Ricorsione

Francesco Isgrò

Un problema semplice

Calcolare la somma dei primi n numeri

- $S(n) = 1 + 2 + \ldots + (n-1) + n$

Un problema semplice

Calcolare la somma dei primi n numeri

$$S(n) = 1 + 2 + \ldots + (n-1) + n$$

- Forma chiusa

$$S(n) = (n*(n+1))/2$$

Un problema semplice

Calcolare la somma dei primi n numeri

$$S(n) = 1 + 2 + \ldots + (n-1) + n$$

- Forma chiusa

$$S(n) = (n*(n+1))/2$$

- Iterativa

```
s = 0;
for (i=0; i<n; i++)
    s += i;
```

- $S(1) = 1$

- $S(1) = 1$
- $S(2) = 1 + 2 = S(1) + 2$

- $S(1) = 1$
- $S(2) = 1 + 2 = S(1) + 2$
- $S(3) = 1 + 2 + 3 = (1 + 2) + 3 = S(2) + 3$

- $S(1) = 1$
- $S(2) = 1 + 2 = S(1) + 2$
- $S(3) = 1 + 2 + 3 = (1 + 2) + 3 = S(2) + 3$
- $S(4) = 1 + 2 + 3 + 4 = (1 + 2 + 3) + 4 = S(3) + 4$

- $S(1) = 1$
- $S(2) = 1 + 2 = S(1) + 2$
- $S(3) = 1 + 2 + 3 = (1 + 2) + 3 = S(2) + 3$
- $S(4) = 1 + 2 + 3 + 4 = (1 + 2 + 3) + 4 = S(3) + 4$
- $S(5) = 1 + 2 + 3 + 4 + 5 = (1 + 2 + 3 + 4) + 5 = S(4) + 5$
- …

- $S(1) = 1$
- $S(2) = 1 + 2 = S(1) + 2$
- $S(3) = 1 + 2 + 3 = (1 + 2) + 3 = S(2) + 3$
- $S(4) = 1 + 2 + 3 + 4 = (1 + 2 + 3) + 4 = S(3) + 4$
- $S(5) = 1 + 2 + 3 + 4 + 5 = (1 + 2 + 3 + 4) + 5 = S(4) + 5$
- …
- $S(n) = 1 + \ldots + (n-1) + n = S(n-1) + n$

- Un problema complesso viene decomposto in istanze più semplici
- Si trovano le soluzioni dei problemi semplici.
- Le soluzioni vengono ricombinate per ottenere la soluzione.

- Ricorsione associata al paradigma divide et impera
  - **divide** il problema di dimensione n viene diviso in $a \geq 1$ sottoproblemi di uguale natura, indipendenti, e di dimensione $m < n$
  - **impera** il problema è talmente semplice che può essere risolto direttamente
  - **ricombina** la soluzione complessiva viene ricostruita ricombinando le soluzioni ai sottoproblemi man mano generati

- $S(1) = 1$
- $S(2) = 1 + 2 = S(1) + 2$
- $S(3) = 1 + 2 + 3 = (1 + 2) + 3 = S(2) + 3$
- $S(4) = 1 + 2 + 3 + 4 = (1 + 2 + 3) + 4 = S(3) + 4$
- $S(5) = 1 + 2 + 3 + 4 + 5 = (1 + 2 + 3 + 4) + 5 = S(4) + 5$
- …
- $S(n) = 1 + \ldots + (n-1) + n = S(n-1) + n$

```
Risolvi(Problema)
  Se il problema e elementare:
    Soluzione = Risolvi_banale(Problema);
  Altrimenti
    Sottoproblema_1, 2, ....., a = Dividi(Problema);
    Per ciascun Sottoproblema_i:
      Sottosoluzione_i = Risolvi(Sottoproblema_i);
    Soluzione = Combina(Sottosoluzione_1, ....., Sottosoluzione_a);
  Return Soluzione
```

Proviamo a definire: funzione ricorsiva

- Una funzione si dice ricorsiva se all’interno della sua definizione vi è una chiamata
  - alla funzione stessa (ricorsione diretta)
  - ad una funzione che chiama la funzione stessa (ricorsione indiretta)

Proviamo a definire: soluzione ricorsiva

- La soluzione di un problema S applicato ai dati D è ricorsiva se si può esprimere come
  - $S(D) = f(S(D'), D)$ con $D \neq D_0$
  - $S(D_0) = S_0$
- D' è più *semplice* di D
- D_0 è una istanza talmente semplice che la soluzione S_0 è nota.
- $S(D_0) = S_0$ viene detta condizione di terminazione o caso base

Somma primi n interi

- $D' = n - 1$
- $D_0 = 1$ con $S_0 = 1$

Somma primi n interi

- D’ = n – 1
- $D_0 = 1$ con $S_0 = 1$

```
int sommaN(int n)
{
   if (n == 1) return 1;
   return (n + sommaN(n-1));
}
```

Algoritmo ricorsivo

- Un algoritmo ricorsivo per la risoluzione di un dato problema deve essere definito nel modo seguente:

Algoritmo ricorsivo

- Un algoritmo ricorsivo per la risoluzione di un dato problema deve essere definito nel modo seguente:
    1. prima si definisce come risolvere dei problemi analoghi a quello di partenza, ma che hanno dimensione piccola e possono essere risolti in maniera estremamente semplice (detti casi base);

Algoritmo ricorsivo

- Un algoritmo ricorsivo per la risoluzione di un dato problema deve essere definito nel modo seguente:
  1. prima si definisce come risolvere dei problemi analoghi a quello di partenza, ma che hanno dimensione piccola e possono essere risolti in maniera estremamente semplice (detti casi base);
  2. poi si definisce come ottenere la soluzione del problema di partenza combinando la soluzione di uno o più problemi analoghi, ma di dimensione inferiore.

Fattoriale

- Dato un intero n>0 si definisce fattoriale
  - $fact(n) = n! = n * (n-1) * (n-2) * \ldots * 2 * 1$
- Il fattoriale può facilmente essere calcolato in maniera iterativa

Fattoriale

- Dato un intero n>0 si definisce fattoriale
  - fact(n) = n! = n * (n-1) * (n-2) * … * 2 * 1
- Il fattoriale può facilmente essere calcolato in maniera iterativa

```
fact = 1;
for (i=1; i<=n; i++) {
   fact = fact*i;
}
```

Fattoriale

- Dato un intero n>0 si definisce fattoriale
  - fact(n) = n! = n * (n-1) * (n-2) * … * 2 * 1
- Oppure possiamo procedere in maniera ricorsiva

Fattoriale

- Dato un intero n>0 si definisce fattoriale
  - $fact(n) = n! = n * (n-1) * (n-2) * \ldots * 2 * 1$
- Oppure possiamo procedere in maniera ricorsiva
  - Caso base

Fattoriale

- Dato un intero n>0 si definisce fattoriale
  - fact(n) = n! = n * (n-1) * (n-2) * … * 2 * 1
- Oppure possiamo procedere in maniera ricorsiva
  - Caso base
    - fact(0) = 1
    - fact(1) = 1

Fattoriale

- Dato un intero n>0 si definisce fattoriale
  - fact(n) = n! = n * (n-1) * (n-2) * … * 2 * 1
- Oppure possiamo procedere in maniera ricorsiva
  - Caso base
    - fact(0) = 1
    - fact(1) = 1
  - Caso generale (n > 1)

Fattoriale

- Dato un intero n>0 si definisce fattoriale
  - fact(n) = n! = n * (n-1) * (n-2) * … * 2 * 1
- Oppure possiamo procedere in maniera ricorsiva
  - Caso base
    - fact(0) = 1
    - fact(1) = 1
  - Caso generale (n > 1)
    - fact(n) = n*fact(n-1)

Fattoriale

- Dato un intero n>0 si definisce fattoriale
  - fact(n) = n! = n * (n-1) * (n-2) * … * 2 * 1
- Oppure possiamo procedere in maniera ricorsiva
  - Caso base
    - fact(0) = 1
    - fact(1) = 1
  - Caso generale (n > 1)
    - fact(n) = n*fact(n-1)

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

Come funziona la ricorsione

$$5! = 5 * 4! = 120$$

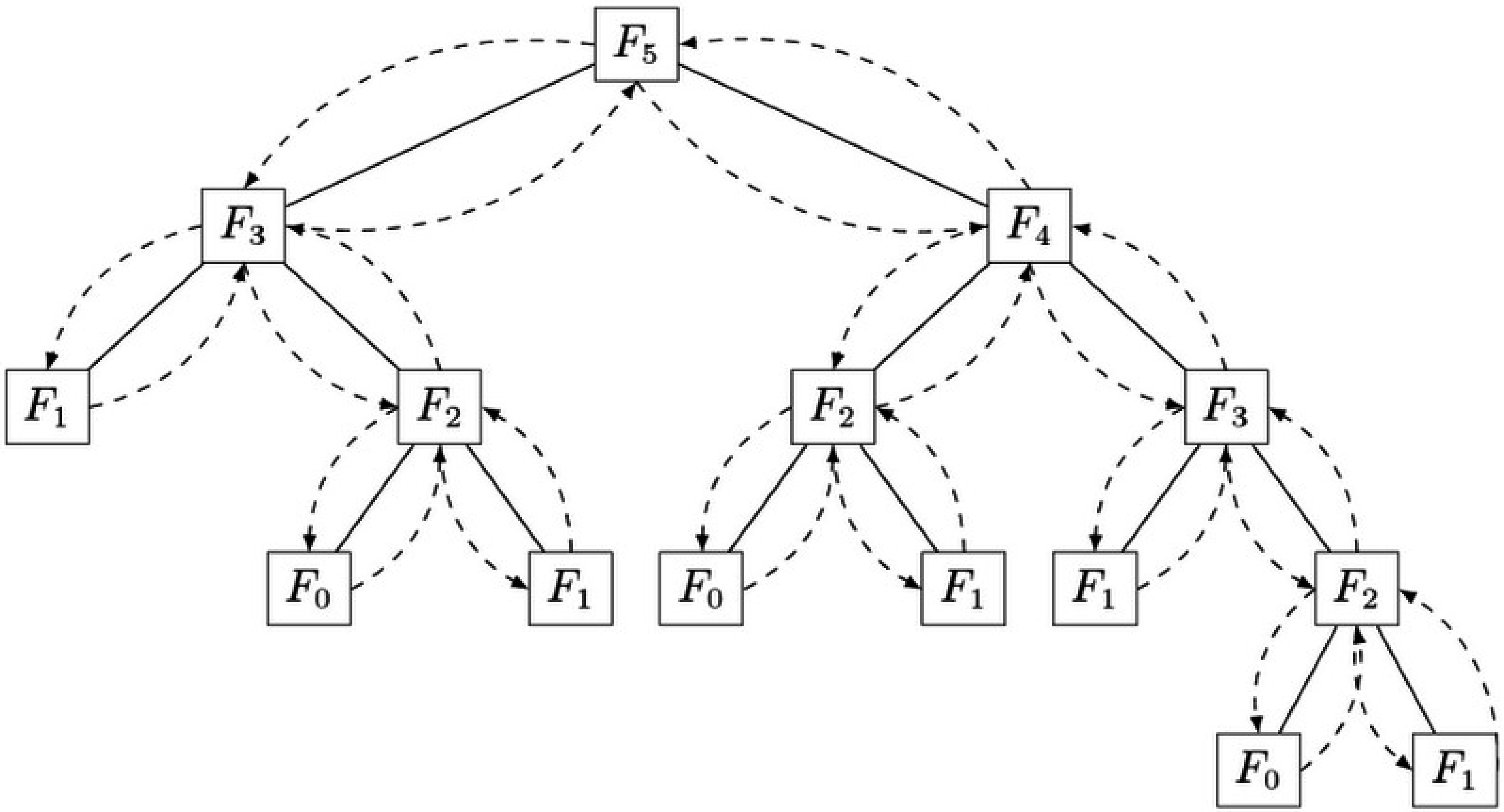
$$4! = 4 * 3! = 24$$

$$3! = 3 * 2! = 6$$

$$2! = 2 * 1! = 2$$

$$1! = 1 * 0! = 1$$

$$0! = 1$$



F_5
F_3
F_4
F_1
F_2
F_2
F_3
F_0
F_1
F_0
F_1
F_1
F_2
F_0
F_1

- Cosa succede quando una funzione generica viene chiamata dal main
  1. si crea una nuova istanza della funzione chiamata
  2. si alloca la memoria per i parametri e le variabili locali
  3. si passano i parametri
  4. il controllo passa dal main alla funzione chiamata
  5. quando la funzione termina il controllo ritorna al main che esegue l’istruzione successiva

Stack

- E’ possibile che una funzione ne chiami un’altra.
- Serve un meccanismo per gestire le chiamate annidate: lo *stack*.
- Sullo stack sono definite due operazioni
  - *push*: inserimento dell’oggetto in cima allo stack
  - pop: prelievo e cancellazione dalla cima dell’oggetto inserito più di recente
- La strategia di gestione dello stack è detta LIFO (Last In First Out).

Stack frame

- Si chiama *stack frame* la struttura che contiene almeno
  - i parametri formali
  - le variabili locali
  - l’indirizzo a cui si ritornerà una volta terminata l’esecuzione della funzione
  - il puntatore al codice della funzione

- Lo stack frame viene creato alla chiamata della funzione e distrutto al suo termine.
- Gli stack frame sono memorizzati nello stack di sistema
- Lo stack di sistema è finito
- Quando si supera lo spazio allocato si ha stack overflow
- Lo stack pointer contiene l’indirizzo al primo stack frame disponibile

```
int f1(int x);
int f2(int x);

int main()
{
   int x, a=10;

   x = f1(a);
   printf("x = %d\n",x);
}

int f1(int x)
{
   return 2*f2(x);
}

int f2(int x)
{
   return x+1;
}
```

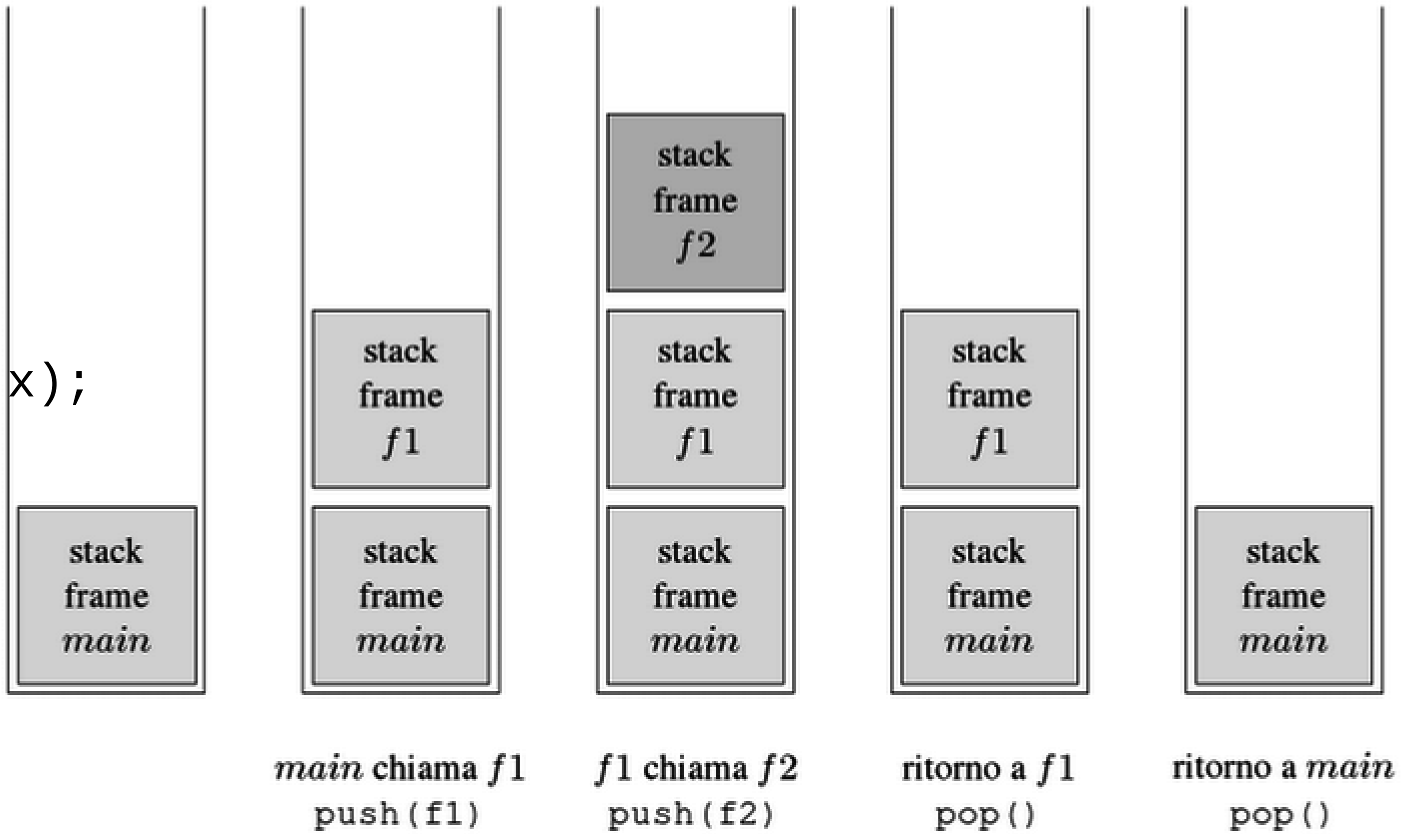
```c
int f1(int x);
int f2(int x);

int main()
{
   int x, a=10;

   x = f1(a);
   printf("x = %d\n",x);
}

int f1(int x)
{
   return 2*f2(x);
}

int f2(int x)
{
   return x+1;
}
```



Stack frame per il fattoriale