# Tecniche della Programmazione – lezione 21

## Ricorsione

- **differenze tra uno schema iterativo e uno schema ricorsivo: nello schema ricorsivo**
  - un'azione su un certo gruppo di dati viene descritta come l'esecuzione di quella stessa azione su quel medesimo gruppo di dati (o un po' piu` piccolo ... meglio)

- **come si definisce un processo ricorsivo (*aka* induttivo)**
  - c'e` un **CASO INDUTTIVO** e c'e` un **CASO BASE,** nei quali l'azione si dirama (parentesi graffa ...)
  - anzi ci possono essere piu` casi induttivi e piu` casi base ...

- **come visualizzare, seguire, analizzare e predire il comportamento di uno schema ricorsivo?**
  - visualizzazione logica (a linee, anzi "alinee")
  - visualizzazione grafica (RDA unleashed!)

- **esercizi** e tecnica di incapsulamento di una chiamata ricorsiva, con troppi parametri, in una funzione che si occupa di chiamarla

**si esegue l'azione S, su un insieme di dati D, mediante eventuale esecuzione di S su D' $\subset$ D**

**esempio**

**CERCA 90 NEL SACCHETTO** = **estrai num**

| Casi | Effetti |
|---|---|
| num ≠ 90 | CERCA 90 NEL SACCHETTO |
| num = 90 | return 1 |

risultato, caso per caso

**si esegue l'azione S, su un insieme di dati D, mediante eventuale esecuzione di S su D' $\subset$ D**

**esempio**

**CERCA 90 NEL SACCHETTO** = **estrai num**

| Casi | Effetti |
|---|---|
| num ≠ 90 | CERCA 90 NEL SACCHETTO |
| num = 90 | return 1 |

**caso induttivo**
(in cui viene attivata un'azione ricorsiva, cioe` una nuova esecuzione di S)

**caso base**

risultato, caso per caso

**L'azione S prevede di eseguire S stessa su un insieme più piccolo di dati: CERCA 90 NEL SACCHETTO viene definita "in termini" di se' stessa**

**in uno schema iterativo, l'azione complessiva e` definita come una sequenza, ripetuta, di** altre azioni diverse da quella complessiva

**CERCA 90 NEL SACCHETTO** = **while !fatto**

```
- estrai num
- if (num == 90)
      fatto=1
```

**no, questo non e` ricorsivo**

# formulazione migliore del primo esempio

**CERCA 90 NEL SACCHETTO** =

- **sacchetto vuoto** → `return 0` **A)**
- **sacchetto non vuoto** → `estrai num`, poi:
  - `num ≠ 90` → **CERCA 90 NEL SACCHETTO** **C)**
  - `num = 90` → `return 1` **B)**

**A)** caso base (90 non è nel sacchetto)

**B)** caso base (trovato!)

**C)** caso induttivo (bisogna cercare ancora)

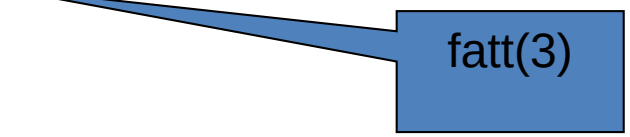
# fattoriale: un esempio di funzione matematica ... ben nota ...

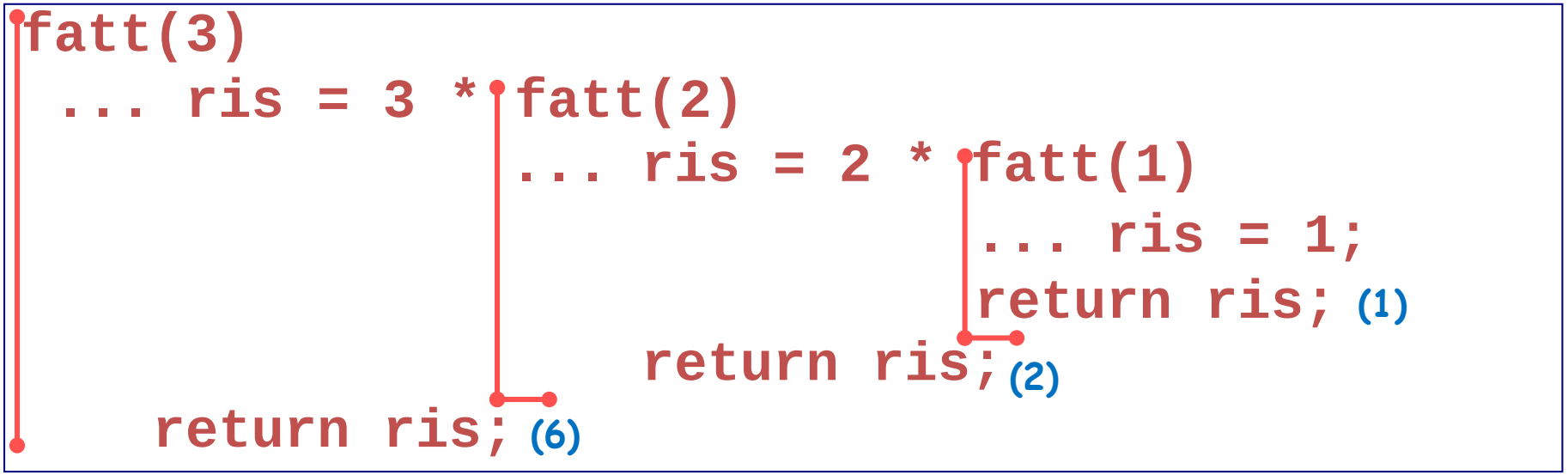
$$fatt(n) = \begin{cases} \text{se } n==1 & ris = 1; \text{ return } ris \\ \text{else} & ris = n * fatt(n-1); \text{ return } ris \end{cases}$$

```
long int fatt (int n) {
 long int ris;
 if (n==1)
   ris = 1;
 else ris = n*fatt(n-1);
return ris;
}
```

esempio

```
int main() {
 ...
 scanf("%d", &k);
 printf("...%ld...", fatt(k))
...
```



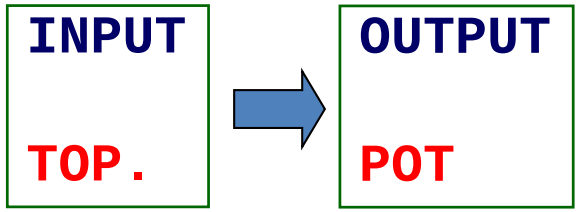



```
long int fatt (int n) {
 long int ris;
 if (n==1)
   ris = 1;
 else ris = n*fatt(n-1);
return ris;
}
```

# Stampa Invertita dell'INPUT

**LEGGI&STAMPAINV**

**legge una sequenza di caratteri da stdin, terminata da '.'**
**e la stampa su video in ordine inverso**

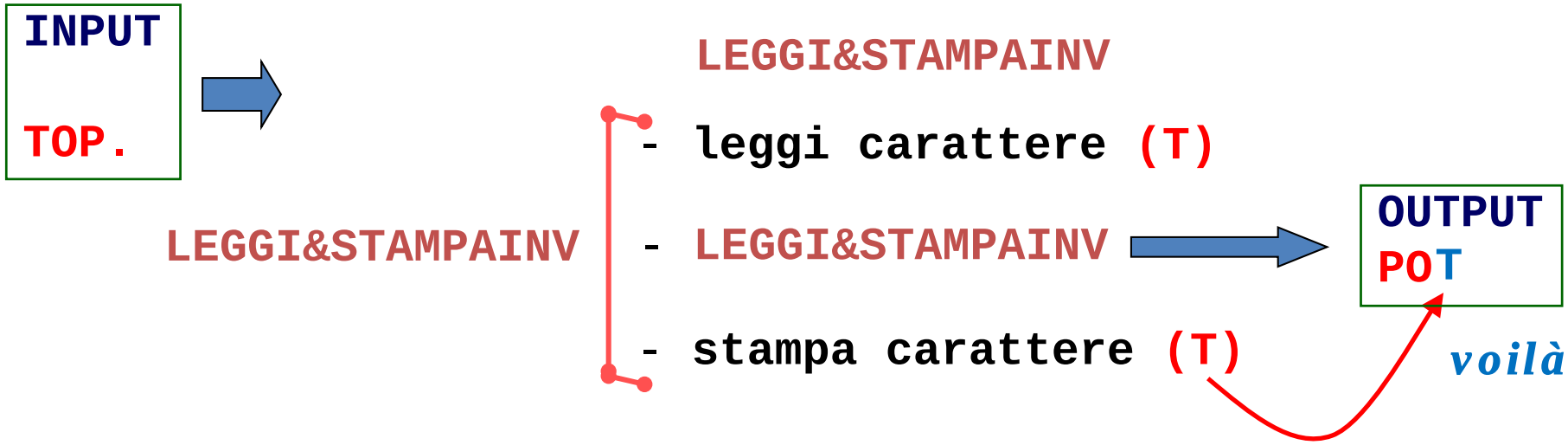



? **leggo un carattere e poi stampo invertito il resto e poi stampo il carattere ...**

# Stampa Invertita dell'INPUT

**LEGGI&STAMPAINV**

**legge una sequenza di caratteri da stdin, terminata da '.'**
**e la stampa su video in ordine inverso**




**questa e` la prima stesura ... quando leggo un '.' non devo stamparlo, ma devo interpretarlo come un caso in cui ci si ferma ... come?**

# Stampa Invertita dell'INPUT: leggi&StampaInv()

**(Parte di) LEGGI&STAMPAINV()**

**Leggi carattere ch**

- **ch ≠ '.'** → **LEGGI&STAMPAINV()**, **print ch**
- **senno`** → **---**

```
leggi ch                                (1)
se(ch != '.') {
                                        (2)


}                                       ☺
```

**LEGGI&STAMPAINV**

(1) - **leggi carattere (T)**

(2) - **LEGGI&STAMPAINV**

(3) - **stampa carattere (T)**

# Stampa Invertita dell'INPUT: leggi&StampaInv()

(Parte di) **LEGGI&STAMPAINV()**


```
leggi ch                          (1)
se (ch != '.') {
   leggi&StampaInv()              (2)
   stampa ch                      (3)
}
```

**visualizzazione** (logica) **dell'esecuzione**

**INPUT:** TOP.

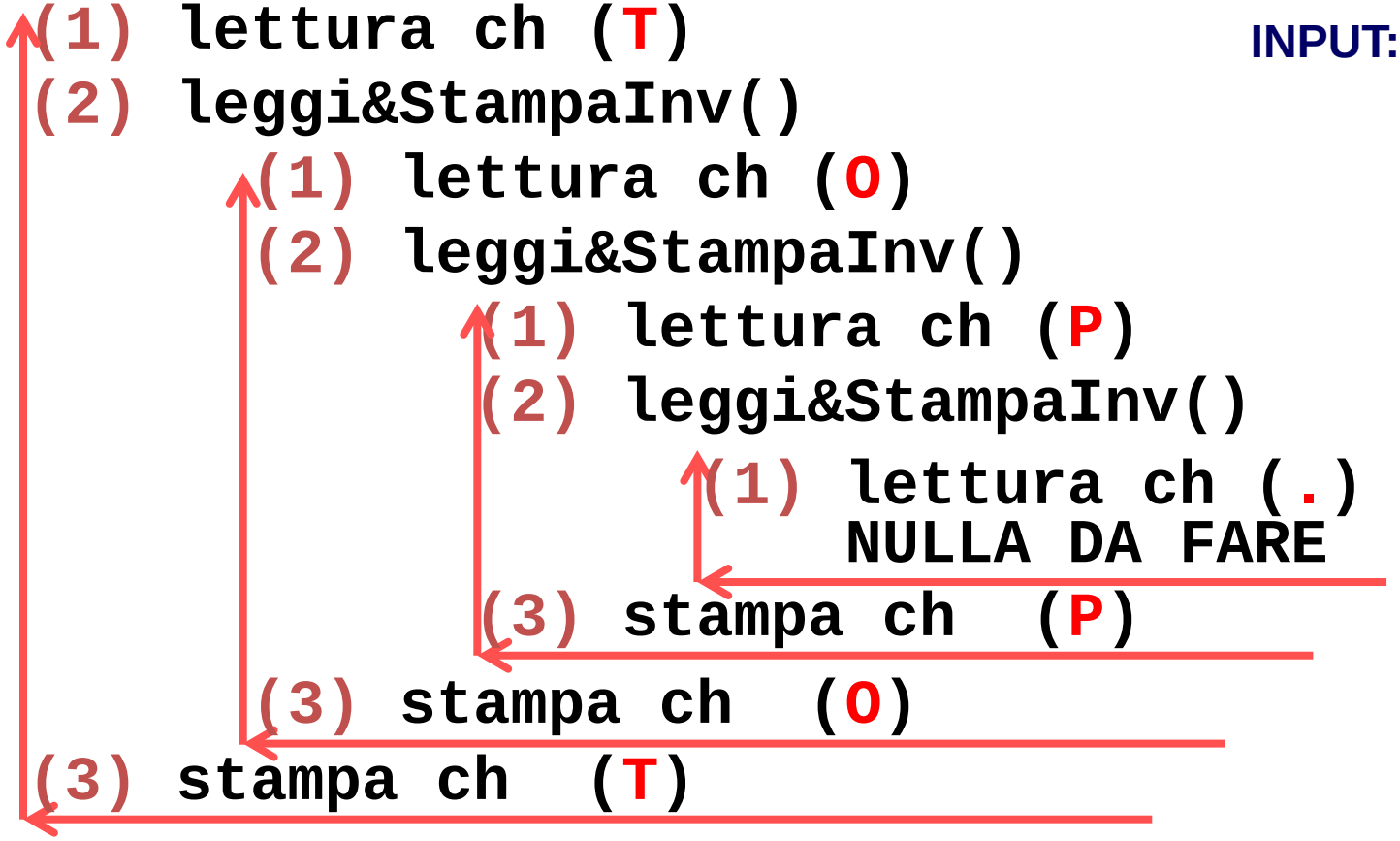
```
(1) lettura ch (T)
     (T!='.')
(2) leggi&StampaInv()
      (1) lettura ch (O)
           (O!='.')
      (2) leggi&StampaInv()
            (1) lettura ch (P)
                 (P!='.')
            (2) leggi&StampaInv()
```

continua ...

# Stampa Invertita dell'INPUT: leggi&StampaInv()

```
leggi ch                          (1)
se (ch != '.') {
   leggi&StampaInv()              (2)
   stampa ch                      (3)
}
```

**visualizzazione logica dell'esecuzione**




**INPUT:** TOP.

**OUTPUT:** POT

# fattoriale: visualizzazione mediante RDA

☺ **RDA della chiamata `fatt(3)` ?**

```
long int fatt (int n) {
 long int ris;
 if (n==1)
   ris = 1;
 else ris = n*fatt(n-1);
return ris;
}
```

```
long int fatt (int n) {
 long int ris;
 if (n==1)
   ris = 1;
 else ris = n*fatt(n-1);
return ris;
}
```

☺ **RDA della chiamata** `fatt(3)` **?**

**fatt(3)**

PAR
n 3
LOC
ris 3 *...

```
{ if (n==1)
     ris = 1;
  else ris =
  n*fatt(n-1);
```

**fatt(2)**

PAR
n 2
LOC
ris 2 *...

```
{ if (n==1)
     ris = 1;
  else ris =
  n*fatt(n-1);
```

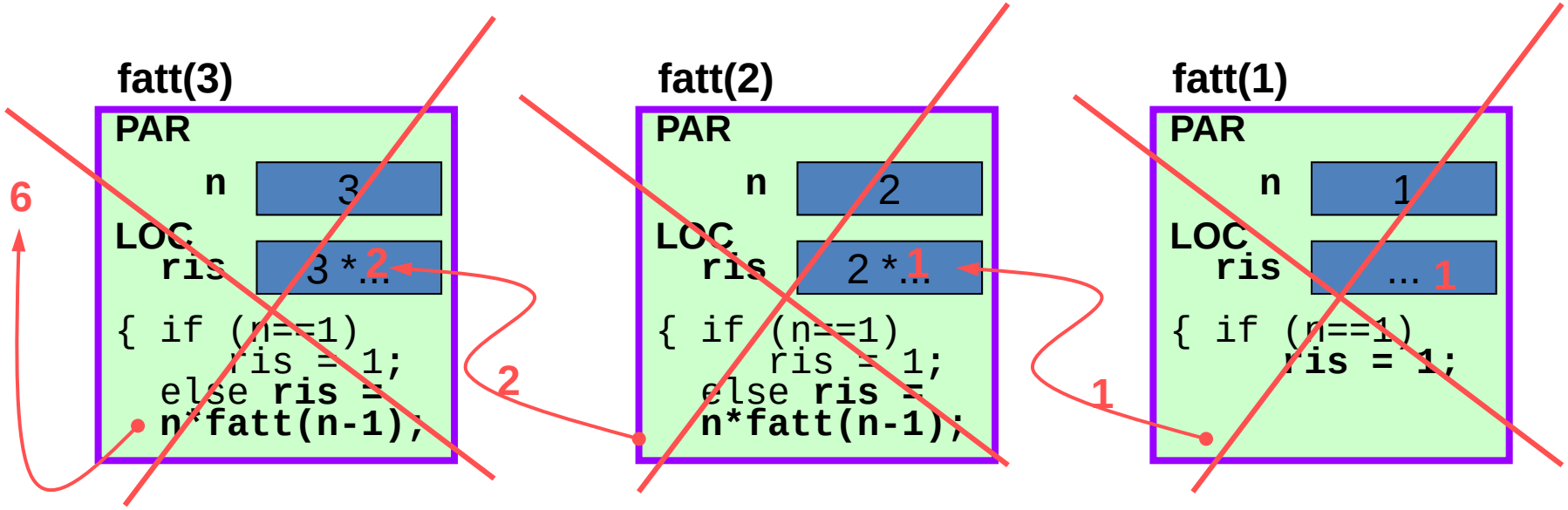
**fatt(1)**

PAR
n 1
LOC
ris ...

```
{ if (n==1)
     ris = 1;
```

```
long int fatt (int n) {
 long int ris;
 if (n==1)
   ris = 1;
 else ris = n*fatt(n-1);
return ris;
}
```

☺ **RDA della chiamata** `fatt(3)` **?**



**deve essere D'$\not\subset$ D:**
**ci si deve garantire che la serie di attivazioni ricorsive abbia un termine**

**es.**

$$sfatt(n) = \begin{cases} \text{se } n==1 & \text{return } 1 \\ \text{else} & \text{return } n * sfatt(n+1); \end{cases}$$

```
long int sfatt (int n) {

 if (n==1)
   return 1;
 else return (n*sfatt(n+1));
}
```

☺ RDA per la chiamata sfatt(3)

**deve essere D'⊄ D:**
**ci si deve garantire che la serie di attivazioni ricorsive abbia un termine**

**es.**

$$sfatt(n) = \begin{cases} \text{se } n==1 & \text{return } 1 \\ \text{else} & \text{return } n * sfatt(n+1); \end{cases}$$

```
long int sfatt (int n) {

 if (n==1)
   return 1;
 else return (n*sfatt(n+1));
}
```


**sequenza infinita di chiamate ricorsive!**

**soluzione ricorsiva**

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

+ conciso
+ elegante
+ aderente alla definizione induttiva formale

**soluzione iterativa**

```
int fattITER (int n) {
  int i, prod=1;

  for (i=2; i<=n; i++)
    prod *= i; /* prod=prod*i */
return prod;
}
```

+ efficiente
+ strutture dati (a cura del programmatore)

Funzione che legge una sequenza di caratteri da INPUT, terminata da '.' e la ristampa invertita (si`, lo abbiamo gia` incontrato ...)

**soluzione iterativa: NMAX fissato (e` un limite, anche se aggirabile …)**

- Usa array [NMAX] caratteri
- Lettura da INPUT in array
- Conta = var contatore (quanti caratteri letti)
- Stampa da [conta-1] a [0]

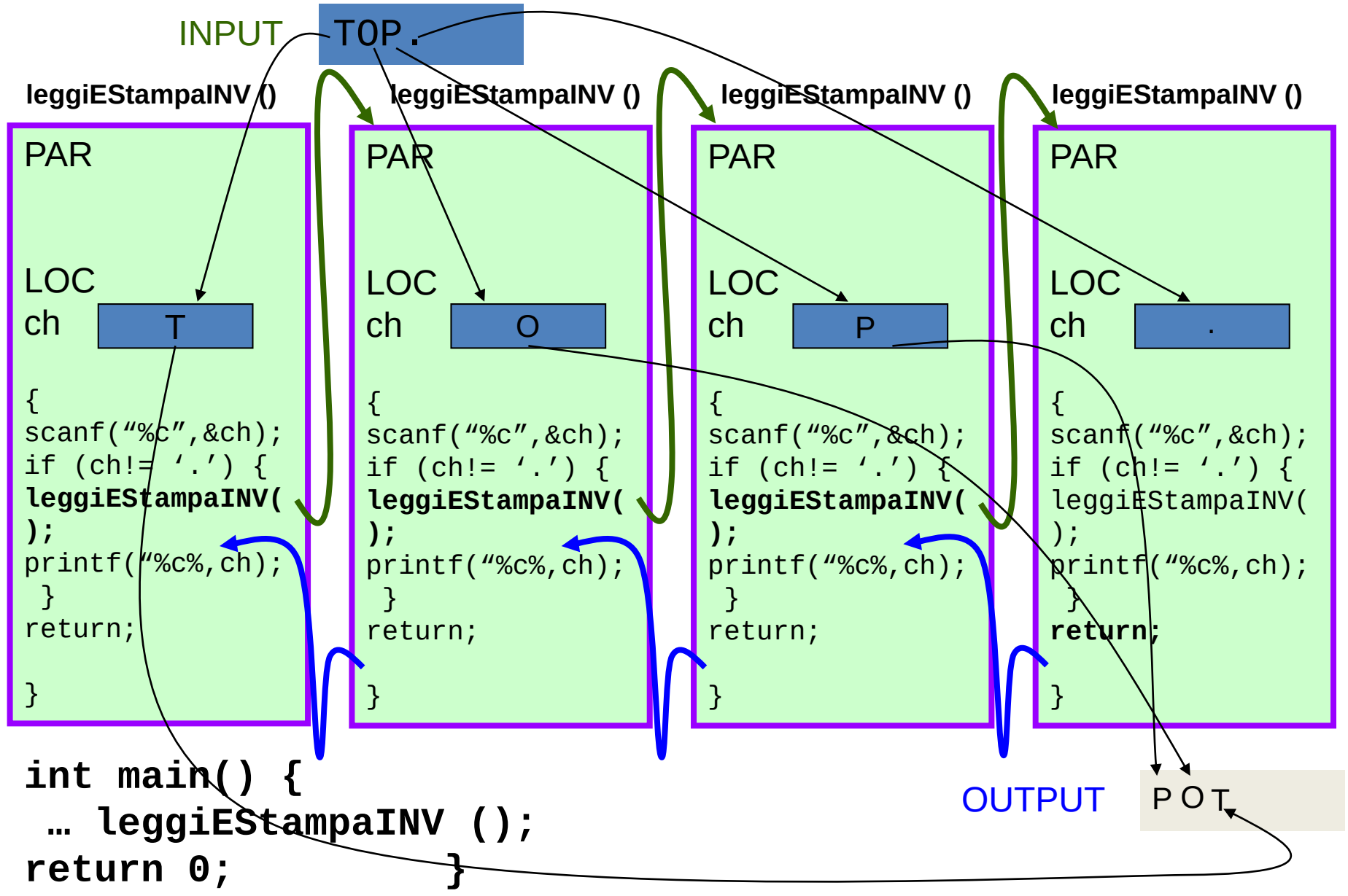
**soluzione ricorsiva**

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

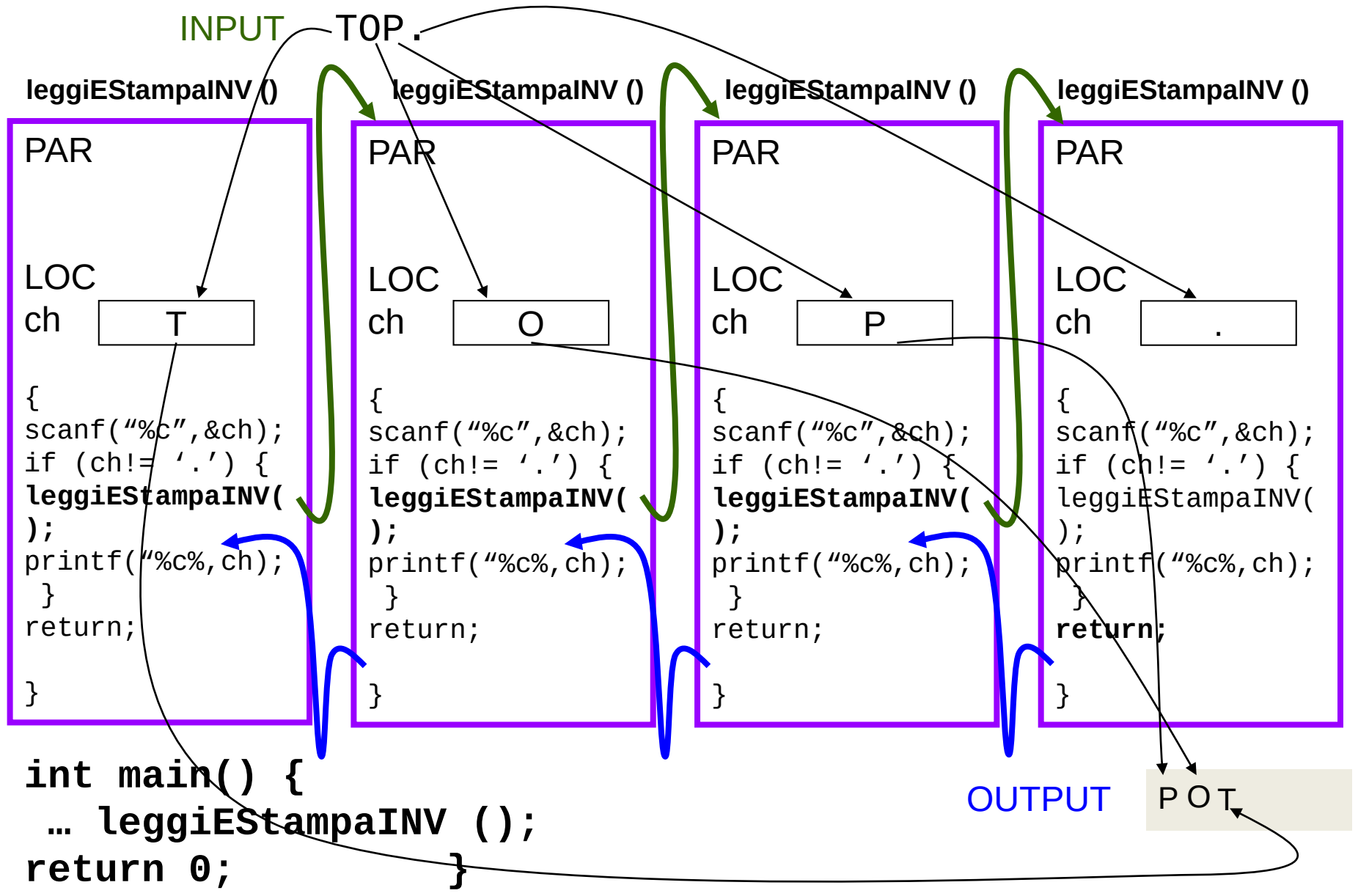
☺

**input TOP.**

# Esecuzione leggi&StampaInv()



# Esecuzione leggi&StampaInv()



PALINDROMIA: funzione intera che riceve una stringa e dice se si tratta di una stringa palindroma

**X Y W Z Z W Y X**

- verifica se primo = ultimo
- " se secondo = penultimo
- " ...
- " se inMezzo = inMezzo+1

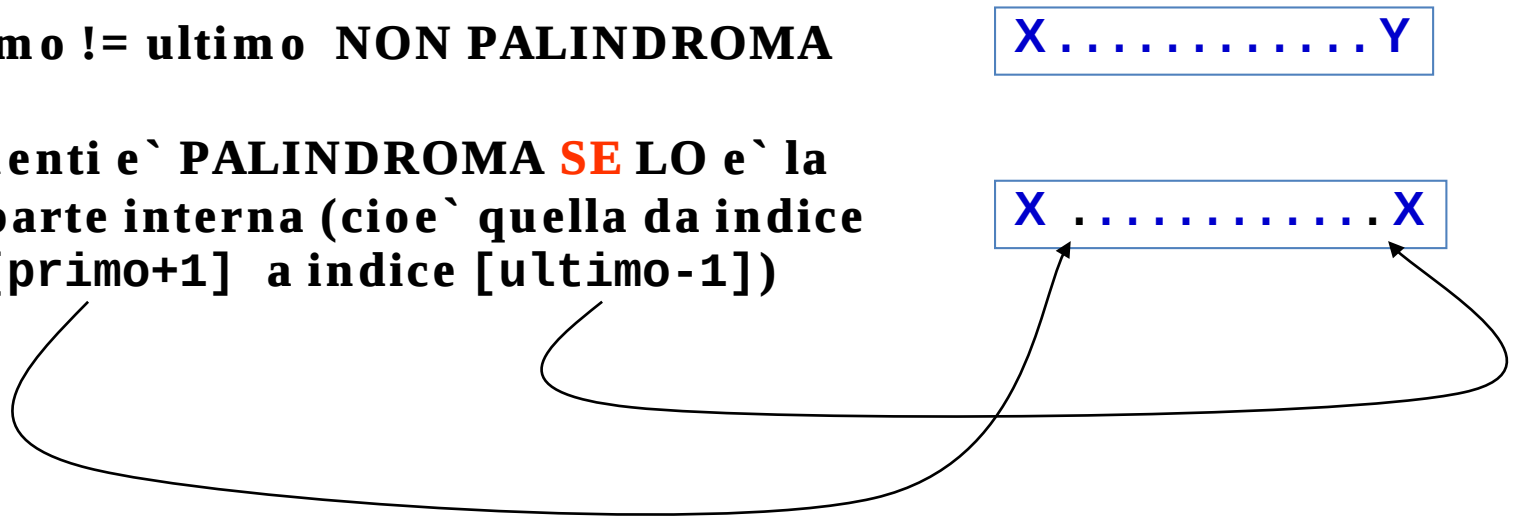
**X Y W Z W Y X**

- verifica se primo = ultimo
- " se secondo = penultimo
- " ...
- carattere centrale non controllato

**soluzione ricorsiva**

**se primo != ultimo NON PALINDROMA**

**altrimenti e` PALINDROMA SE LO e` la parte interna (cioe` quella da indice [primo+1] a indice [ultimo-1])**



**soluzione ricorsiva**

**se primo != ultimo NON PALINDROMA**

X.....Y

**altrimenti e` PALINDROMA SE LO e` la parte interna (cioe` quella da indice [primo+1] a indice [ultimo-1])**

X.....X

```
int palin ( char *parola,
            int indicePrimoCar,
            int indiceUltimoCar ) {
```

per realizzare questo schema, bisogna avere a disposizione gli indici su cui si lavora

i
j

palin(par,i,j) =

- par[i]==par[j] return palin(par, i+1, j-1)
- par[i]!=par[j] return 0

```
palin(par,i,j) =
```

```
par[i]==par[j]   return palin(par, i+1, j-1)
```

```
par[i]!=par[j]   return 0
```

memoria

```
par   A S O R R O S A
```

```
          i j
palin(par,0,7)
A==A -> palin(par,1,6)
        S==S -> palin(par,2,5)
                O==O -> palin(par,3,4)
                                  i j
                        R==R -> palin(par,4,3)   !!!!
```

(i>j) e` un caso base in cui

```
return 1
```

```
palin(par,i,j) =
    par[i]==par[j]   return palin(par, i+1, j-1)
    par[i]!=par[j]   return 0
```

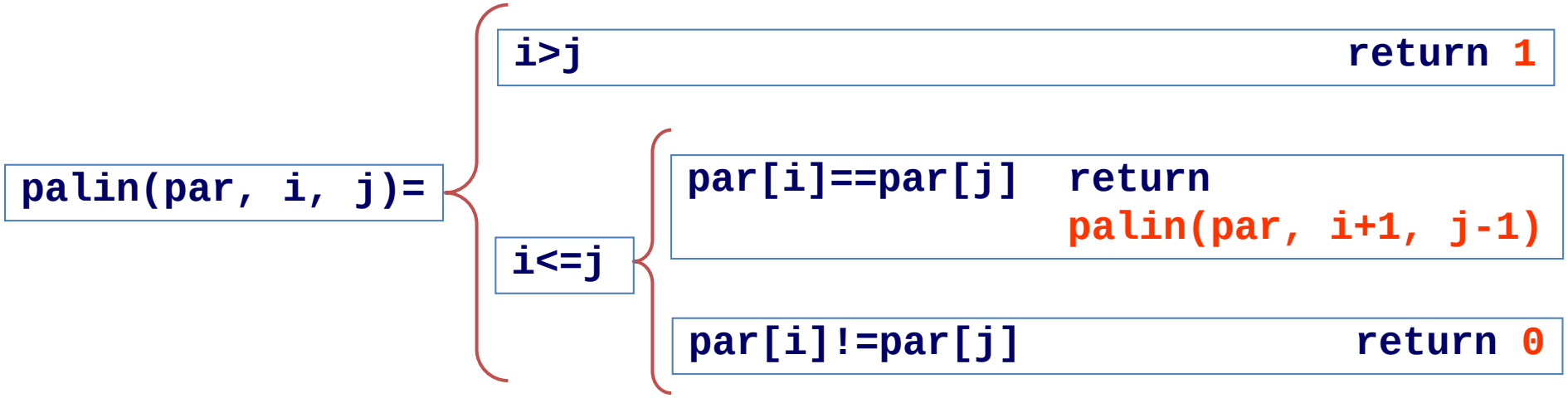
memoria

| par | A S O R R O S A |
|---|---|

```
return 1
```

(i>j) e` un caso base in cui

```
palin(par, i, j)=
    i>j                          return 1
    i<=j
        par[i]==par[j]   return palin(par, i+1, j-1)
        par[i]!=par[j]               return 0
```




memoria

| par | A | N | I | L | I | N | A |
|---|---|---|---|---|---|---|---|

```
palin(par,0,6)
A==A -> palin(par,1,5)
        N==N -> palin(par,2,4)
                I==I -> palin(par,3,3)    !!!!
```

(i==j) e` un caso base
in cui return 1

```
palin(par, i, j)=
    i>j                                  return 1
    i<=j
        par[i]==par[j]   return
                         palin(par, i+1, j-1)
        par[i]!=par[j]                   return 0
```

par ANILINA

(i==j) e` un caso base
in cui return 1

```
palin(par,i,j) =
    i > j                                return 1
    i == j                               return 1
    i < j
        par[i]==par[j]    return
                          palin(par, i+1, j-1)
        par[i]!=par[j]    return 0
```

**algoritmo**

```
palin(par,i,j) = { i > j                          return 1
                 { i == j                         return 1
                 { i < j  { par[i]==par[j]        return palin(par, i+1, j-1)
                          { par[i]!=par[j]        return 0
```

**funzione palin()**

☺

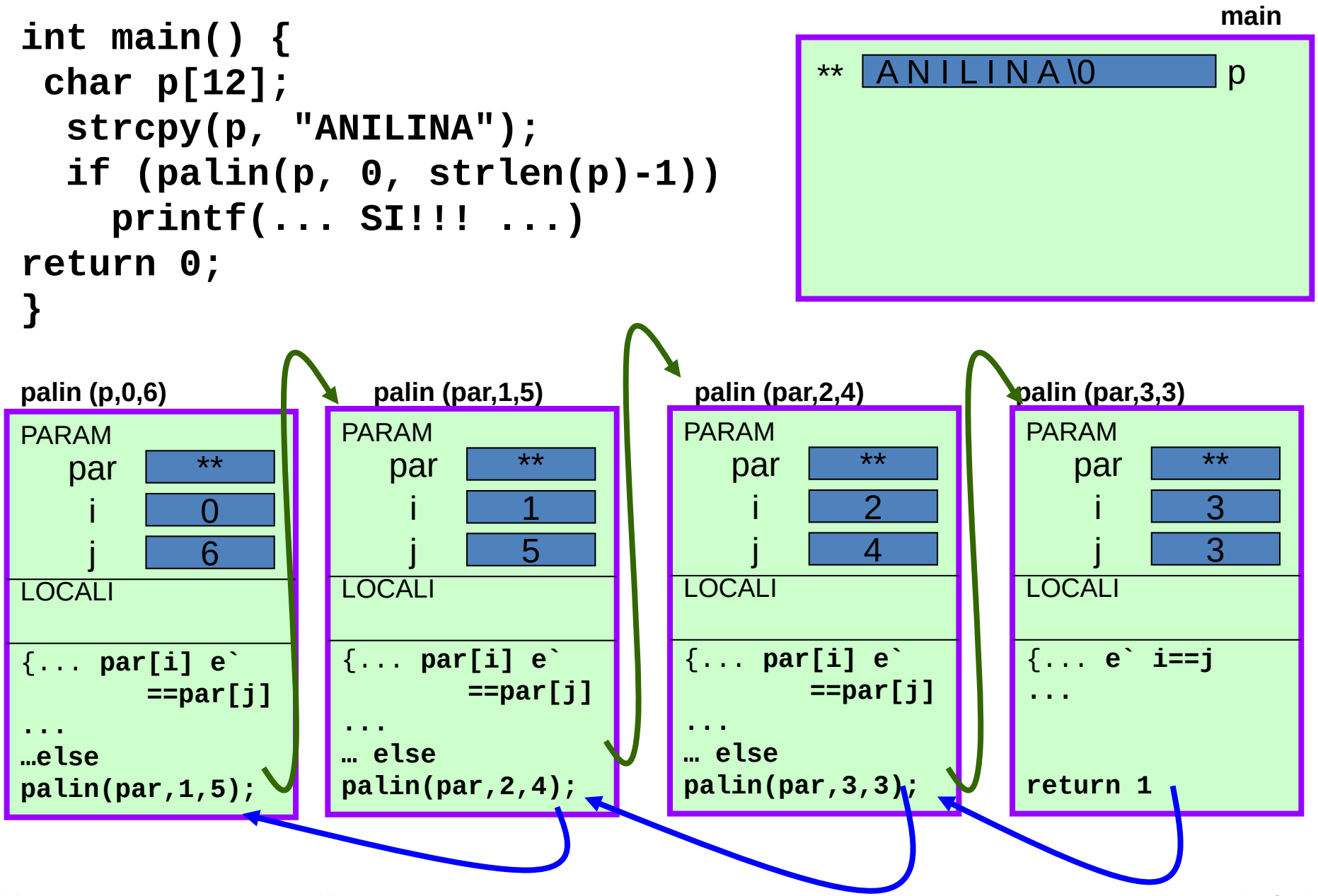
**algoritmo**

```
palin(par,i,j) = { i > j                          return 1
                 { i == j                         return 1
                 { i < j  { par[i]==par[j]        return palin(par, i+1, j-1)
                          { par[i]!=par[j]        return 0
```

**funzione palin()**

```
int palin (char *par, int i, int j) {
 if (i>=j)
    return 1;
 else if (par[i]!=par[j])
         return 0;
      else return palin(par, i+1, j-1);
}
```

```
int main() {
 char p[12];
  strcpy(p, "ANILINA");
  if (palin(p, 0, strlen(p)-1))
    printf(... SI!!! ...)
return 0;
}
```



# Funzione che incapsula la chiamata a palin

in `palin`, la chiamata va corredata di informazioni che non sono essenziali dal punto di vista della "logica" della funzione: se la funzione calcola la palindromia di una parola, ci si aspetterebbe di dover solo passare la parola ... e non di dover anche specificare un intervallo di caratteri da prendere in considerazione ...

se si vuole che l'utente abbia a disposizione una funzione di uso più semplice e logico si può definire una funzione che riceva solo la parola e poi incapsuli una chiamata completa alla funzione palin; in tal modo la chiamata alla funzione palindroma() viene fatta passando solo la parola da considerare.

☺ **qual e` l'intestazione della funzione?**

```
int main() {      char p[12];
  strcpy(p, "ANILINA");
  if (palindroma(p))
    printf(... SI!!! ...)
return 0;
}
```

# Funzione che incapsula la chiamata a palin

in `palin`, la chiamata va corredata di informazioni che non sono essenziali dal punto di vista della "logica" della funzione: se la funzione calcola la palindromia di una parola, ci si aspetterebbe di dover solo passare la parola ... e non di dover anche specificare un intervallo di caratteri da prendere in considerazione ...

se si vuole che l'utente abbia a disposizione una funzione di uso più semplice e logico si può definire una funzione che riceva solo la parola e poi incapsuli una chiamata completa alla funzione palin; in tal modo la chiamata alla funzione palindroma() viene fatta passando solo la parola da considerare.

```
int palindroma(char *parola) {

        ☺  qual e` il codice da eseguire per
    restituire il valore di palindromia di "parola"?
}

int main() {      char p[12];
  strcpy(p, "ANILINA");
  if (palindroma(p))
    printf(... SI!!! ...)
return 0;
}
```

# Funzione che incapsula la chiamata a palin

in `palin`, la chiamata va corredata di informazioni che non sono essenziali dal punto di vista della "logica" della funzione: se la funzione calcola la palindromia di una parola, ci si aspetterebbe di dover solo passare la parola ... e non di dover anche specificare un intervallo di caratteri da prendere in considerazione ...

se si vuole che l'utente abbia a disposizione una funzione di uso più semplice e logico si può definire una funzione che riceva solo la parola e poi incapsuli una chiamata completa alla funzione palin; in tal modo la chiamata alla funzione palindroma() viene fatta passando solo la parola da considerare.

```
int palindroma(char *parola) {
  return palin(parola, 0, strlen(parola)-1);
}


int main() {      char p[12];
  strcpy(p, "ANILINA");
  if (palindroma(p))
    printf(... SI!!! ...)
return 0;
}
```