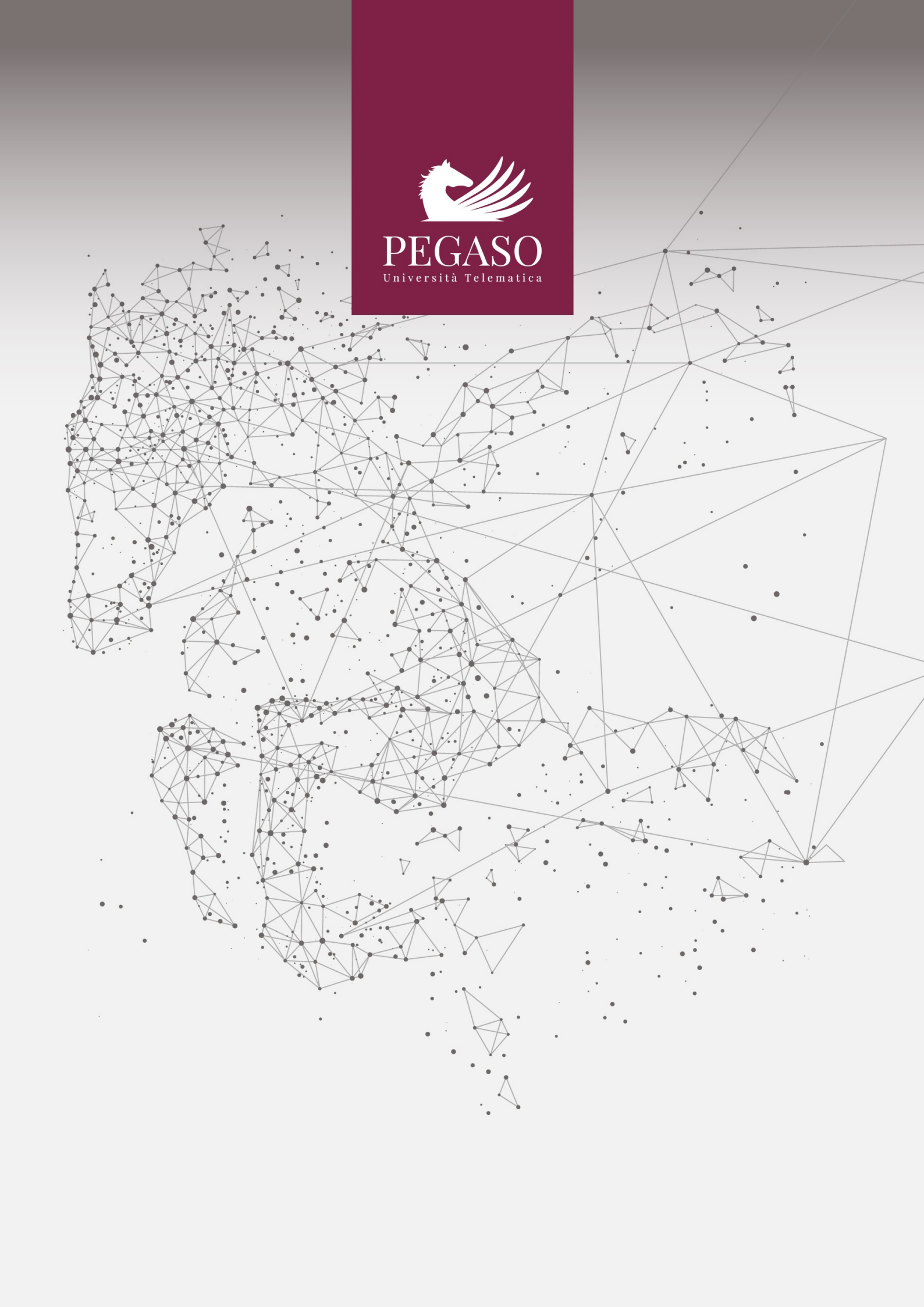

PEGASO
Università Telematica

Indice

1. Stack

Per comprendere bene come il C esegua le chiamate delle funzioni, dobbiamo innanzitutto considerare la sua struttura di dati (cioè un insieme di dati correlati) nota come pila o stack. Pensate a uno stack come a qualcosa di analogo a una pila di piatti.

Quando si mette un piatto su una pila, normalmente lo si colloca in cima, operazione che chiameremo *push*.

In modo simile, quando si toglie un piatto dalla pila, normalmente si prende quello in cima, operazione che chiameremo *pop*.

Gli stack sono noti come strutture di dati *last-in first-out (LIFO)* (ossia l'ultimo elemento inserito nello stack è il primo elemento a essere rimosso da esso).

Un meccanismo che va compreso bene è lo stack delle chiamate delle funzioni (detto anche pila di esecuzione del programma).

Questa struttura di dati, che lavora "dietro le quinte", supporta il meccanismo di chiamata e ritorno delle funzioni.

Essa supporta anche la creazione, il mantenimento e la distruzione delle variabili locali (chiamate anche variabili automatiche) di tutte le funzioni chiamate.

Abbiamo illustrato il comportamento last-in first-out (LIFO) dello stack col nostro esempio della pila di piatti.

Questo comportamento LIFO è esattamente ciò che una funzione fa quando il controllo ritorna alla funzione che l'ha chiamata.

Quando una funzione è chiamata, essa può chiamare altre funzioni, le quali possono a loro volta chiamarne altre; tutto prima che una funzione torni alla funzione chiamante, restituendo a essa il controllo.

Ogni funzione alla fine deve restituire il controllo alla funzione che l'ha chiamata.

Così, dobbiamo tenere traccia degli indirizzi di ritorno che servono a ogni funzione per tornare alla funzione che l'ha chiamata.

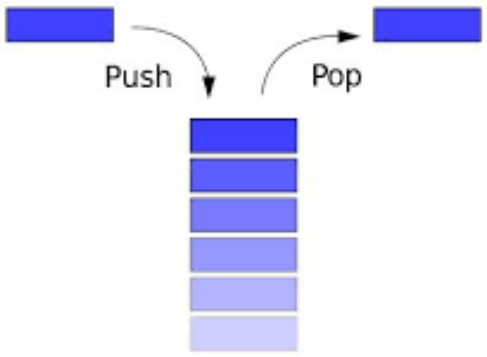
Lo stack delle chiamate delle funzioni è la struttura di dati perfetta per trattare queste informazioni.

Ogni volta che una funzione chiama un'altra funzione, con un push un nuovo elemento viene inserito in cima allo stack.

Questo elemento, chiamato *record di attivazione*, contiene l'indirizzo di ritorno che serve alla funzione chiamata per tornare alla funzione chiamante.

Se la funzione chiamata torna alla funzione chiamante, senza chiamare un'altra funzione prima di restituire il controllo, con un pop il record di attivazione della chiamata della funzione viene estratto dalla cima dello stack.

Il controllo si trasferisce all'indirizzo di ritorno memorizzato in quello stesso record di attivazione.



LIFO

Record di attivazione

Ogni funzione chiamata trova sempre le informazioni che le occorrono per tornare alla sua funzione chiamante in cima allo stack delle chiamate.

E se una funzione chiama un'altra funzione, con un push il record di attivazione per la chiamata della nuova funzione viene inserito in cima allo stack delle chiamate.

In questo modo, l'indirizzo di ritorno richiesto dalla funzione chiamata più di recente per tornare alla sua funzione chiamante è ora posizionato in cima allo stack.

I record di attivazione hanno un altro importante compito.

La maggior parte delle funzioni posseggono variabili locali (*automatiche*) – parametri e alcune o tutte le loro variabili locali.

Le variabili automatiche devono esistere mentre una funzione è in esecuzione e devono inoltre rimanere attive se la funzione chiama altre funzioni.

Ma quando una funzione chiamata torna alla sua funzione chiamante, le variabili automatiche della funzione chiamata devono "sparire".

Il record di attivazione della funzione chiamata è un posto perfetto per memorizzare le sue variabili automatiche.

Il record di attivazione esiste solo finché la funzione chiamata è attiva.

Quando quella funzione torna alla funzione chiamante – e non ha più bisogno delle sue variabili automatiche locali – con un pop il record di attivazione è rimosso dallo stack e quelle variabili automatiche locali non esistono più per il programma.

Naturalmente, la quantità di memoria in un computer è limitata, quindi se ne può utilizzare solo una certa quantità per memorizzare i record di attivazione nello stack delle chiamate delle funzioni.

Se avvengono più chiamate di funzioni di quanti record di attivazione possono essere memorizzati nello stack delle chiamate, si verifica un errore irreversibile noto come *stack overflow*.

Esempio di uso dello stack

Consideriamo ora come lo stack delle chiamate supporta l'esecuzione di una funzione `square` chiamata da `main`.

```
01  // Programma C
02  //  Meccanismo dello stack delle chiamate delle funzioni
03  // e dei record di attivazione con la funzione square.
04  #include <stdio.h>
05
06  int square(int); // prototipo per la funzione square
07
08  int main()
09  {
10     int a = 10; // variabile automatica locale in main
11
12     printf("%d squared: %d\n", a, square(a));
13  }
14
15  // restituisce il quadrato di un intero
16  int square(int x) // x e' una variabile locale
17  {
18     return x * x; // calcola e restituisce il quadrato
19  }
```

```
10 squared: 100
```

Per prima cosa il sistema operativo chiama `main`;

ciò implica un push di un record di attivazione nello stack (Figura 1).

Il record di attivazione indica alla funzione `main` come tornare al sistema operativo (cioè come trasferire il controllo all'indirizzo di ritorno R1) e contiene lo spazio per la variabile automatica di `main` (cioè a, che è inizializzata a 10).

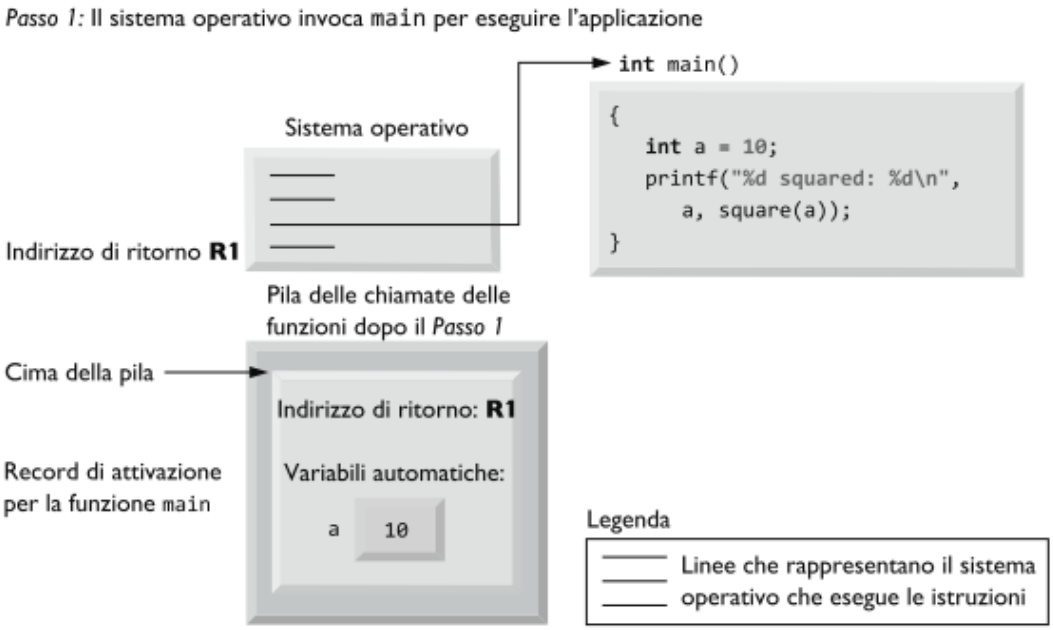


Figura 1: *Stack (pila) delle chiamate delle funzioni dopo che il sistema operativo invoca main per eseguire il programma*

La funzione `main` – prima di tornare al sistema operativo – chiama ora la funzione `square` nella riga 12.

Questo causa un push di un record di attivazione per `square` (righe 16–19) nello stack delle chiamate delle funzioni (Figura 2).

Questo record di attivazione contiene l'indirizzo di ritorno che occorre a `square` per tornare a `main` (cioè R2) e la memoria per la variabile automatica di `square` (cioè x).

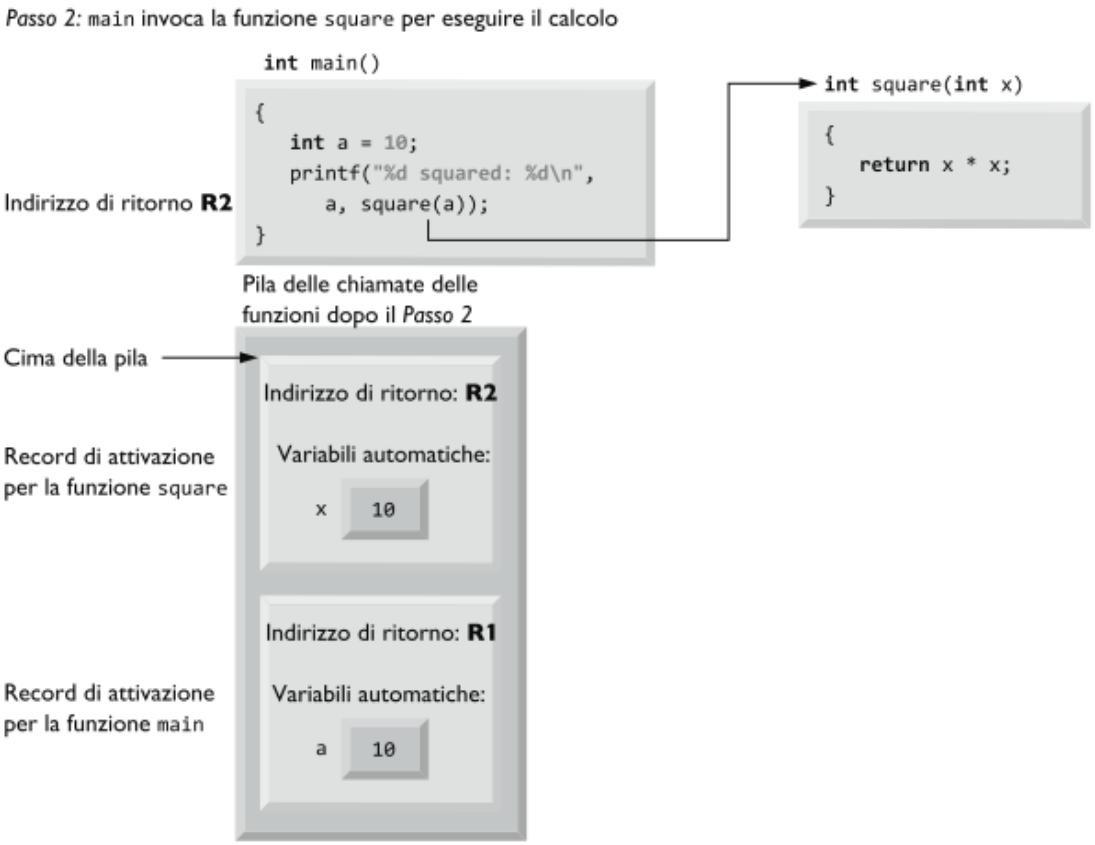


Figura 2: *Stack (pila) delle chiamate delle funzioni dopo che main invoca square per eseguire il calcolo*

Dopo che `square` calcola il quadrato del suo argomento, deve tornare a `main` e non ha più bisogno della memoria per la sua variabile automatica `x`.

Così si effettua un pop dallo stack, fornendo a `square` l'indirizzo di ritorno in `main` (cioè R2), mentre la variabile automatica di `square` viene persa.

La Figura 3 mostra lo stack delle chiamate della funzione dopo la rimozione del record di attivazione di `square`.

La funzione `main` stampa ora il risultato della chiamata a `square`.

Quando si raggiunge la parentesi graffa destra di chiusura di `main`, il record di attivazione viene rimosso dallo stack, `main` riceve l'indirizzo che le occorre per tornare al sistema operativo (cioè R1 nella Figura 1) e la memoria per la variabile automatica di `main` (cioè a) non è più accessibile.

Avete appena visto quanto sia preziosa la struttura a stack per implementare un meccanismo chiave in grado di supportare l'esecuzione dei programmi.

Le strutture di dati hanno applicazioni molto importanti nell'ambito dell'informatica.

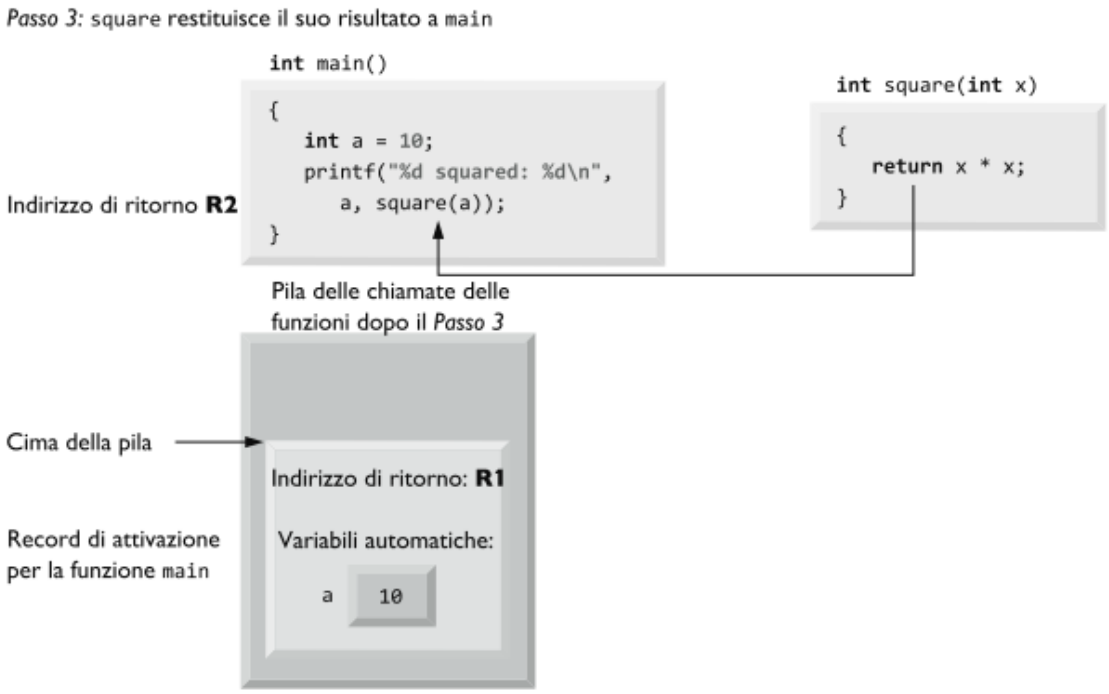


Figura 3: *Stack (pila) delle chiamate delle funzioni dopo il ritorno della funzione square a main*

Riferimenti bibliografici

- Paul Deitel, Harvey Deitel, "Il linguaggio C – Fondamenti e tecniche di programmazione", Libro edito da Pearson Italia. Include anche utili esercizi di autovalutazione.