Linguaggio C:
Stack e Ricorsione

FUNZIONI: IL MODELLO A RUN-TIME

Ogni volta che viene invocata una funzione:

- si crea di una nuova **attivazione** (istanza) del servitore
- viene **allocata la memoria** per i parametri e per le variabili locali
- si effettua il **passaggio dei parametri**
- si **trasferisce il controllo** al servitore
- si **esegue il codice** della funzione

Record di Attivazione

- Al momento dell'invocazione:

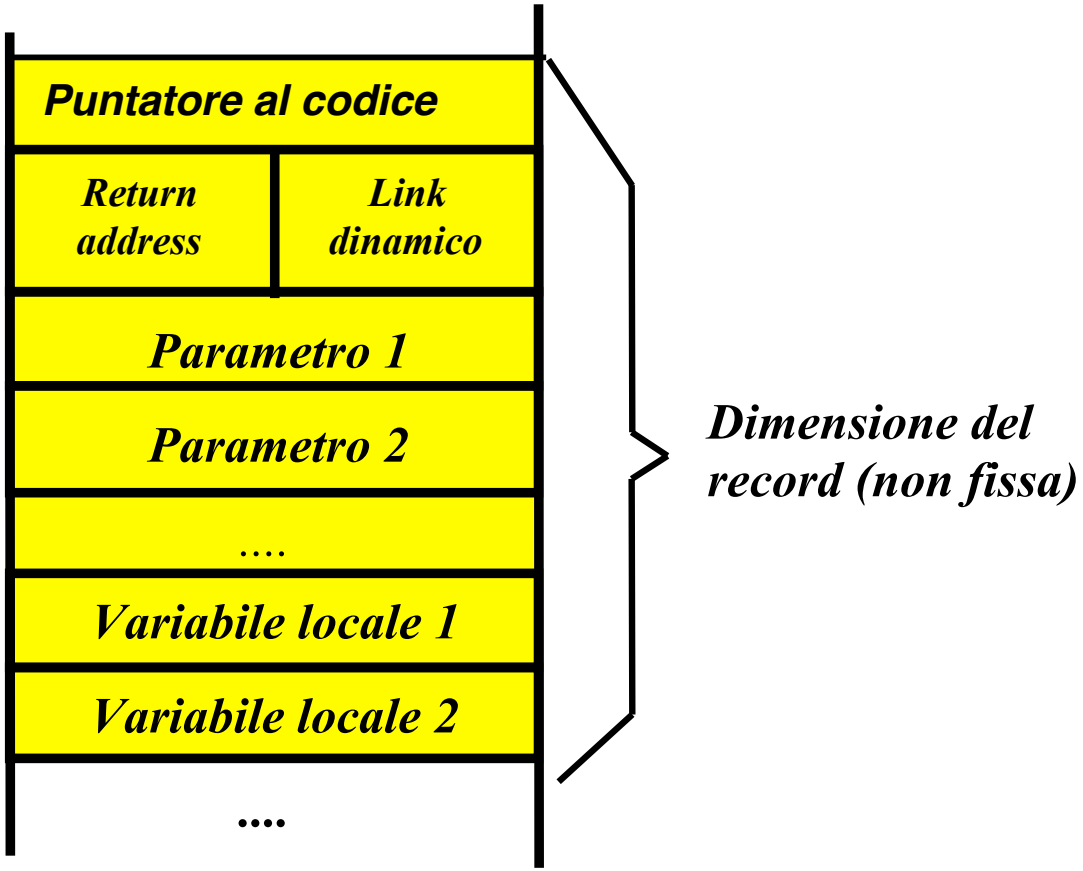
  viene creata dinamicamente una struttura dati che contiene i binding dei parametri e degli identificatori definiti localmente alla funzione detta **RECORD DI ATTIVAZIONE.**

Record di Attivazione

È il "***mondo della funzione***": contiene tutto ciò che serve per la chiamata alla quale e` associato:

- i **parametri** formali
- le **variabili locali**
- l'**indirizzo di ritorno** (***Return address RA***) che indica il punto a cui tornare (nel codice del cliente) al termine della funzione, per permettere al cliente di proseguire una volta che la funzione termina.
- un collegamento al record di attivazione del cliente (*Link Dinamico DL*)
- l'***indirizzo del codice*** della funzione (puntatore alla prima istruzione del corpo)

Record di Attivazione



Record di Attivazione

- Il record di attivazione associato a una chiamata di una funzione f:
  - è creato al momento della invocazione di f
  - permane per tutto il tempo in cui la funzione f è in esecuzione
  - è distrutto (*deallocato*) al termine dell'esecuzione della funzione stessa.
- Ad ogni chiamata di funzione viene *creato un nuovo record, specifico per quella chiamata di quella funzione*
- La dimensione del record di attivazione
  - varia da una funzione all'altra
  - *per una data funzione, è fissa e calcolabile a priori*

Record di Attivazione

- *Funzioni che chiamano altre funzioni* danno luogo a una *sequenza* di record di attivazione
  - allocati secondo l'ordine delle chiamate
  - deallocati in ordine inverso

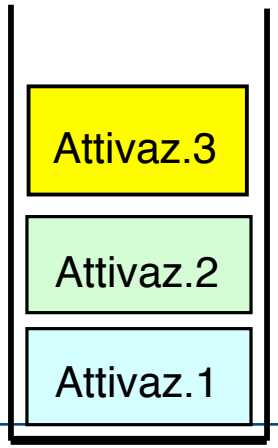
- La sequenza dei link dinamici costituisce la cosiddetta *catena dinamica*, che rappresenta *la storia delle attivazioni ("chi ha chiamato chi")*

Stack

L'area di memoria in cui vengono allocati i record di attivazione viene gestita *come una* ***pila*** :

STACK

E` una struttura dati gestita a tempo di esecuzione con politica *LIFO* (***Last In, First Out*** - l'ultimo a entrare è il primo a uscire) nella quale ogni elemento e` un record di attivazione.



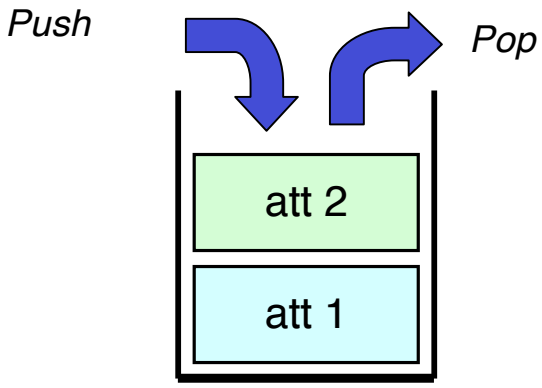
La gestione dello stack avviene mediante due operazioni:

push: aggiunta di un elemento (in cima alla pila)

pop: prelievo di un elemento (dalla cima della pila)

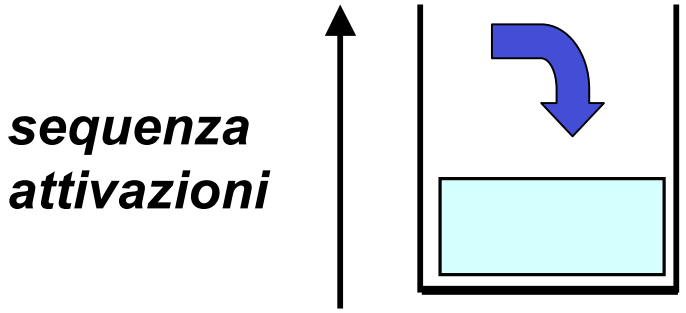
Stack

- L'ordine di collocazione dei record di attivazione nello stack indica la cronologia delle chiamate:

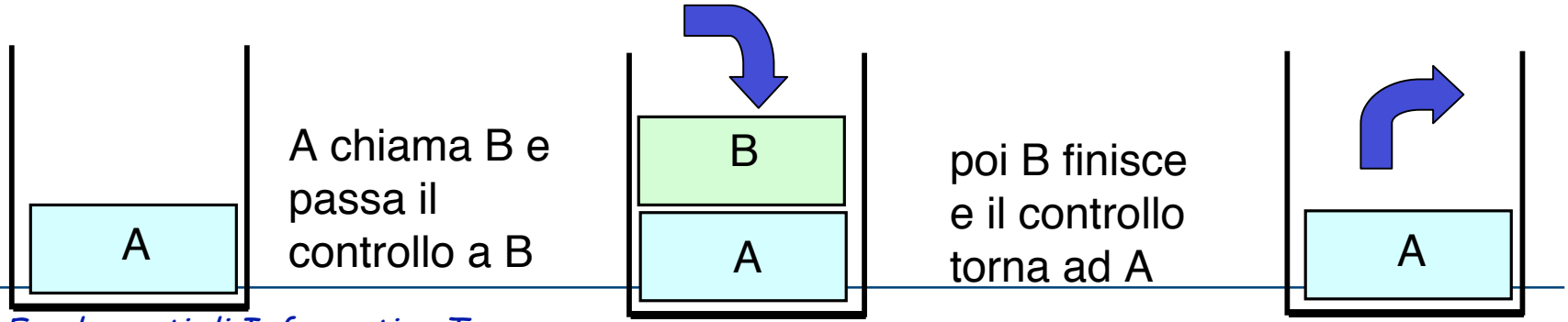


Record di Attivazione

- Normalmente lo STACK dei record di attivazione si disegna nel modo seguente:



- Quindi, se la funzione A chiama la funzione B, lo stack evolve nel modo seguente



Esempio: chiamate annidate

Programma:

```
int R(int A) { return A+1; }
int Q(int x) { return R(x); }
int P(void) { int a=10; return Q(a); }
main() { int x = P(); }
```

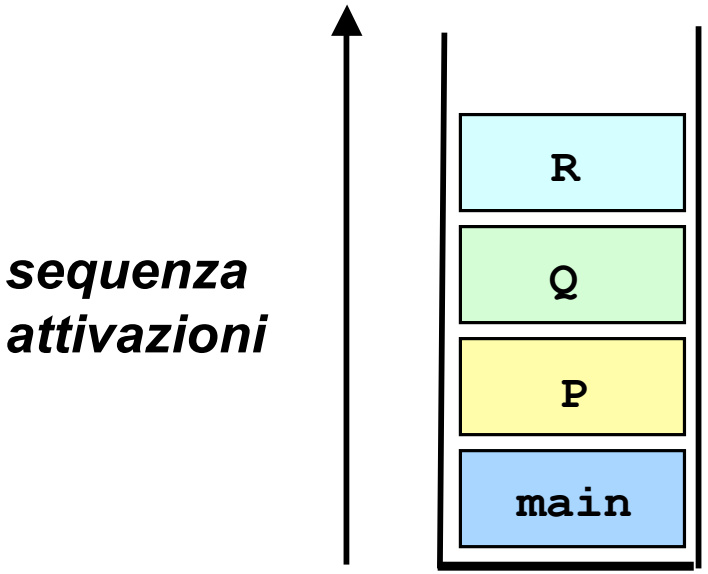
Sequenza chiamate:

S.O. → **main** → **P()** → **Q()** → **R()**

Esempio: chiamate annidate

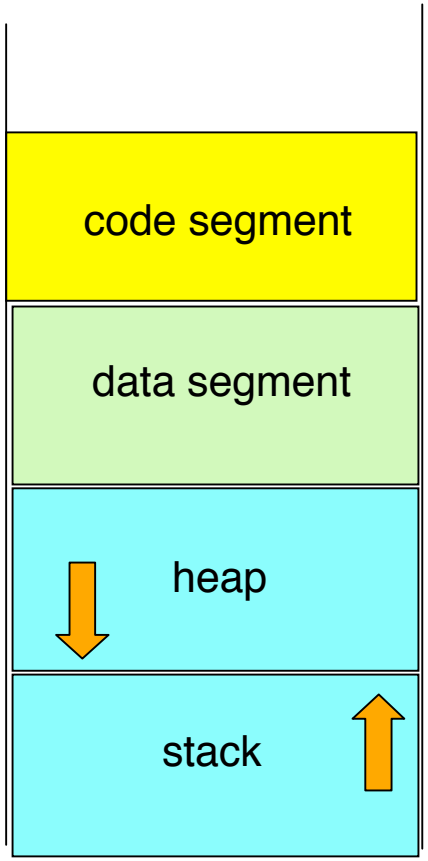
Sequenza chiamate:

S.O. → `main` → `P()` → `Q()` → `R()`



Spazio di indirizzamento

La memoria allocata a ogni programma in esecuzione e` suddivisa in varie parti (segmenti), secondo lo schema seguente:



- code segment: contiene il codice eseguibile del programma
- data segment: contiene le variabili globali
- heap: contiene le variabili dinamiche
- stack: e` l'area dove vengono allocati i record di attivazione

➔ ***Code*** segment e ***data*** segment sono di dimensione fissata staticamente (a tempo di compilazione).

➔ La dimensione dell'area associata a ***stack*** + ***heap*** e` fissata staticamente: man mano che lo stack cresce, diminuisce l'area a disposizione dell'heap, e viceversa.

La ricorsione

- Una funzione matematica è definita ***ricorsivamente*** quando nella sua definizione compare un riferimento a se stessa

- 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 $n>=n_0$

La ricorsione

- **Operativamente, risolvere un problema con un approccio ricorsivo comporta**
  - **di identificare un "caso base" (n=n0) in cui la soluzione sia nota**
  - **di riuscire a esprimere la soluzione al caso generico *n* in termini dello *stesso problema in uno o più casi più semplici*** (n-1, n-2, etc).

La ricorsione: esempio

Esempio: il fattoriale di un numero naturale

`fact(n) = n!`

```
n!: N → N
  { n! vale 1          se n == 0
  { n! vale n*(n-1)!   se n > 0
```

Ricorsione in C

In C e` 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);
}
```

Ricorsione in C: esempio

- **Servitore & Cliente**: `fact` e` sia servitore che cliente (di se stessa):

```
int fact(int n)
{  if (n==0) return 1;
       else return n*fact(n-1);
}
main()
{      int fz,f6,z = 5;
       fz = fact(z-2);
   }
```

Ricorsione in C: esempio

- **Servitore & Cliente:**

```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
main(){
    int fz,f6,z = 5;
    fz = fact(z-2);
}
```

Si valuta l'espressione che costituisce il parametro attuale (nell'environment del main) e si trasmette alla funzione fatt una copia del valore così ottenuto (3).

Ricorsione in C: esempio

La funzione fact lega il parametro n a 3. Essendo 3 positivo si passa al ramo else. Per calcolare il risultato della funzione e' necessario effettuare una nuova chiamata di funzione `fact(2)`

- **Servitore & Cliente:**

```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
main(){
    int fz,f6,z = 5;
    fz = fact(z-2);
    }
```

Ricorsione in C: esempio

- **Servitore & Cliente:**

```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
main(){
    int fz,f6,z = 5;
    fz = fact(z-2);
    }
```

La funzione fact lega il parametro n a 3. Essendo 3 positivo si passa al ramo else. Per calcolare il risultato della funzione e' necessario effettuare una nuova chiamata di funzione. n-1 nell'environment di fact vale 2 quindi viene chiamata `fact(2)`

Ricorsione in C: esempio

- **Servitore & Cliente:**

```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
main(){
    int fz,f6,z = 5;
    fz = fact(z-2);
    }
```

Il nuovo servitore lega il parametro n a 2. Essendo 2 positivo si passa al ramo else. Per calcolare il risultato della funzione e' necessario effettuare una nuova chiamata di funzione. n-1 nell'environment di fact vale 1 quindi viene chiamata `fact(1)`

Ricorsione in C: esempio

Il nuovo servitore lega il parametro n a 1. Essendo 1 positivo si passa al ramo else. Per calcolare il risultato della funzione e' necessario effettuare una nuova chiamata di funzione. n-1 nell'environment di fact vale 0 quindi viene chiamata `fact(0)`

- **Servitore & Cliente:**

```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
main(){
    int fz,f6,z = 5;
    fz = fact(z-2);
    }
```

Ricorsione in C: esempio

Il nuovo servitore lega il parametro n a 0. La condizione n <=0 e' vera e la funzione `fact(0)` *torna come risultato 1 e termina.*

- **Servitore & Cliente:**

```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
main(){
    int fz,f6,z = 5;
    fz = fact(z-2);
    }
```

Ricorsione in C: esempio

Il controllo torna al servitore precedente `fact(1)` *che puo' valutare l'espressione n * 1 (valutando n nel suo environment dove vale 1) ottenendo come risultato 1 e terminando.*

- **Servitore & Cliente:**

```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
main(){
    int fz,f6,z = 5;
    fz = fact(z-2);
    }
```

Ricorsione in C: esempio

Il controllo torna al servitore precedente `fact(2)` *che puo' valutare l'espressione n * 1 (valutando n nel suo environment dove vale 2) ottenendo come risultato 2 e terminando.*

- **Servitore & Cliente:**

```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
main(){
    int fz,f6,z = 5;
    fz = fact(z-2);
    }
```

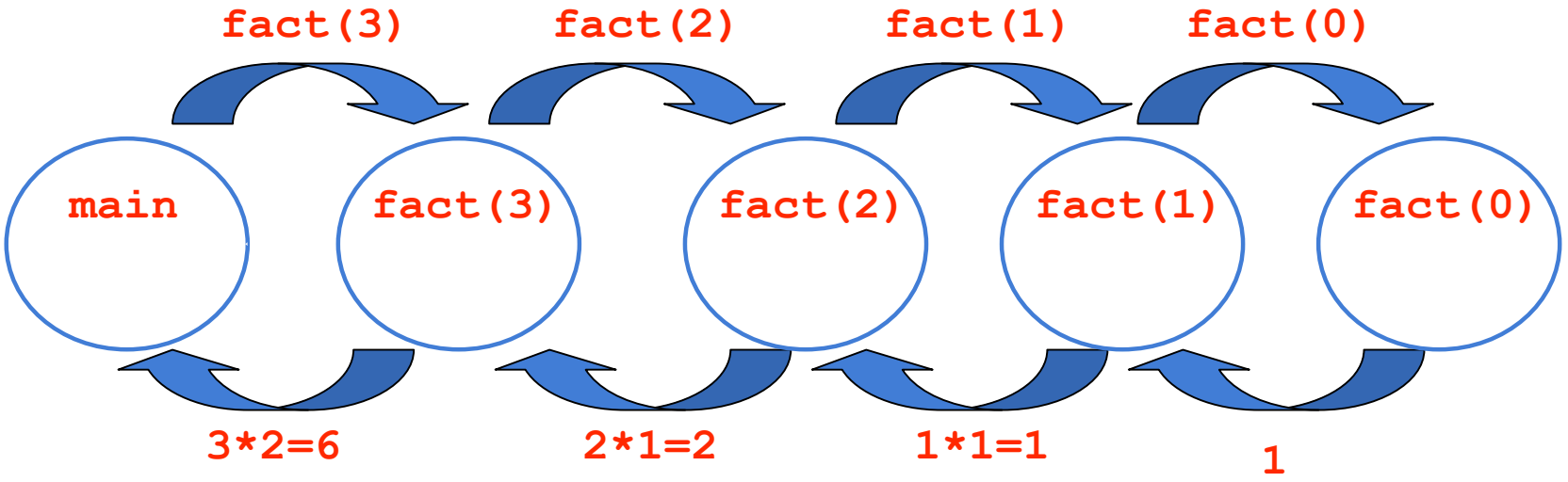
Ricorsione in C: esempio

Il controllo torna al servitore precedente `fact(3)` *che puo' valutare l'espressione n * 2 (valutando n nel suo environment dove vale 3) ottenendo come risultato 6 e terminando.*
IL CONTROLLO PASSA AL MAIN CHE ASSEGNA A fz IL VALORE 6

- **Servitore & Cliente:**

```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
main(){
    int fz,f6,z = 5;
    fz = fact(z-2);
    }
```

Ricorsione in C: esempio



main	fact(3)	fact(2)	fact(1)	fact(0)
Cliente di fact(3)	Cliente di fact(2) Servitore del main	Cliente di fact(1) Servitore di fact(3)	Cliente di fact(0) Servitore di fact(2)	Servitore di fact(1)

Cosa succede nello stack ?

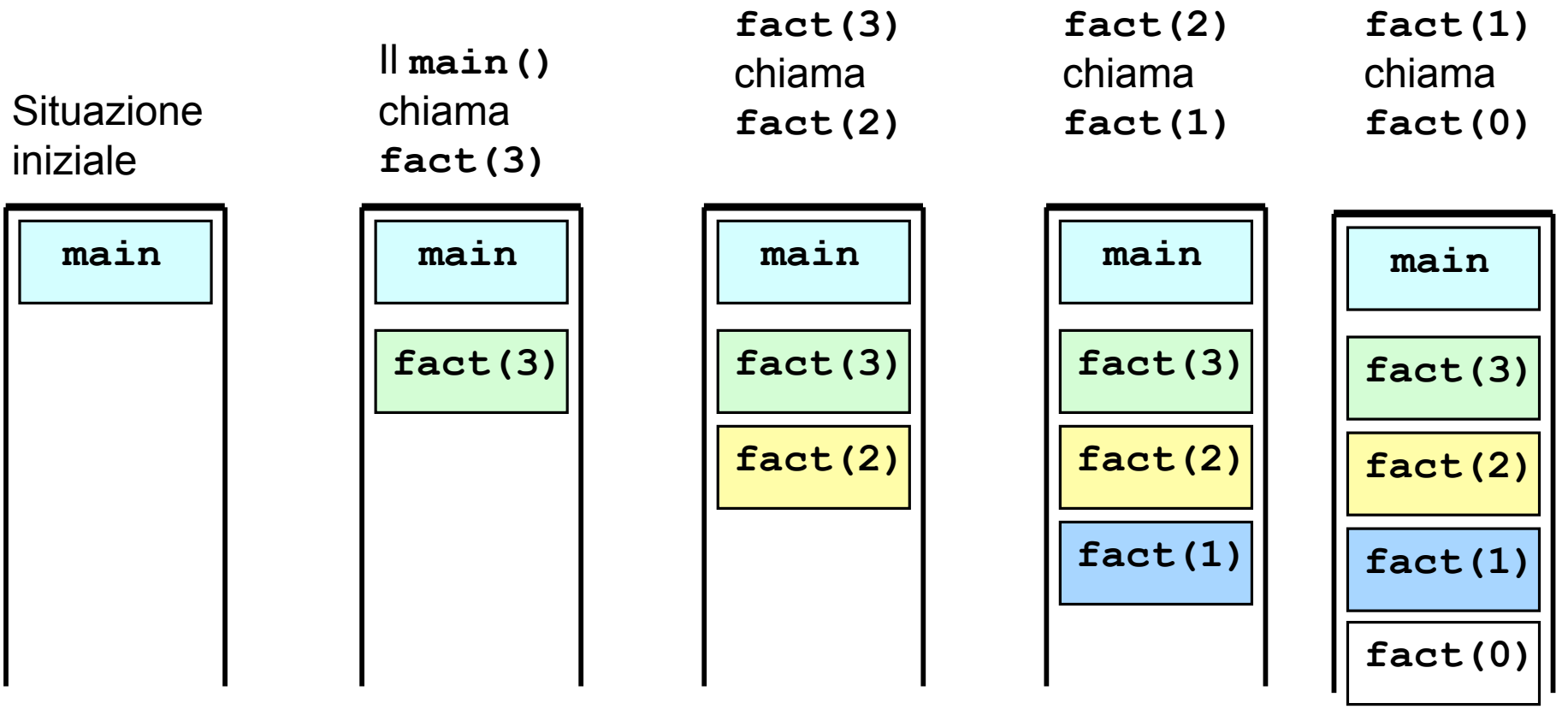
```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
                          }

main(){
    int fz,f6,z = 5;
    fz = fact(z-2);
    }
```

NOTA: Anche il **`main()`** e' una funzione

Seguiamo l'evoluzione dello stack durante l'esecuzione:

Cosa succede nello stack ?



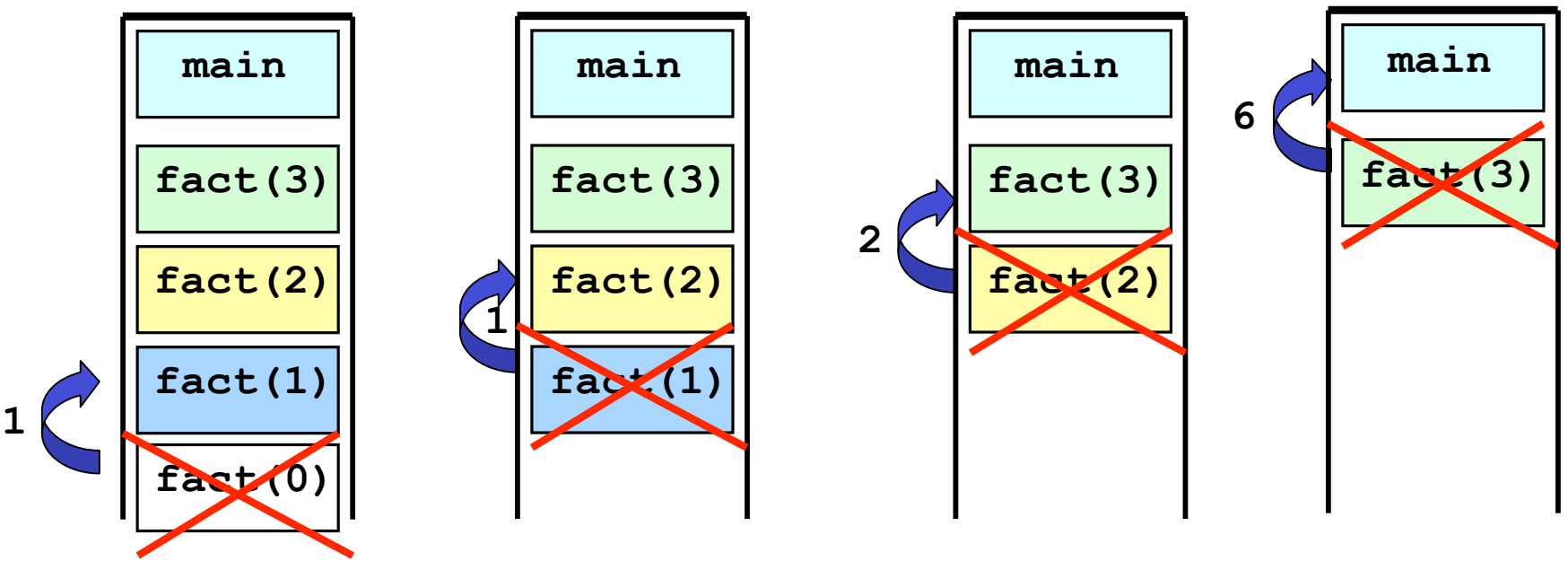
Cosa succede nello stack ?

`fact(0)` termina restituendo il valore 1. Il controllo torna a `fact(1)`

fact(1) effettua la moltiplicazione e termina restituendo il valore 1. Il controllo torna a fact(2)

fact(2) effettua la moltiplicazione e termina restituendo il valore 2. Il controllo torna a fact(3)

fact(6) effettua la moltiplicazione e termina restituendo il valore 6. Il controllo torna al main.



Calcolo iterativo del fattoriale

- Il fattoriale puo` essere anche calcolato mediante un'algoritmo iterativo:

```
int fact(int n){
      int i;
      int F=1; /*inizializzazione del fattoriale*/
      for (i=2;i <= n; i++)
        F=F*i;
      return F;
      }
```

DIFFERENZA CON LA VERSIONE RICORSIVA: ad ogni passo viene accumulato un risultato intermedio

Calcolo iterativo del fattoriale

```
int fact(int n){
      int i;
      int F=1; /*inizializzazione del fattoriale*/
      for (i=2;i <= n; i++)
            F=F*i;
      return F;
      }
```

La variabile F accumula risultati intermedi: se n = 3 inizialmente `F=1` *poi al primo ciclo for* `i=2` `F` *assume il valore 2. Infine all'ultimo ciclo for* `i=3` `F` *assume il valore 6.*

- Al primo passo F accumula il fattoriale di 1
- Al secondo passo F accumula il fattoriale di 2
- Al i-esimo passo F accumula il fattoriale di i

Processo computazionale iterativo

- Nell'esempio precedente il risultato viene sintetizzato *“in avanti”*
- L'esecuzione di un algoritmo di calcolo che computi “in avanti”, per accumulo, e` un *processo computazionale iterativo.*
- La caratteristica fondamentale di un processo computazionale *iterativo* è che *a ogni passo è disponibile un risultato parziale*
  - dopo k passi, si ha a disposizione il risultato parziale relativo al caso k
  - questo *non è vero* nei processi computazionali *ricorsivi,* in cui nulla è disponibile finché non si è giunti fino al caso elementare.