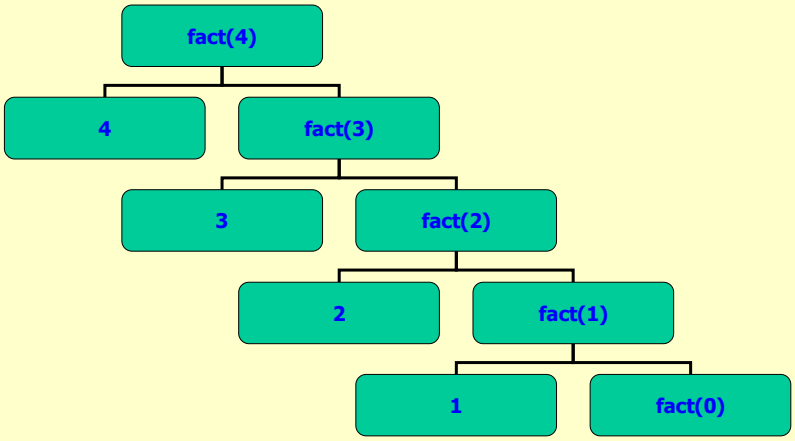
fact(n)= n * (n-1) * (n-2) ……* 2 +1

fact(n) = n * fact(n – 1) se n > 0

CALCOLO DEL FATTORIALE



Analisi dei passi



Analisi dei passi

- Analiticamente possiamo scrivere:

fact(4) = 4 * fact(3) =
= 4 * (3 * fact(2)) =
= 4 * (3 * (2 * fact(1))) =
= 4 * (3 * (2 * (1 * fact(0)))) =
= 4 * (3 * (2 * (1 * 1))) =
= … =
= 24

Ricorsione in C

In C è possibile definire funzioni *ricorsive*:

➔ Il corpo di ogni funzione ricorsiva contiene almeno una chiamata alla funzione stessa.

- Esempio: definizione in C della funzione ricorsiva *fattoriale*.

```
int fact(int n)
{
    if (n == 0) return 1;
    else return n * fact ( n - 1);
}
main()
{
    int fz;
    fz = fact(4);
}
```

Ricorsione in C

In C è possibile definire funzioni *ricorsive*:

➔ Il corpo di ogni funzione ricorsiva contiene almeno una chiamata alla funzione stessa.

- Esempio: definizione in C della funzione ricorsiva *fattoriale*.

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

CASO BASE

Ricorsione in C

In C è possibile definire funzioni *ricorsive*:

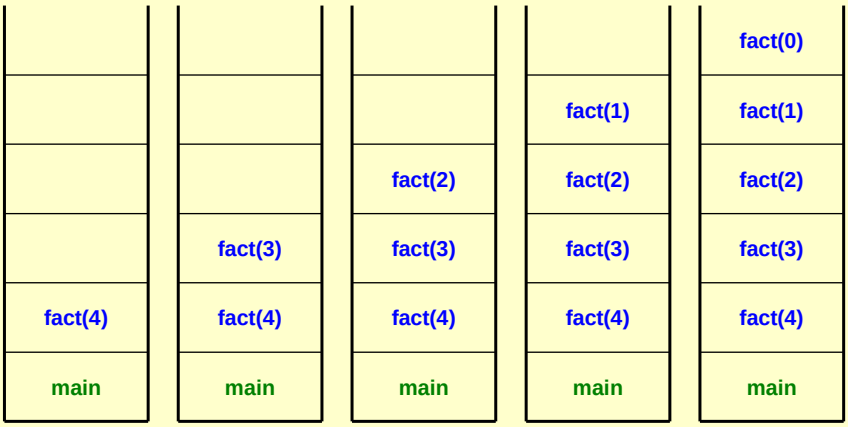
➔ Il corpo di ogni funzione ricorsiva contiene almeno una chiamata alla funzione stessa.

- Esempio: definizione in C della funzione ricorsiva *fattoriale*.

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

Passo ricorsivo

Implementazione della ricorsione

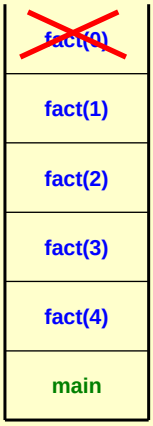


Implementazione della ricorsione

- Analiticamente:

fact(4) = 4 * fact(3) =
= 4 * (3 * fact(2)) =
= 4 * (3 * (2 * fact(1))) =
= 4 * (3 * (2 * (1 * fact(0)))) =
= 4 * (3 * (2 * (1 * 1)))
= … =
= 24

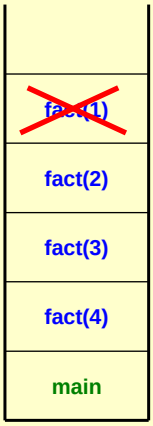
CASO BASE



Implementazione della ricorsione

- Analiticamente:

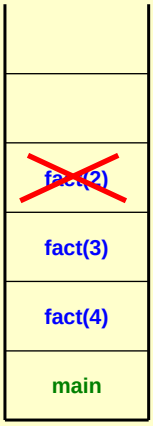
fact(4) = 4 * fact(3) =
= 4 * (3 * fact(2)) =
= 4 * (3 * (2 * fact(1))) =
= 4 * (3 * (2 * (1 * fact(0)))) =
= 4 * (3 * (2 * (1 * 1))) =
= … =
= 24



Implementazione della ricorsione

- Analiticamente:

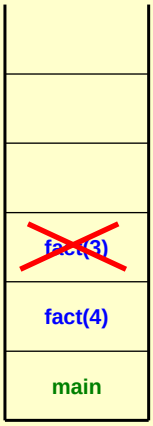
fact(4) = 4 * fact(3) =
= 4 * (3 * fact(2)) =
= 4 * (3 * (2 * fact(1))) =
= … =
= 4 * (3 * (2 * 1)) =
= … =
= 24



Implementazione della ricorsione

- Analiticamente:

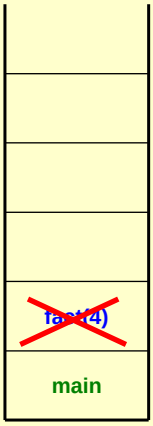
fact(4) = 4 * fact(3) =
= 4 * (3 * fact(2)) =
= … =
= 4 * (3 * 2) =
= … =
= 24



Implementazione della ricorsione

- Analiticamente:

fact(4) = 4 * fact(3) =
= … =
= 4 * 6 =
= 24

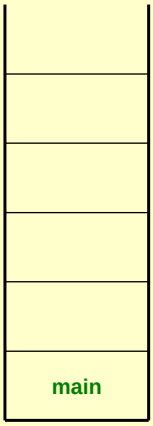


Implementazione della ricorsione

- Analiticamente:

```
int fact(int n)
{
    if (n == 0) return 1;
    else return n * fact ( n - 1);
}
main()
{
    int fz;
    fz = fact(4);
}
```

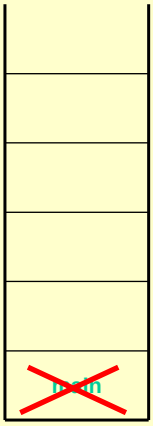
24



Implementazione della ricorsione

- Analiticamente:

```
int fact(int n)
{
    if (n == 0) return 1;
    else return n * fact ( n - 1);
}
main()
{
    int fz;
    fz = fact(4);
}
```



Ricorsione/Iterazione

- La ricorsione può essere molto inefficiente (dipende dal sistema operativo) perché ha un inconveniente: i calcoli vengono effettuati tutti alla fine, cioè il risultato può essere costruito solo dopo essere arrivati al caso base
- Mentre in un programma iterativo non si ha questo inconveniente
- Inoltre si ha un gran numero di chiamate al Sistema operativo poiché la funzione viene chiamata ripetutamente
- Salvataggio del contesto

Ricorsione/Iterazione

RICORSIONE

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

ITERAZIONE

```
int fact(int n)
{
    int i = 2;
    int F = 1;
    while (i <= n)
    {
        F = F * i;
        i++;
    }
    return F;
}
```

Programma per il calcolo del Fattoriale

- /* Calcolo del fattoriale con una funzione ricorsiva */

```
#include <stdio.h>
Int fat(int);
main()
{
int n;
printf("CALCOLO DI n!\n\n");
printf("Inser. n: \t");
scanf("%d", &n);
printf("Il fattoriale di: %d ha valore: %d\n", n, fat(n));
}
Int fat(int n)
{
if(n==0)
return(1);
else
return(n*fat(n-1));
}
```

Esempio

- La ricorsione può essere applicata a qualsiasi funzione che utilizzi assegnazioni, istruzioni if-else e while.
- si propone il programma per il calcolo della somma di una serie di numeri, terminanti con il valore nullo, scritto utilizzando una funzione ricorsiva al posto di una struttura ciclica:

```
int prossimo(void)
    {
    int questo,somm;
     printf("\nNumero da Sommare =");
     scanf("%d",&questo);
    if (questo){ /*1*/
          somm = questo+prossimo();
          return somm;
     }else
    return 0;
    }
```

Somma di una serie di numeri (termina con 0)

```
#include <stdio.h>

int prossimo(void);

main(){
  int somma;
  somma = prossimo();
  printf("\nSomma = %d",somma);
}
/* Funzione ricorsiva per il calcolo della somma */
int prossimo(void){
  int questo,somm;
  printf("\nNumero da Sommare =");
  scanf("%d",&questo);
  if (questo){        /*1*/
    somm = questo+prossimo();
    return somm;
  }else
    return 0;
}
```

Funzione potenza

- Funzione potenza(int x, int n), che calcola x^n.

```
// iterativa:
int potenza(int x, int n)
{
int i;
int ris = 1;
for (i = 0; i < n; ++i) ris *= x;
return ris;
}
```

- // ricorsiva: sfrutta l'uguaglianza x^n = x * x^(n-1), con x^0 = 1.

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

RICORSIVITÀ MUTUA

- In una funzione **x** può essere presente una chiamata a una funzione **y**. Se y a sua volta contiene una chiamata a x, si ha una **ricorsività indiretta o mutua ricorsione.**
- Nel processo di mutua ricorsività possono essere coinvolte più di due funzioni che indirettamente invocano se stesse, tramite una successione di chiamate: dalla funzione x viene chiamata y, da y t, da t z, da z ancora x.

RICORSIVITÀ MUTUA