# 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

# IL MODELLO A RUN-TIME: ENVIRONMENT

- La definizione di una funzione introduce un *nuovo binding* nell'environment in cui la funzione è definita (C: *global environment*)

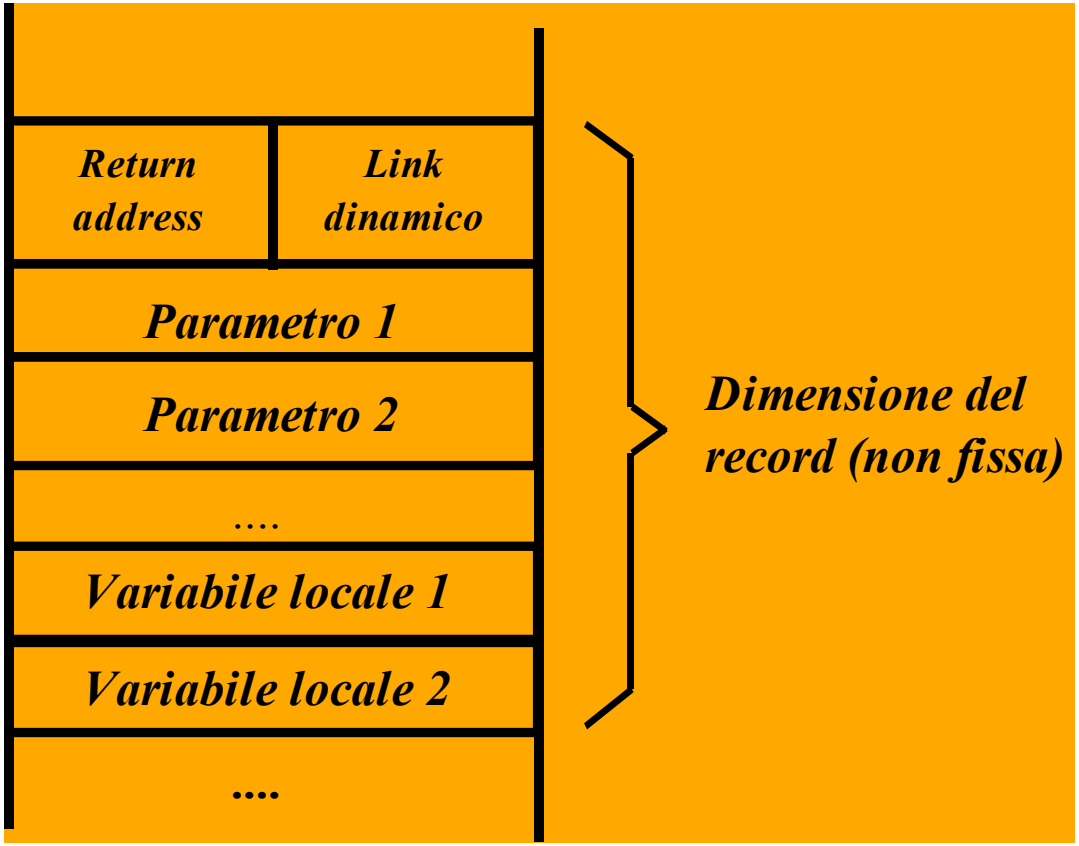
- Al momento dell'*invocazione*, si crea un *nuovo environment*
  - viene creata una struttura dati che contiene i *binding* dei parametri e degli identificatori definiti localmente alla funzione detta *RECORD DI ATTIVAZIONE*

# RECORD DI ATTIVAZIONE

**È il "*mondo della funzione*": *contiene tutto ciò che ne caratterizza l'esistenza***

- **i parametri ricevuti**
- **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)***

# RECORD DI ATTIVAZIONE



# RECORD DI ATTIVAZIONE

- Rappresenta il "*mondo della funzione*": *nasce e muore con essa*
  - è creato al momento della invocazione di una funzione
  - permane per tutto il tempo in cui la funzione 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

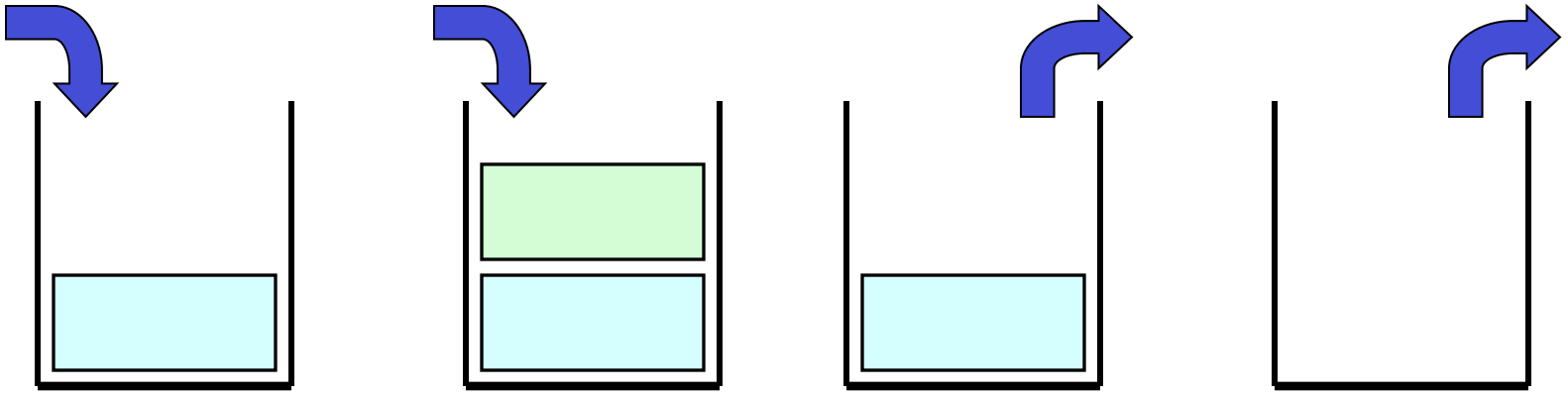
- La sequenza dei link dinamici costituisce la cosiddetta *catena dinamica*, che rappresenta *la storia delle attivazioni*
  *("chi ha chiamato chi")*

# RECORD DI ATTIVAZIONE

- Per catturare la semantica delle chiamate annidate (una funzione che chiama un'altra funzione che...), l'area di memoria in cui vengono allocati i record di attivazione deve essere gestita *come una pila*

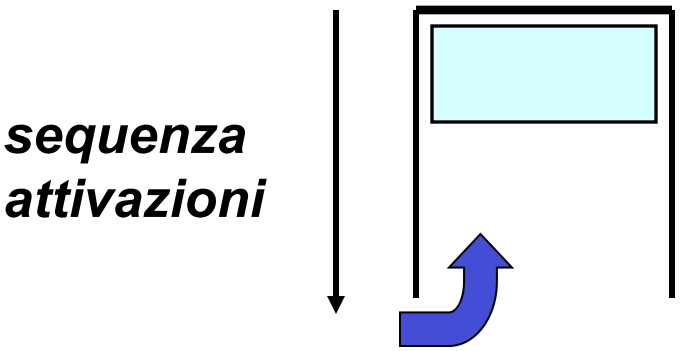
## STACK

Una struttura dati gestita con politica LIFO (Last In, First Out - l'ultimo a entrare è il primo a uscire)

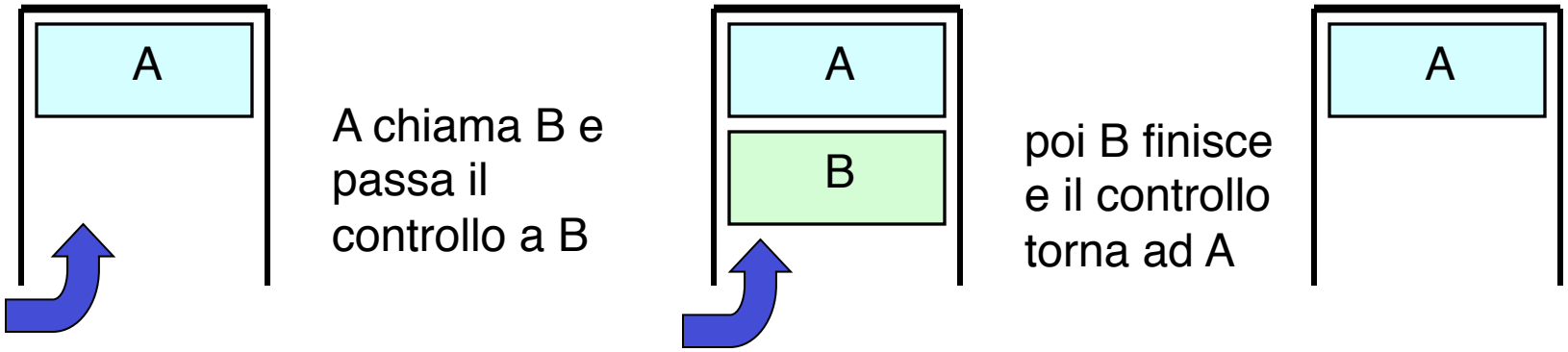


# RECORD DI ATTIVAZIONE

- 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



# RECORD DI ATTIVAZIONE

**Il valore di ritorno calcolato dalla funzione può essere *restituito al cliente* in due modi:**

- **inserendo un apposito "slot" nel record di attivazione**
  - il cliente deve copiarsi il risultato da qualche parte *prima* che il record venga distrutto
- **tramite un registro della CPU**
  - soluzione più semplice ed efficiente, privilegiata ovunque possibile.

# ESEMPIO DI 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); }
void main()
   { int x = P(); }
```

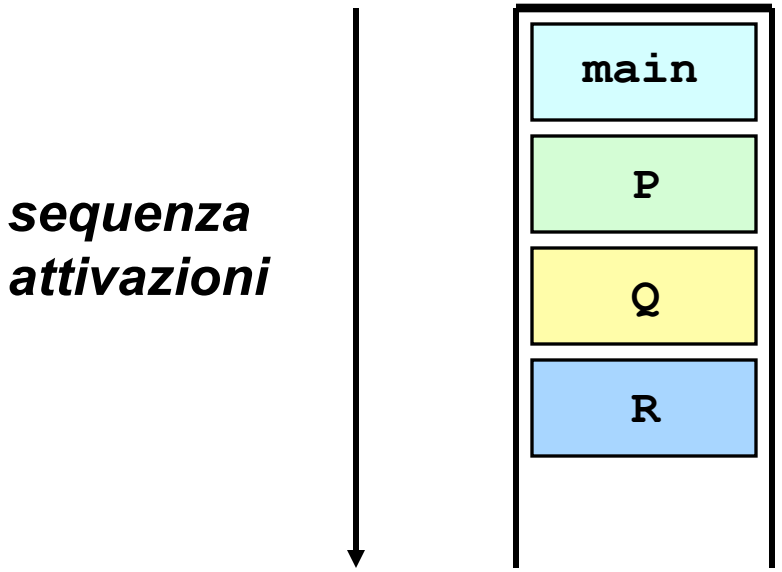
**Sequenza chiamate:**

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

# ESEMPIO DI CHIAMATE ANNIDATE

**Sequenza chiamate:**

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



# ESEMPIO: FATTORIALE

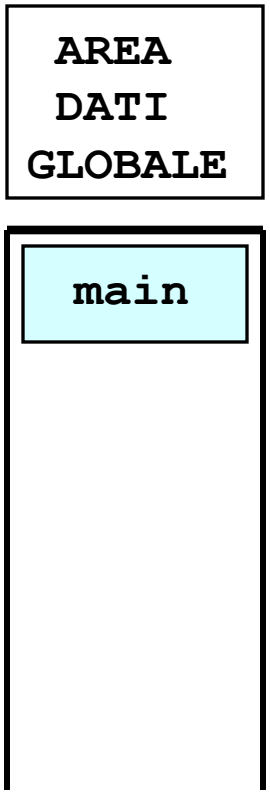
```
int fact(int n) {
    if n<=0 return 1;
    else return n*fact(n-1);
                        }

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

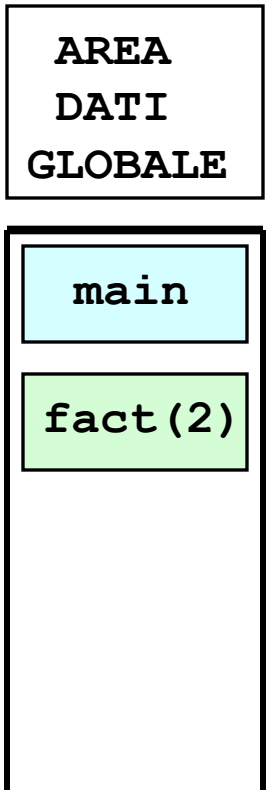
NOTA: Anche il `main()` e' una funzione

# ESEMPIO: FATTORIALE

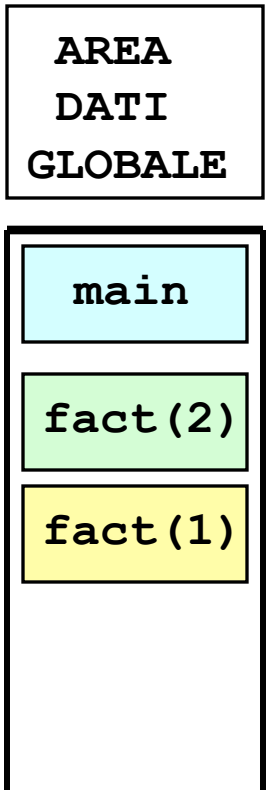
Situazione all'inizio dell'esecuzione del `main()`



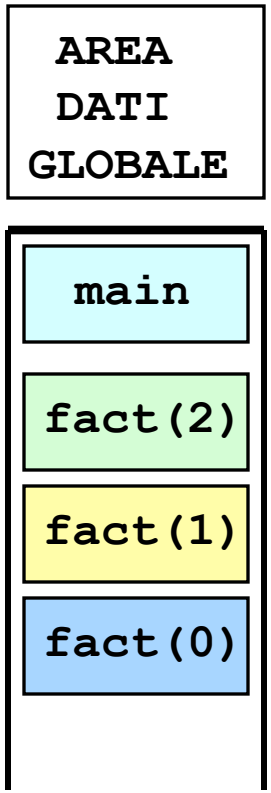

Il `main()` chiama `fact(2)`




`fact(2)` chiama `fact(1)`

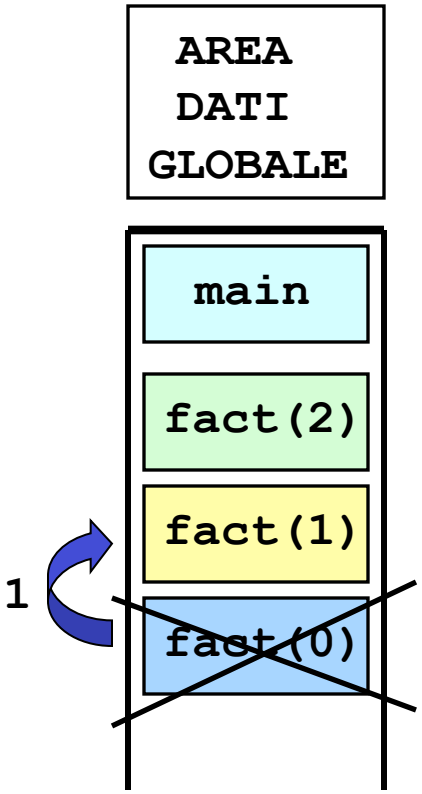



`fact(1)` chiama `fact(0)`

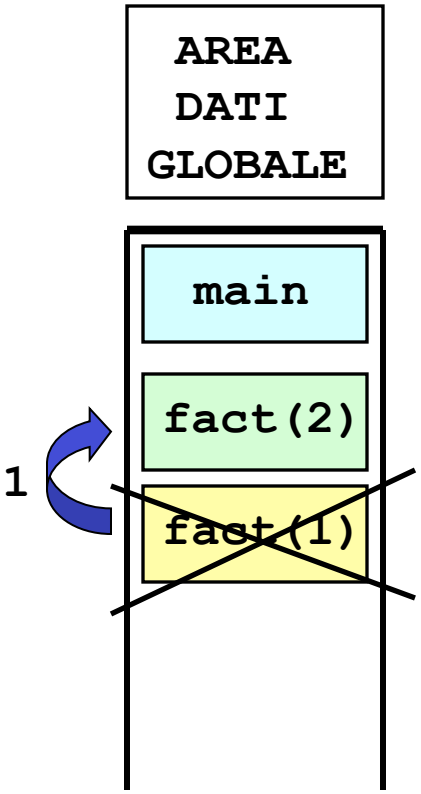


# ESEMPIO: FATTORIALE

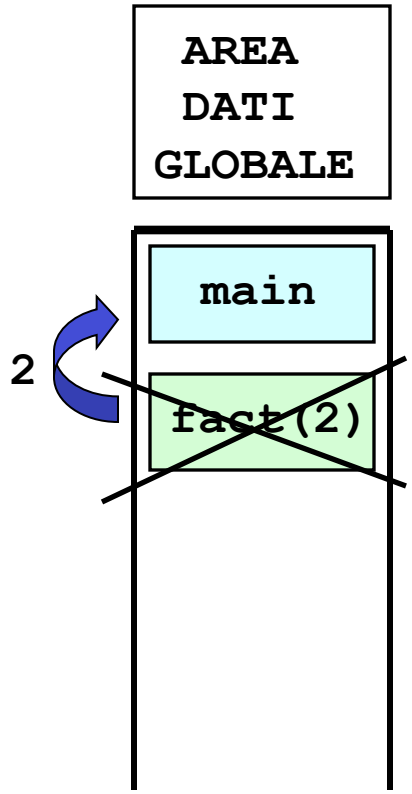
`fact(0)` termina restituendo il valore 1. Il controllo torna a `fact(1)`



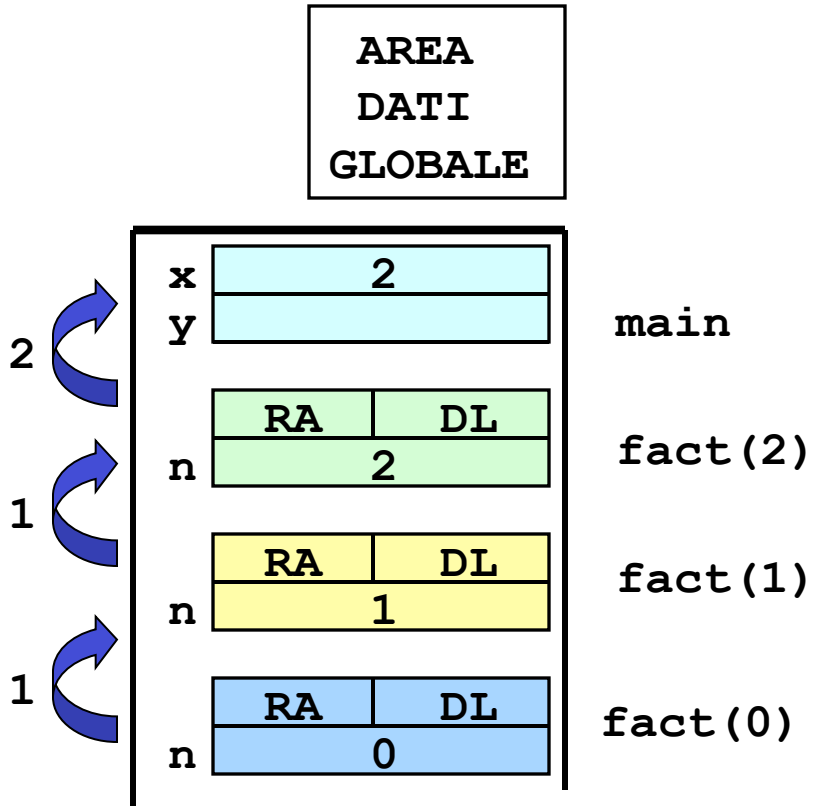

`fact(1)` effettua la moltiplicazione e termina restituendo il valore 1. Il controllo torna a `fact(2)`




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



# RECORD DI ATTIVAZIONE IN DETTAGLIO




**SI RICORDA CHE …**

nei processi computazionali ricorsivi ogni funzione che effettua una chiamata ricorsiva deve aspettare il risultato del servitore per effettuare operazioni su questo e poi può terminare.

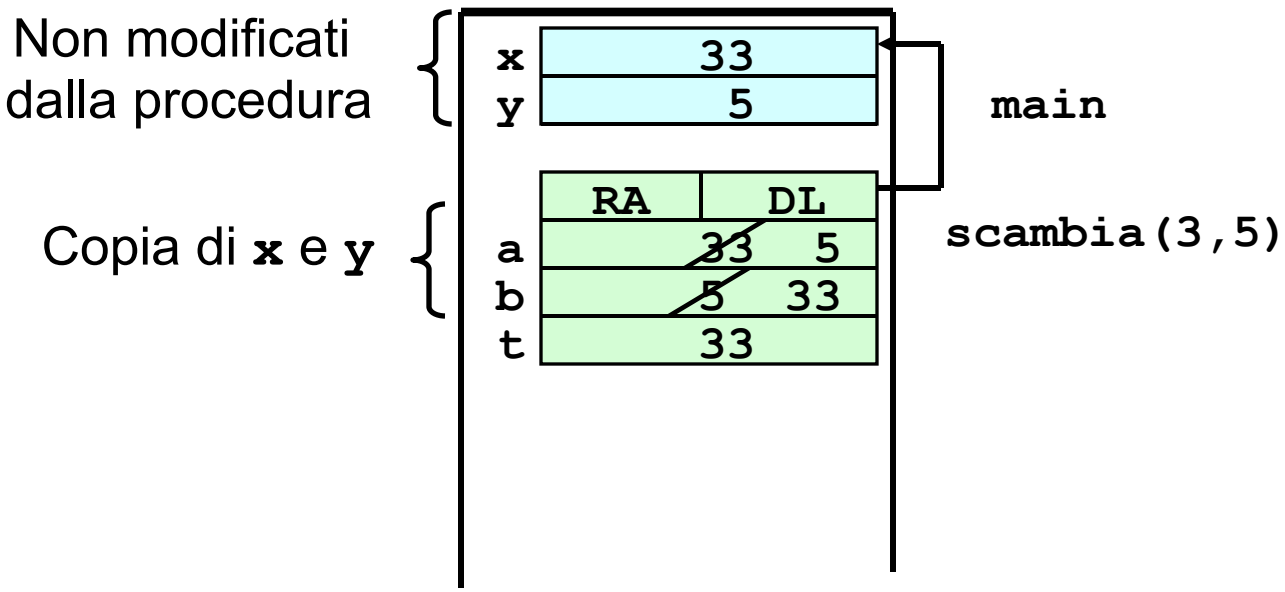
# REALIZZARE IL PASSAGGIO PER RIFERIMENTO IN C

```
void scambia(int* a, int* b) {
 int t;
 t = *a;  *a = *b;  *b = t;
}

void main(){
 int y = 5, x = 33;
 scambia(&x, &y);
printf("%d %d", x,y);
}
```

# ESEMPIO: RECORD DI ATTIVAZIONE

Caso del passaggio *per valore*:



# ESEMPIO: RECORD DI ATTIVAZIONE

Caso del passaggio *per riferimento*: