

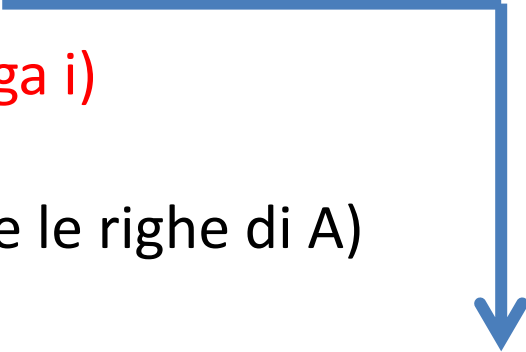
# esempi di correttezza

vedere testo: esercizio 3 (20/1) è 4.7  
del testo

vedere anche 6.1 e 6.2

lettura di int A[4][5] da INP

```
for(int i=0; i<4; i++) //R1=(0<=i<=4, lette righe 0..i-1 di A)
{
  //leggere la riga i
  //POST2=(letta la riga i)
}
//POST1=(lette tutte le righe di A)
```



```
for(int j=0; j<5; j++) //R2=(0<=j<=5, letto A[i][0..j-1])
```

```
INP>>A[i][j];
```

```
//POST2
```

**//C=A \* B**

```
for(int i=0; i<4; i++) //R1=(0<=i<=4, calcolate righe 0..i-1 di C)
{
  for(int j=0; j<6; j++)//R2=(0<=j<=6, calcolata C[i][0..j-1]
  {
    //calcola C[i][j]
  }
  //POST2=(calcolata C[i])
}
//POST1=(calcolata C)
```

```
{  
  int som=0;  
  for(int k=0; k<5; k++)//R3  
    som=som+A[i][k]*B[k][j];  
  //POST3  
  C[i][j]=som;  
}
```

//R3=(som=A[i][0]\*B[0][j] + A[i][1]\*B[1][j]+.. A[i][k-1]\*B[k-1][j]

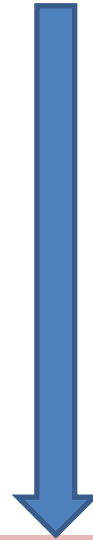
//POST3=(som= somma dei prodotti A[i][n]\*B[n][j] per  
//tutti gli n in [0..4])

## //esercizio 2 della settimana 27-3

```
for(int i=0; i<6; i++)  
//R1=(0<=i<=4, calcolate righe 0..i-1 di B)  
{  
    for(int j=0; j<8; j++)  
        //R2=(0<=j<=6, calcolata B[i][0..j-1]  
        {  
            //calcola B[i][j] → cioè un booleano  
        }  
    //calcolata la riga B[i])  
}  
//POST1=(calcolata tutta B).
```

```
bool ok=true;
for (int k=0; k<8; k++) // scorre riga A[i]
{
    bool trovato= ??
    for(int n=0; n<6; n++) { //A[i][k] è in A[][j]?

        ok=ok && trovato;
    }
    //ok sse A[i] è contenuta in A[][j]
    B[i][j]=ok;
}
```



```
if(A[i][k]==A[n][j]) trovato=true;
```

```
bool trovato=false; //all'inizio non l'abbiamo trovato
for(int n=0; n<6; n++) //R4
    if(A[i][k]==A[n][j])
        trovato=true;
//POST4=(trovato sse A[i][k] è in A[0..5][j])
```

R4=(  $0 \leq n \leq 6$ , trovato sse  $A[i][k]$  è in  $A[0..n-1][j]$ )

1. trovato=false  $\Rightarrow A[i][k]$  non è in  $A[0..-1][j]$  ovvio
2. se  $A[i][k]==A[n][j]$  facciamo trovato=true e  $n++$
3.  $R4 \ \&\& \ n=6 \Rightarrow$  trovato sse  $A[i][k]$  è in  $A[0..5][j]$  = POST4

2. se  $A[i][k] == A[n][j]$  facciamo  $trovato = true$  e  $n++$

quindi se  $trovato$  era già  $true$  lo resta, mentre se era  $false$  (questa è la prima volta che troviamo  $A[i][k]$ ) da  $false$  diventa  $true$

inoltre se  $A[i][k] != A[n][j]$

$trovato$  non cambia visto che aver considerato  $A[n][j]$  non aggiunge nulla



questa prova non funzionerebbe per questo ciclo:

```
for(int n=0; n<6; n++) //R
```

```
    if(A[i][k]==A[n][j])
```

```
        trovato=true;
```

```
    else
```

```
        trovato=false;
```

```
R=(trovato sse A[i][k]==A[n-1][j])
```

Non va     R && n=6 (non =>) POST4

questo programma non avrebbe senso. Perché ?

torniamo all'invariante che funziona:

$R4 = (0 \leq n \leq 6, \text{ trovato sse } A[i][k] \text{ è in } A[0..n-1][j])$

ci dice che stiamo facendo cose inutili se  $A[i][k]$  l'abbiamo trovato prima di  $A[n-1][j]$ , perché dovremmo considerare  $A[n-1][j]$  ?

E' inutile !!! Possiamo affinare la condizione di permanenza in modo da uscire dal ciclo non appena troviamo  $A[i][k]$

```
bool trovato=false;  
for(int n=0; n<6 && !trovato; n++) //R4'  
    if(A[i][k]==A[n][j])  
        trovato=true;  
//POST4=(trovato sse A[i][k] è in A[0..5][j])
```

$R4' = (0 \leq n \leq 6, A[i][k] \text{ non è in } A[0..n-2][j],$   
 $\text{trovato sse } A[i][k] == A[n-1][j])$

bisogna rifare la prova 3:

$R4' \ \&\& \ ! (n < 6 \ \&\& \ ! \text{trovato}) = R4' \ \&\& (n \geq 6 \ || \ \text{trovato})$

OR implica che ci sono 2 casi:

i) trovato = true e n in [0..6]

ii) trovato = false && n=6

i)  $R4' \ \&\& \text{ trovato } (0 \leq n \leq 6)$

$R4' = (0 \leq n \leq 6, A[i][k] \text{ non è in } A[0..n-2][j],$   
 $\text{trovato sse } A[i][k] == A[n-1][j])$

$\Rightarrow \text{POST4} = (\text{trovato sse } A[i][k] \text{ è in } A[0..5][j])$

potremmo dimostrare una POST **più forte**  
(trovato sse  $A[i][k] = A[n-1][j]$  ed  $n$  è il minimo indice in  
 $[1..6]$  per cui questa uguaglianza vale)

Ma qui non ci serve

ii)  $R4' \ \&\& \ (\text{trovato}=\text{false} \ \&\& \ n=6)$

$R4' = (0 \leq n \leq 6, \ A[i][k] \text{ non è in } A[0..n-2][j],$   
 $\text{trovato sse } A[i][k] == A[n-1][j])$

$\Rightarrow A[i][k] \text{ non è in } A[0..5][j]$

$\Rightarrow \text{POST4} = (\text{trovato sse } A[i][k] \text{ è in } A[0..5][j])$

```
bool ok=true;
for (int k=0; k<8; k++) // R3
{
    bool trovato= false;
    for(int n=0; n<6 && !trovato; n++)//R4'
        {if (A[i][k]==A[n][j]) trovato=true;}
    ok=ok && trovato;
}
//POST3=(ok sse A[i] è contenuta in A[][j])
```

$R3 = (0 \leq k \leq 8, \text{ ok sse } A[i][0..k-1] \text{ in } A[][j])$

(1) e (3) sono facili, l'invarianza di R3 usa la prova del ciclo interno

R3 && k<8

invarianza

```
bool trovato= false;  
for(int n=0; n<6 && !trovato; n++)//R4'  
    {if (A[i][k]==A[n][j]) trovato=true;}
```

POST4=(trovato sse A[i][k] è in A[][j])

ok=ok && trovato;

k++;

R3=(0<=k<=8, ok sse A[i][0..k-1] in A[][j])

anche qui stiamo facendo cose inutili: se ok=false  
inutile continuare il ciclo

```

bool ok=true;
for (int k=0; k<8 && ok; k++) // R3'
{
    bool trovato= false;
    for(int n=0; n<6 && !trovato; n++)//R4'
        {if (A[i][k]==A[n][j]) trovato=true;}
    ok=ok && trovato;
}
//POST3=(ok sse A[i] è contenuta in A[][j])

```

$R3' = (0 \leq k \leq 8, A[i][0..k-1] \text{ sono in } A[][j], \text{ ok sse } A[i][k-1] \text{ è in } A[][j])$

(1) e (2) come prima, ma (3) va cambiata perchè la condizione di permanenza è doppia



$R3' = (0 \leq k \leq 8, A[i][0..k-1] \text{ sono in } A[][j], \text{ ok sse } A[i][k-1] \text{ è in } A[][j])$

$R3' \ \&\& \ !(k < 8 \ \&\& \text{ ok}) = R3' \ \&\& \ (k = 8 \ || \ !\text{ok})$

i)  $\text{ok} = \text{false} \ \&\& \ (0 \leq k \leq 8)$

ii)  $\text{ok} = \text{true} \ \&\& \ k = 8$

da (i) e  $R3' \Rightarrow A[i][k-1] \text{ non è in } A[][j] \Rightarrow \text{POST3}$

da (ii) e  $R3' \Rightarrow A[i][0..7] \text{ sono in } A[][j] \Rightarrow \text{POST3}$

si può fare con una sola var booleana

```
bool ok=true;
for (int k=0; k<8 && ok; k++) // R3'
{
    ok= false;
    for(int n=0; n<6 && !ok; n++)//R4'
        {if (A[i][k]==A[n][j]) ok=true;}

}
//POST3 garantisce che la prox istruz. è corretta
B[i][j]=ok;
```