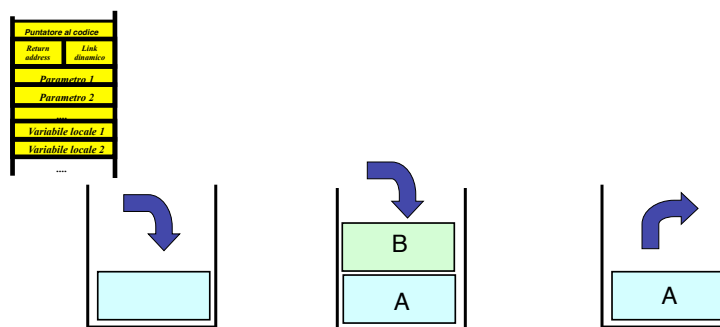


Il linguaggio C Il modello a run-time



Fondamenti di Informatica L- A

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

Fondamenti di Informatica L- A

Record di Attivazione

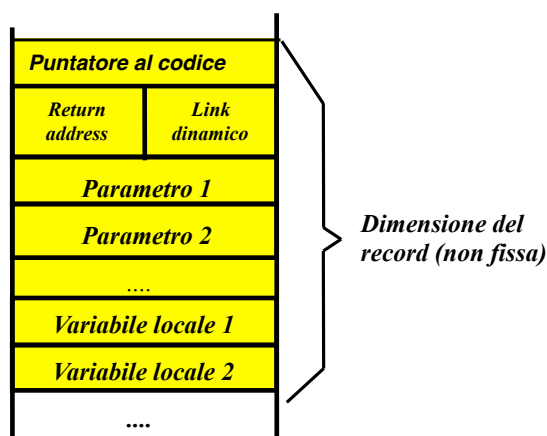
- Al momento dell'invocazione:
viene creata dinamicamente una struttura dati che contiene i binding dei parametri e degli identificatori definiti localmente alla funzione detta **RECORD DI ATTIVAZIONE**.

Record di Attivazione

È il “**mondo della funzione**”: contiene tutto ciò che serve per la chiamata alla quale è 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*), per sapere dove finisce il record di attivazione corrente (utile per la gestione della memoria)
- l'**indirizzo del codice** della funzione (puntatore alla prima istruzione)

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

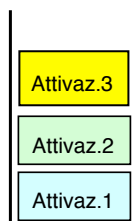
- *Funzioni che chiamano altre funzioni* danno luogo a una *sequenza* di record di attivazione
 - allocati secondo l'ordine delle chiamate
 - deallocati in ordine inverso
- La sequenza dei link dinamici costituisce la cosiddetta *catena dinamica*, che rappresenta *la storia delle attivazioni* (“chi ha chiamato chi”)

Stack

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

STACK

È 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 è un record di attivazione.



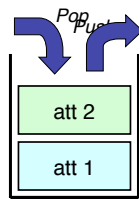
La gestione dello stack avviene mediante due operazioni:

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

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

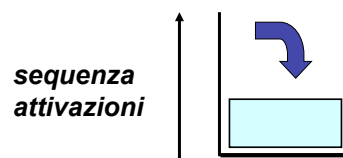
Stack

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

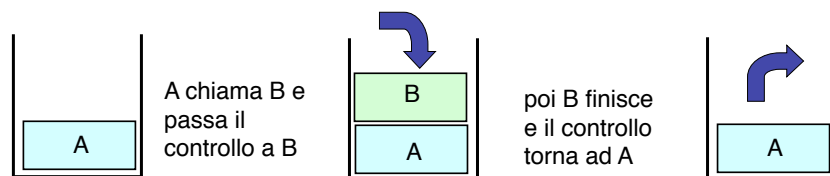


Record di Attivazione

- 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



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: 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()

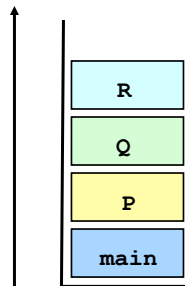
Esempio: chiamate annidate

Sequenza chiamate:

S.O. → main → P() → Q() → R()



*sequenza
attivazioni*



Fondamenti di Informatica L- A

Esempio: parametri di tipo puntatore

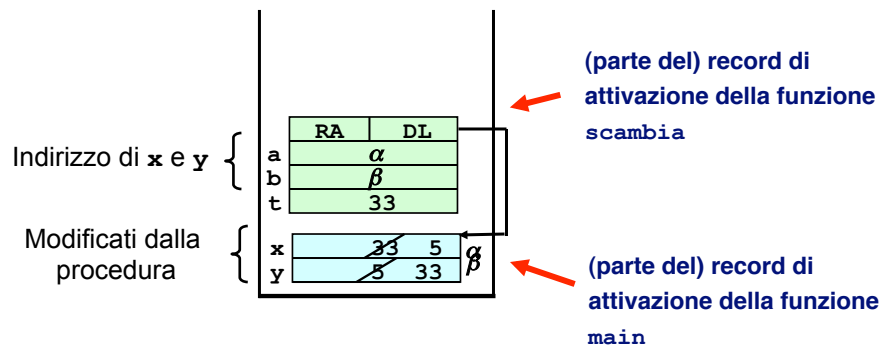
```
void scambia(int* a, int* b) {  
    int t;  
    t = *a; *a = *b; *b = t;  
}
```

```
main() {  
    int y = 5, x = 33;  
    scambia(&x, &y);  
}
```

Fondamenti di Informatica L- A

Esempio: parametri di tipo puntatore

Caso del passaggio *per riferimento*:

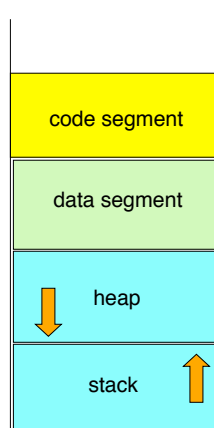


Fondamenti di Informatica L- A

1 3

Spazio di indirizzamento

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



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

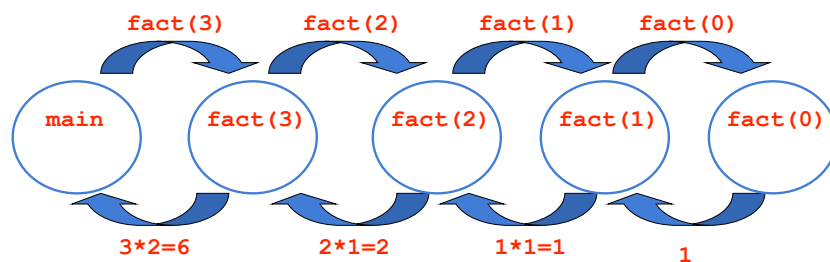
→ **Code segment** e **data segment** sono di dimensione fissata staticamente (a tempo di compilazione).

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

Fondamenti di Informatica L- A

Il linguaggio C

La ricorsione



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 $n=n_0$ \Rightarrow **CASO BASE**
 - e si può provare che, assumendola valida per n , allora vale per $n+1$
 allora P vale per ogni $n \geq n_0$

La ricorsione

- Operativamente, risolvere un problema con un approccio ricorsivo comporta
 - di identificare un “caso base” ($n=n_0$) 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

$\text{fact}(n) = n!$

$$\begin{array}{l}
 n! : \mathbb{N} \rightarrow \mathbb{N} \\
 \left\{ \begin{array}{ll}
 n! \text{ vale } 1 & \text{se } n == 0 \\
 n! \text{ vale } n * (n-1)! & \text{se } n > 0
 \end{array} \right.
 \end{array}$$

Ricorsione in C

In C è 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` è 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, 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, 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).

Fondamenti di Informatica L- A

Ricorsione in C: esempio

- **Servitore & Cliente:**

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

Fondamenti di Informatica L- A

Ricorsione in C: esempio

- **Servitore & Cliente:**

```
int fact(int n) {  
    if (n==0) return 1;  
    else return n*fact(n-1);  
}  
  
main() {  
    int fz, 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 (fact(n-1)). n-1 nell'environment di fact vale 2 quindi viene chiamata fact(2)

Fondamenti di Informatica L- A

Ricorsione in C: esempio

- **Servitore & Cliente:**

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

Fondamenti di Informatica L- A

Ricorsione in C: esempio

- **Servitore & Cliente:**

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

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)

Fondamenti di Informatica L- A

Ricorsione in C: esempio

- **Servitore & Cliente:**

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

Il nuovo servitore lega il parametro n a 0. La condizione $n \leq 0$ e' vera e la funzione fact(0) torna come risultato 1 e termina.

Fondamenti di Informatica L- A

Ricorsione in C: esempio

- **Servitore & Cliente:**

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

*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.*

Fondamenti di Informatica L- A

Ricorsione in C: esempio

- **Servitore & Cliente:**

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

*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.*

Fondamenti di Informatica L- A

Ricorsione in C: esempio

- **Servitore & Cliente:**

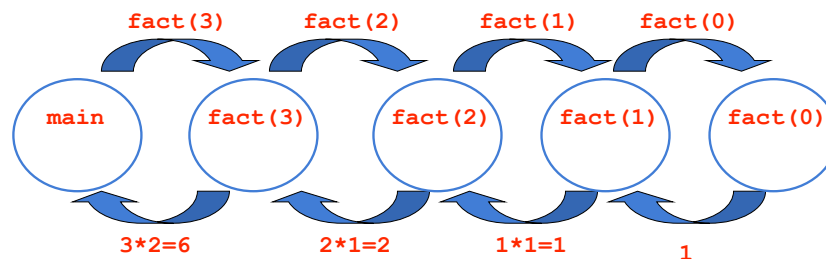
```
int fact(int n) {
    if (n==0) return 1;
    else return n*fact(n-1);
}
```

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

*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*

Fondamenti di Informatica L- A

Ricorsione in C: esempio



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

Fondamenti di Informatica L- A

Cosa succede nello stack ?

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

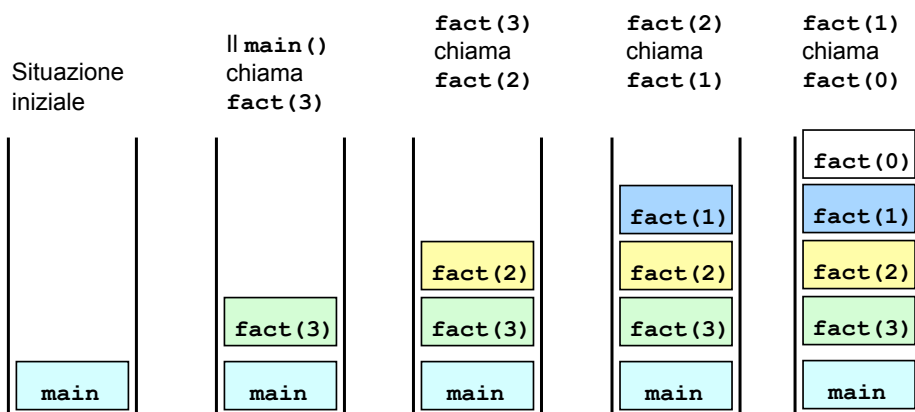
```
main() {  
    int fz, z = 5;  
    fz = fact(z-2);  
}
```

NOTA: Anche il
main() e' una funzione

Seguiamo l'evoluzione dello stack durante
l'esecuzione:

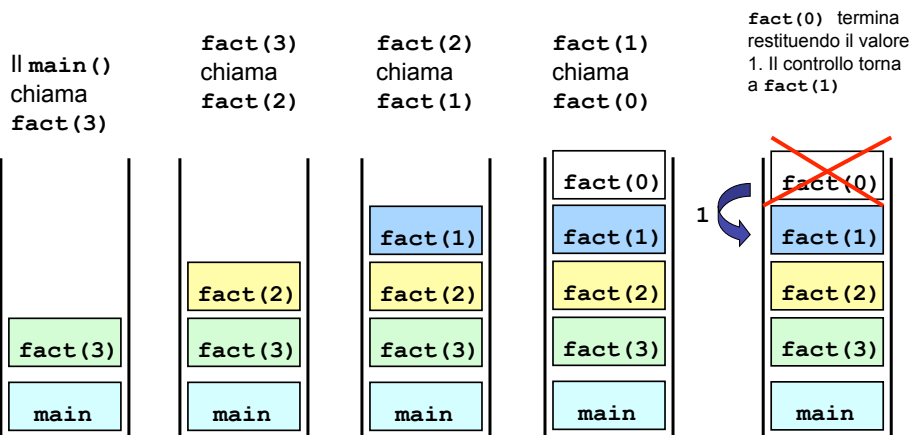
Fondamenti di Informatica L- A

Cosa succede nello stack ?

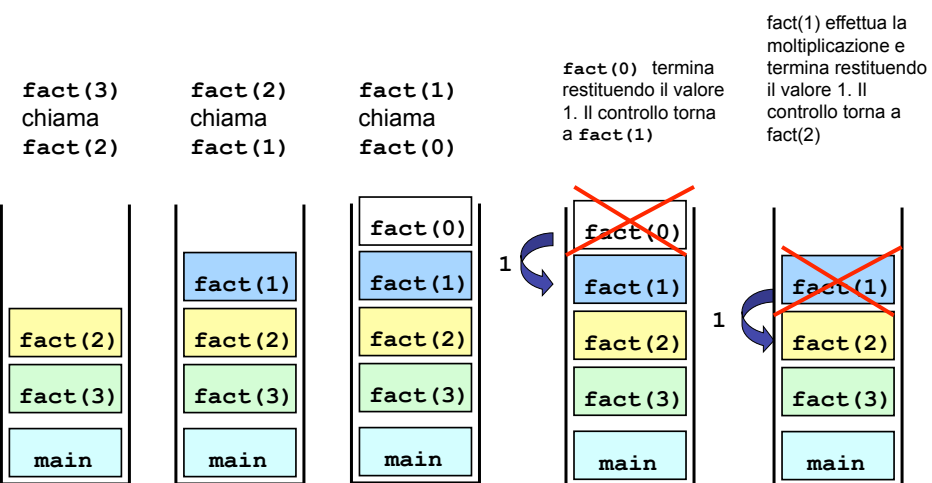


Fondamenti di Informatica L- A

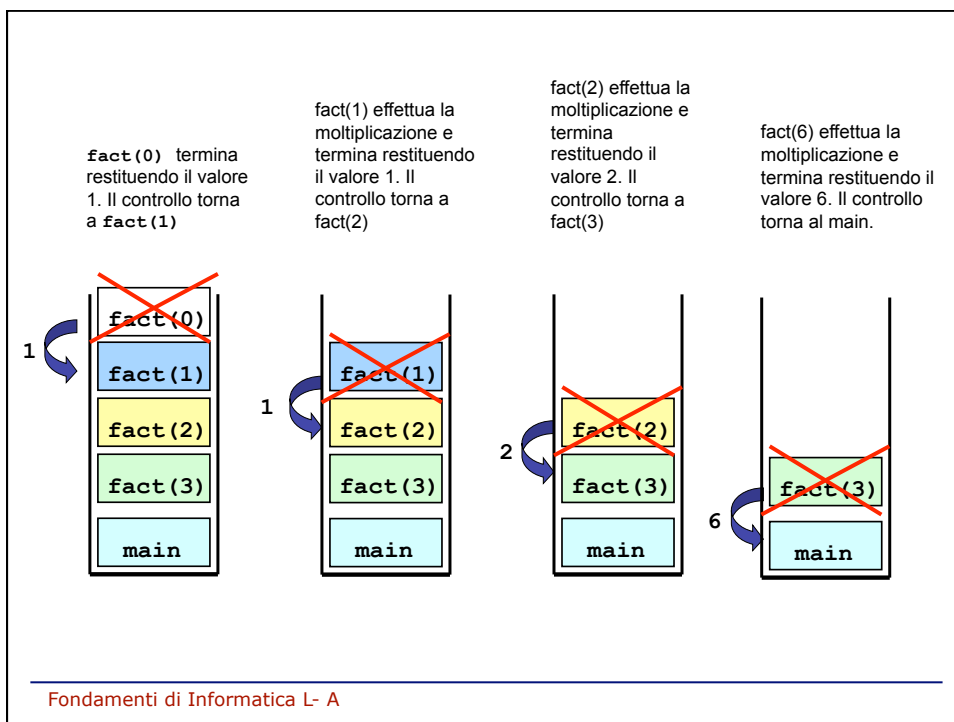
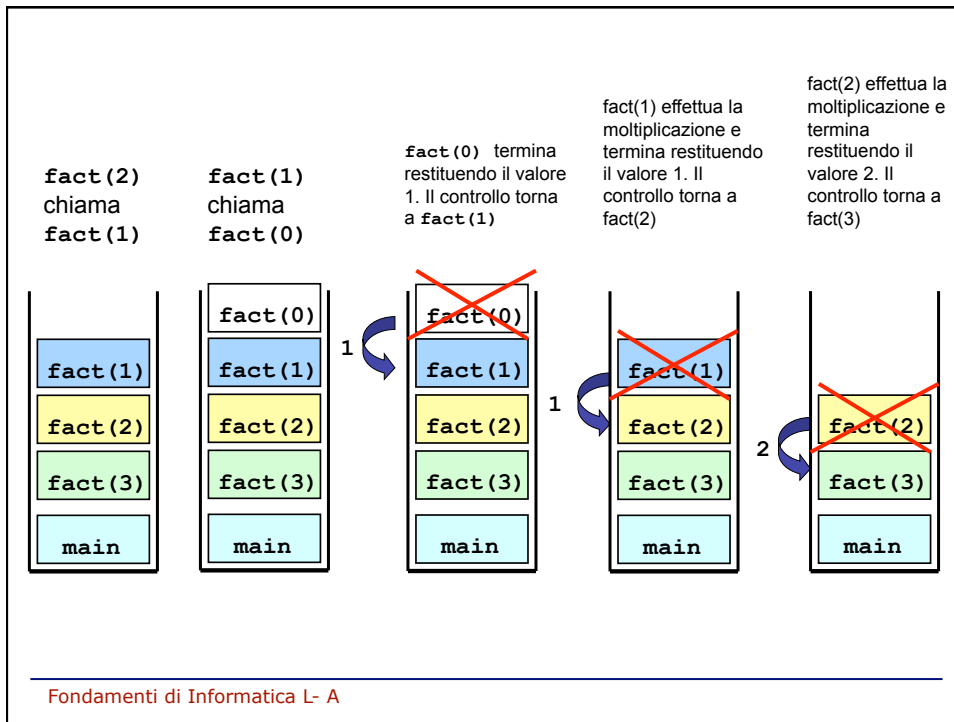
Cosa succede nello stack ?



Fondamenti di Informatica L- A



Fondamenti di Informatica L- A



Esempio: somma dei primi N naturali

Problema:
calcolare la somma dei primi N naturali

Specifica:

Considera la somma $1+2+3+\dots+(N-1)+N$ come composta di due termini:

- $(1+2+3+\dots+(N-1))$ →
- N → Valore noto

Il primo termine non è altro che lo stesso problema in un caso più semplice: calcolare la somma dei primi N-1 interi

Esiste un caso banale ovvio: **CASO BASE**

- la somma fino a 1 vale 1.

Fondamenti di Informatica L- A

Esempio: somma dei primi N naturali

Problema:
calcolare la somma dei primi N naturali

Algoritmo ricorsivo:

Somma: $N \rightarrow N$

{	Somma(n) vale 1	se $n == 1$
	Somma(n) vale $n + \text{Somma}(n-1)$	se $n > 0$

Fondamenti di Informatica L- A

Esempio: somma dei primi N naturali

Codifica:

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

Fondamenti di Informatica L- A

Esempio: somma dei primi N naturali

```
#include<stdio.h>
int sommaFinoA(int n);

main()
{ int dato;
  printf("\ndammi un intero positivo: ");
  scanf("%d", &dato);
  if (dato>0)
    printf("\nRisultato: %d", sommaFinoA(dato));
  else printf("ERRORE!");
}

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

Esercizio: seguire l'evoluzione dello stack nel caso in cui dato=4.

Fondamenti di Informatica L- A

Esempio: il numero di Fibonacci

Problema:
calcolare l'N-esimo numero di Fibonacci

$$\text{fib}(n) = \begin{cases} 0, & \text{se } n=0 \\ 1, & \text{se } n=1 \\ \text{fib}(n-1) + \text{fib}(n-2), & \text{altrimenti} \end{cases}$$

Fondamenti di Informatica L- A

Esempio: il numero di Fibonacci

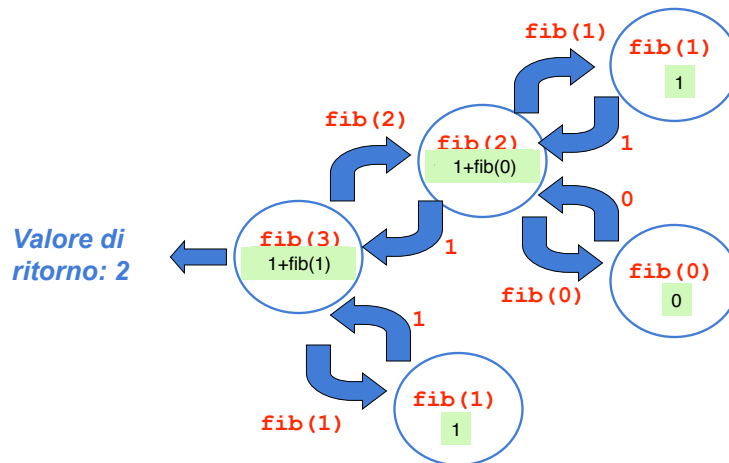
Codifica:

```
unsigned fibonacci(unsigned n) {  
    if (n<2) return n;  
    else return fibonacci(n-1)+fibonacci(n-2);  
}
```

Ricorsione non lineare: una invocazione del servitore causa due nuove chiamate al servitore medesimo.

Fondamenti di Informatica L- A

Esempio: il numero di Fibonacci



Fondamenti di Informatica L- A

1

Osservazioni

- Negli esempi visti finora si inizia a sintetizzare il risultato solo dopo che si sono *aperte* tutte le chiamate, "a ritroso", mentre le chiamate si *chiudono*.

Le chiamate ricorsive decompongono via via il problema, ma non calcolano nulla

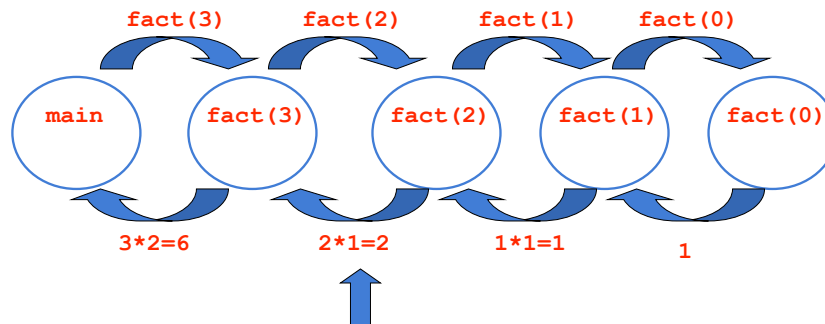
- Il risultato viene sintetizzato a partire dalla fine, perché prima occorre arrivare al caso base
 - il caso base fornisce il valore di partenza
 - poi si sintetizzano, "a ritroso", i successivi risultati parziali.



Processo di calcolo effettivamente ricorsivo

Fondamenti di Informatica L- A

Esempio



PASSI:

- 1) `fact(3)` chiama `fact(2)` passandogli il controllo,
- 2) `fact(2)` calcola il fattoriale di 2 e termina restituendo 2
- 3) `fact(3)` riprende il controllo ed effettua la moltiplicazione $3*2$
- 4) termina anche `fact(3)` e torna il controllo al main

Fondamenti di Informatica L- A

Calcolo iterativo del fattoriale

- Il fattoriale può 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

Fondamenti di Informatica L- A

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

Fondamenti di Informatica L- A

Processo computazionale iterativo 1

- Nell'esempio precedente (ciclo `for`) il risultato viene sintetizzato *"in avanti"*
- L'esecuzione di un algoritmo di calcolo che computi "in avanti", per accumulo, è 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.

Fondamenti di Informatica L- A

Processo computazionale iterativo

- Un processo computazionale iterativo si può realizzare anche tramite **funzioni ricorsive**
- Si basa sulla disponibilità di una variabile, detta *accumulatore*, destinata a esprimere in ogni istante la soluzione corrente
- Si imposta identificando quell'operazione di *modifica dell'accumulatore* che lo porta a esprimere, dal valore relativo al passo k , il valore relativo al passo $k+1$.

Esempio: calcolo iterativo del fattoriale

Definizione:

$$n! = 1 * 2 * 3 * \dots * n$$

Detto:

$$v_k = 1 * 2 * 3 * \dots * k:$$

$$\left\{ \begin{array}{l} 1! = v_1 = 1 \\ (k+1)! = v_{k+1} = (k+1) * v_k \quad \text{per } k \geq 1 \\ n! = v_n \quad \text{per } k = n \end{array} \right.$$

Esempio: calcolo iterativo del fattoriale

```
int fact(int n){
    return factIt(n, 1, 1);
}
```

Inizializzazione dell'accumulatore:
corrisponde al fattoriale di 1

Contatore del passo

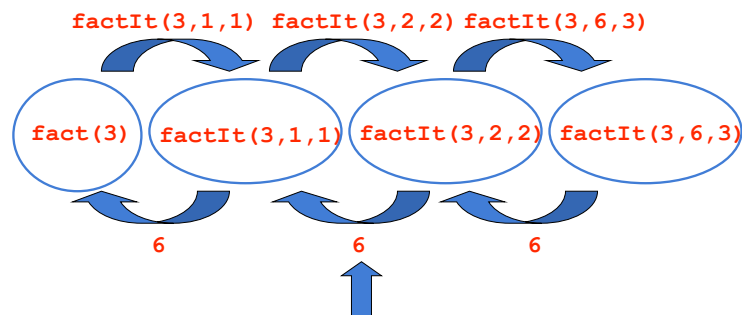
```
int factIt(int n, int F, int i){
    if (i < n)
        {F = (i+1)*F;
         i = i+1;
         return factIt(n, F, i);
        }
    return F;
}
```

Ad ogni chiamata ricorsiva
viene accumulato un
risultato intermedio in F

Accumulatore del risultato parziale

Fondamenti di Informatica L- A

Esempio: calcolo iterativo del fattoriale



Al passo i-esimo viene calcolato il fattoriale di i. Quando i = n l'attivazione della funzione corrispondente calcola il fattoriale di n.

NOTA: ciascuna funzione che effettua una chiamata ricorsiva si sospende, aspetta la terminazione del servitore e poi termina, cioè **NON EFFETTUA ALTRE OPERAZIONI DOPO**. [diversamente, nel caso del fattoriale ricorsivo vero e proprio, dopo la fine del servitore veniva effettuata una moltiplicazione]

Fondamenti di Informatica L- A

Esempio: calcolo iterativo del fattoriale

Confrontiamo la realizzazione mediante la funzione `factIt` con una funzione iterativa (senza ricorsione) :

```
int factIt(int n, int F, int i)
{
  if (i < n)
  {
    F = (i+1)*F;
    i = i+1;
    return factIt(n,F,i);
  }
  return F;
}
```

```
int fact(int n)
{
  int i;
  int F=1, i=2;
  while(i <= n)
  {
    F=F*i;
    i++;
  }
  return F;
}
```

Fondamenti di Informatica L- A

1 3

Fattoriale: Iterazione e Ricorsione

- il ciclo diventa un **if** con, in fondo, la **chiamata ricorsiva** (che realizza l'iterazione successiva)

```
while (condizione) {
  <corpo del ciclo>
}
```

```
if (condizione) {
  <corpo del ciclo>
  <chiamata ricorsiva>
}
```

Fondamenti di Informatica L- A

Fattoriale: Iterazione e Ricorsione

- La soluzione ricorsiva individuata per il fattoriale è *sintatticamente ricorsiva* ma dà luogo a un *processo computazionale ITERATIVO*

Ricorsione *apparente*, detta **RICORSIONE TAIL**

- Il risultato viene sintetizzato *in avanti*
 - ogni passo *decompone e calcola*
 - e *porta in avanti il nuovo risultato parziale* quando le chiamate si chiudono non si fa altro che riportare indietro, fino al cliente, il risultato ottenuto.

Ricorsione tail

- Una ricorsione che realizza un processo computazionale *ITERATIVO* è **una ricorsione apparente**
- la chiamata ricorsiva è sempre ***l'ultima istruzione***
 - *i calcoli sono fatti prima*
 - *la chiamata serve solo, dopo averli fatti, per proseguire la computazione*
- questa forma di ricorsione si chiama ***RICORSIONE TAIL*** (“ricorsione in coda”)

Esempio

Scrivere una versione ricorsiva dell'algoritmo di ordinamento di un vettore:

Soluzione:

```
#include <stdio.h>
#define n 5
void leggi(int*, int);
void stampa(int*, int);
void ordina (int*, int);
void scambia (int*, int*);

main()
{int i, a[n];

  leggi(a, n);
  ordina(a, n);
  stampa(a, n);
}
```

Fondamenti di Informatica L- A

Definizione tail-ricorsiva della funzione ordina:

```
void ordina(int *V, int N)
{int j, max, tmp;
  if (N==1) return;
  else
  {   for( max=N-1, j=0; j<N; j++)
      if (V[j]>V[max]) max=j;
      if (max<N-1)
          scambia( &V[N-1], &V[max] );
  }
  ordina(V, N-1); /* chiamata tail-ricorsiva*/
}
```

Fondamenti di Informatica L- A

```
void leggi(int a[], int dim)
{int i;

  printf ("Scrivi %d interi\n", dim);
  for (i = 0; i < dim; i++)
    scanf ("%d", &a[i]);
}

void stampa(int a[], int dim)
{int i;

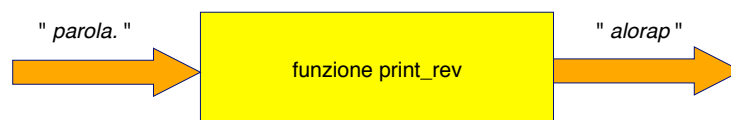
  printf ("Vettore:\n");
  for (i = 0; i < dim; i++)
    printf ("%d\t", a[i]);
}
```

Fondamenti di Informatica L- A

Esercizio

Scrivere una funzione ricorsiva `print_rev` che, data una sequenza di caratteri (terminata dal carattere '.') stampi i caratteri della sequenza in ordine inverso. La funzione non deve utilizzare stringhe.

Ad esempio:



Fondamenti di Informatica L- A

Esercizio

Osservazione: l'estrazione (*pop*) dei record di attivazione dallo stack avviene sempre in ordine inverso rispetto all'ordine di inserimento (*push*).

→ associamo ogni carattere letto a una nuova chiamata ricorsiva della funzione

Soluzione:

```
void print_rev(char car);
{ char c;
  if (car != '.')
  {   scanf("%c", &c);
      print_rev(c);
      printf("%c", car);
  }
  else return;
}
```

ogni record di attivazione nello stack memorizza un singolo carattere letto (*push*); in fase di *pop*, i caratteri vengono stampati nella sequenza inversa

Fondamenti di Informatica L- A

Soluzione

```
#include <stdio.h>
#include <string.h>
void print_rev(char car);
main()
{ char k;
  printf("\nIntrodurre una sequenza terminata da .:\t");
  scanf("%c", &k);
  print_rev(k);
}
void print_rev(char car)
{ char c;
  if (car != '.')
  {   scanf("%c", &c);
      print_rev(c);
      printf("%c", car);
  }
  else return;
}
```

Fondamenti di Informatica L- A

Stack

Codice

```
...
main(void)
{
    ...
    print_rev(k);
}
void print_rev(char car);
{
    if (car != '.')
    {
        ...
        print_rev(c);
        printf("%c", car);
    }
    else return;
}
```

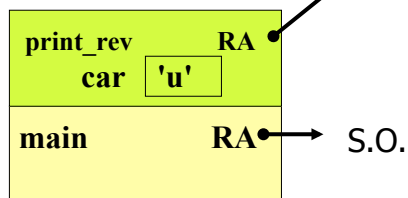


Standard Input:
"uno."

Stack

Codice

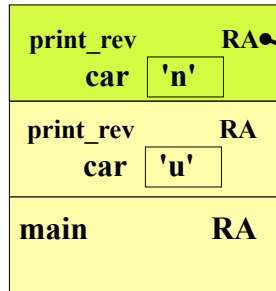
```
...
main(void)
{
    ...
    print_rev(k);
}
void print_rev(char car);
{
    if (car != '.')
    {
        ...
        print_rev(c);
        printf("%c", car);
    }
    else return;
}
```



Standard Input:
"u"no."

Stack

Codice

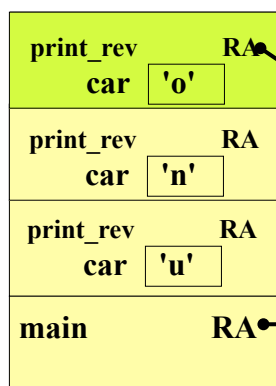


```
...
main(void)
{
    ...
    print_rev(k);
}
void print_rev(char car);
{
    if (car != '.')
    {
        ...
        print_rev(c);
        printf("%c", car);
    }
    else return;
}
```

Standard Input:
"uno."

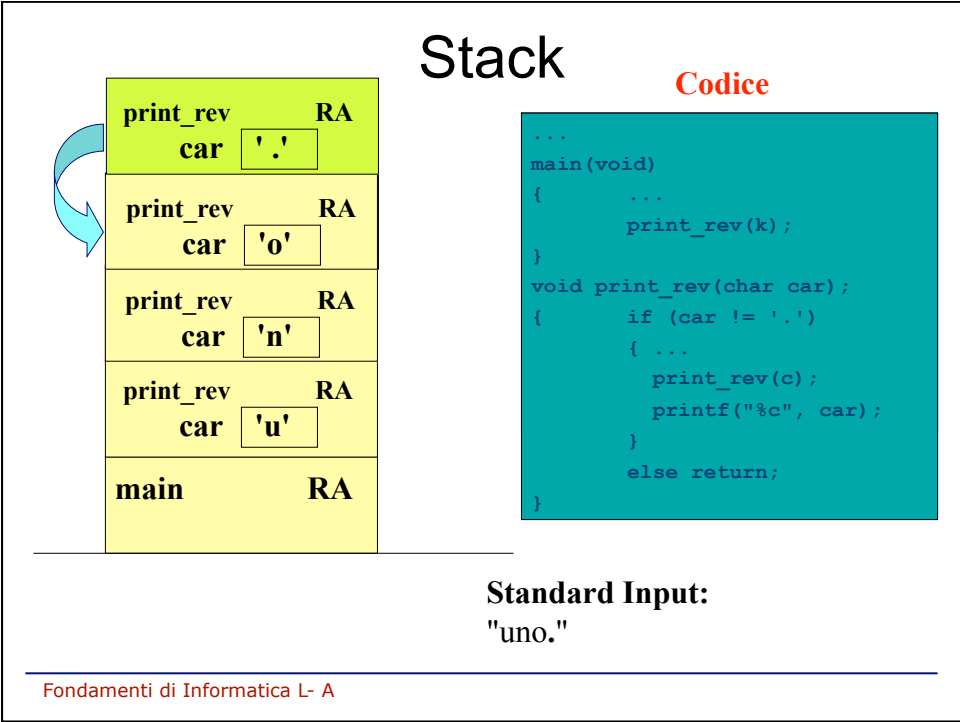
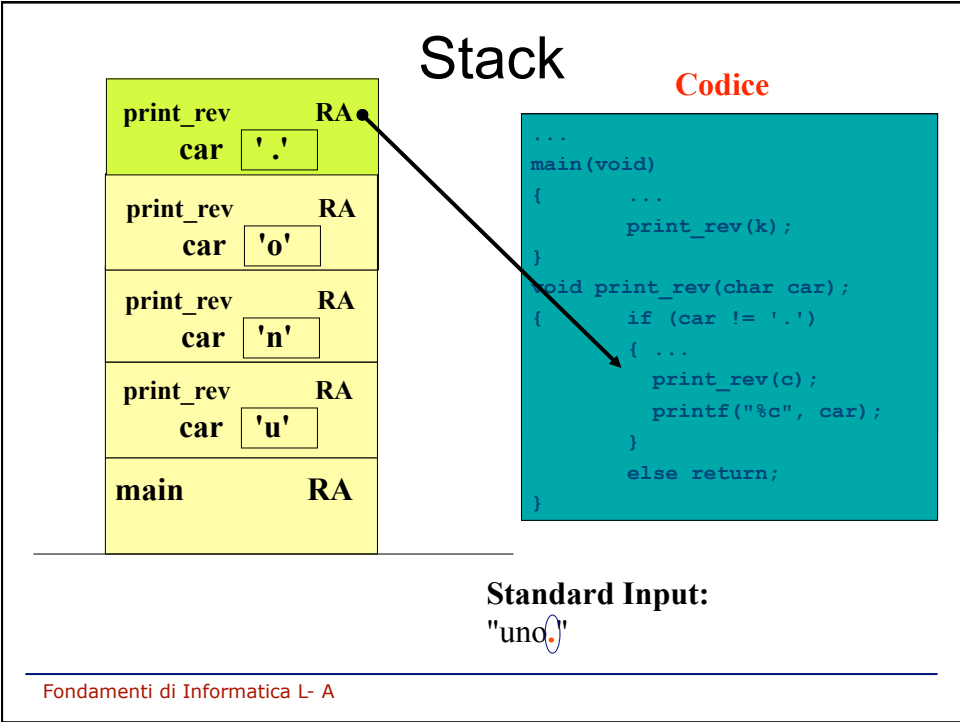
Stack

Codice



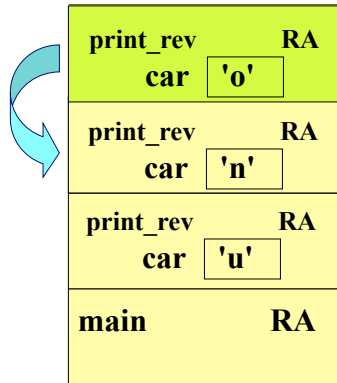
```
...
main(void)
{
    ...
    print_rev(k);
}
void print_rev(char car);
{
    if (car != '.')
    {
        ...
        print_rev(c);
        printf("%c", car);
    }
    else return;
}
```

Standard Input:
"uno."



Stack

Codice



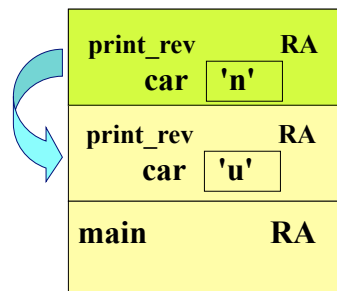
```
...
main(void)
{
    ...
    print_rev(k);
}
void print_rev(char car);
{
    if (car != '.')
    { ...
        print_rev(c);
        printf("%c", car);
    }
    else return;
}
```

Standard output:
"o"

Standard Input:
"uno."

Stack

Codice



```
...
main(void)
{
    ...
    print_rev(k);
}
void print_rev(char car);
{
    if (car != '.')
    { ...
        print_rev(c);
        printf("%c", car);
    }
    else return;
}
```

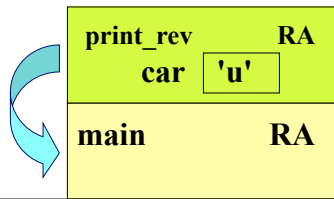
Standard output:
"on"

Standard Input:
"uno."

Stack

Codice

```
...
main(void)
{
    ...
    print_rev(k);
}
void print_rev(char car;
{
    if (car != '.')
    { ...
        print_rev(c);
        printf("%c", car);
    }
    else return;
}
```



Standard output:
"onu"

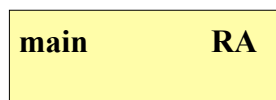
Standard Input:
"uno."

Fondamenti di Informatica L- A

Stack

Codice

```
...
main(void)
{
    ...
    print_rev(k);
}
void print_rev(char car;
{
    if (car != '.')
    { ...
        print_rev(c);
        printf("%c", car);
    }
    else return;
}
```



Standard output:
"onu"

Standard Input:
"uno."

Fondamenti di Informatica L- A