RECORD DI ATTIVAZIONE

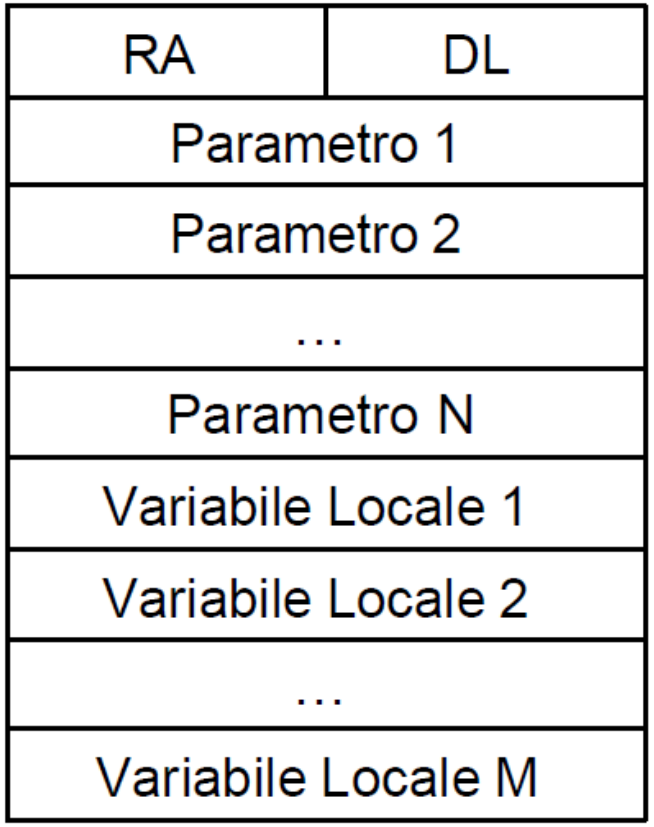
INVOCAZIONE DI FUNZIONI

- Ogni volta che viene invocata una funzione
  1. Si crea una nuova attivazione (istanza) della funzione
  2. Viene allocata la memoria per i parametri e per le variabili locali
  3. Si effettua il passaggio dei parametri
  4. Si esegue il codice della funzione

RECORD DI ATTIVAZIONE

- Contiene tutte le informazioni necessarie all'esecuzione della funzione
  - **Parametri**
  - **Variabili**
  - **Indirizzo di ritorno** (*Return Address* – RA)
    - Indica il punto a cui tornare (nel codice della funzione chiamante) al termine dell'esecuzione della funzione chiamata
    - Permette perciò alla funzione chiamante di proseguire una volta che la funzione chiamata termina
  - **Collegamento al record di attivazione della funzione chiamante** (*Dynamic Link* – DL)
    - Indica dove finisce il record di attivazione corrente

RECORD DI ATTIVAZIONE



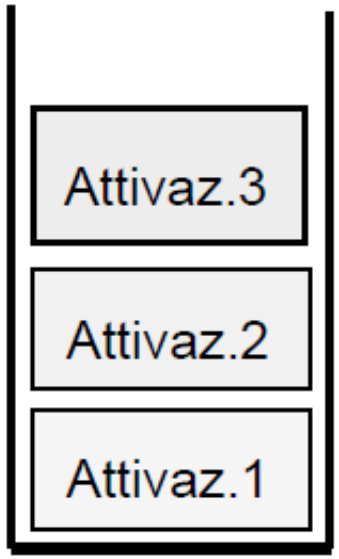
- La dimensione del record di attivazione
  - varia da una funzione all'altra
  - per una data funzione, è fissa e calcolabile a priori

CICLO DI VITA DEL RECORD DI ATTIVAZIONE

- È creato (*allocato*) al momento dell'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

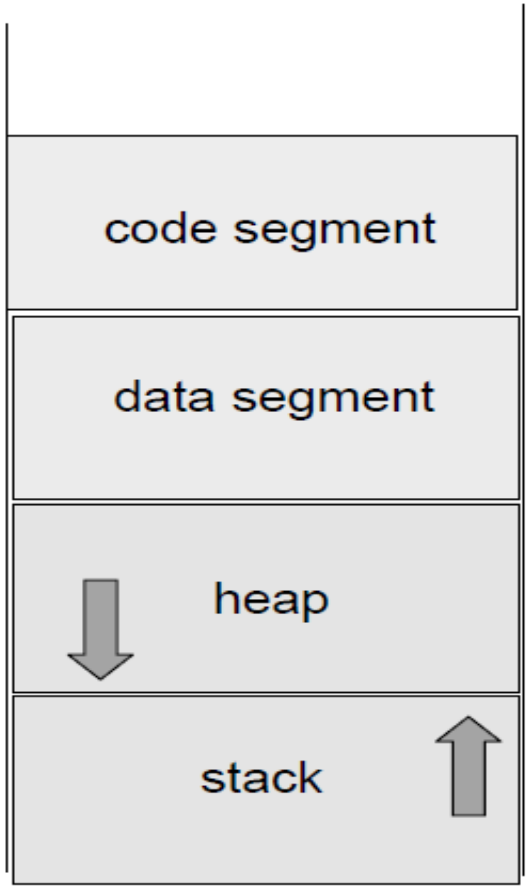
- Ad ogni chiamata di funzione viene creato un **nuovo** record, **specifico per quella chiamata di quella funzione**

CICLO DI VITA DEL 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
- L'area di memoria in cui vengono allocati i record di attivazione deve perciò essere gestita come una **pila** (*stack*)
  - Gestione **last in first out** (LIFO)
- La sequenza dei DL costituisce la cosiddetta «catena dinamica», che rappresenta la «storia» delle attivazioni

SPAZI DI INDIRIZZAMENTO IN MEMORIA



- **Code segment**: contiene il codice eseguibile
- **Data segment:** contiene dati **globali** (comuni a tutte le funzioni)
- **Heap**: contiene dati **dinamici** (creati durante l'esecuzione, ad esempio liste Python o array Java)
- **Stack**: contiene i record di attivazione

- Code segment e data segment sono di dimensione fissata staticamente
- La dimensione dell'area associata a stack+heap è fissata staticamente
  - Man mano che lo stack cresce, diminuisce l'area a disposizione dell'heap e viceversa

RESTITUZIONE DEI RISULTATI

- Il valore restituito dalla funzione chiamata può essere restituito alla funzione chiamante in due modi
  1. Inserendo uno spazio aggiuntivo nel record di attivazione
     - In questo caso la funzione chiamante deve recuperare il risultato prima che il record venga distrutto
  2. Utilizzando un registro della CPU

ESEMPIO

```
def f_a(a):
    return a+1

def f_b(b):
    return f_a(b)
```

```
def f_c():
    c = 10
    return f_b(c)

def main():
    x = f_c()

main()
```

ESEMPIO

```
def f_a(a):
    return a+1

def f_b(b):
    return f_a(b)
```

```
def f_c():
    c = 10
    return f_b(c)

def main():
    x = f_c()
```

main()

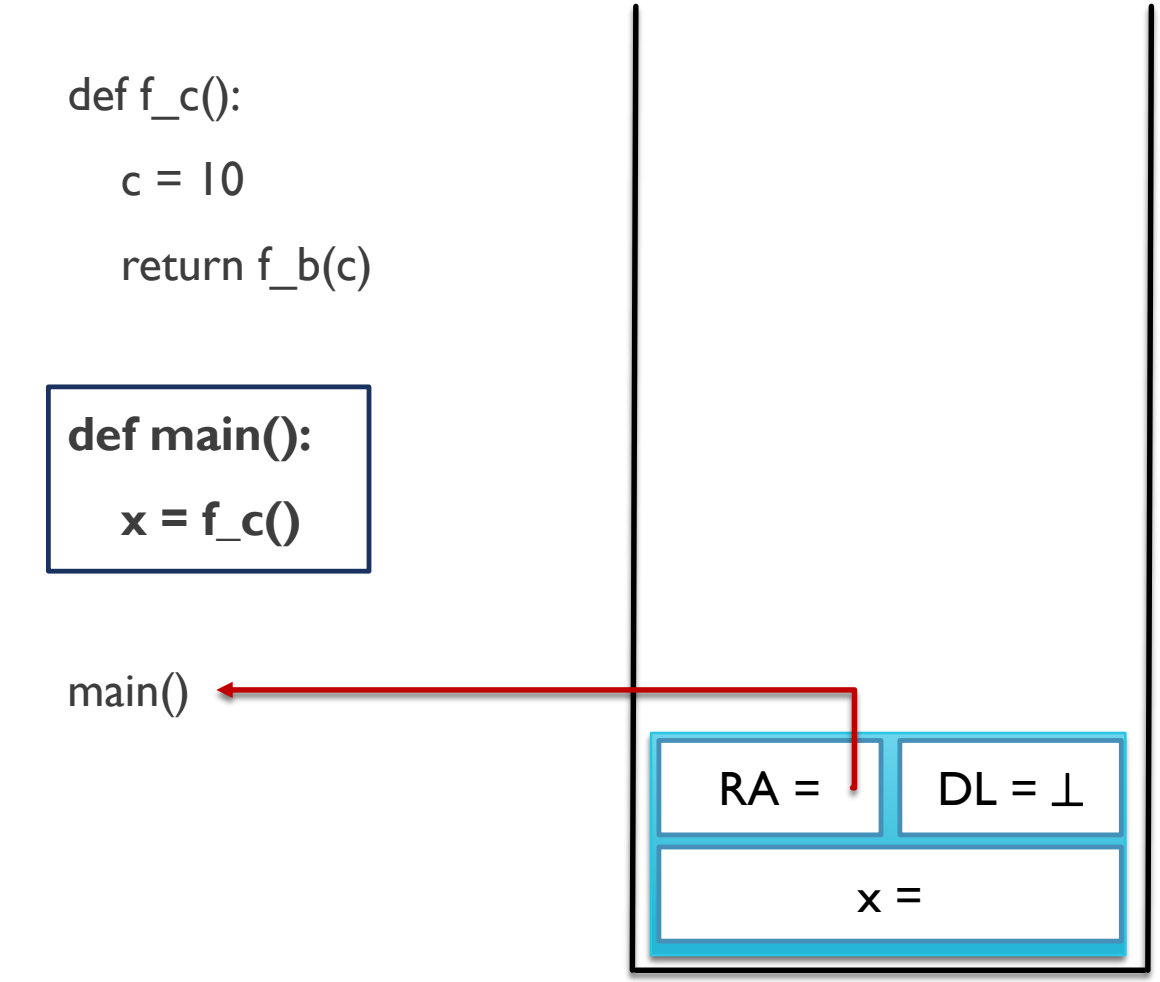
ESEMPIO

```
def f_a(a):
    return a+1

def f_b(b):
    return f_a(b)

def f_c():
    c = 10
    return f_b(c)

def main():
    x = f_c()
```



ESEMPIO

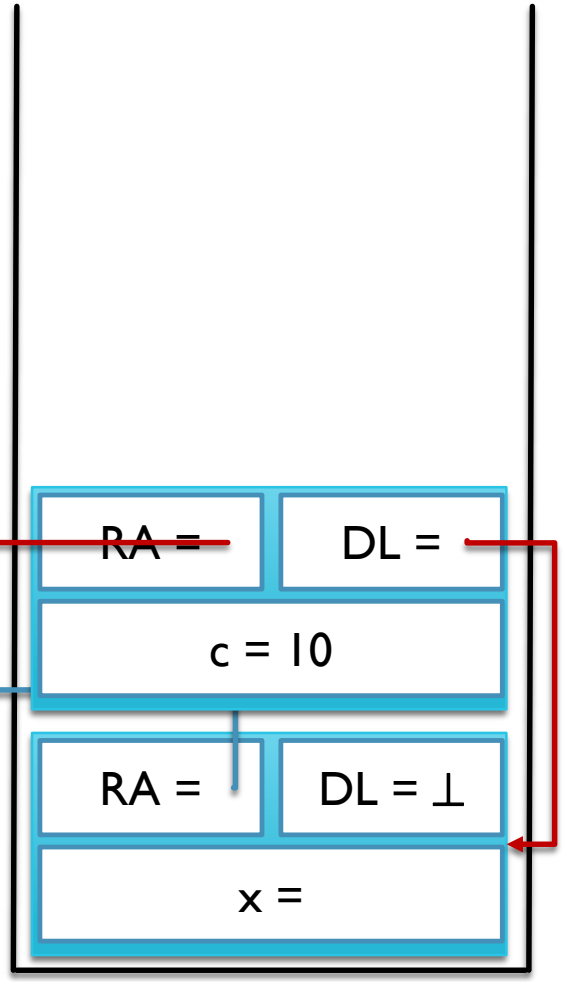
```
def f_a(a):
    return a+1

def f_b(b):
    return f_a(b)
```

```
def f_c():
    c = 10
    return f_b(c)

def main():
    x = f_c()
```

main()



ESEMPIO

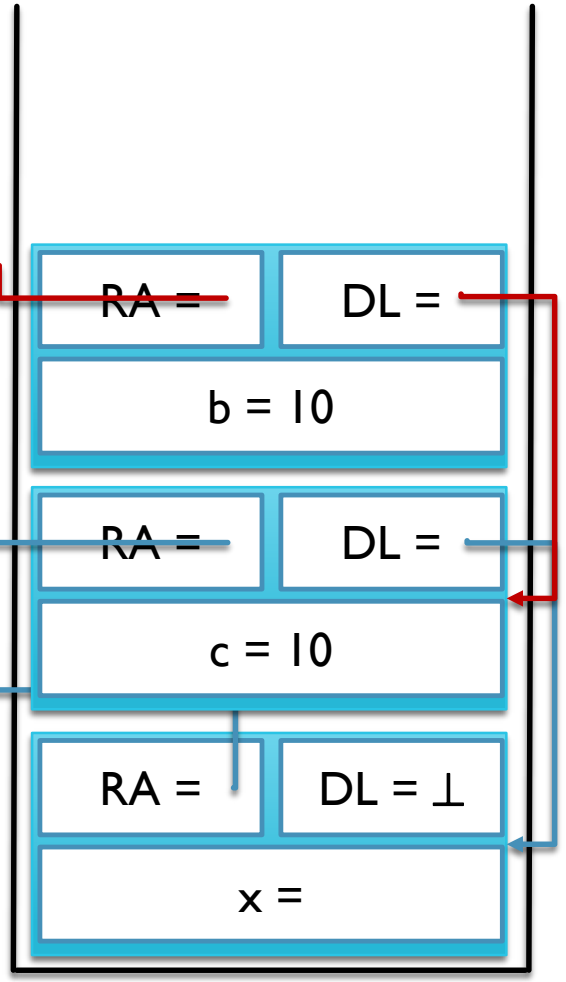
```
def f_a(a):
    return a+1

def f_c():
    c = 10
    return f_b(c)

def f_b(b):
    return f_a(b)

def main():
    x = f_c()

main()
```



ESEMPIO

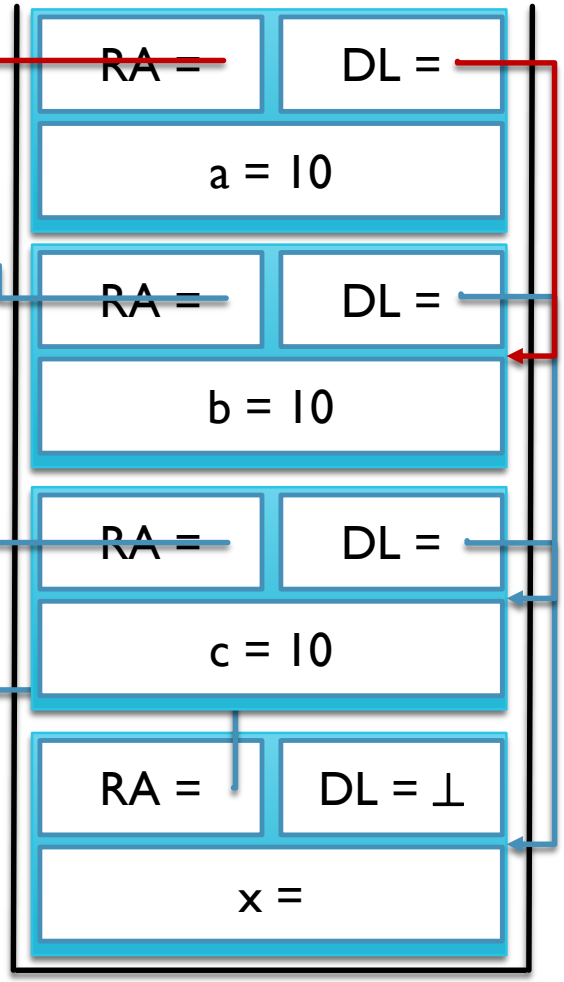
```
def f_a(a):
    return a+1
```

```
def f_c():
    c = 10
    return f_b(c)

def f_b(b):
    return f_a(b)

def main():
    x = f_c()

main()
```



ESEMPIO

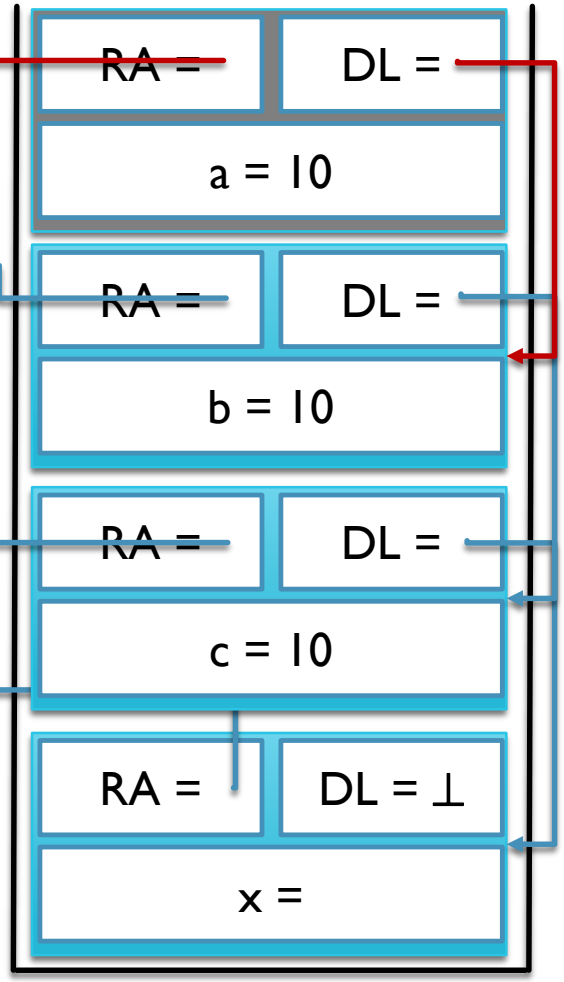
```
def f_a(a):
    return a+1

def f_b(b):
    return f_a(b)

def f_c():
    c = 10
    return f_b(c)

def main():
    x = f_c()

main()
```



- f_a restituisce il valore 11
- l'esecuzione si sposta al suo RA (in f_b)
- viene eliminato il record di f_a

ESEMPIO

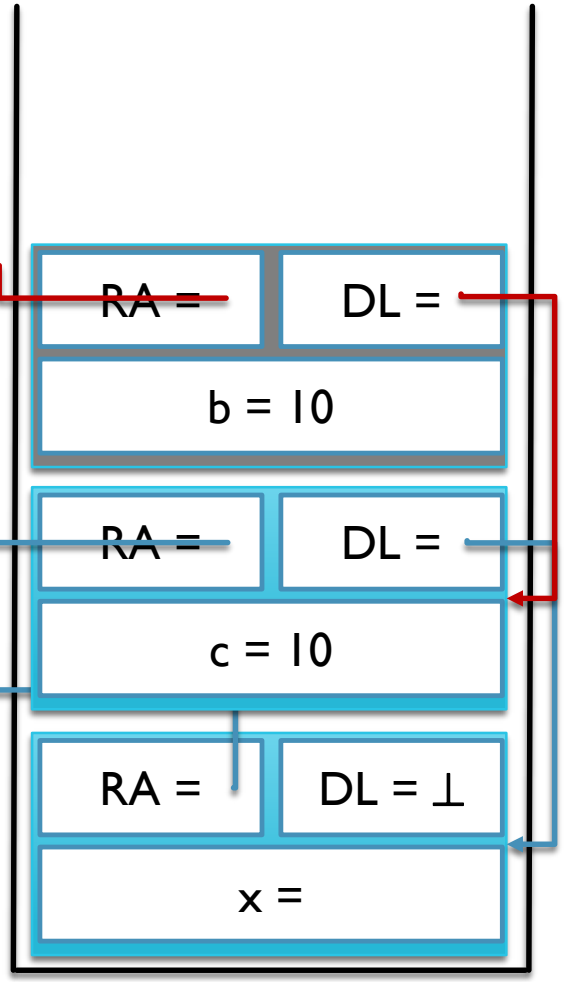
```
def f_a(a):
    return a+1

def f_b(b):
    return f_a(b)

def f_c():
    c = 10
    return f_b(c)

def main():
    x = f_c()

main()
```



- f_b restituisce il valore 11
- l'esecuzione si sposta al suo RA (in f_c)
- viene eliminato il record di f_b

ESEMPIO

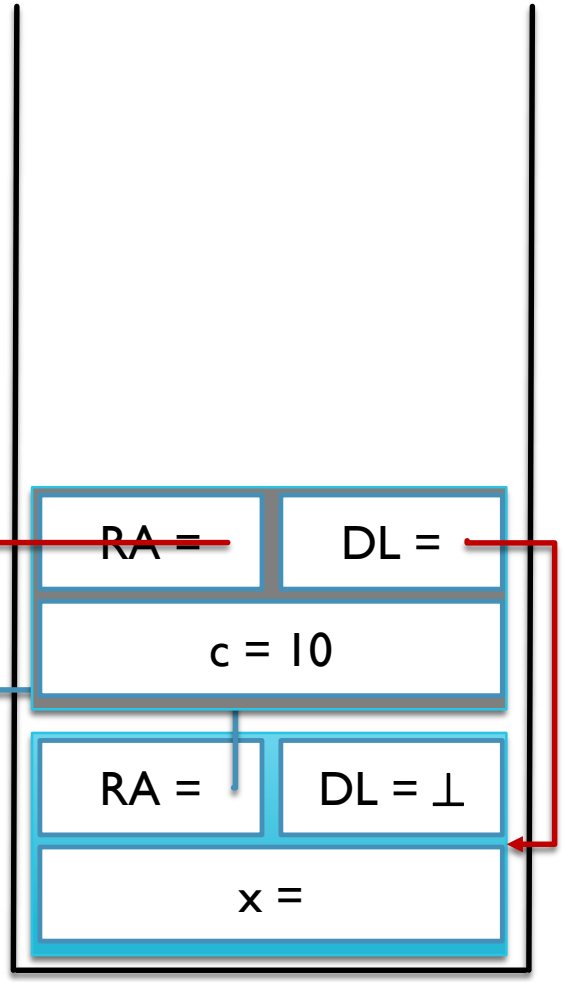
```
def f_a(a):
    return a+1

def f_b(b):
    return f_a(b)

def f_c():
    c = 10
    return f_b(c)

def main():
    x = f_c()

main()
```



- f_c restituisce il valore 11
- l'esecuzione si sposta al suo RA (in main)
- viene eliminato il record di f_c

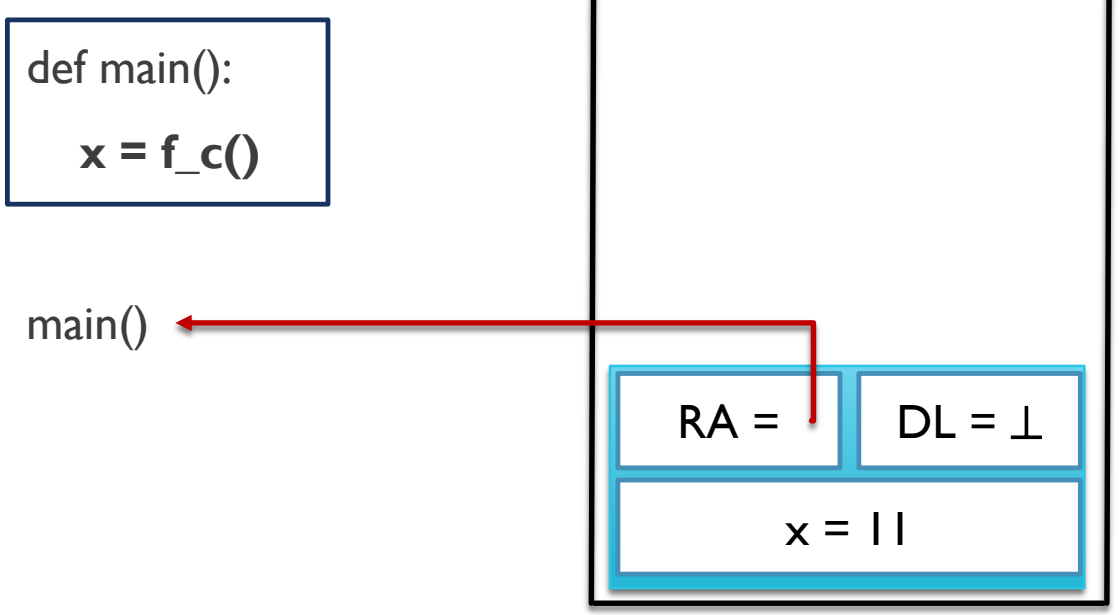
ESEMPIO

```
def f_a(a):
    return a+1

def f_b(b):
    return f_a(b)

def f_c():
    c = 10
    return f_b(c)

def main():
    x = f_c()
```



- main assegna 11 ad x
- l'esecuzione si sposta al suo RA
- viene eliminato il record di main