



**POLITECNICO
DI MILANO**

INFORMATICA

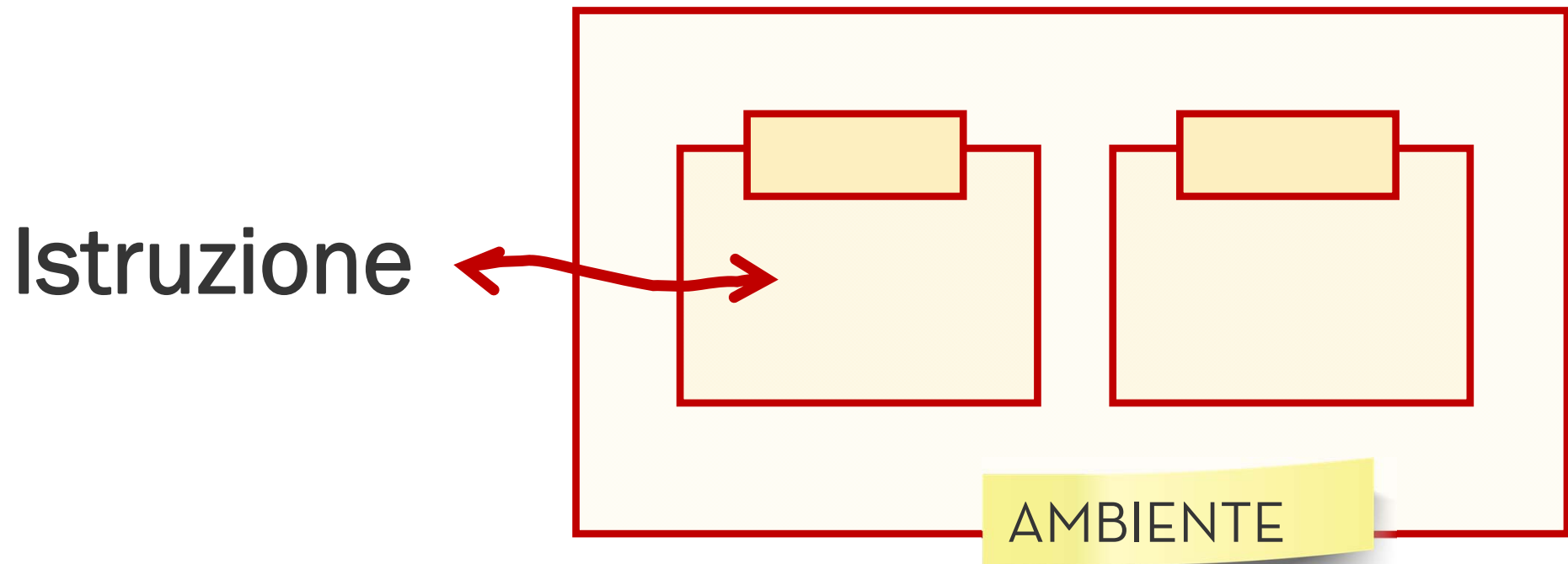
Lo scambio di
informazioni fra
funzioni tramite
ambiente globale

DEFINIZIONE E CHIAMATA

```
void elevaAPotenza()  
{ //versione con esponente positivo  
    potenza = 1;  
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)  
        potenza *= base;  
}  
  
void main()  
{  
    //presenta le funzionalità del programma  
    cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl  
        << "Se x, y, z sono interi positivi e n intero > 2" << endl  
        << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"  
        << endl;  
  
    //leggi i dati e verifica che rispondano alle specifiche  
    cout << "Inserisci x,y,z n, separati da almeno uno spazio: "  
        << endl;  
    cin >> x >> y >> z >> n;  
    if (x <= 0 || y <= 0 || z <= 0 || n < 3) return;  
    //calcola x elevati a n, con risultato in xAllaN  
    base = x; esponente = n;  
    elevaAPotenza();  
    xAllaN = potenza;
```

SEMANTICA

Modello ad ambienti



ambiente globale

```
* Si fa uso di una funzione  
* Mancano le dichiarazioni  
*/
```

```
#include <iostream.h>
```

```
void elevaAPotenza()  
{ //versione con esponen  
    potenza = 1;  
    for (prodMancanti = esp  
        potenza *= base;  
}
```

```
void main()  
{  
    //presenta le funzionali  
    cout << "Semplice verif  
        << "Se x, y, z sono
```

ambiente globale

cin

cout

```
* Si fa uso di una funzione  
* Mancano le dichiarazioni  
*/
```

```
#include <iostream.h>
```

```
void elevaAPotenza()  
{ //versione con esponen  
  potenza = 1;  
  for (prodMancanti = esp  
      potenza *= base;  
}
```

```
void main()  
{
```

```
//presenta le funzionali  
  cout << "Semplice verif  
    << "Se x, y, z sono
```

ambiente globale

cin

cout

```
* Si fa uso di una funzione  
* Mancano le dichiarazioni  
*/
```

```
#include <iostream.h>
```

```
void elevaAPotenza()
```

```
{ //versione con esponen  
    potenza = 1;  
    for (prodMancanti = esp  
        potenza *= base;  
}
```

```
void main()
```

```
{  
    //presenta le funzionali  
    cout << "Semplice verif  
        << "Se x, y, z sono
```

ambiente globale

cin

cout

```
* Si fa uso di una funzione  
* Mancano le dichiarazioni  
*/
```

```
#include <iostream.h>
```

```
void elevaAPotenza()
```

```
{ //versione con esponen
```

```
    potenza = 1;
```

```
    for (prodMancanti = esp
```

```
        potenza *= base;
```

```
}
```

```
void main()
```

```
{
```

```
    //presenta le funzionali
```

```
    cout << "Semplice verif
```

```
    << "Se x, y, z sono
```

```
// Tentativo senza speranza di dimostrare la falsità dell'ultimo
// teorema di Fermat

#include <iostream.h>

void main()
{ int x, y, z, n,                // valori su cui operare
  xAllaN, yAllaN, zAllaN        // contiene x,y,z, elevati a n
  int base, esponente, potenza, // variabili per utilizzo codice
  prodMancanti;                // già esistente

//presenta le funzionalità del programma
  cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl
        << "Se x, y, z sono interi positivi e n intero > 2" << endl
        << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"
        << endl;

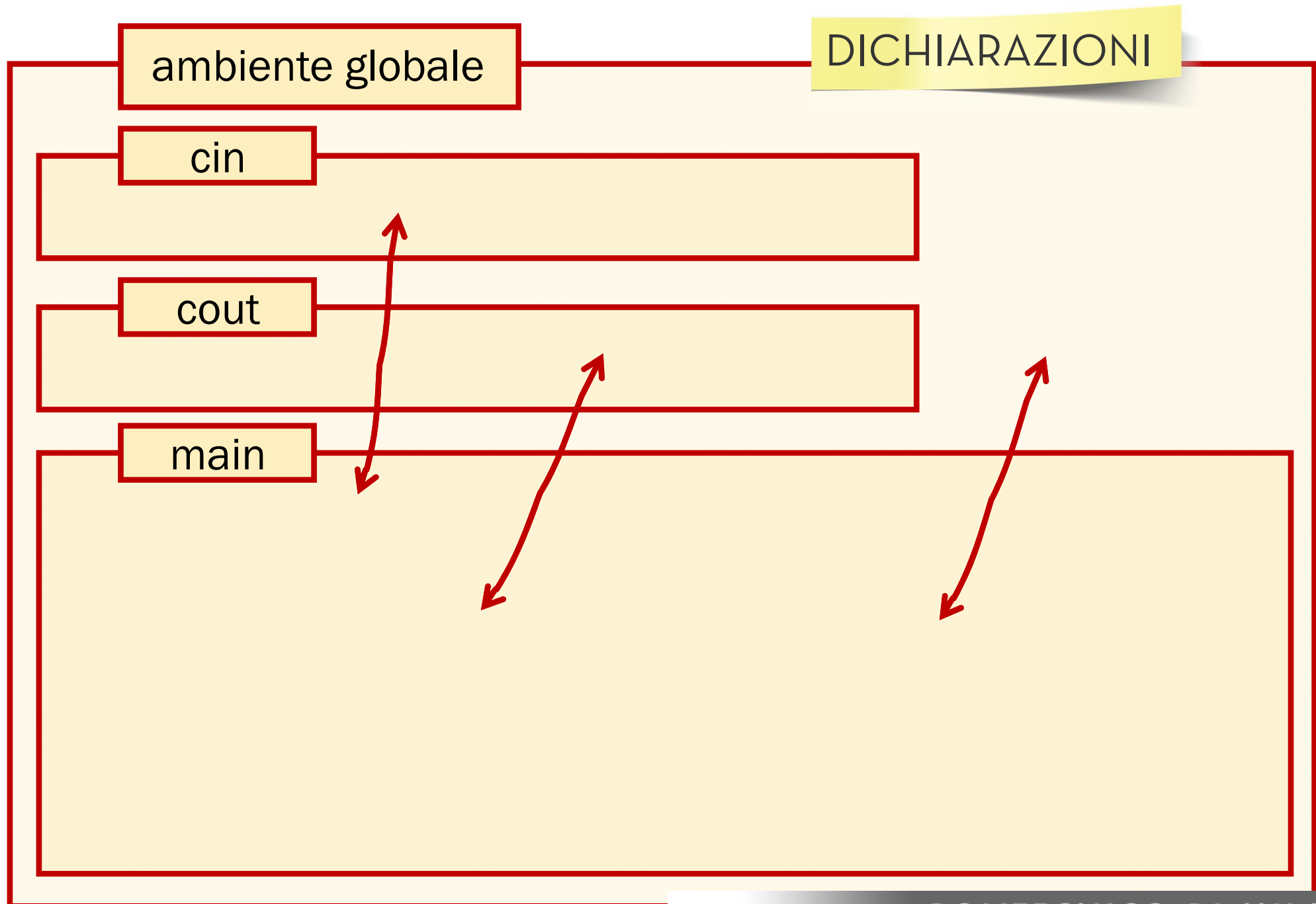
//leggi i dati e verifica che rispondano alle specifiche
  cout << "Inserisci x,y,z n, separati da almeno uno spazio: "
        << endl;
  cin >> x >> y >> z >> n;
  if (x <= 0 || y <= 0 || z <= 0 || n < 3) return;

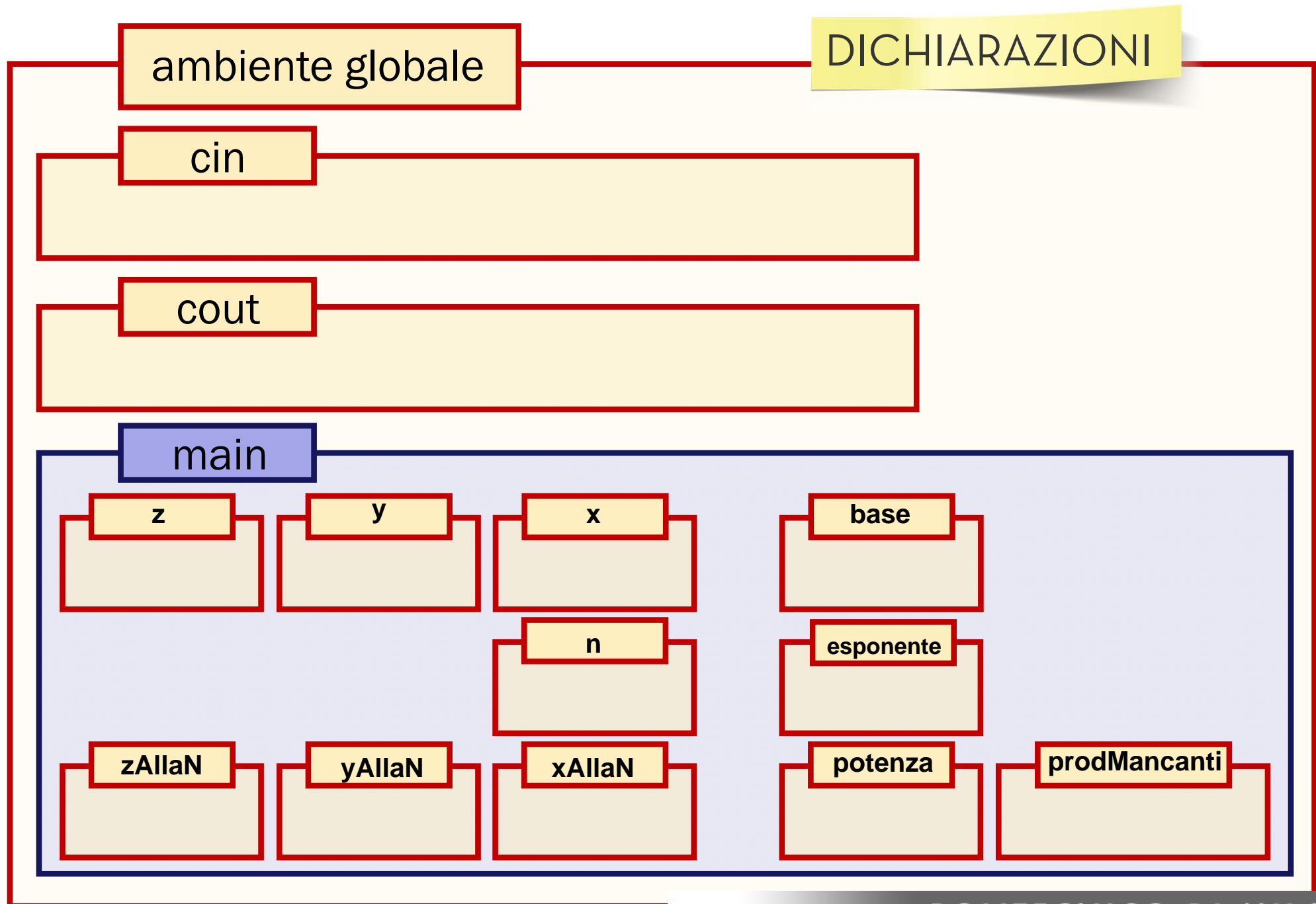
//calcola x elevato a n, con risultato in xAllaN
  base = x; esponente = n;
  //calcola base elevato a esponente, con risultato in potenza
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
  xAllaN = potenza;

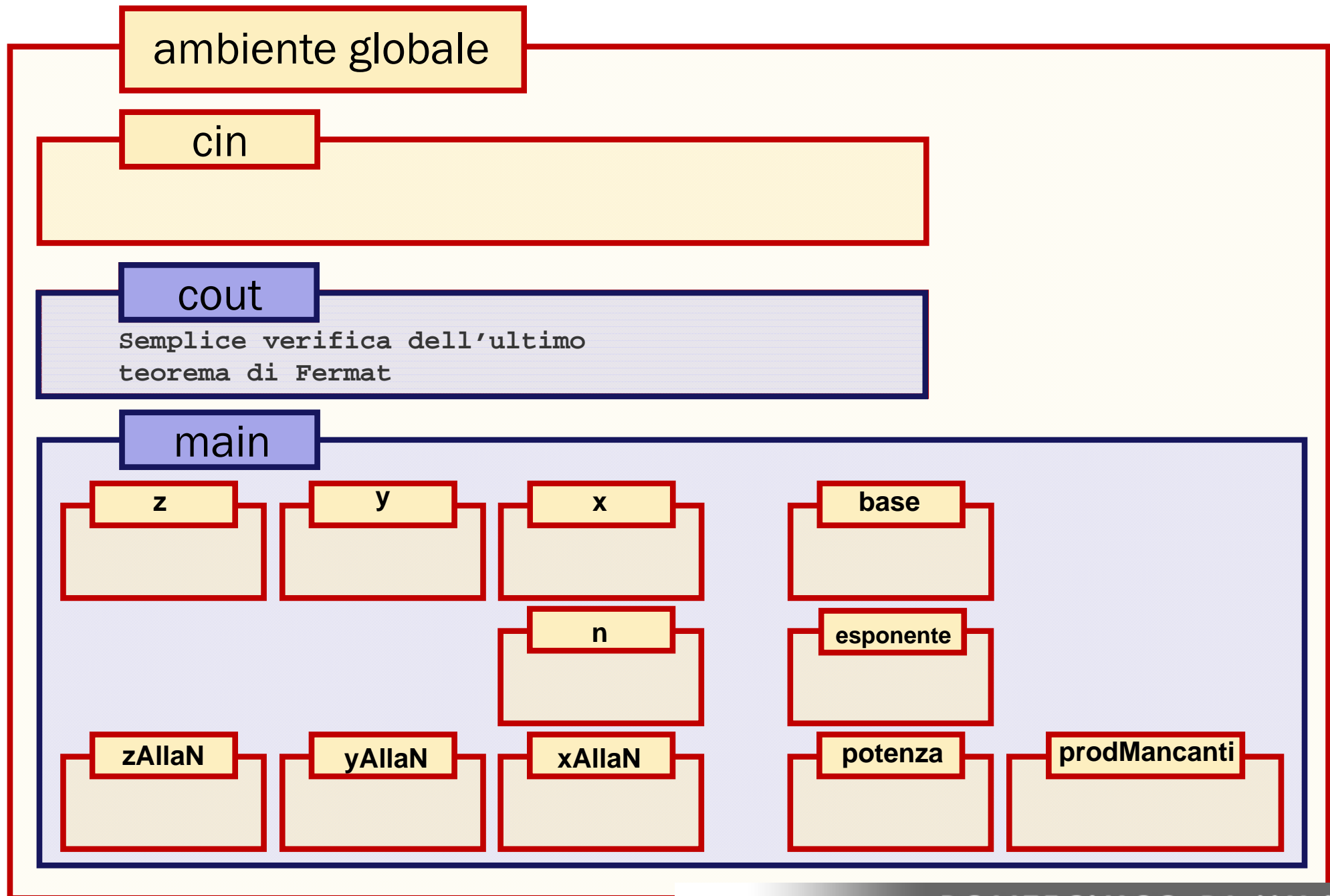
//calcola y elevato a n, con risultato in yAllaN
  base = y; esponente = n;
  //calcola base elevato a esponente, con risultato in potenza
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
  yAllaN = potenza;

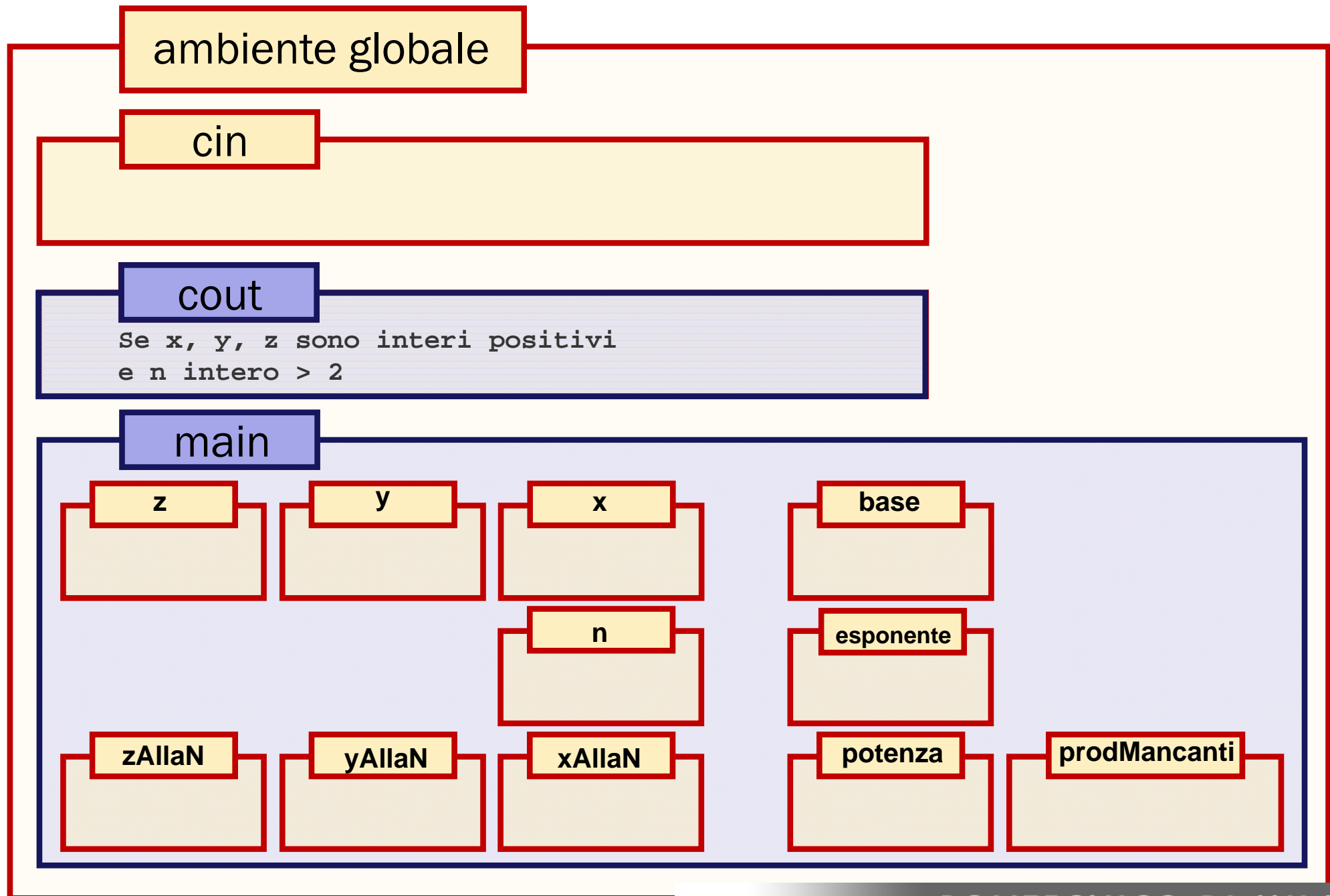
//calcola z elevato a n, con risultato in zAllaN
  base = z; esponente = n;
  //calcola base elevato a esponente, con risultato in potenza
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
  zAllaN = potenza;

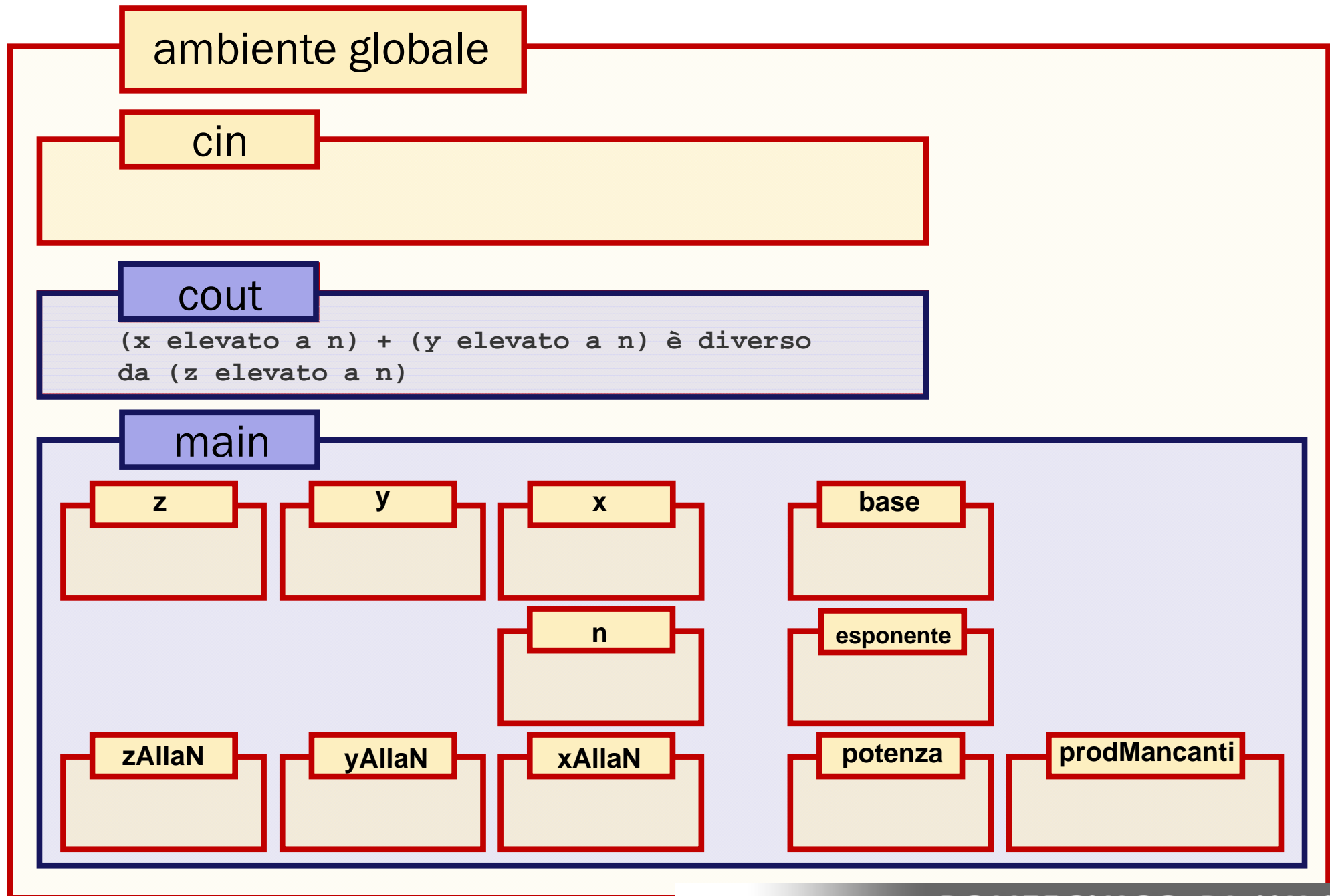
  if (xAllaN + yAllaN == zAllaN)
    cout << "Fermat's Last Theorem is false!" << endl;
  else
    cout << "Fermat's Last Theorem is true!" << endl;
}
```

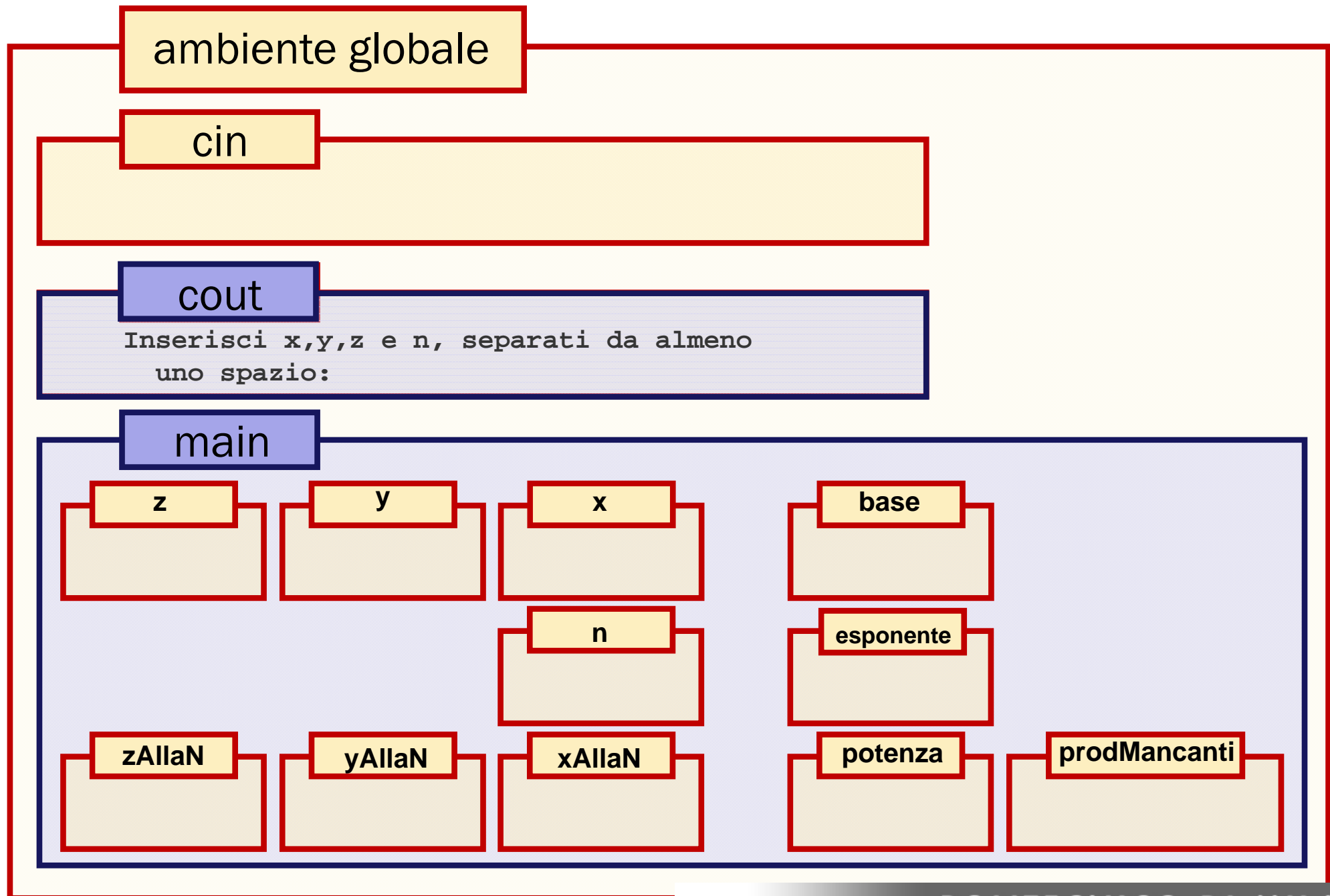



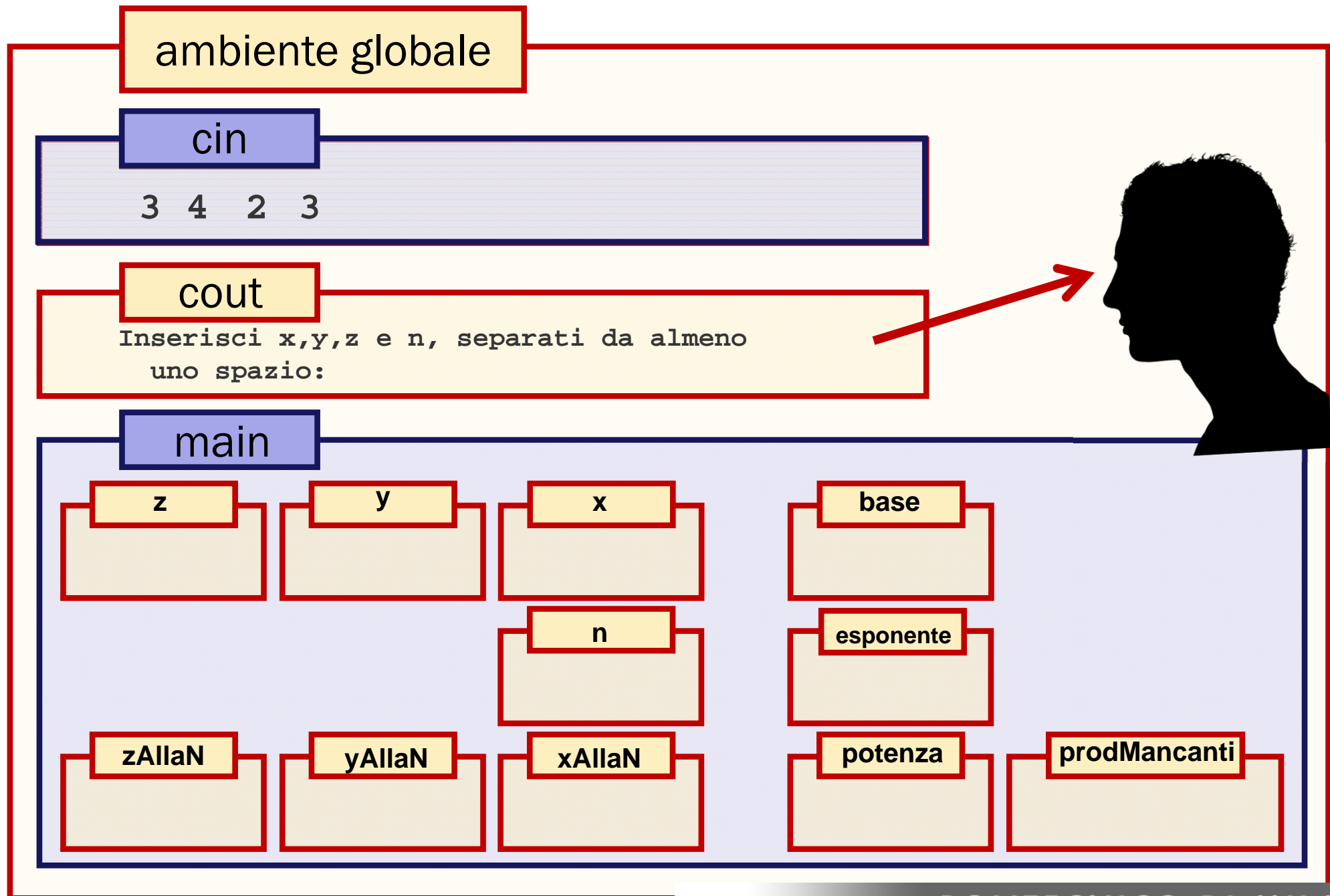


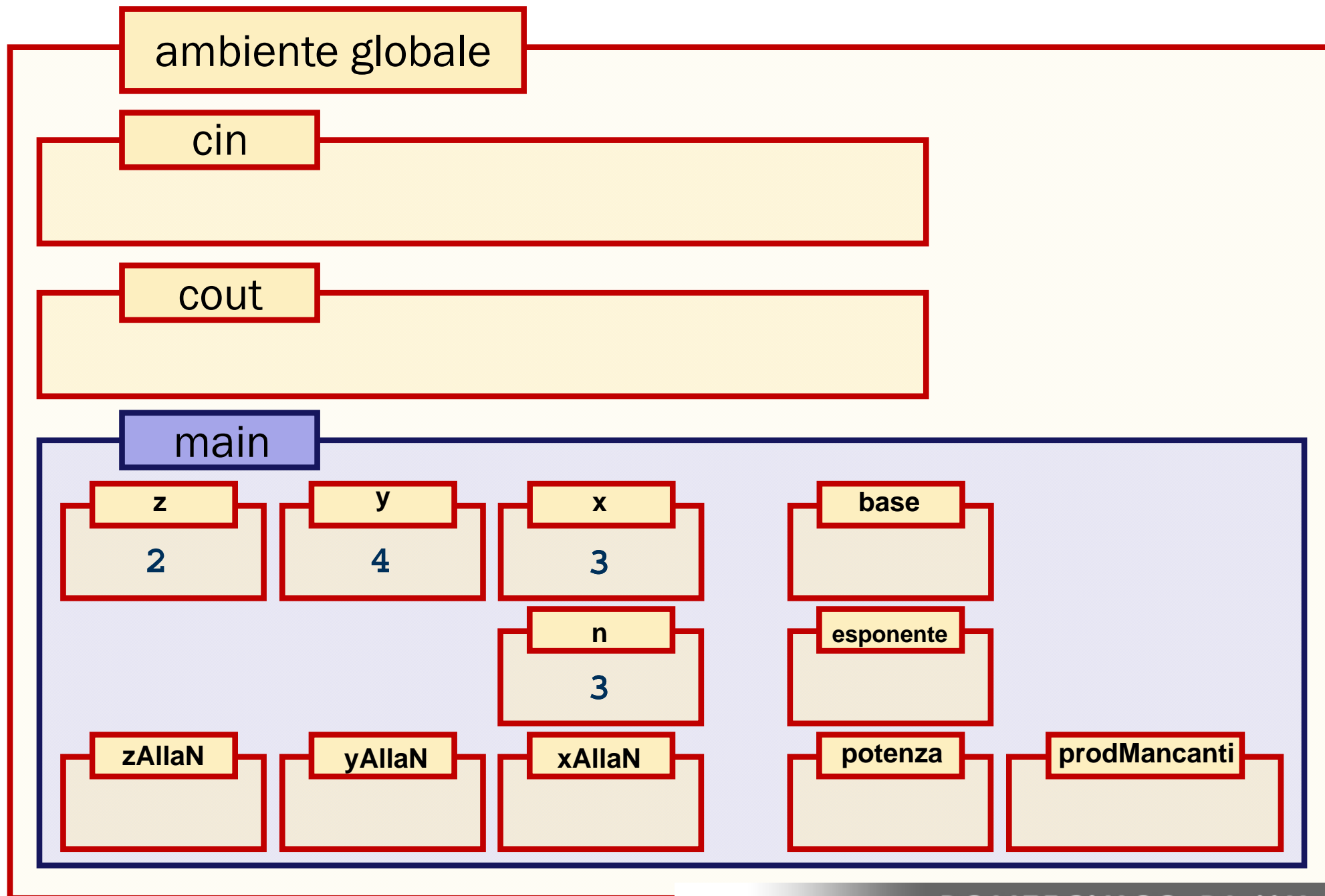


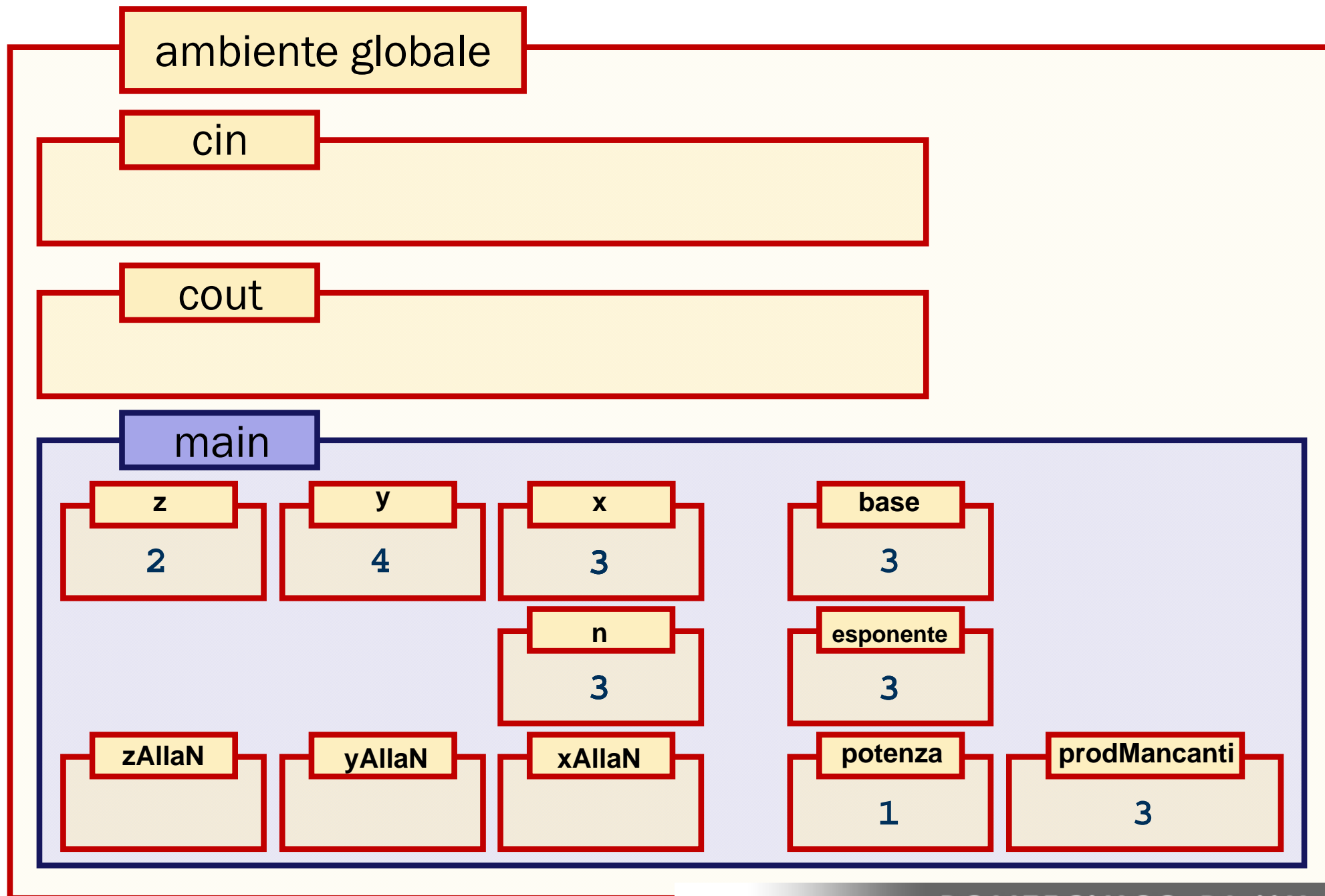


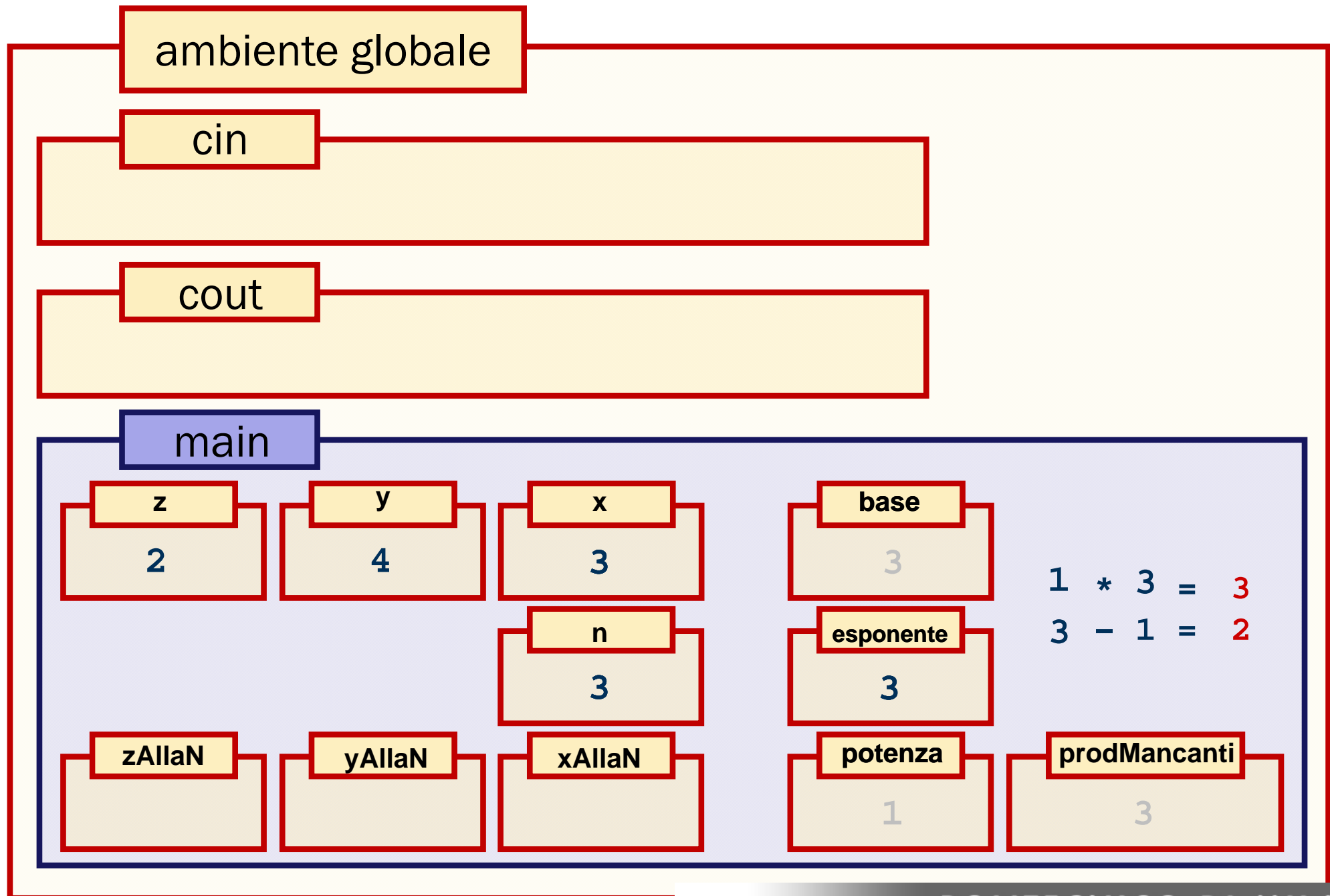


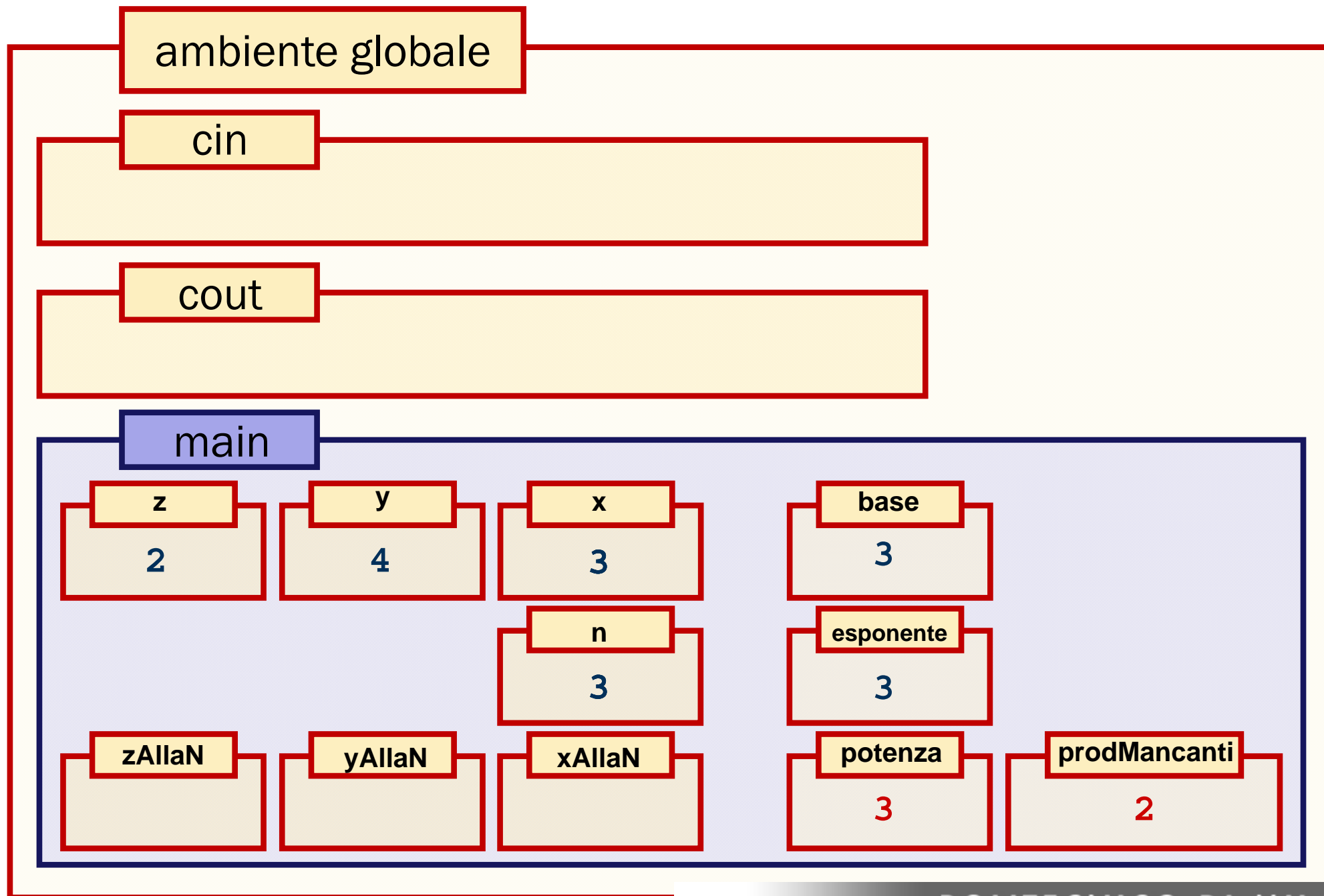


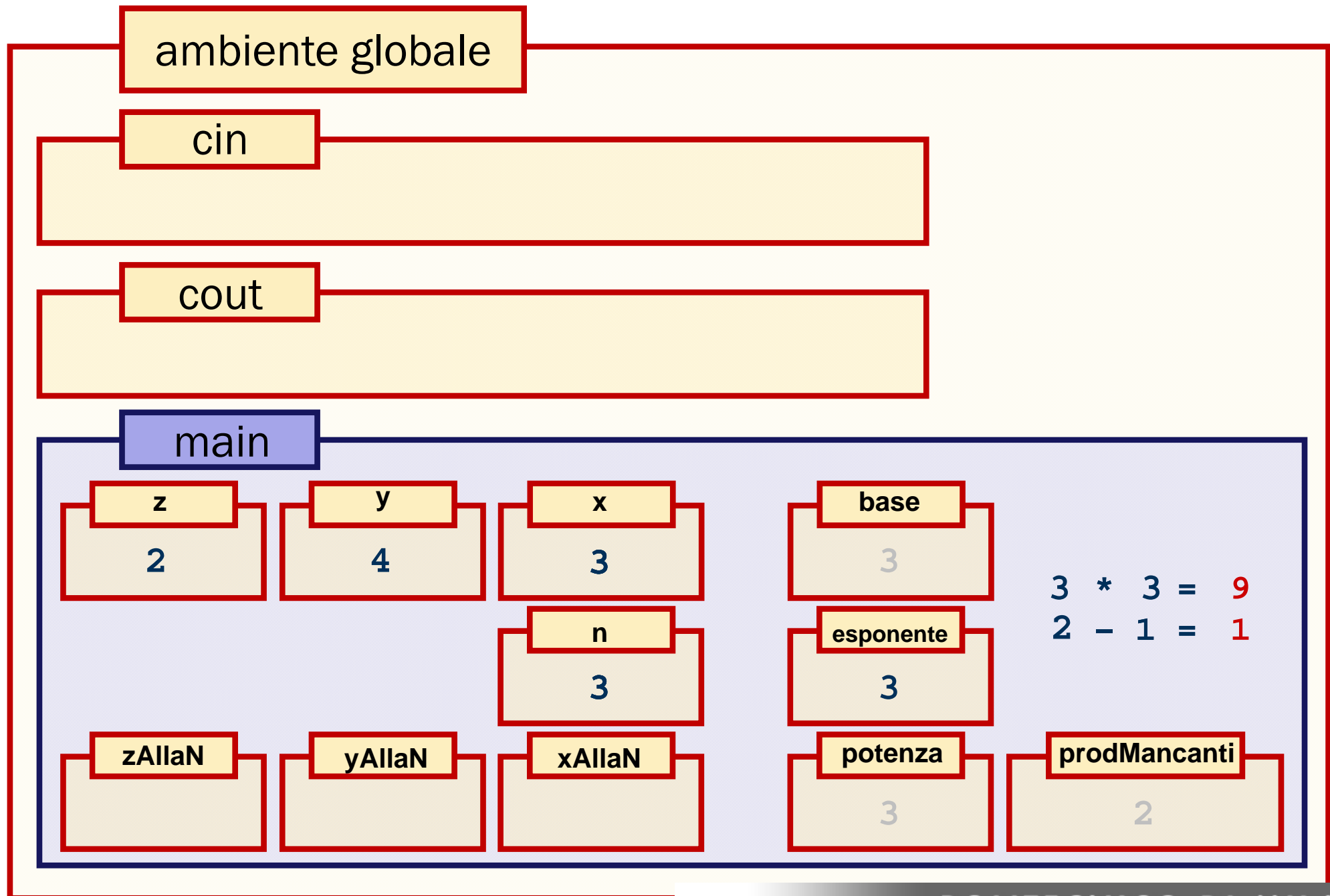


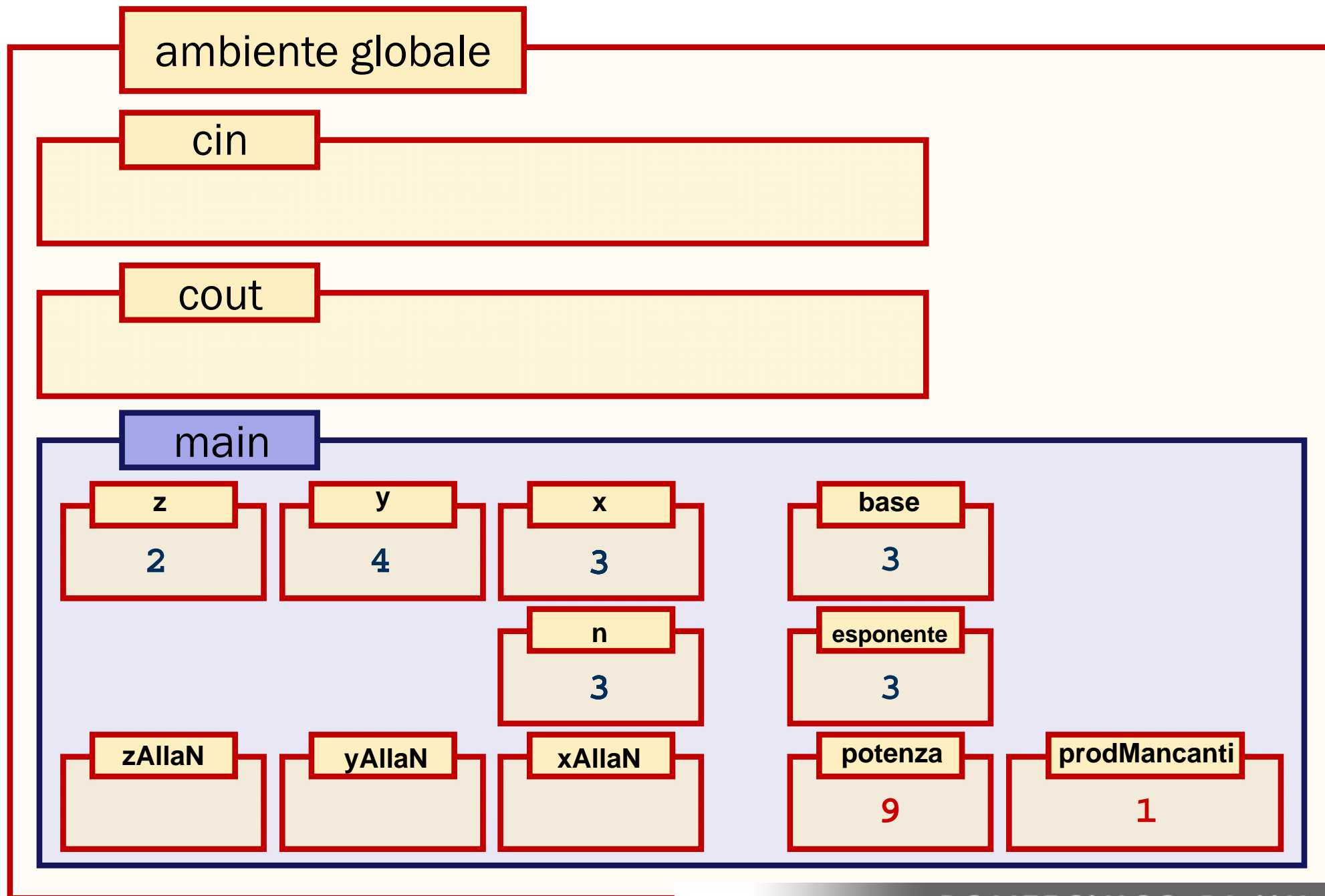


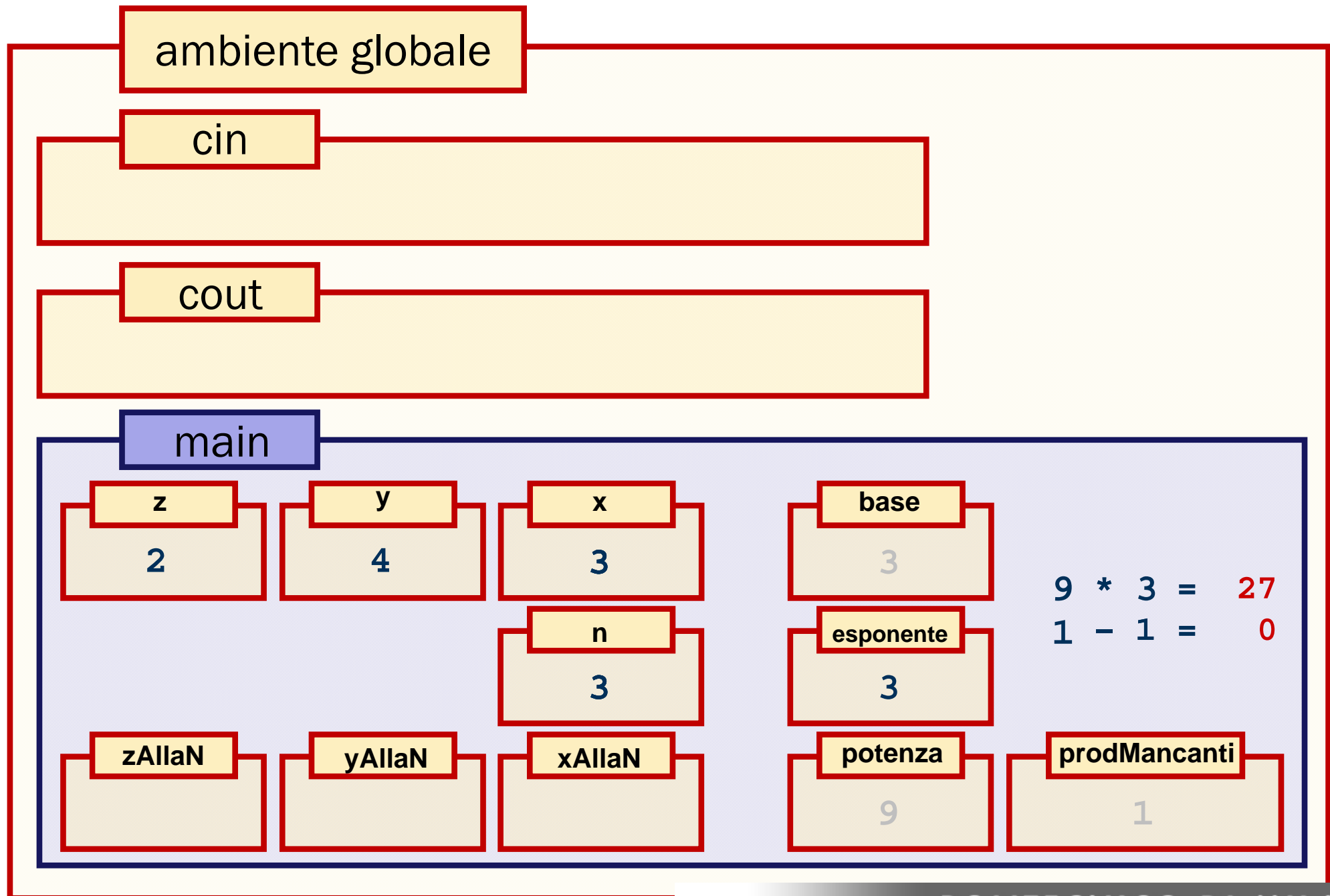


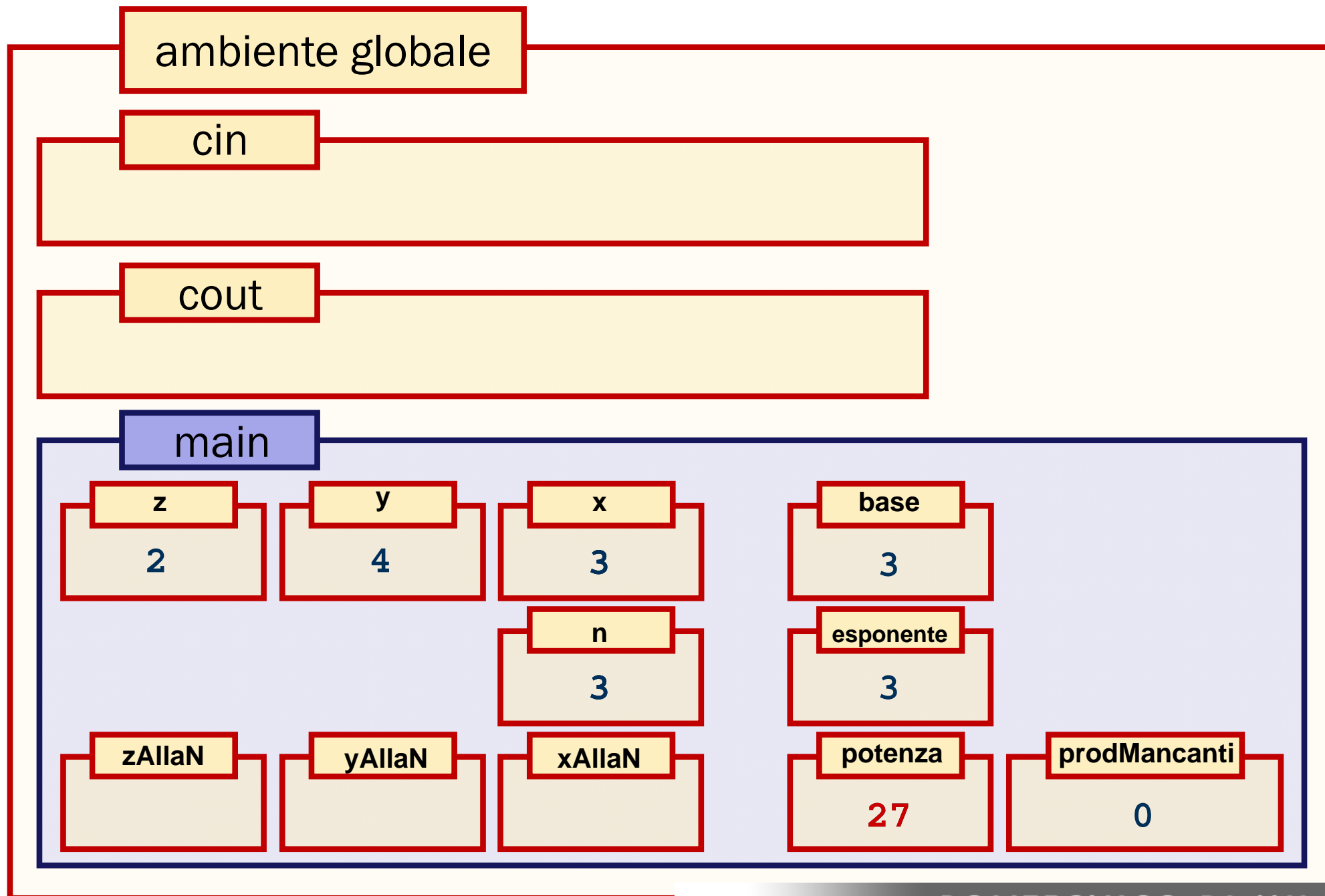


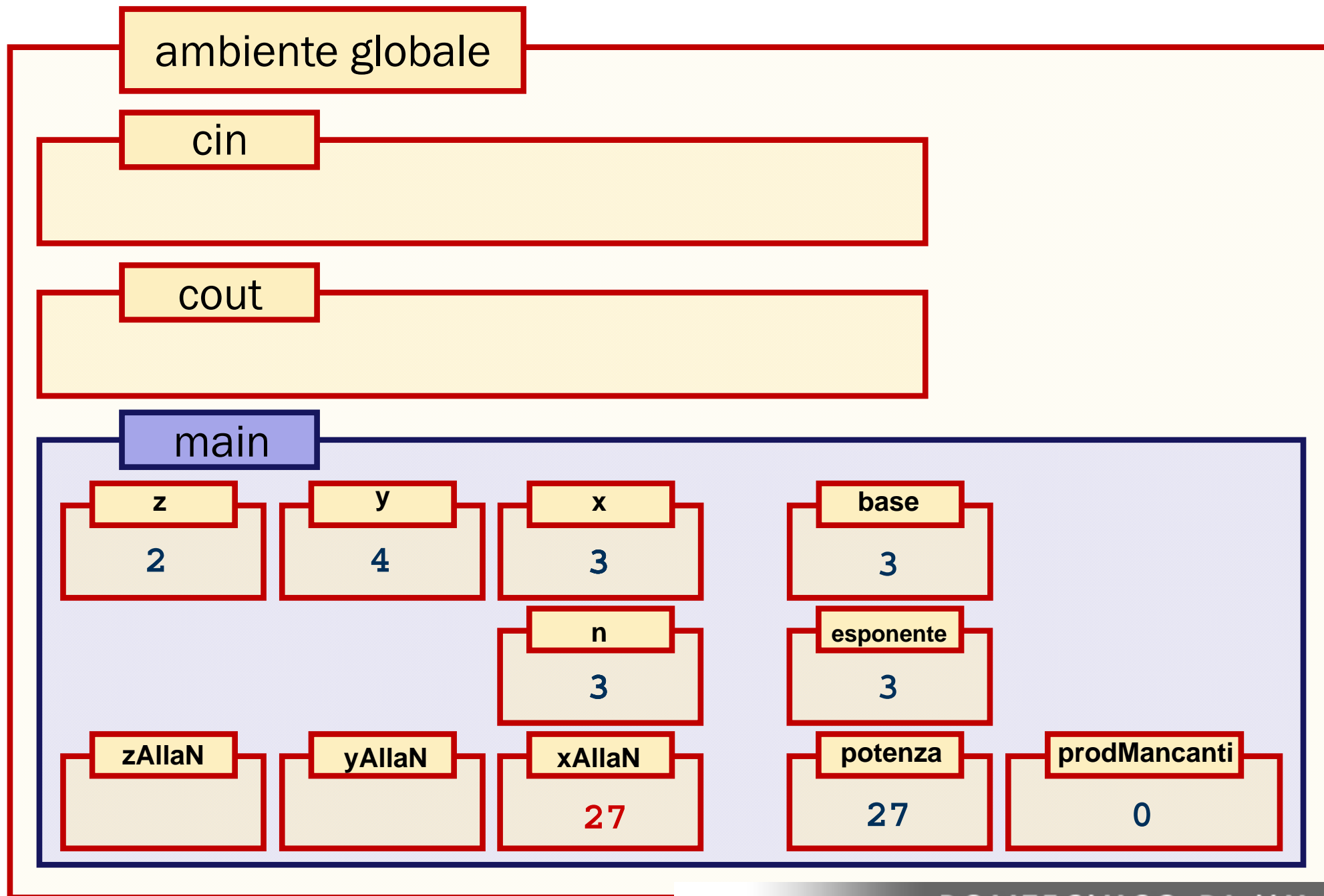













```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * Si fa uso di una funzione senza parametri
 * Mancano le dichiarazioni di variabili.
 */

#include <iostream.h>

void elevaAPotenza()
{ //versione con esponente positivo
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
        potenza *= base;
}

void main()
{
    //presenta le funzionalità del programma
    cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl
         << "Se x, y, z sono interi positivi e n intero > 2" << endl
         << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"
         << endl;

    //leggi i dati e verifica che rispondano alle specifiche
    cout << "Inserisci x,y,z n, separati da almeno uno spazio: "
         << endl;
    cin >> x >> y >> z >> n;
    if (x <= 0 || y <= 0 || z <= 0 || n < 3) return;

    //calcola x elevato a n, con risultato in xAllaN
    base = x; esponente = n;
    elevaAPotenza();
    xAllaN = potenza;
}
```

AMBIENTE LOCALE O
GLOBALE

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * Si fa uso di una funzione senza parametri
 * Mancano le dichiarazioni di variabili.
 */

#include <iostream.h>

void elevaAPotenza()
{ //versione con esponente positivo
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
        potenza *= base;
}

void main()
{
    //presenta le funzionalità del programma
    cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl
         << "Se x, y, z sono interi positivi e n intero > 2" << endl
         << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"
         << endl;

    //leggi i dati e verifica che rispondano alle specifiche
    cout << "Inserisci x,y,z n, separati da almeno uno spazio: "
         << endl;
    cin >> x >> y >> z >> n;
    if (x <= 0 || y <= 0 || z <= 0 || n < 3) return;

    //calcola x elevati a n, con risultato in xAllaN
    base = x; esponente = n;
    elevaAPotenza();
    xAllaN = potenza;
```

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo  
 * teorema di Fermat  
 * Si fa uso di una funzione senza parametri.  
 */
```

```
#include <iostream.h>
```

```
//variabili globali usate per la comunicazione fra main ed  
//elevaAPotenza  
int base, esponente, potenza;
```

```
void elevaAPotenza()  
{ //versione con esponente positivo  
    potenza = 1;  
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)  
        potenza *= base;  
}
```

```
void main()  
{  
    //presenta le funzionalità del programma  
    cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl  
        << "Se x, y, z sono interi positivi e n intero > 2" << endl  
        << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"  
        << endl;
```

```
// Leggi i dati e verifica che rispondano alle specifiche
```

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * Si fa uso di una funzione senza parametri.
 */

#include <iostream.h>

//variabili globali usate per la comunicazione fra main ed
//elevaAPotenza
int base, esponente, potenza;

void elevaAPotenza()
{ //versione con parametri
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
        potenza *= base;
}

void main()
{
    // Presenta le funzionalità del programma
    cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl
        << "Se x, y, z sono interi positivi e n intero > 2" << endl
        << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"
        << endl;

    // Leggi i dati e verifica che rispondano alle specifiche
}
```

base
esponente
potenza
elevaAPotenza
main()

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * Si fa uso di una funzione senza parametri.
 */

#include <iostream.h>

//variabili globali usate per la comunicazione fra main ed
//elevaAPotenza
int base, esponente, potenza;

void elevaAPotenza()
{ //versione con argomenti globali
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
        potenza *= base;
}

void main()
{
    // Presenta le funzionalità del programma
    cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl
        << "Se x, y, z sono interi positivi e n intero > 2" << endl
        << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"
        << endl;

    // Leggi i dati e verifica che rispondano alle specifiche
}
```

The diagram illustrates the flow of information between the `main()` function and the `elevaAPotenza()` function. Three global variables are defined: `base`, `esponente`, and `potenza`. Blue arrows indicate the flow of data: one arrow points from `base` to `elevaAPotenza()`, another from `esponente` to `elevaAPotenza()`, and a third from `potenza` back to `main()`. The entire code block is enclosed in a red border.

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * Si fa uso di una funzione senza parametri.
 */

#include <iostream.h>

//variabili globali usate per la comunicazione fra main ed
//elevaAPotenza
int base, esponente, potenza;

void elevaAPotenza()
{ //versione con parametri
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
        potenza *= base;
}

void main()
{
    // Presenta le funzionalità del programma
    cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl
        << "Se x, y, z sono interi positivi e n intero > 2" << endl
        << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"
        << endl;

    // Leggi i dati e verifica che rispondano alle specifiche
}
```

base
esponente
potenza

elevaAPotenza

main()

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * Si fa uso di una funzione senza parametri.
 */

#include <iostream.h>

//variabili globali usate per la comunicazione fra main ed
//elevaAPotenza
int base, esponente, potenza;

void elevaAPotenza()
{ //versione con esponente positivo
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
        potenza *= base;
}

void main()
{ int x, y, z, n          // valori su cui operare
  xAllaN, yAllaN, zAllaN; // contengono x,y,z elevati a n

    // Presenta le funzionalità del programma
    cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl
         << "Se x, y, z sono interi positivi e n intero > 2" << endl
         << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"
         << endl;
```

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * Si fa uso di una funzione senza parametri.
 */

#include <iostream.h>

//variabili globali usate per la comunicazione fra main ed
//elevaAPotenza
int base, esponente, potenza;

void elevaAPotenza()
{ //versione con esponente positivo
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
        potenza *= base;
}

void main()
{ int x, y, z, n          // valori su cui operare
  xAllaN, yAllaN, zAllaN; // contengono x,y,z elevati a n

    // Presenta le funzionalità del programma
    cout << "Semplice verifica dell'ultimo t
        << "Se x, y, z sono interi positivi
        << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"
        << endl;
```

VARIABILI LOCALI A MAIN()


```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * Si fa uso di una funzione senza parametri
 */
#include <iostream.h>

//variabili globali usate per la comunicazione fra main ed
//elevaAPotenza
int base, esponente, potenza;

void elevaAPotenza()
{ //versione con esponente positivo
    int prodMancanti; // variabile locale ad elevaAPotenza
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
        potenza *= base;
}

void main()
{ int x, y, z, n // valori su cui operare
  xAllaN, yAllaN, zAllaN; // contengono x,y,z elevati a n

  // Presenta le funzionalità del programma
  cout << "Semplice verifica dell'ultimo teorema di Fermat." << endl
        << "Se x, y, z sono interi positivi e n intero > 2" << endl
        << "(x elevato a n)+(y elevato a n) è diverso da (z elevato a n)"
        << endl;
```

MODELLO AD AMBIENTI

```
* Si fa uso di una funzione senza parametri
```

```
*/
```

```
#include <iostream.h>
```

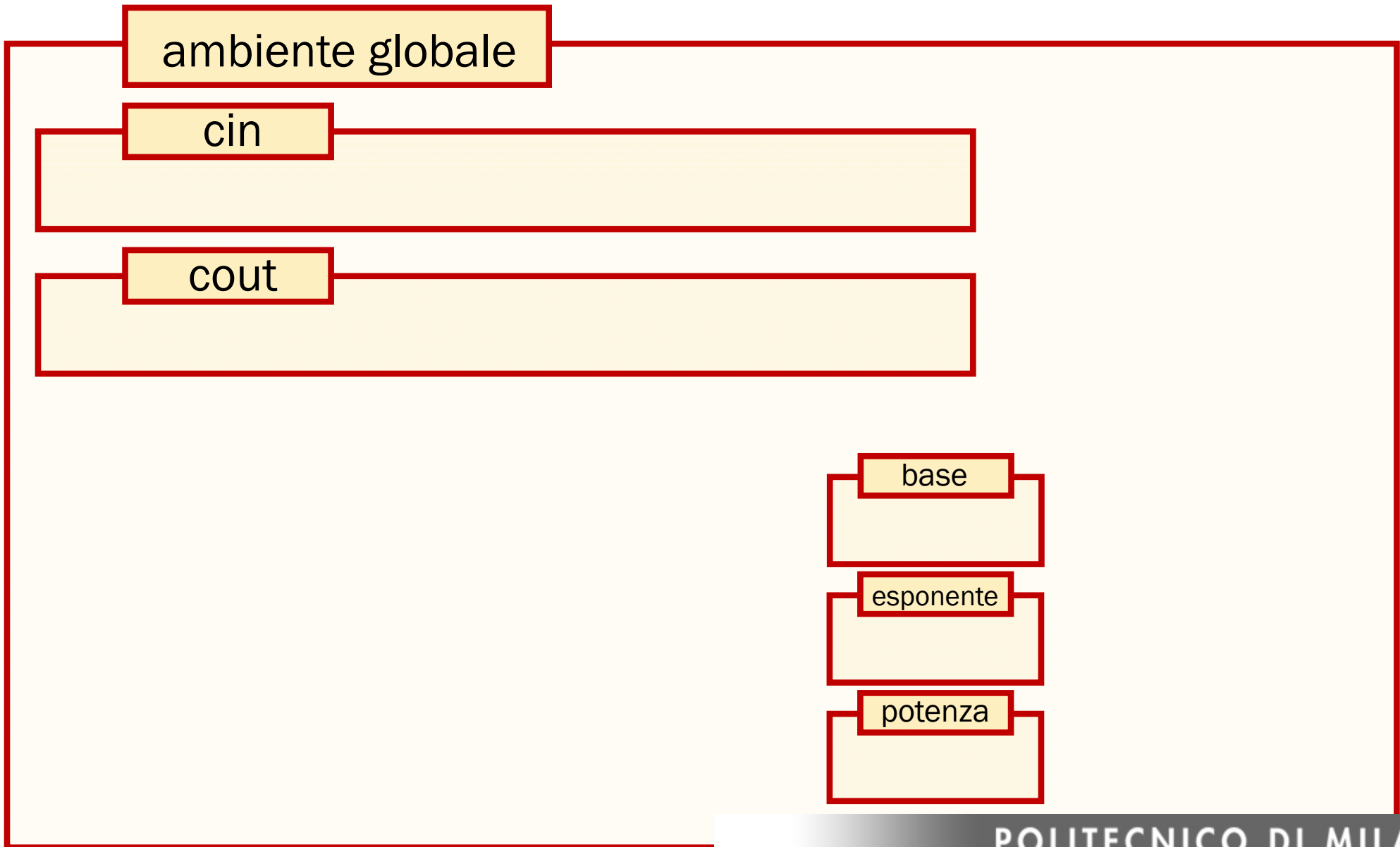
```
//variabili globali usate per la comunicazione fra main ed
```

ambiente globale

cin

cout

```
//variabili globali usate per la comunicazione fra main ed  
//elevaAPotenza  
int base, esponente, potenza;
```

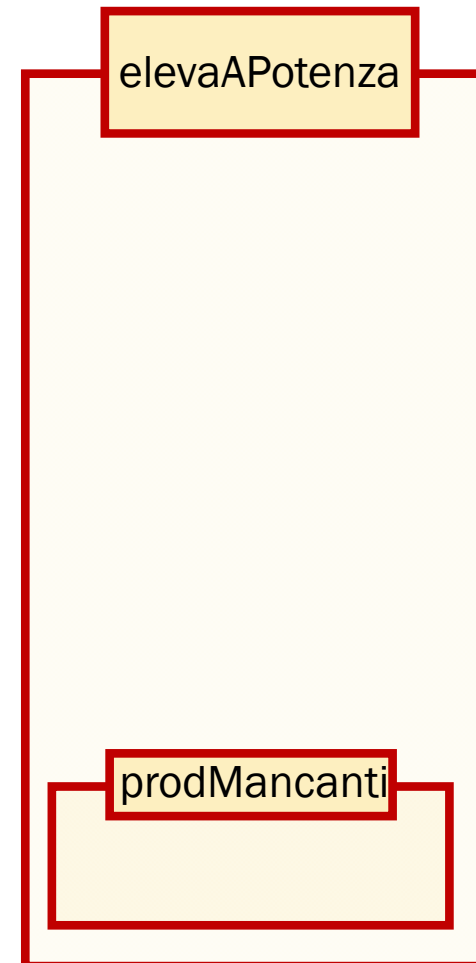


```
void elevaAPotenza()  
{ //versione con esponente positivo  
    int prodMancanti;    // variabile locale ad elevaAPotenza
```

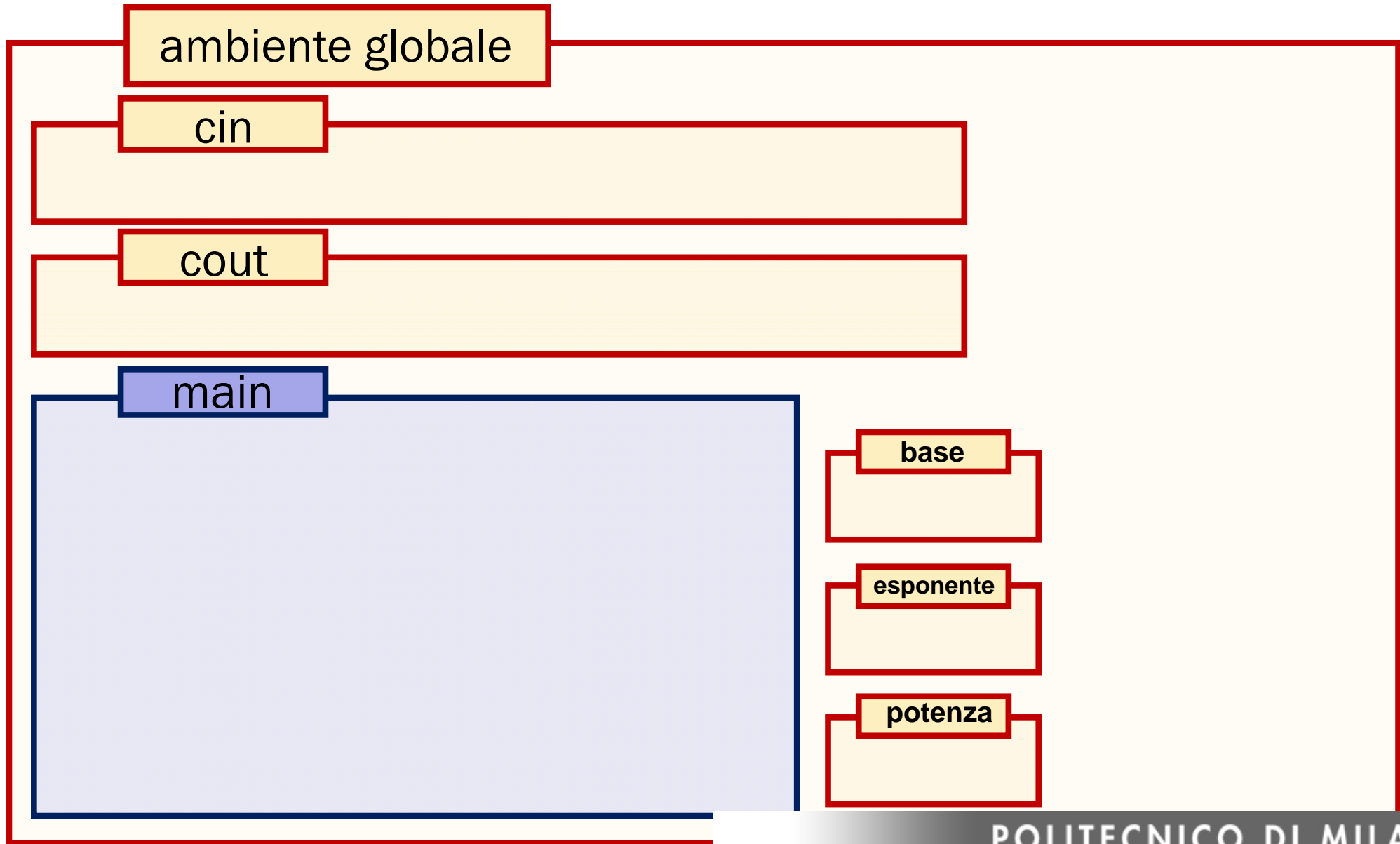


elevaAPotenza

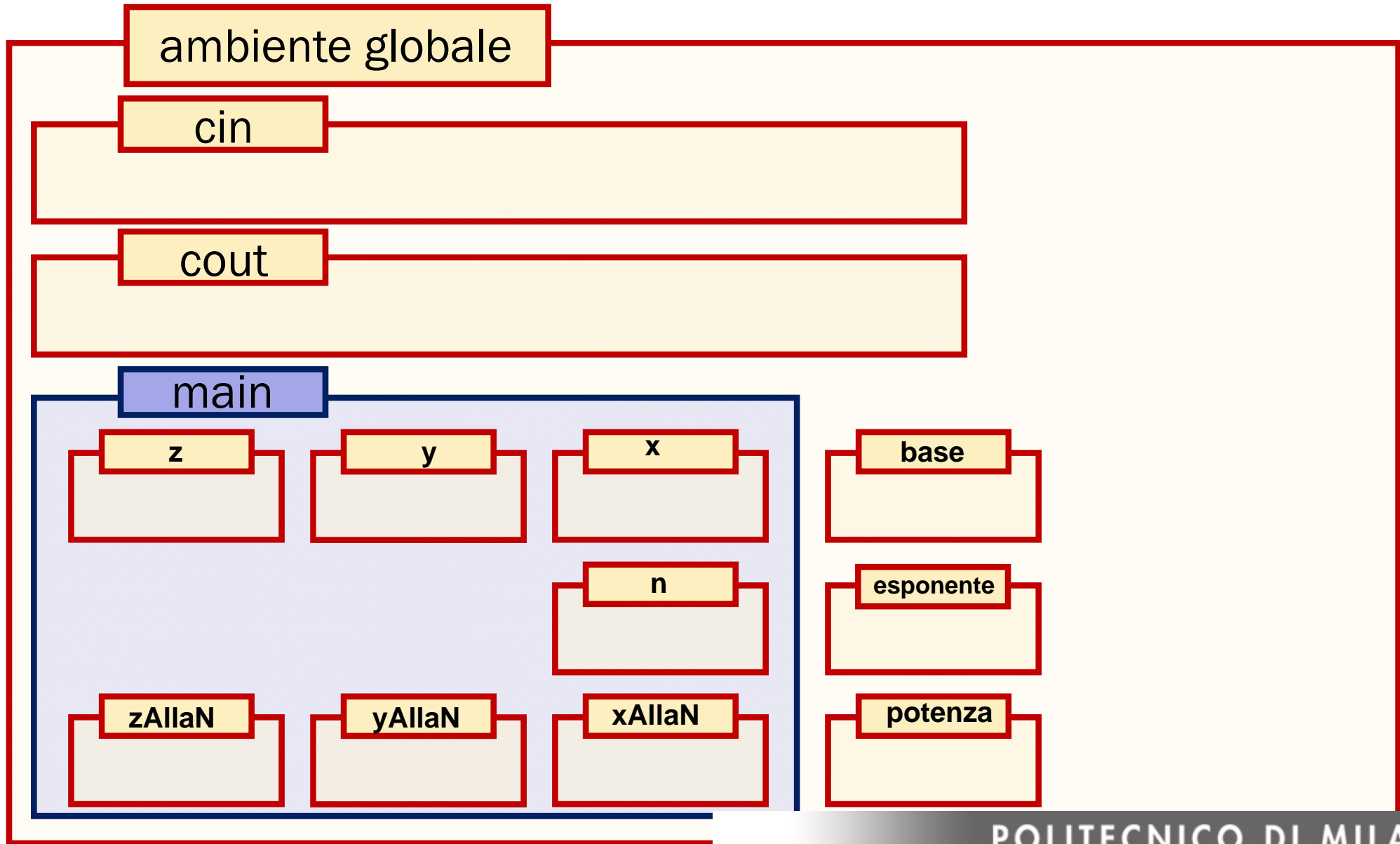
```
void elevaAPotenza()  
{ //versione con esponente positivo  
    int prodMancanti;    // variabile locale ad elevaAPotenza  
    potenza = 1;  
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```



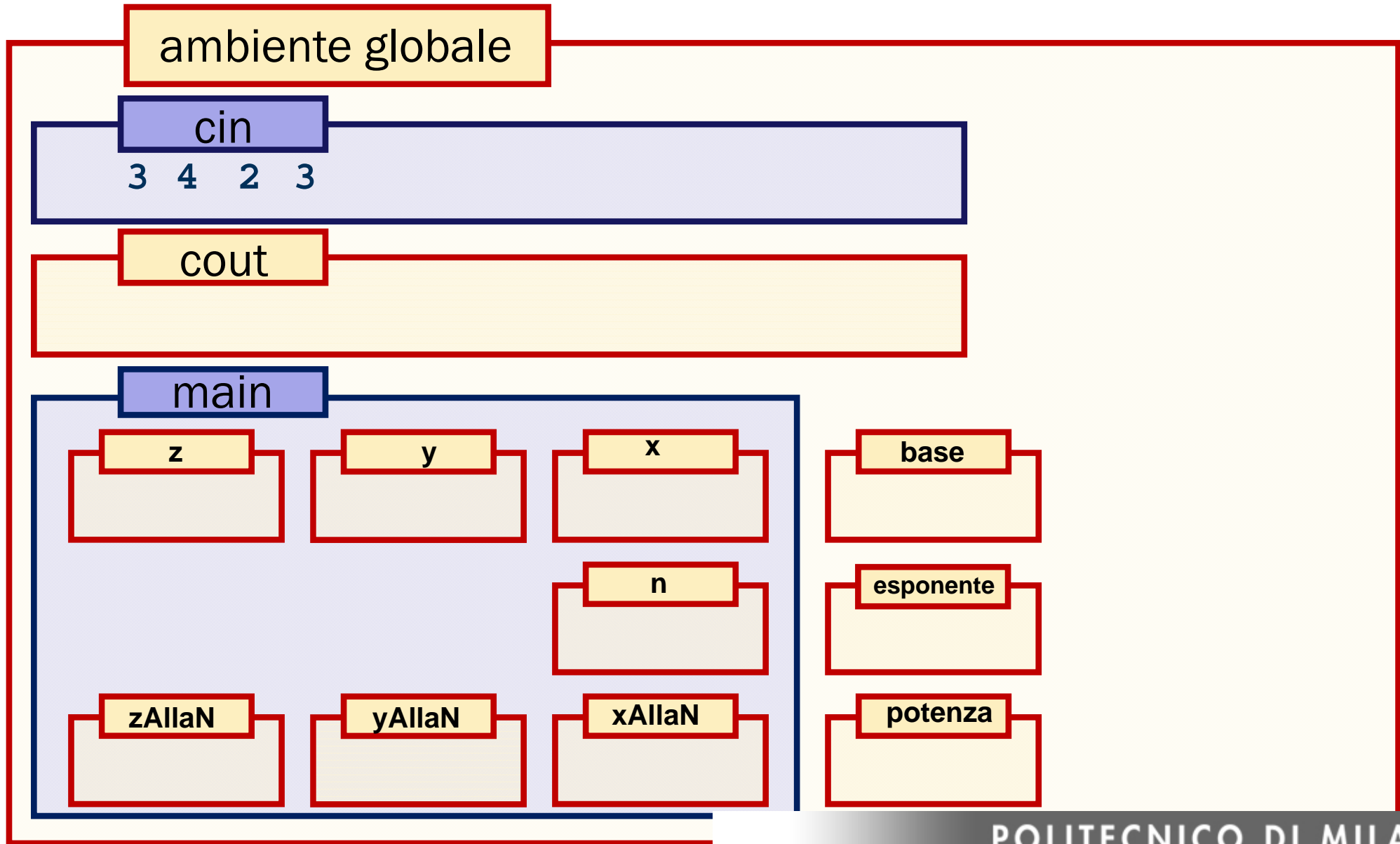
```
void main()  
{ int x, y, z, n,           // valori su cui operare  
  xAllaN, yAllaN, zAllaN;  // contengono x, y, z elevati a n
```



```
void main()  
{ int x, y, z, n,           // valori su cui operare  
  xAllaN, yAllaN, zAllaN;   // contengono x, y, z elevati a n
```



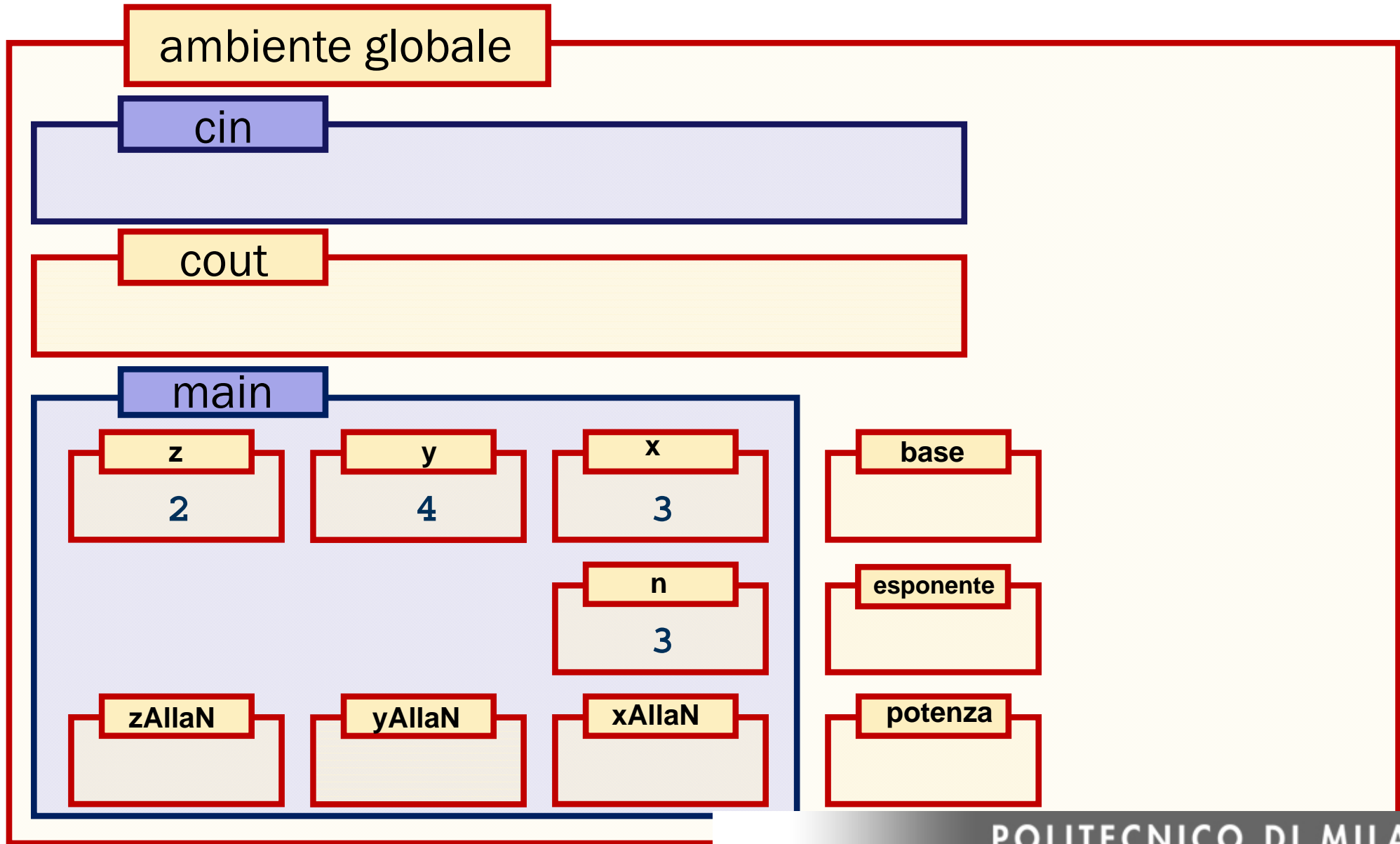

```
<< endl;  
cin >> x >> y >> z >> n;  
if (x <= 0 || y <= 0 || z <= 0 || n < 3) return;  
//calcola x elevati a n, con risultato in xAllaN
```



```
cin >> x >> y >> z >> n;
```

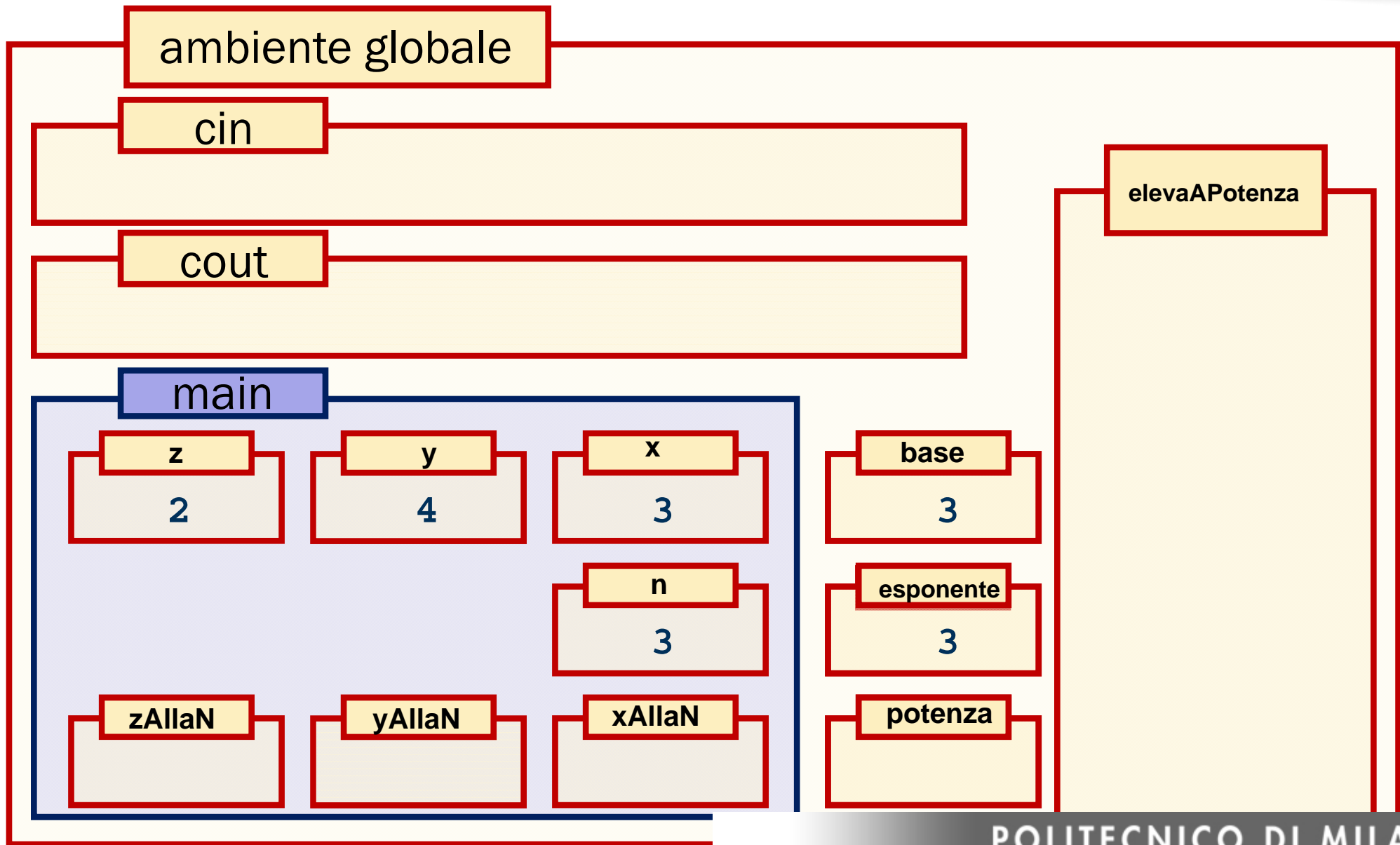
```
if (x <= 0 || y <= 0 || z <= 0 || n < 3) return;
```

```
//calcola x elevati a n, con risultato in xAllaN
```



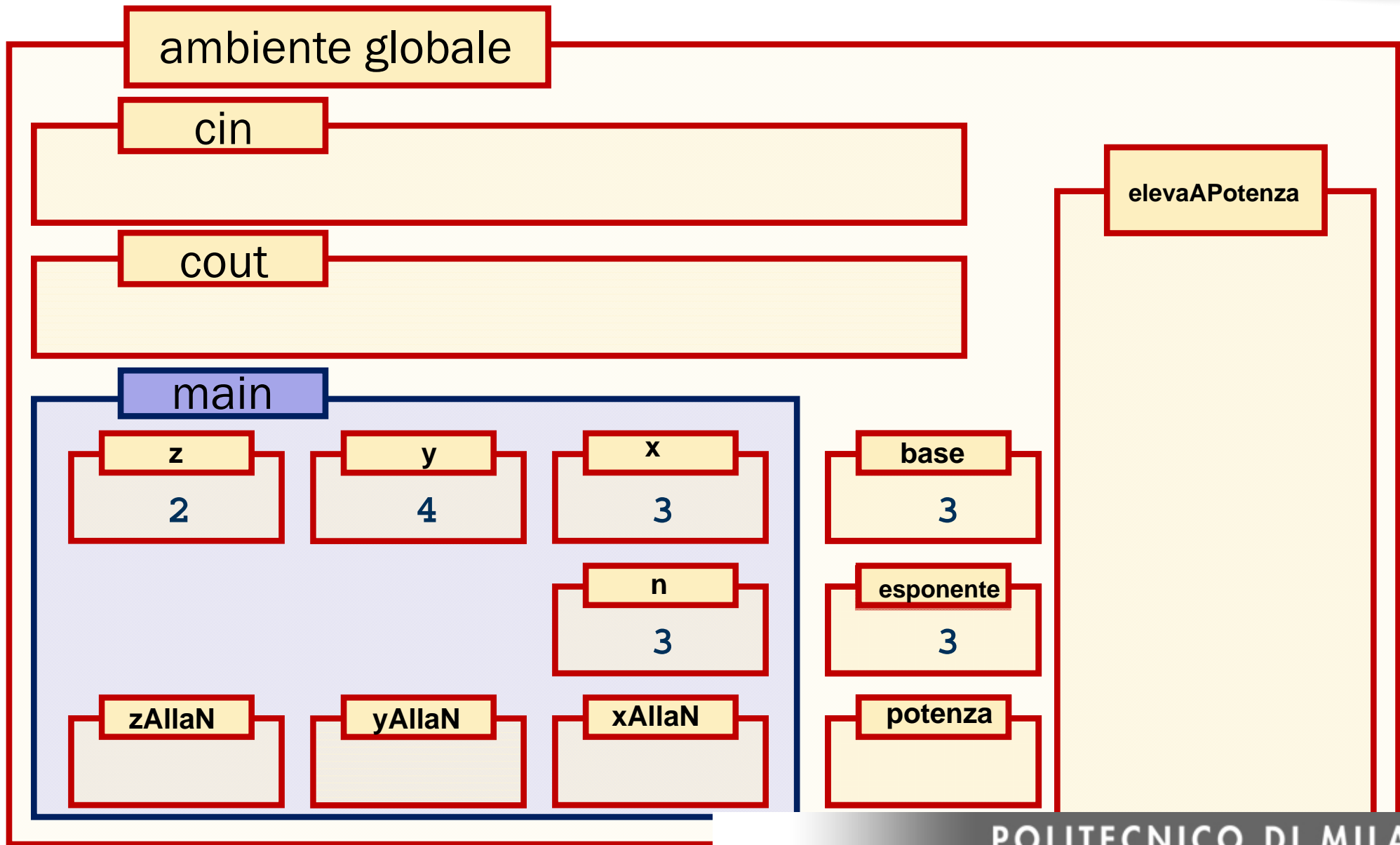
```
//calcola x elevato a n, con risultato in xAllaN  
base = x; esponente = n;  
elevaAPotenza();  
xAllaN = potenza;
```

COPIA DELL'AMBIENTE LOCALE



```
//calcola x elevato a n, con risultato in xAllaN  
base = x; esponente = n;  
elevaAPotenza();  
xAllaN = potenza;
```

MODELLO AD AMBIENTI



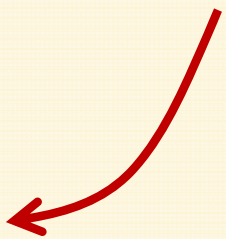
funzione chiamante

```
void main()  
{  
    . . .  
    elevaAPotenza( )  
    . . .  
}
```

funzione chiamata

```
void elevaAPotenza( )
```

*ritorno dalla
funzione chiamata*



```
//calcola x elevato a n, con risultato in xAllaN  
base = x; esponente = n;  
elevaAPotenza();  
xAllaN = potenza;
```

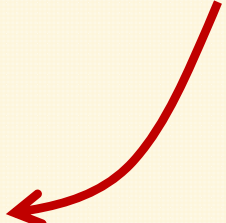
funzione chiamante

```
void main()  
{  
    . . .  
    elevaAPotenza()  
    xAllaN = potenza  
    . . .  
}
```

funzione chiamata

```
void elevaAPotenza()
```

*ritorno dalla
funzione chiamata*

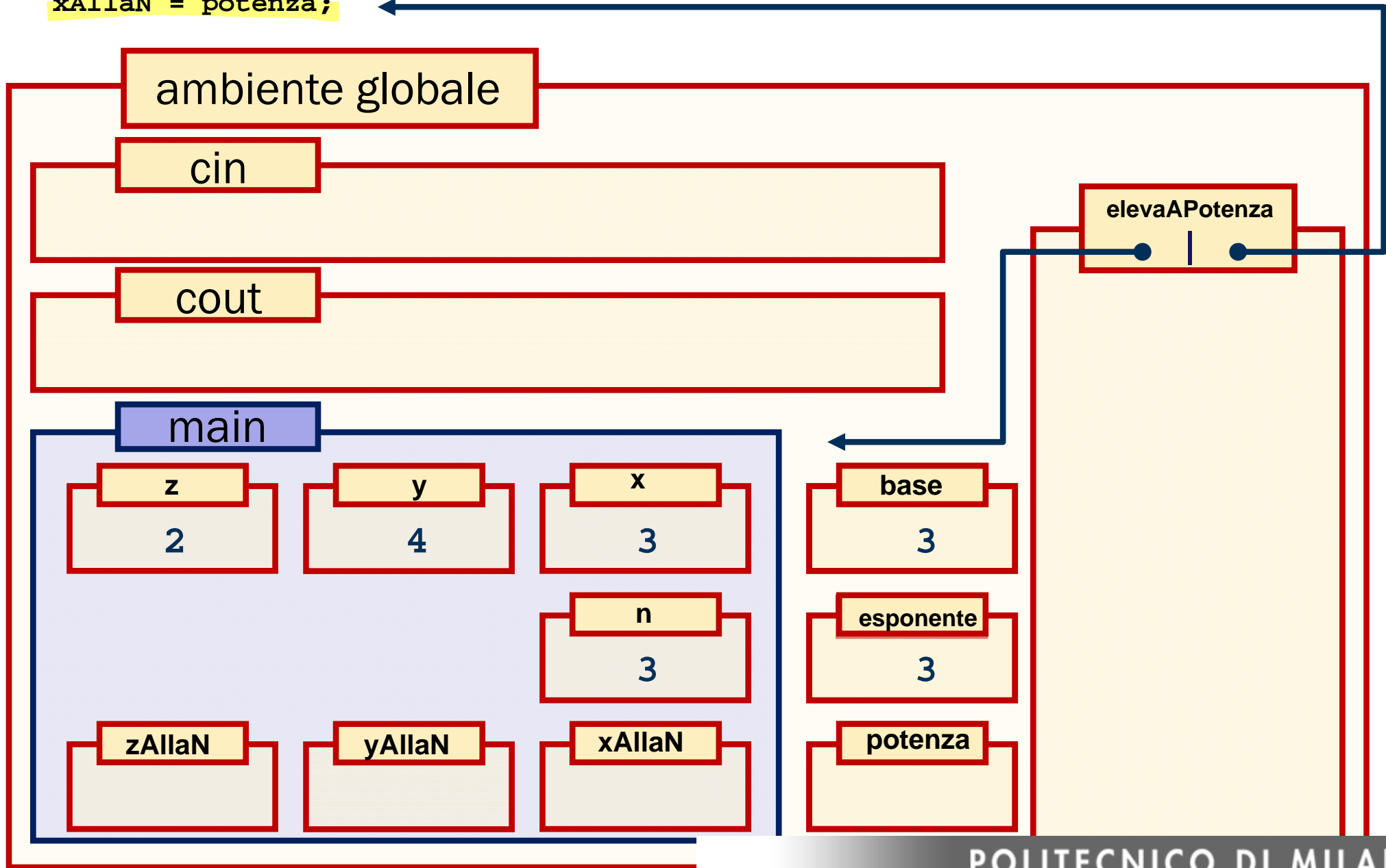


```
//calcola x elevato a n, con risultato in xAllaN
```

```
base = x; esponente = n;
```

```
elevaAPotenza();
```

```
xAllaN = potenza;
```

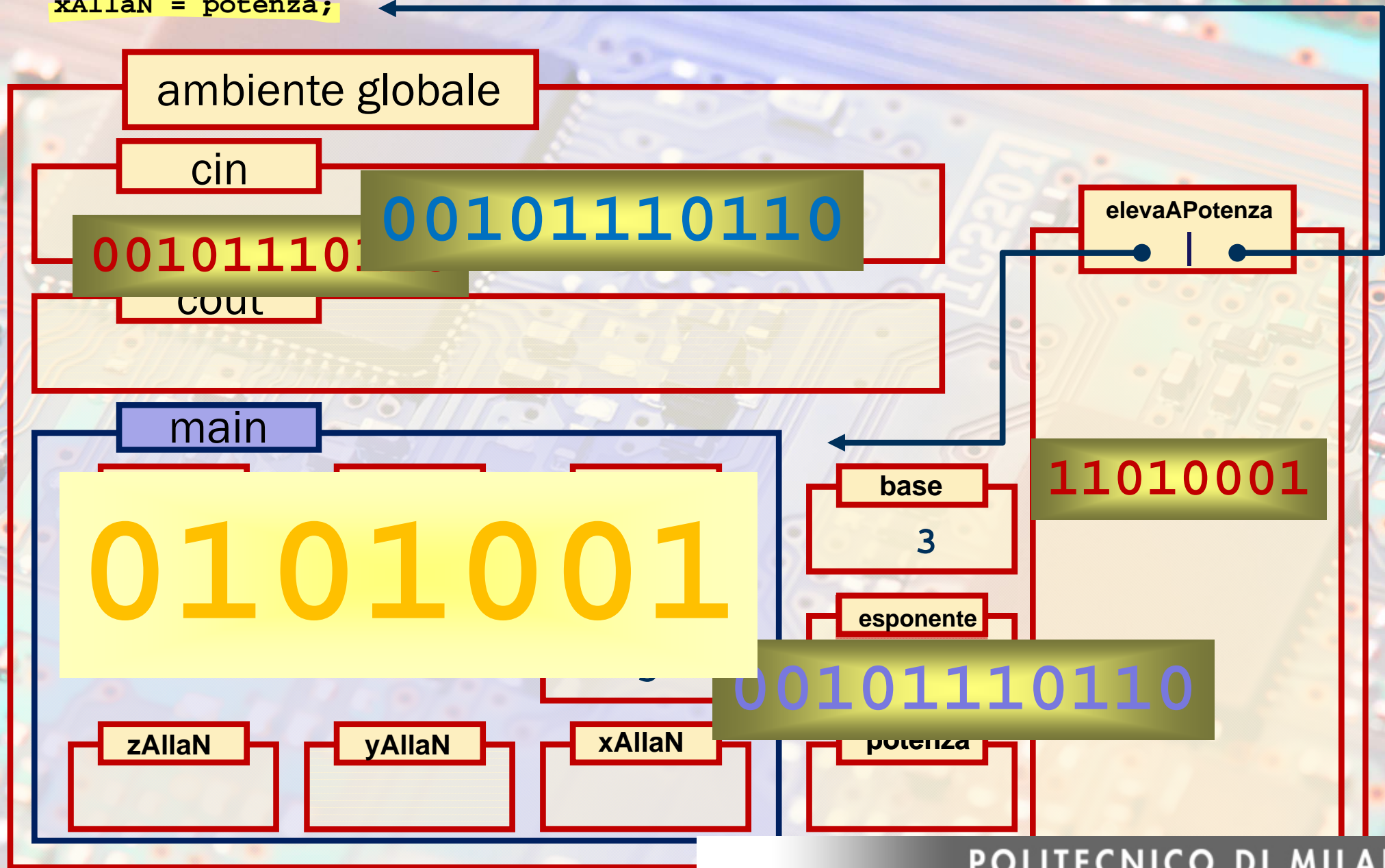



```
//calcola x elevato a n, con risultato in xAllaN
```

```
base = x; esponente = n;
```

```
elevaAPotenza();
```

```
xAllaN = potenza;
```

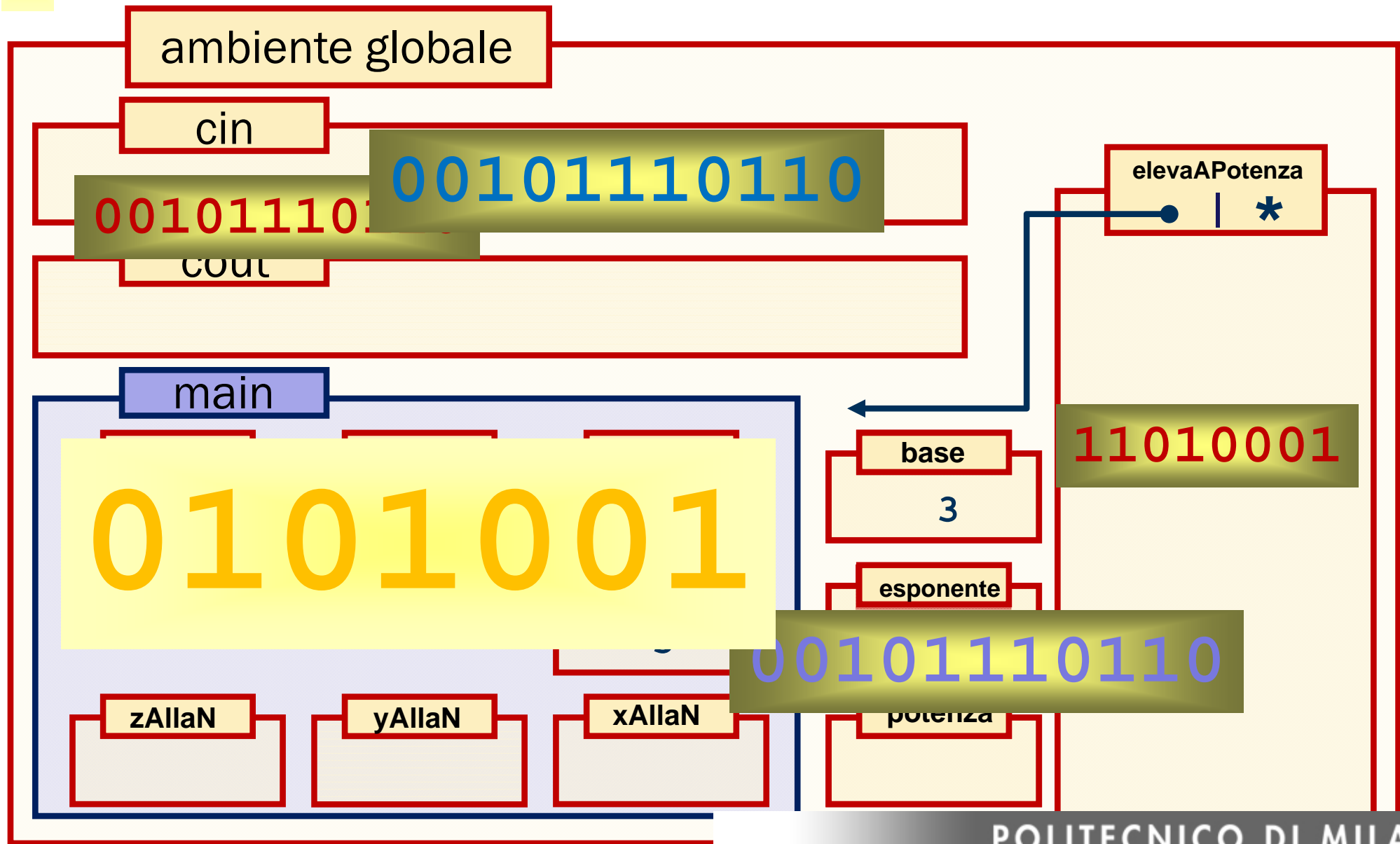



```
//calcola x elevato a n, con risultato in xAllaN
```

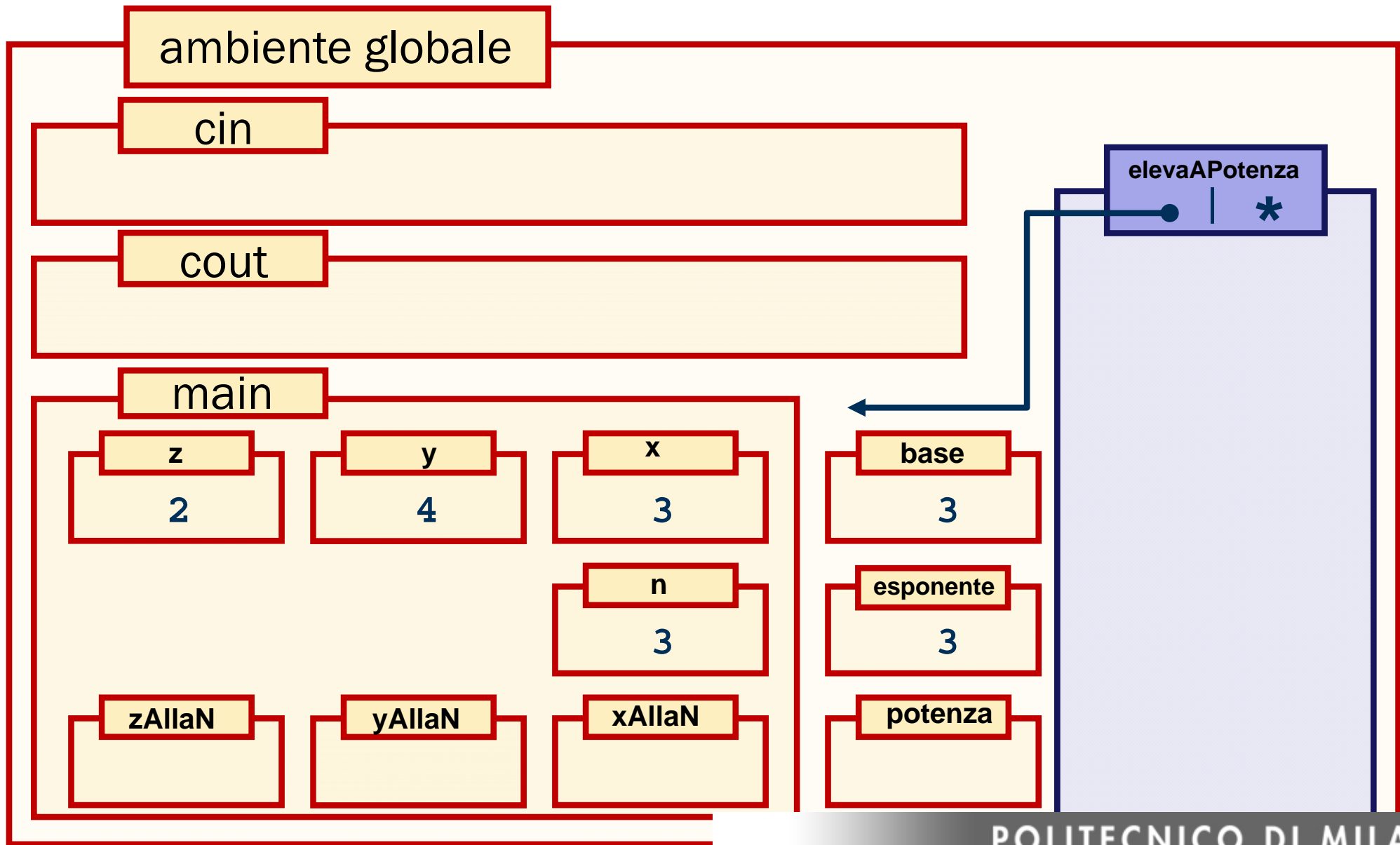
```
base = x; esponente = n;
```

```
elevaAPotenza();
```

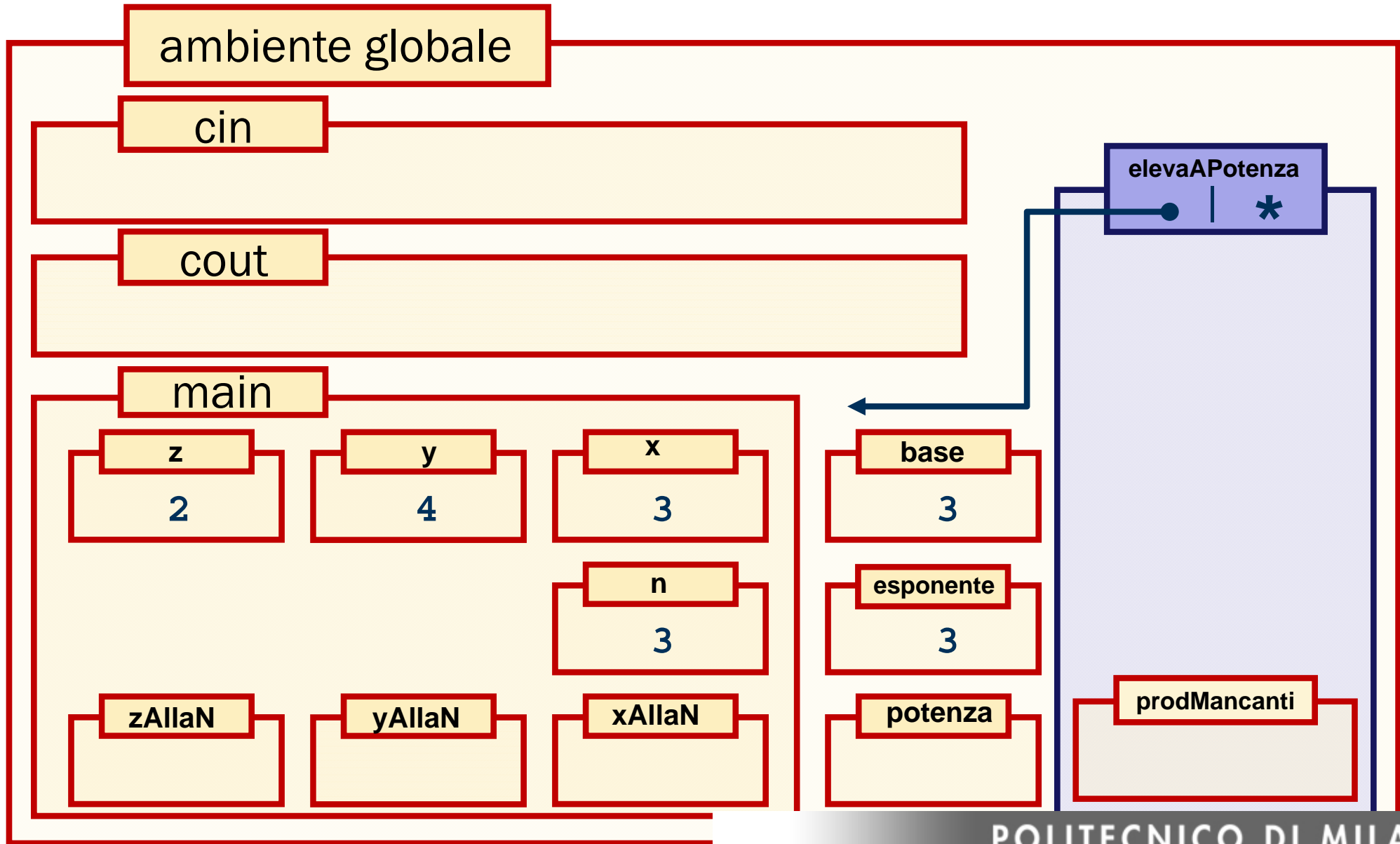
```
* xAllaN = potenza;
```



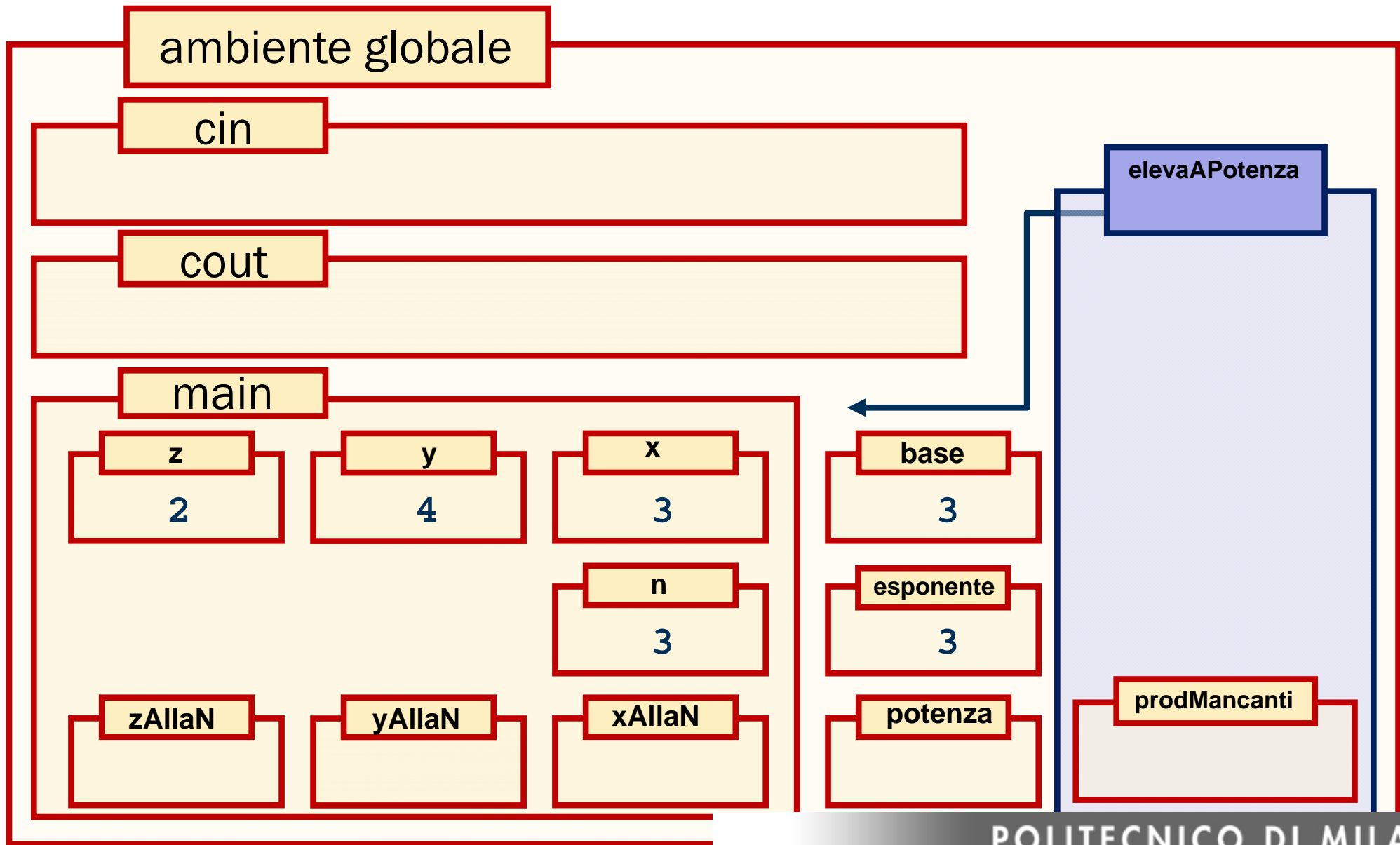
```
void elevaAPotenza()  
{ //versione per esponente positivo  
    int prodMancanti;           // variabile locale ad elevaAPotenza
```



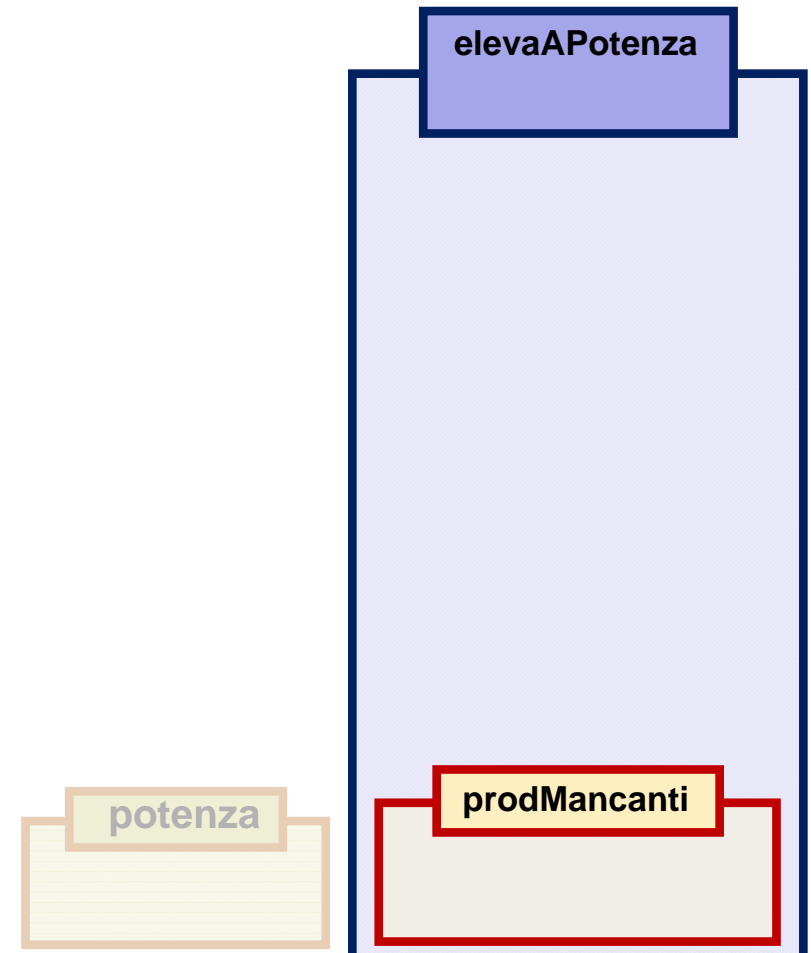
```
void elevaAPotenza()  
{ //versione per esponente positivo  
  int prodMancanti;           // variabile locale ad elevaAPotenza  
  potenza = 1;  
}
```



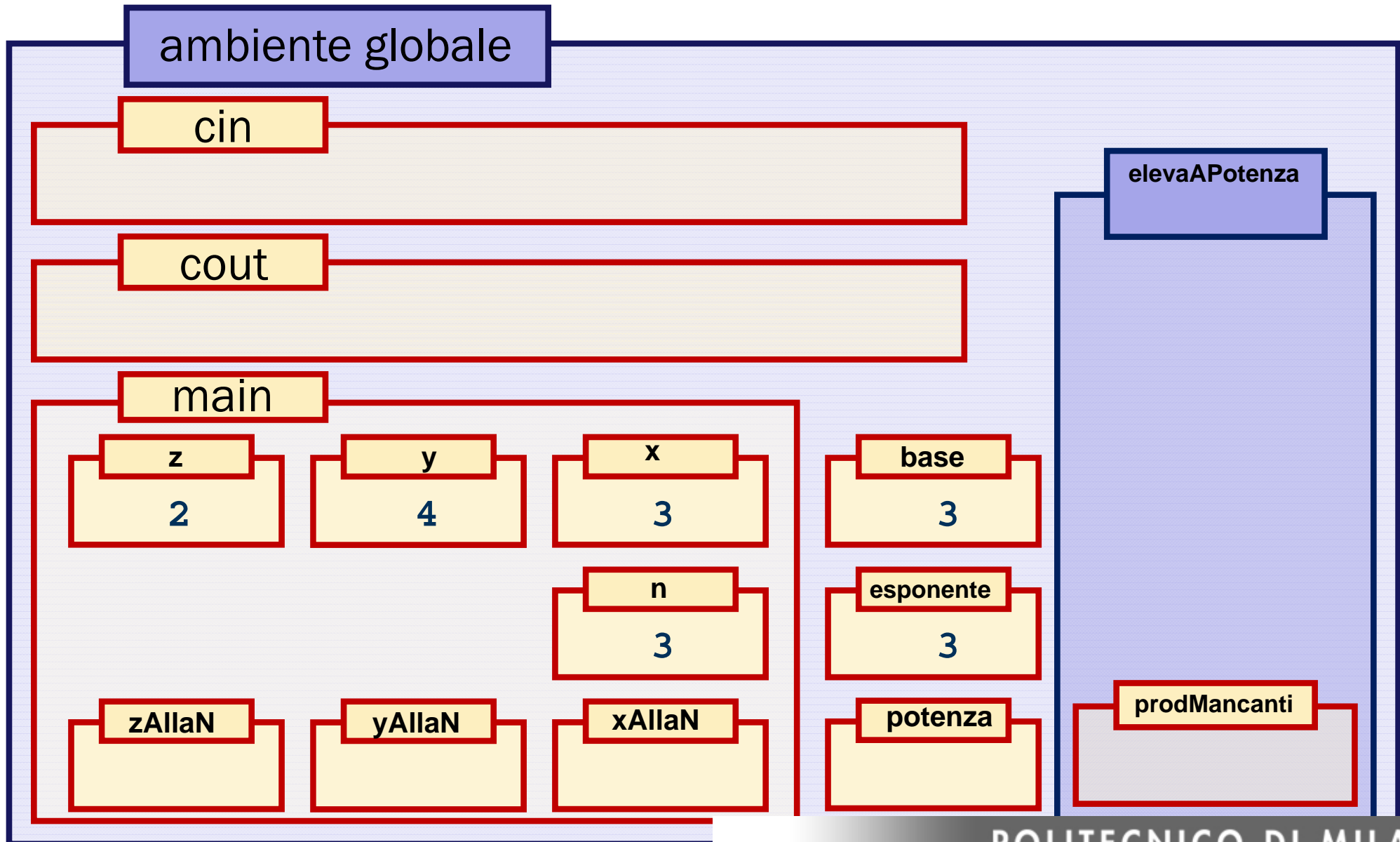
```
{ //versione per esponente positivo
  int prodMancanti;           // variabile locale ad elevaAPotenza
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```



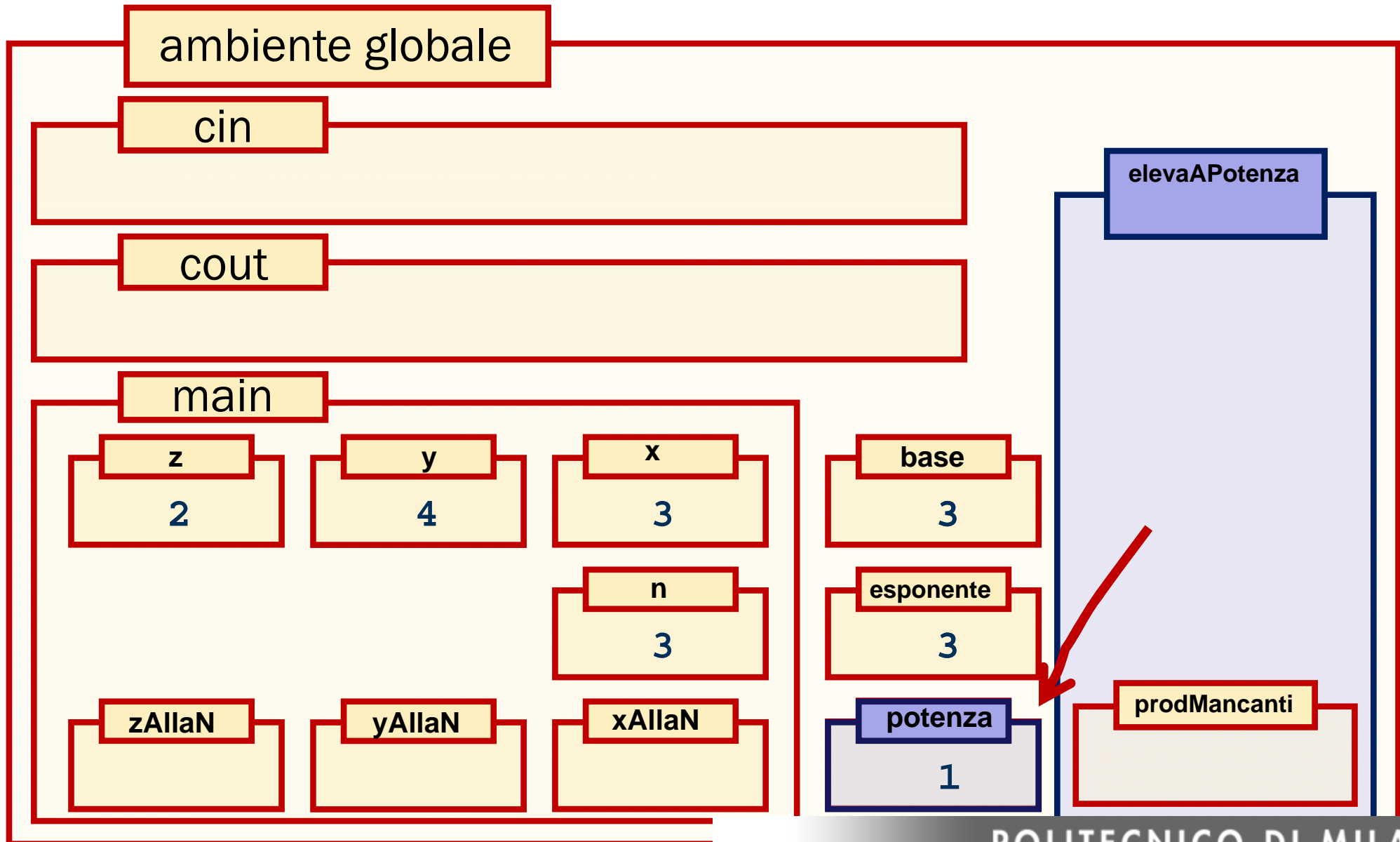
```
{ // versione per esponente positivo
  int prodMancanti;                // variabile locale ad elevaAPotenza
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```



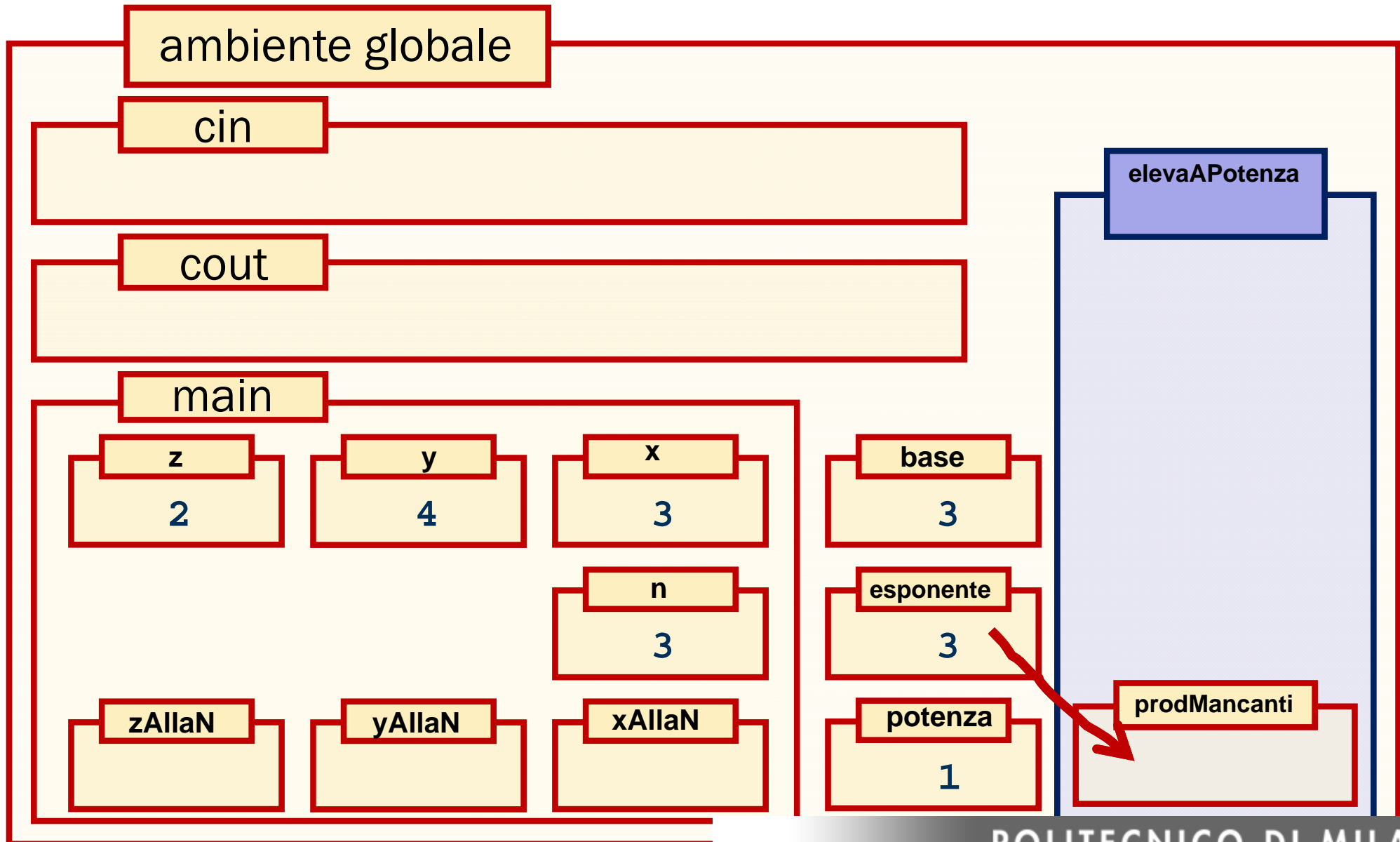
```
{ // versione per esponente positivo
  int prodMancanti;                // variabile locale ad elevaAPotenza
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```



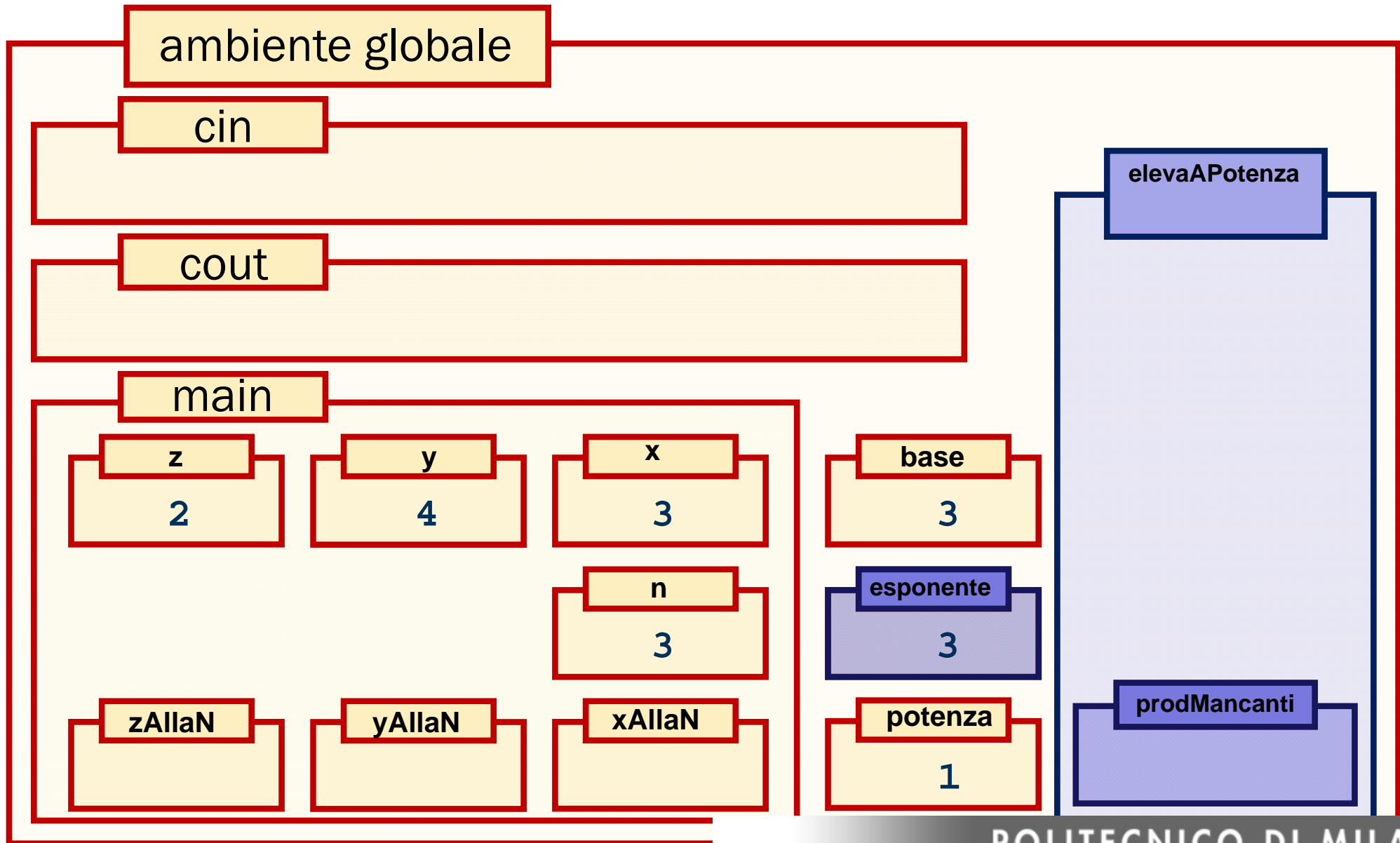
```
{ // versione per esponente positivo
  int prodMancanti;                // variabile locale ad elevaAPotenza
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```



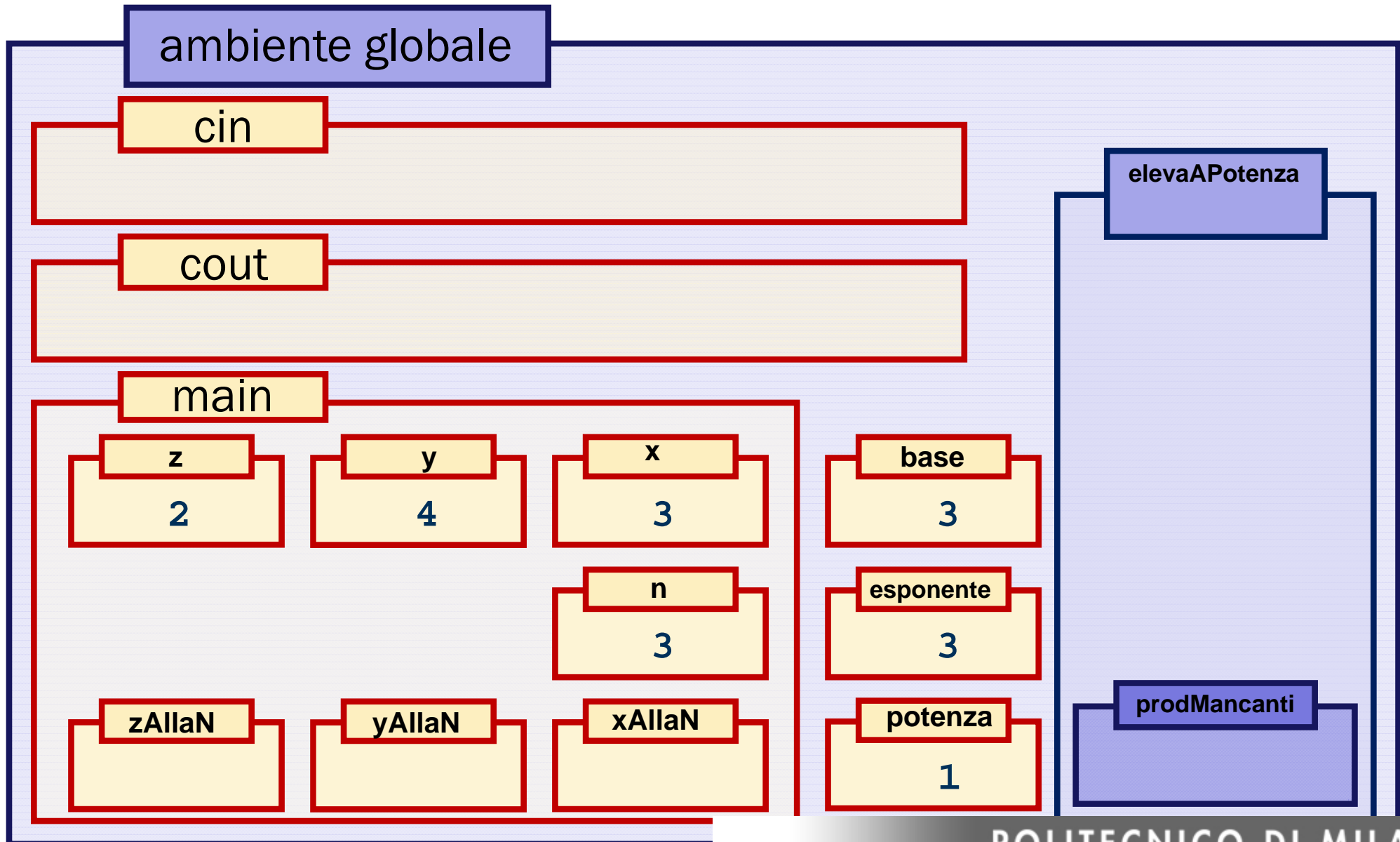

```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



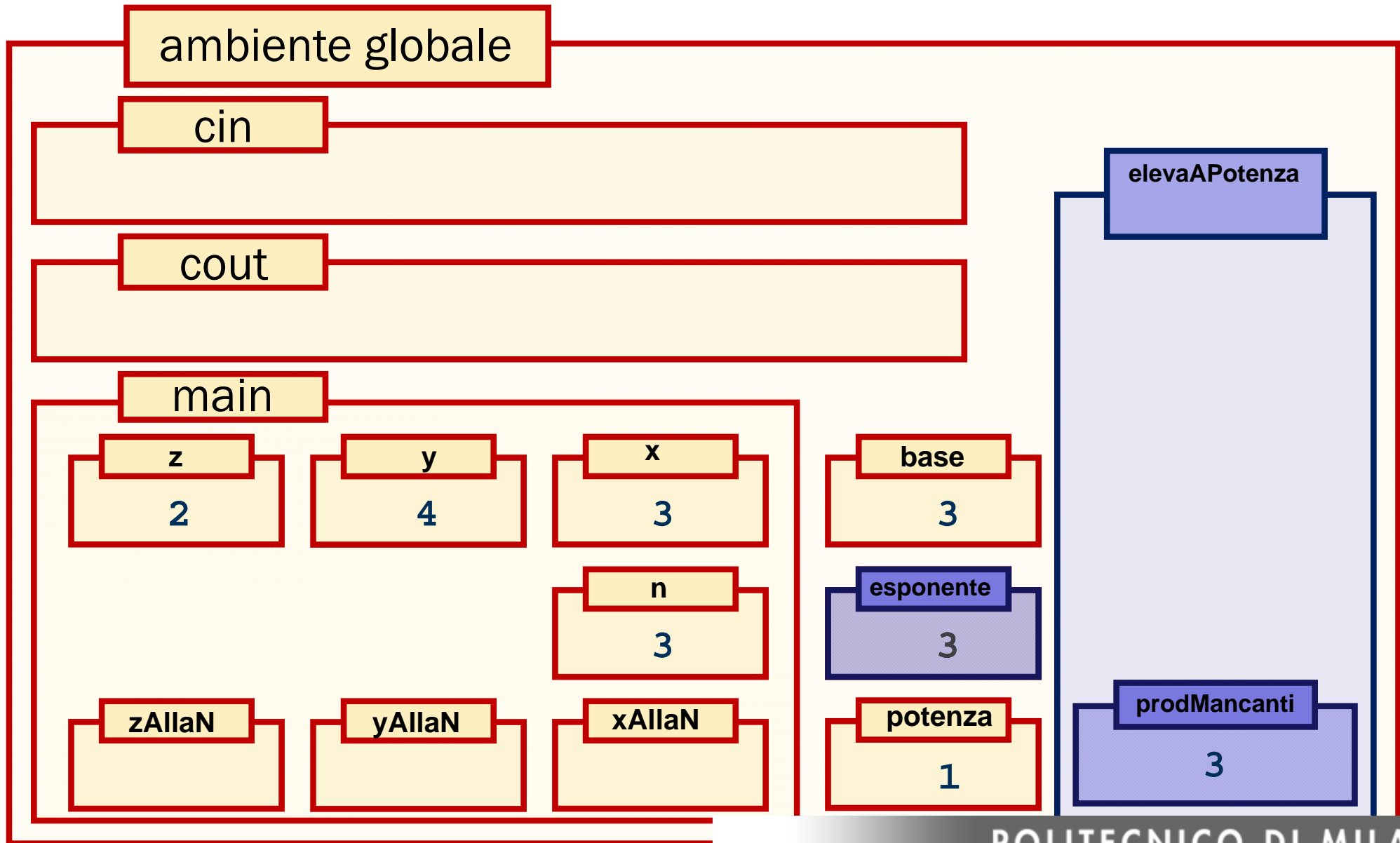

```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



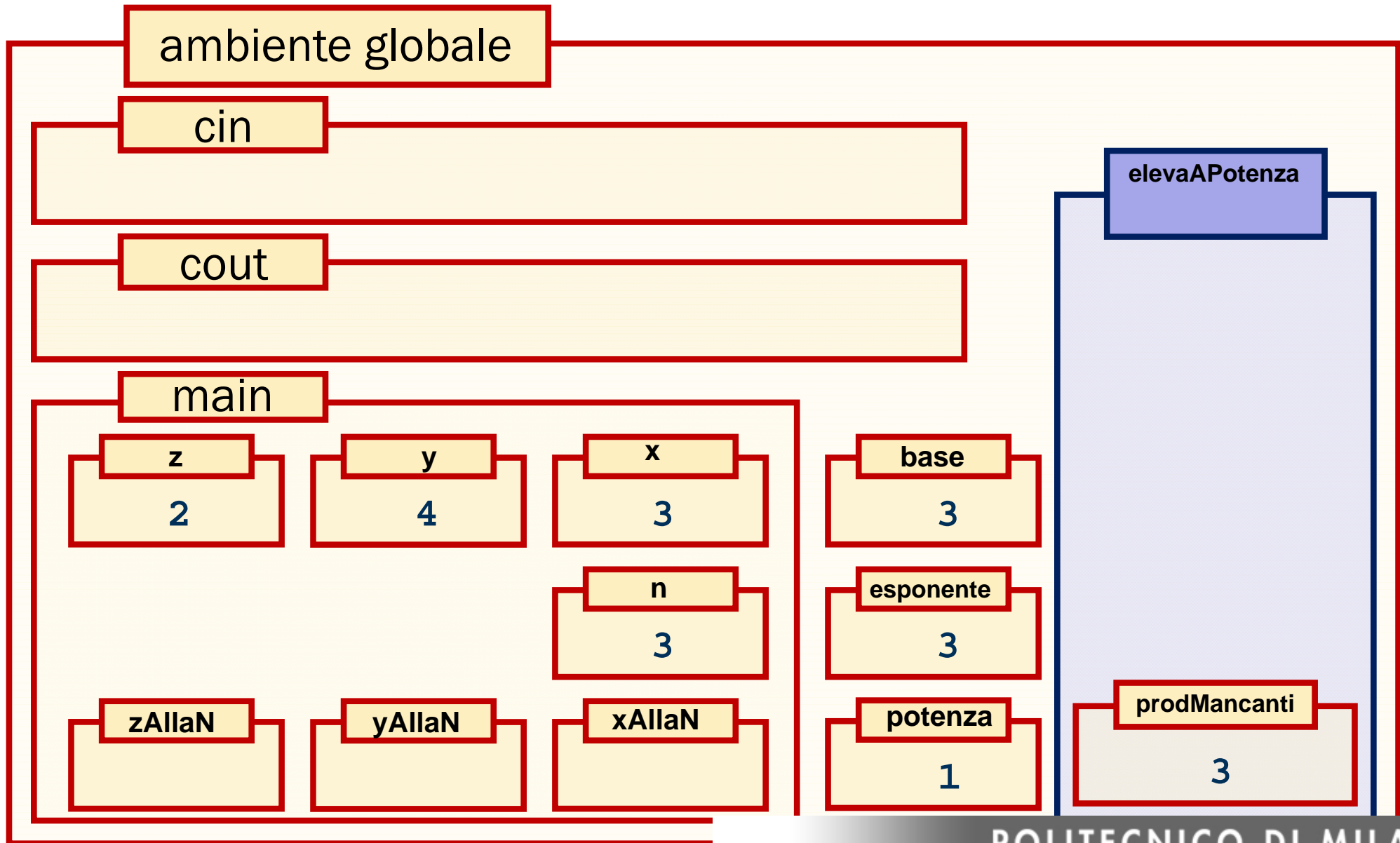
```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



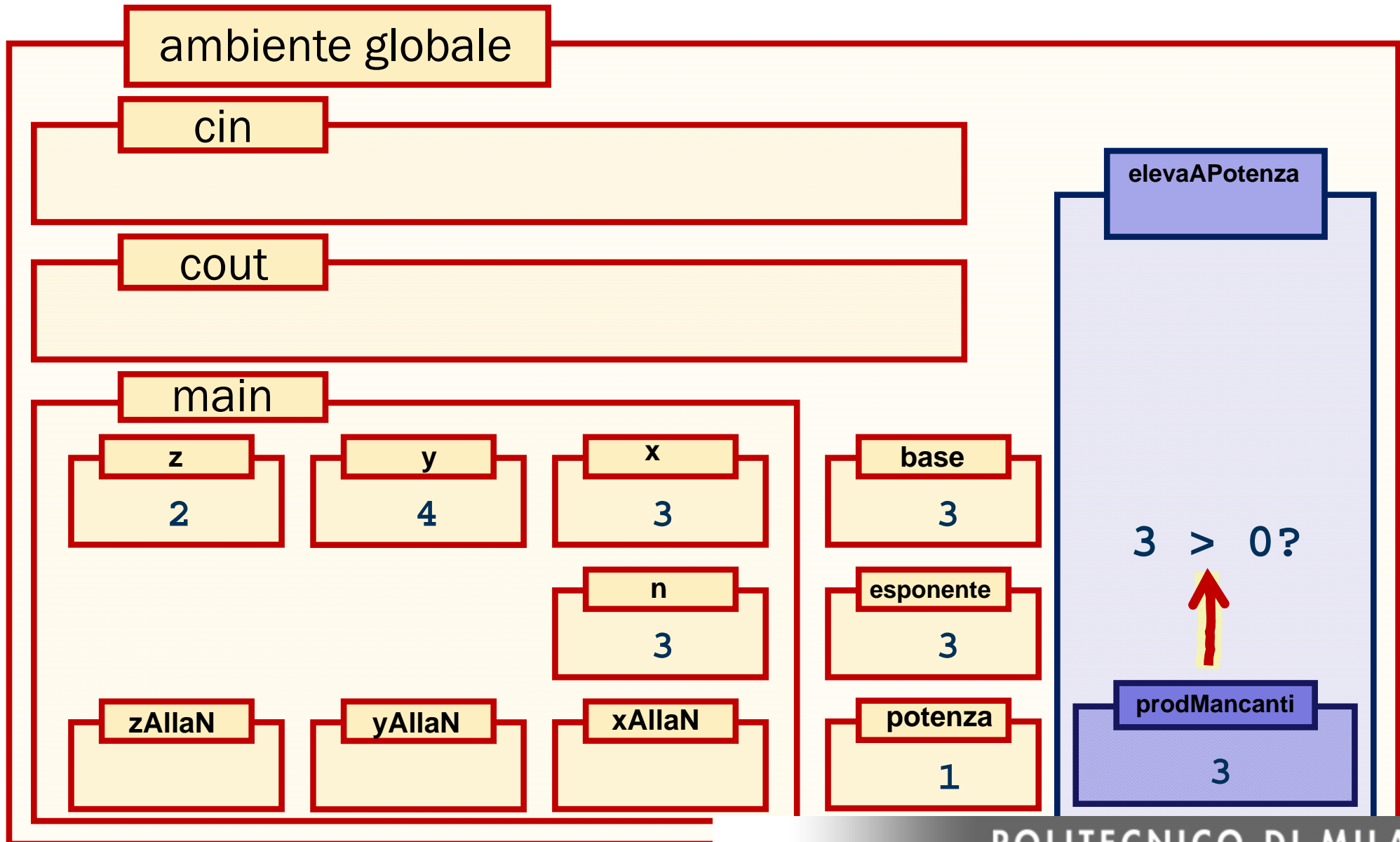
```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



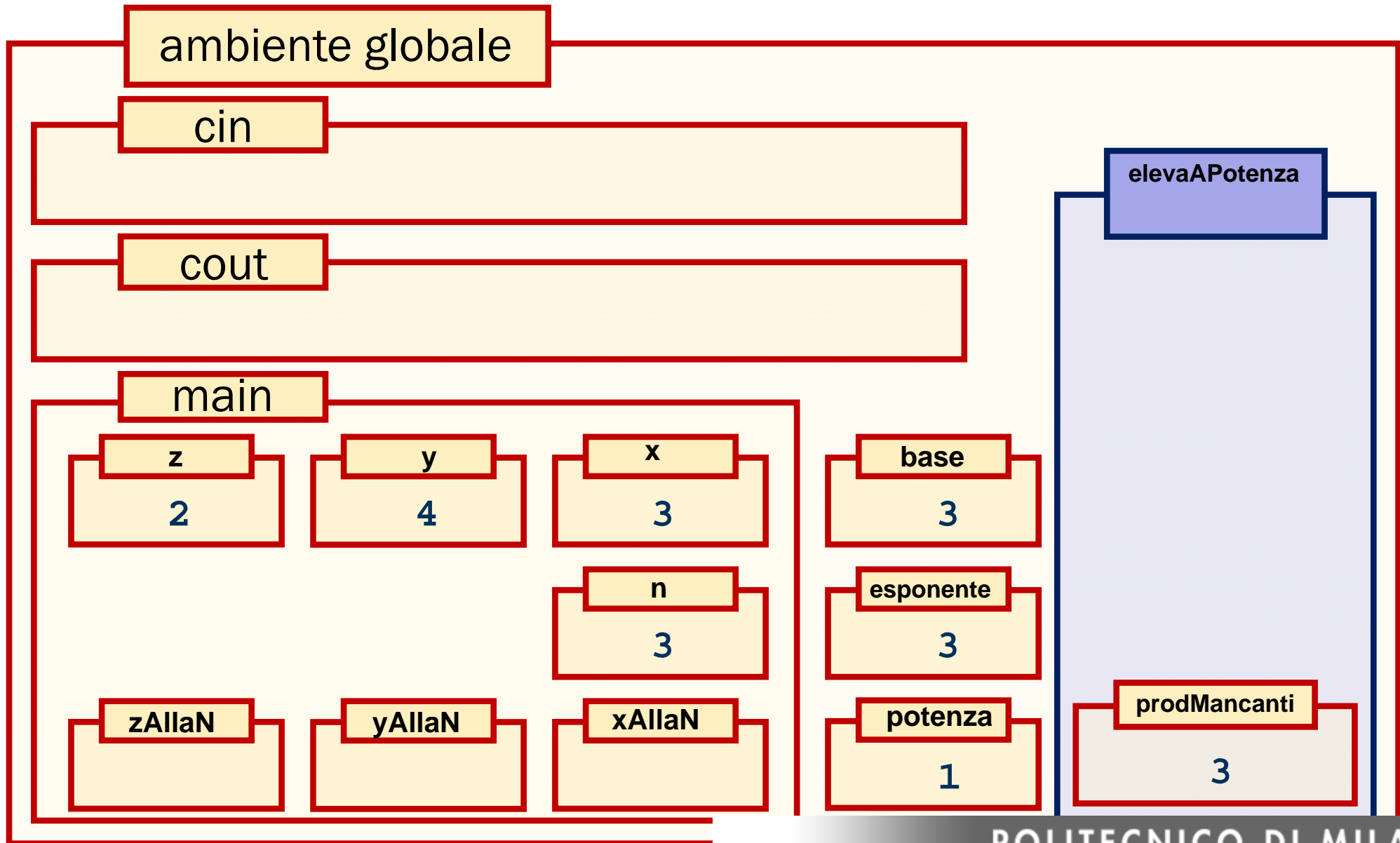
```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



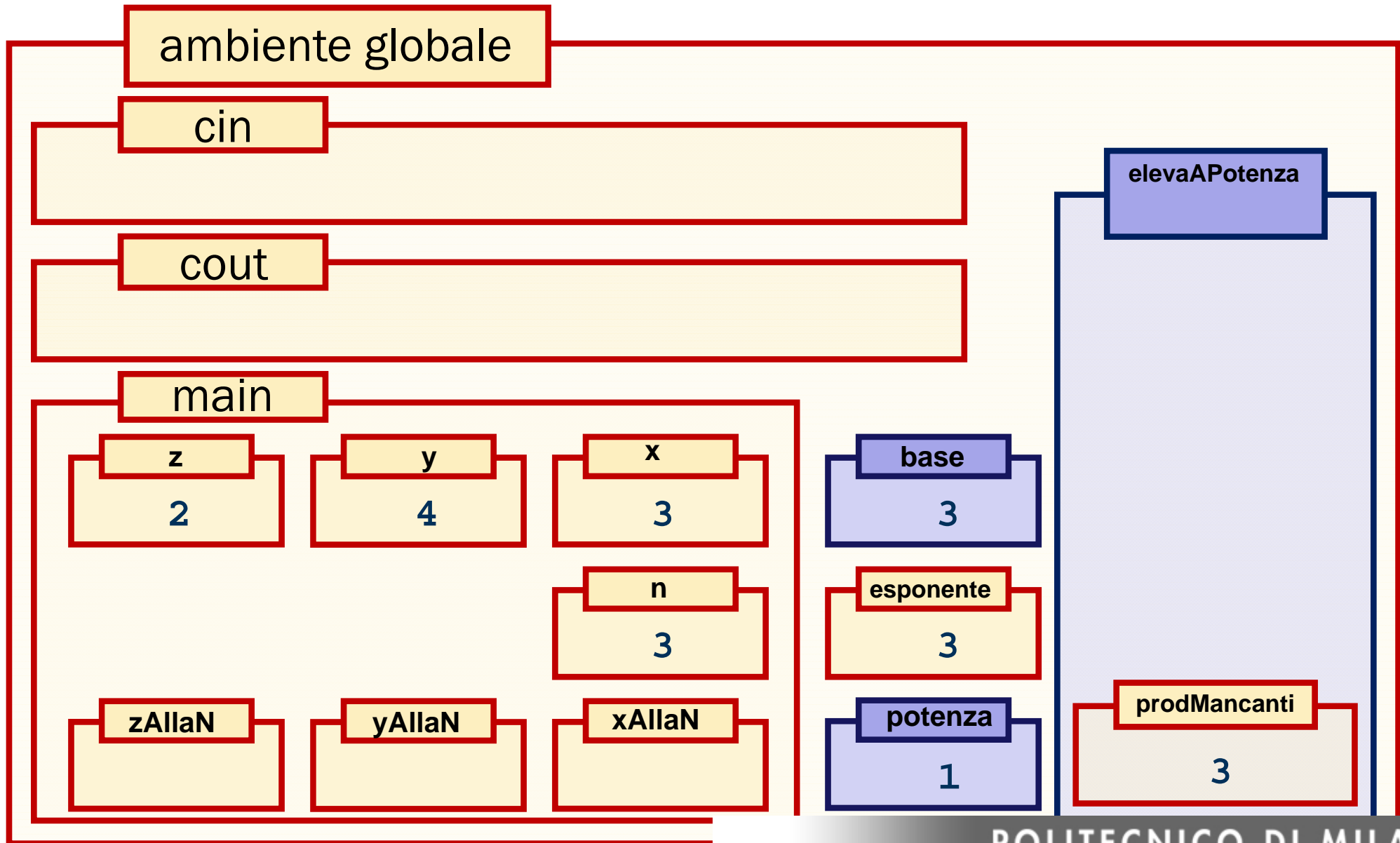
```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



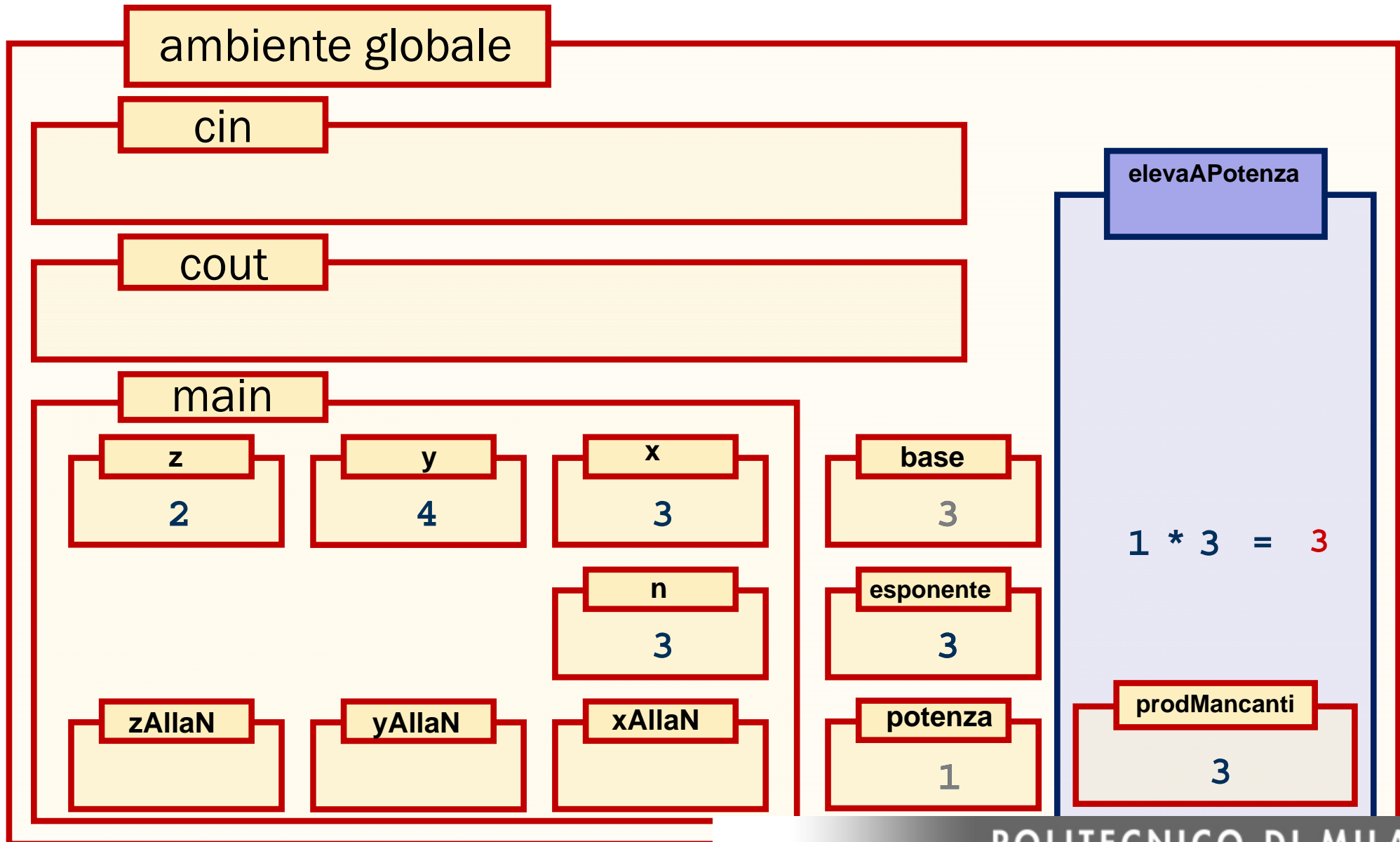
```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



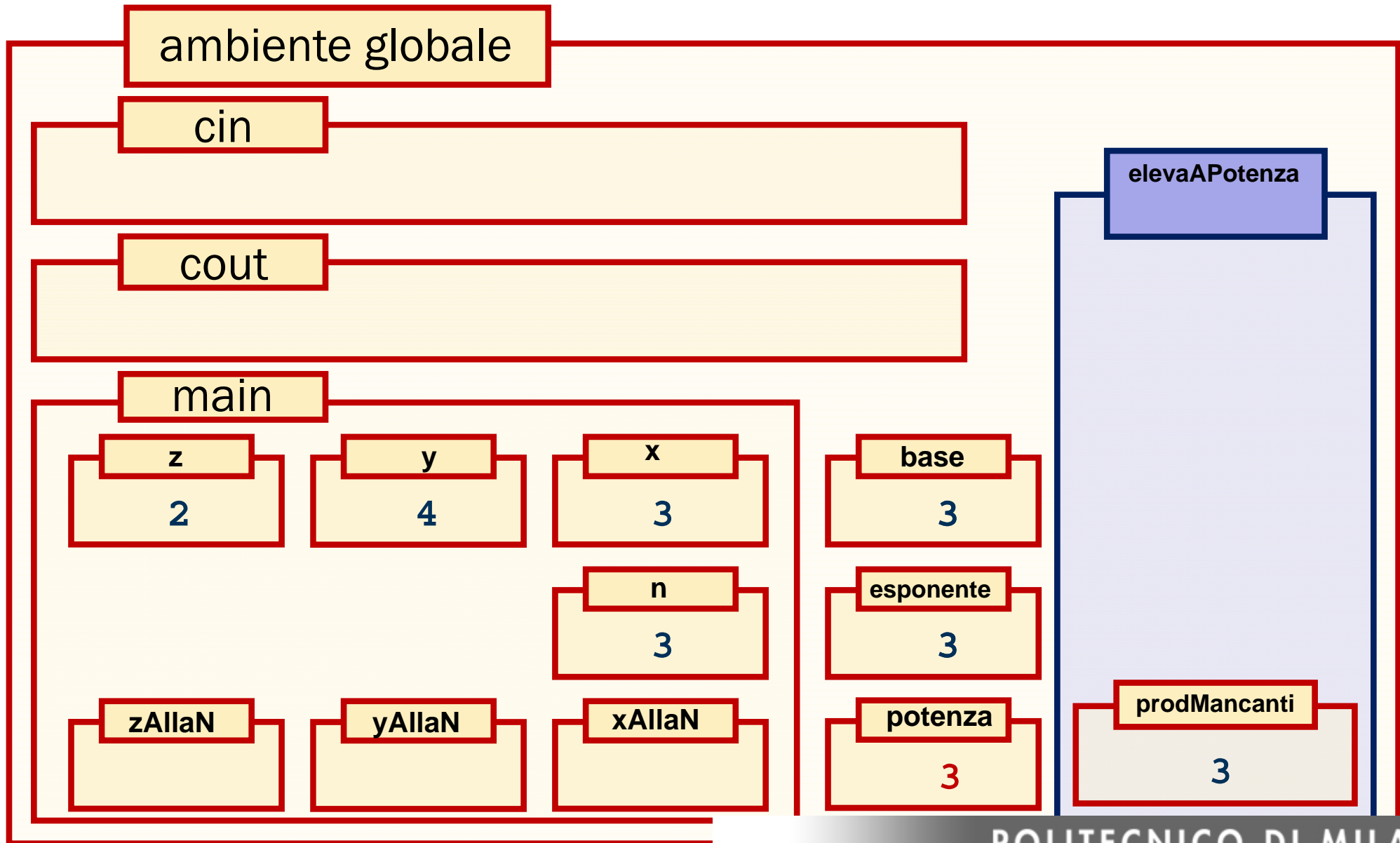
```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```




```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```

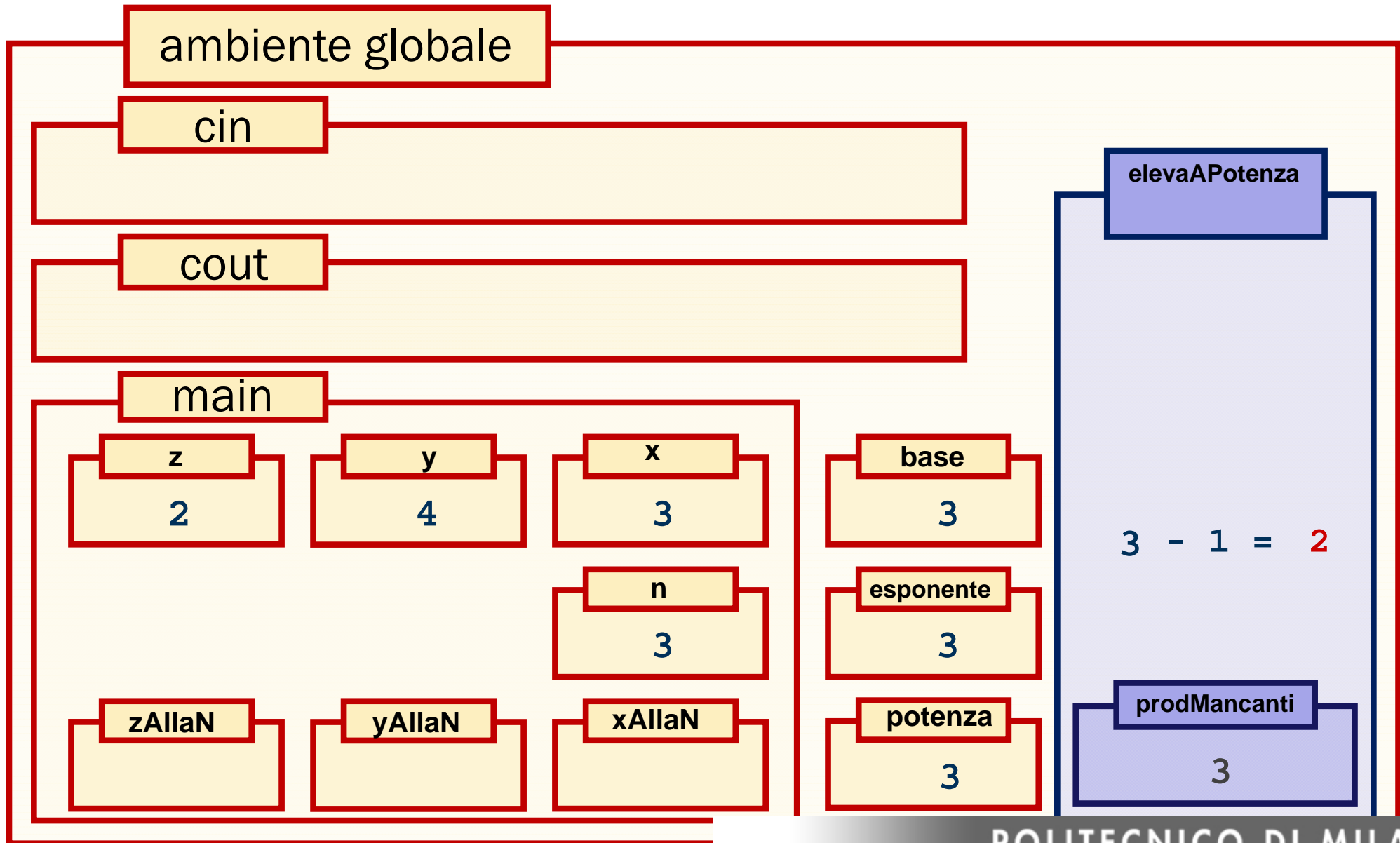



```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```

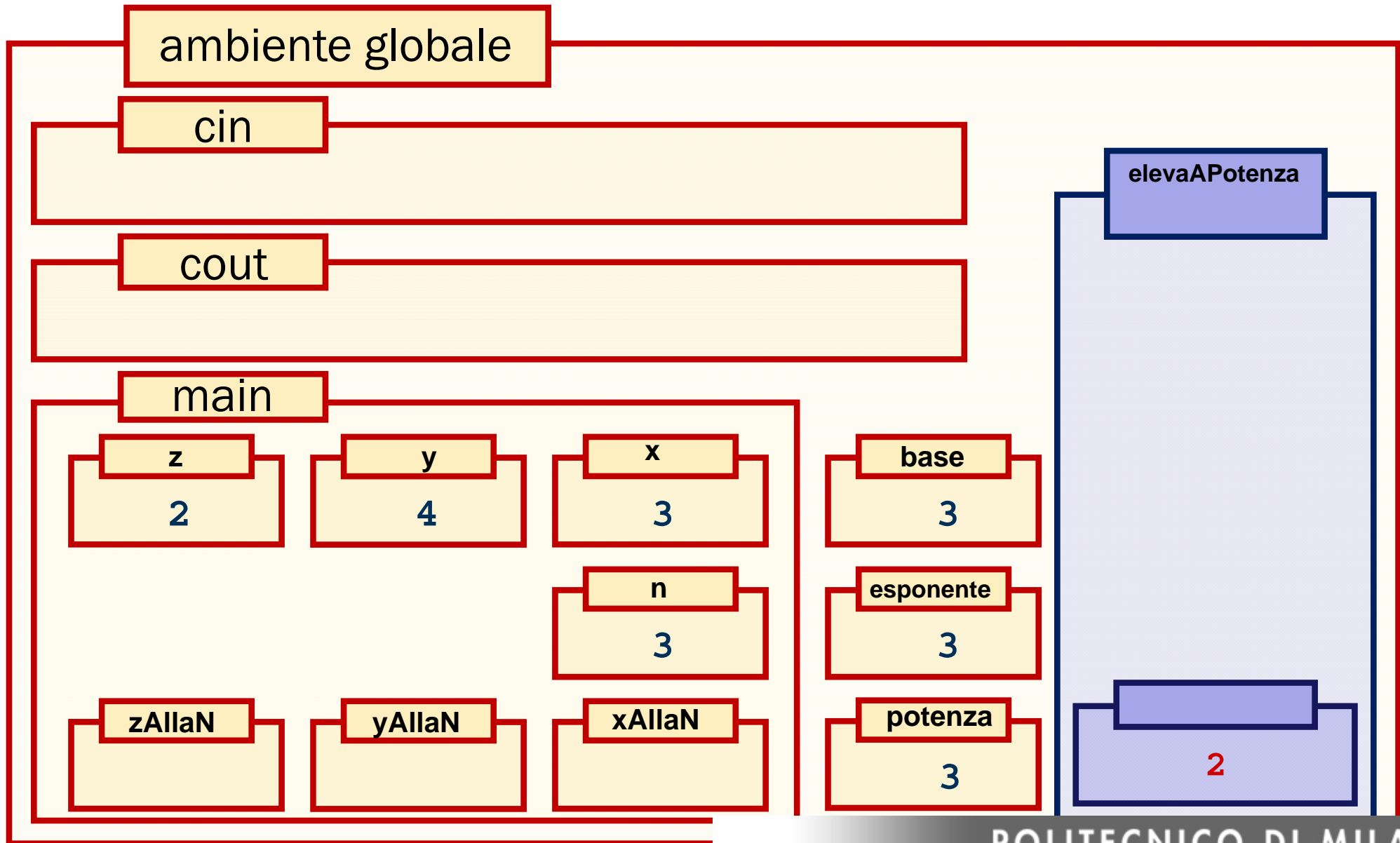


```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```

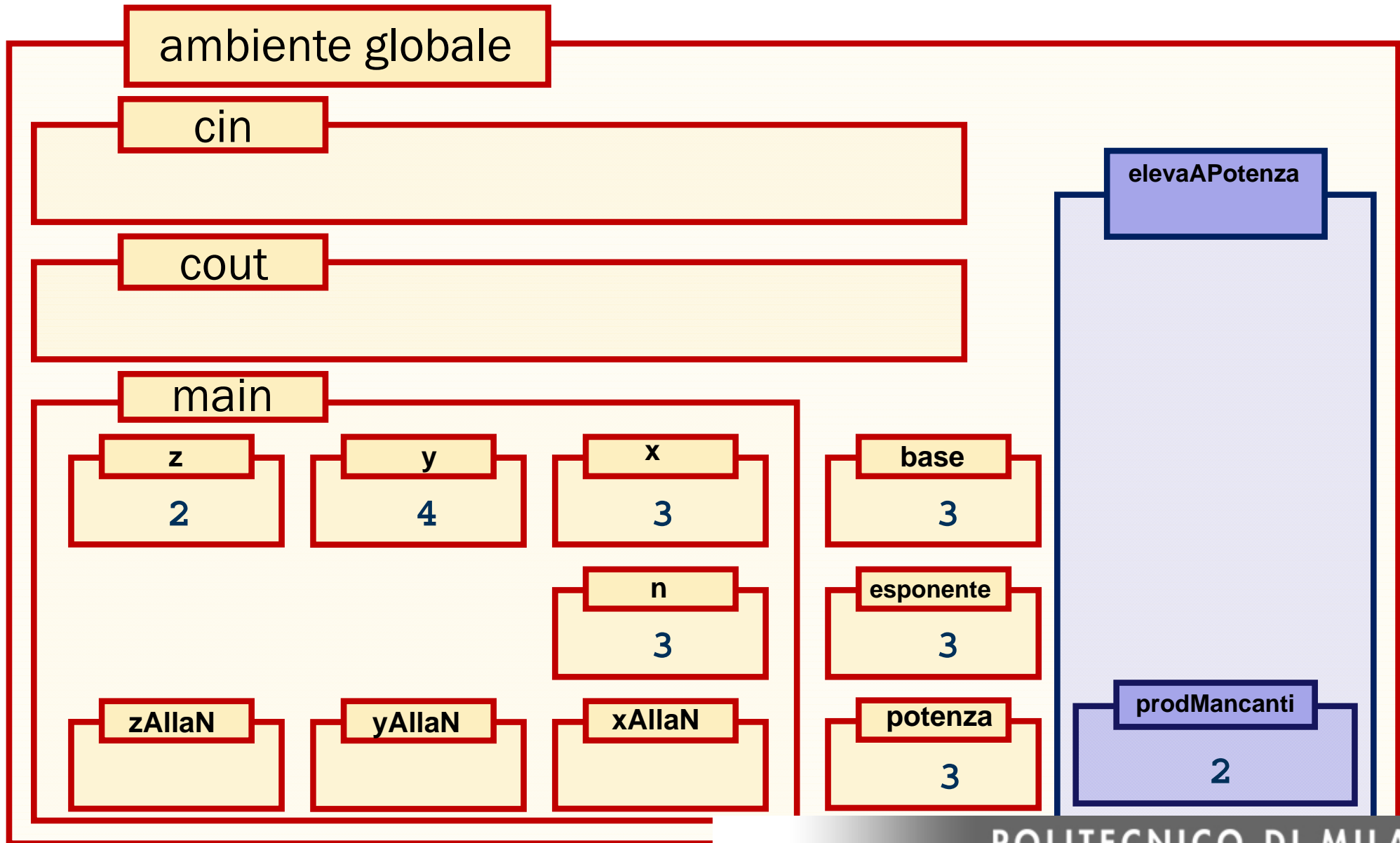
potenza *= base;



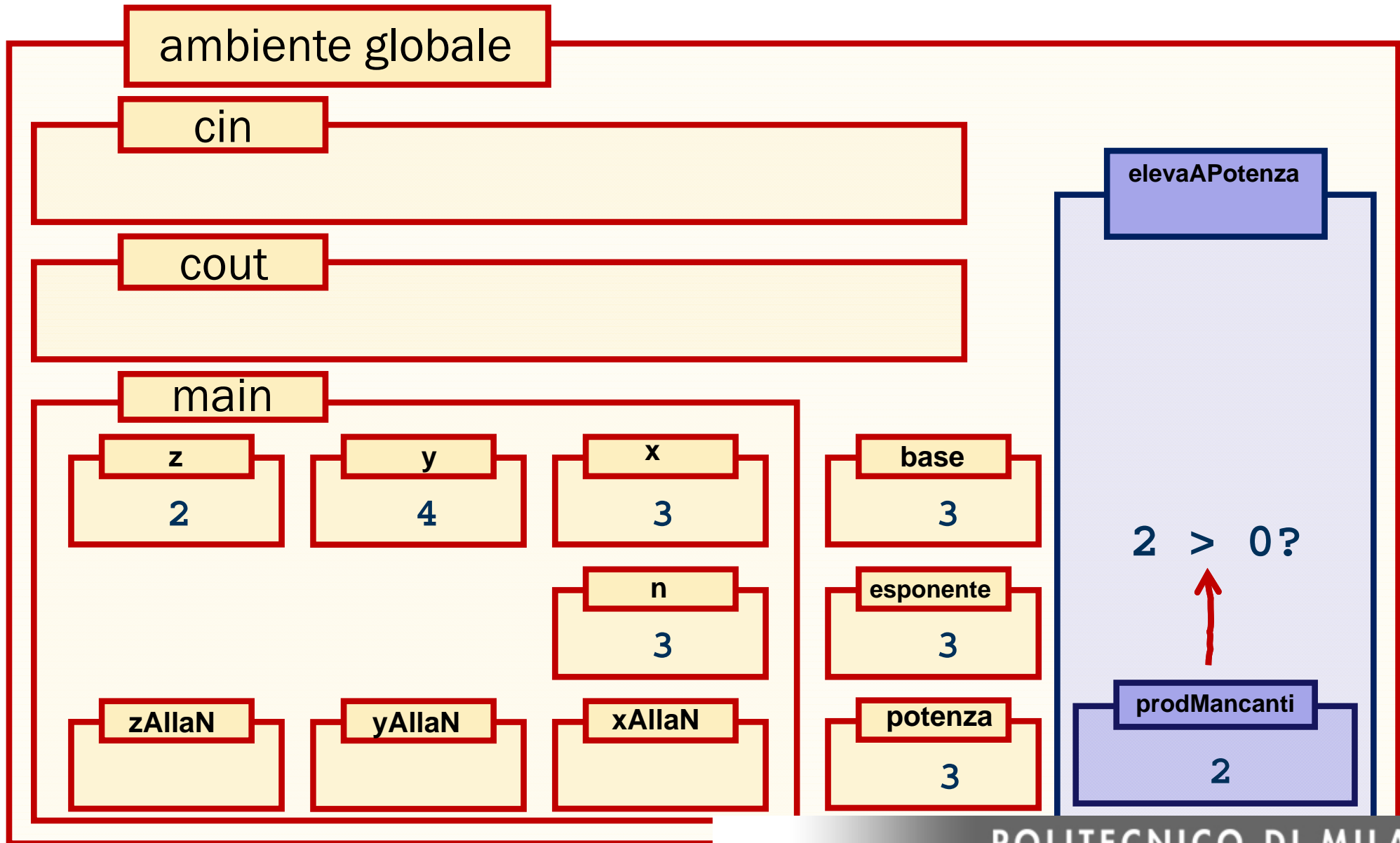
```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



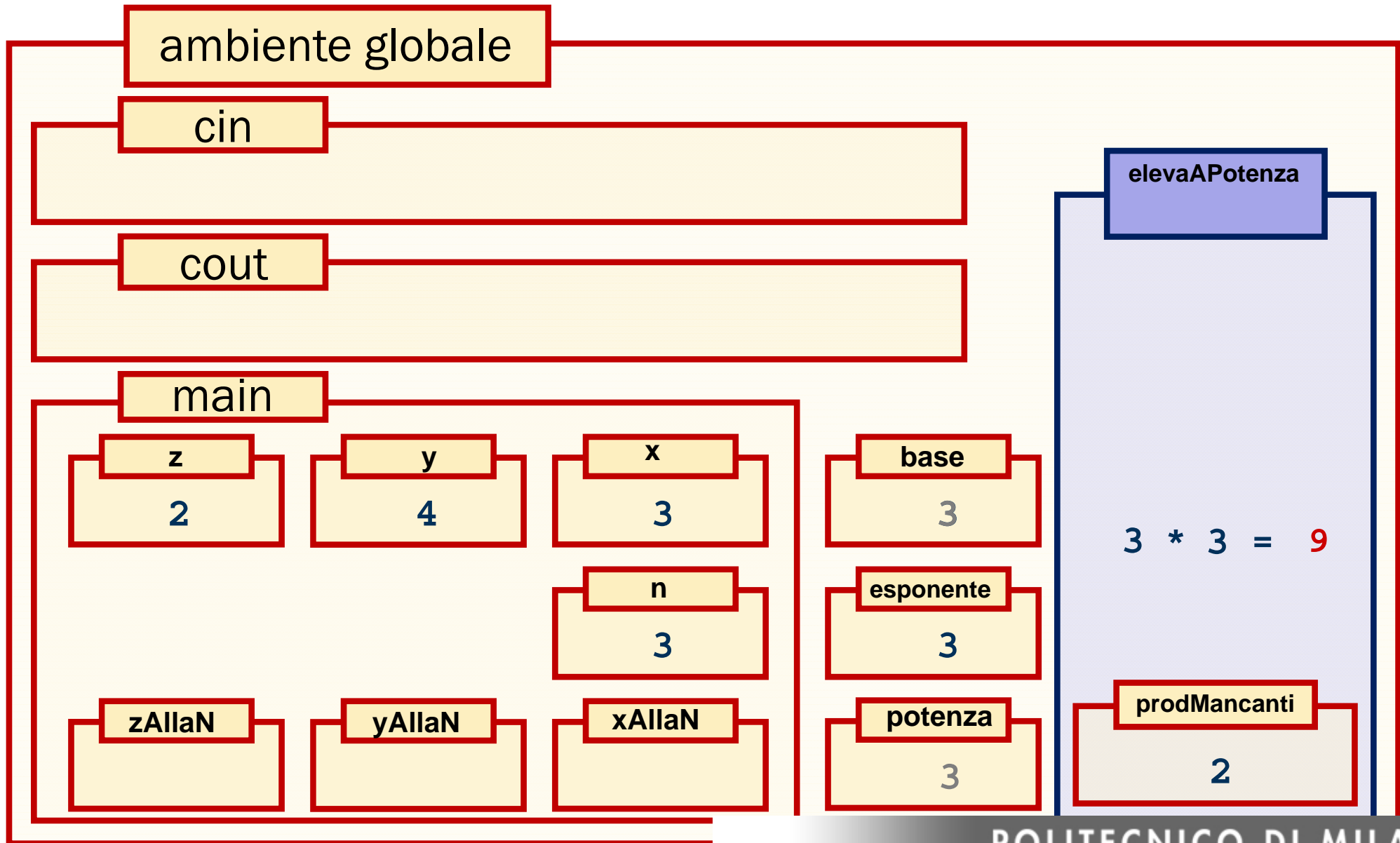
```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



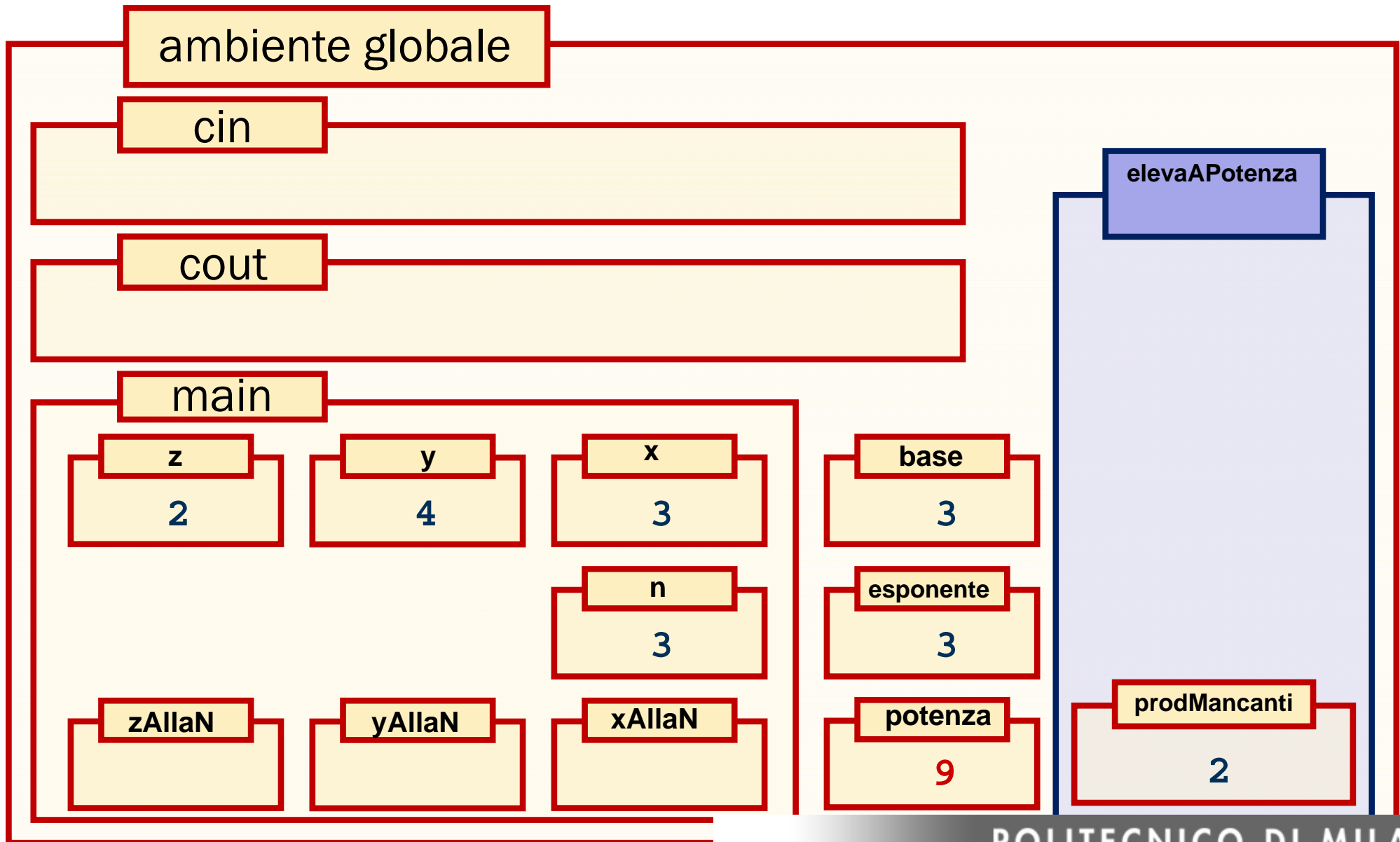
```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```




```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```

ambiente globale

cin

cout

main

z

2

y

4

x

3

n

3

zAllaN

yAllaN

xAllaN

base

3

esponente

3

potenza

9

elevaAPotenza

2 - 1 = 1

prodMancanti

2


```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```

ambiente globale

cin

cout

main

z

2

y

4

x

3

base

3

n

3

esponente

3

zAllaN

yAllaN

xAllaN

potenza

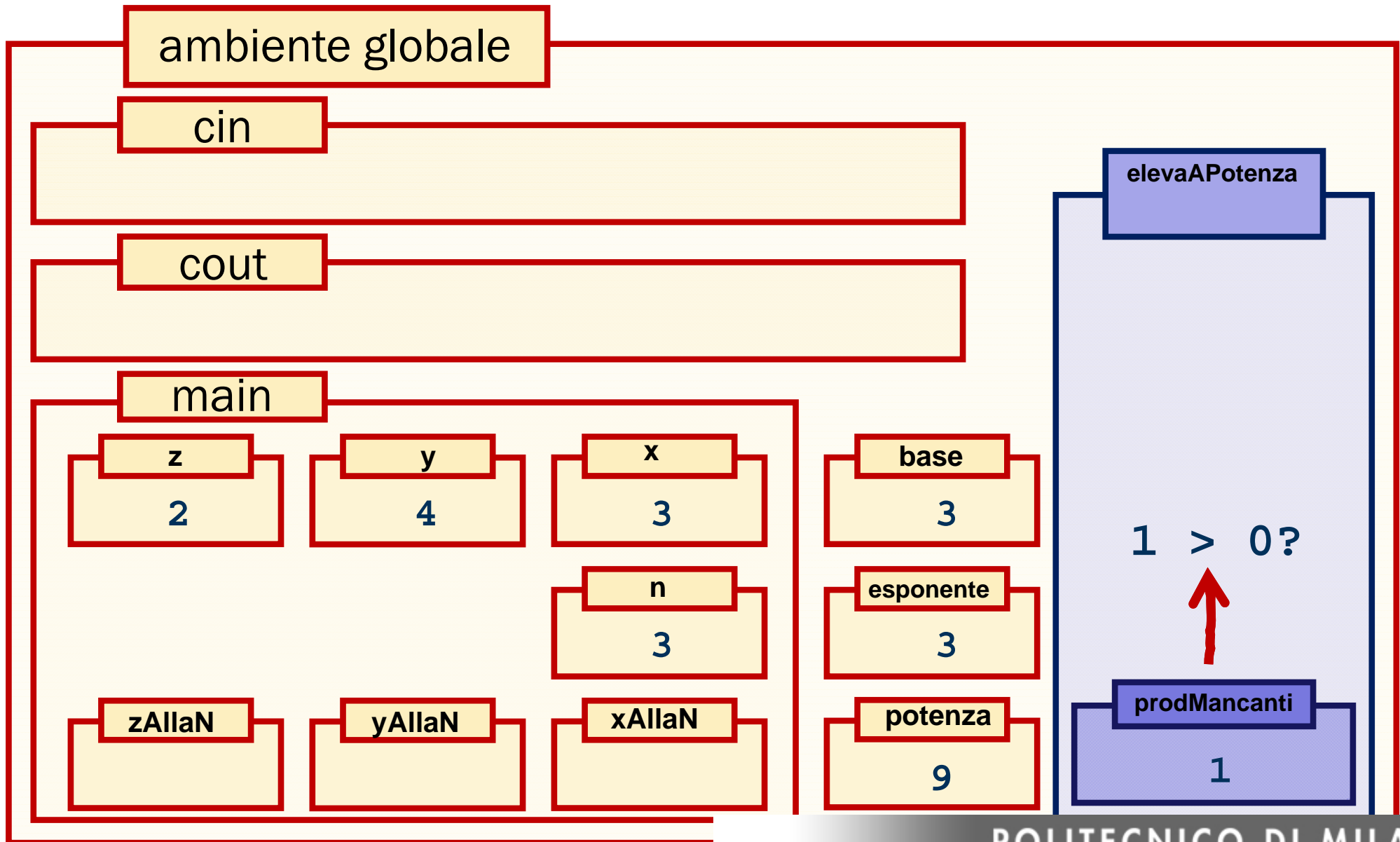
9

elevaAPotenza

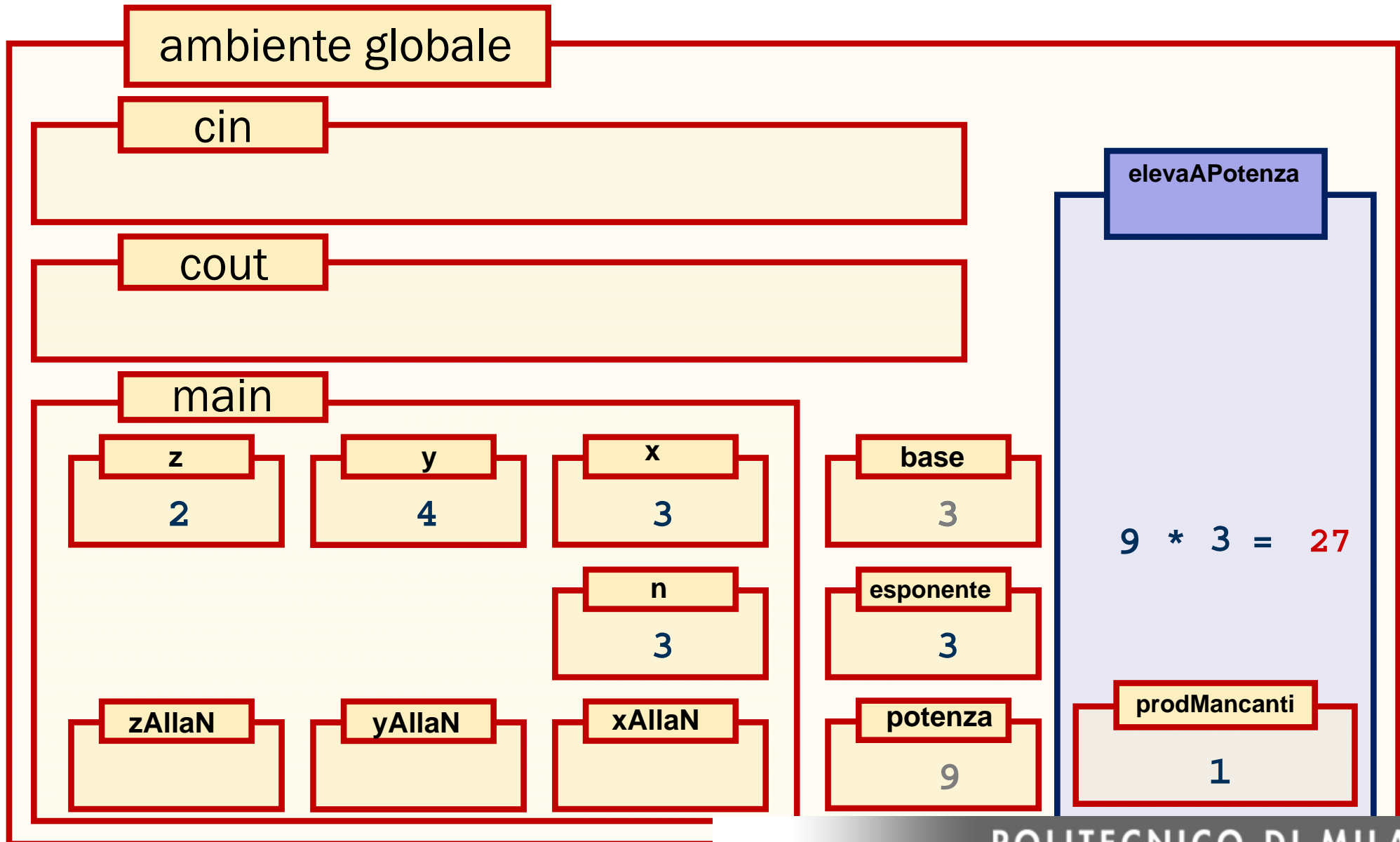
prodMancanti

1

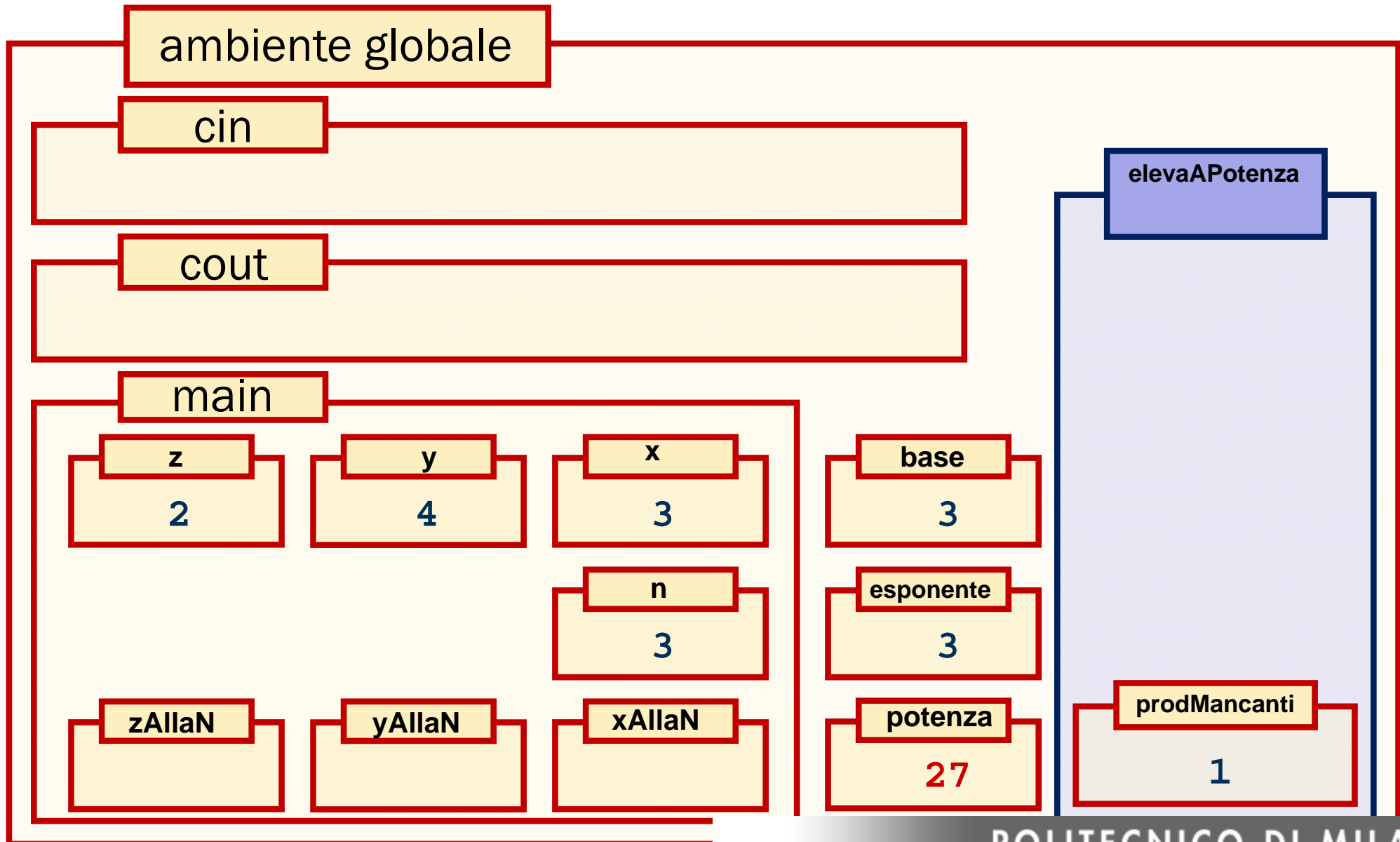
```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```



```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```

ambiente globale

cin

cout

main

z

2

y

4

x

3

n

3

zAllaN

yAllaN

xAllaN

base

3

esponente

3

potenza

27

elevaAPotenza

1 - 1 = 0

prodMancanti

1

```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```

ambiente globale

cin

cout

main

z

2

y

4

x

3

n

3

zAllaN

yAllaN

xAllaN

base

3

esponente

3

potenza

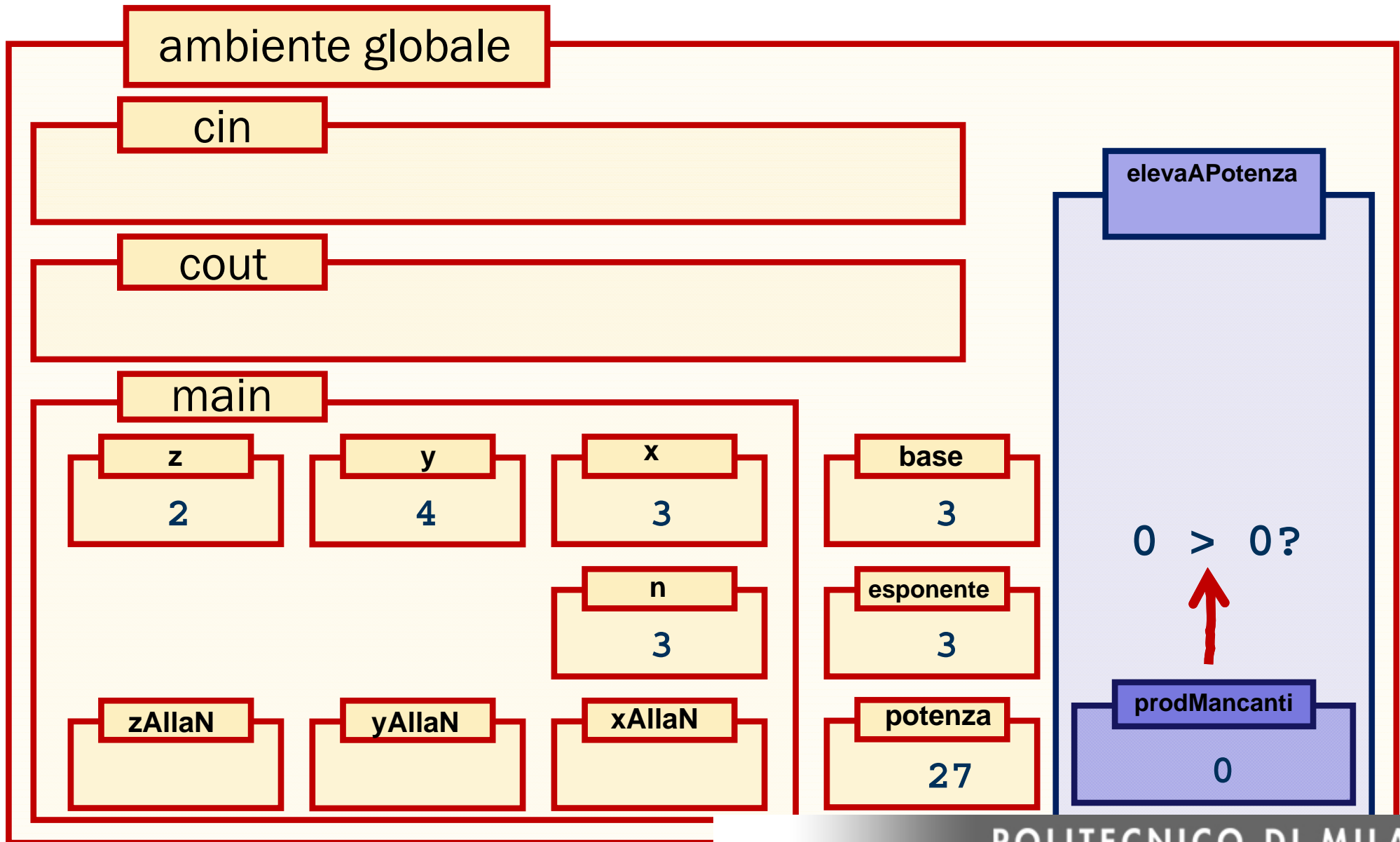
27

elevaAPotenza

prodMancanti

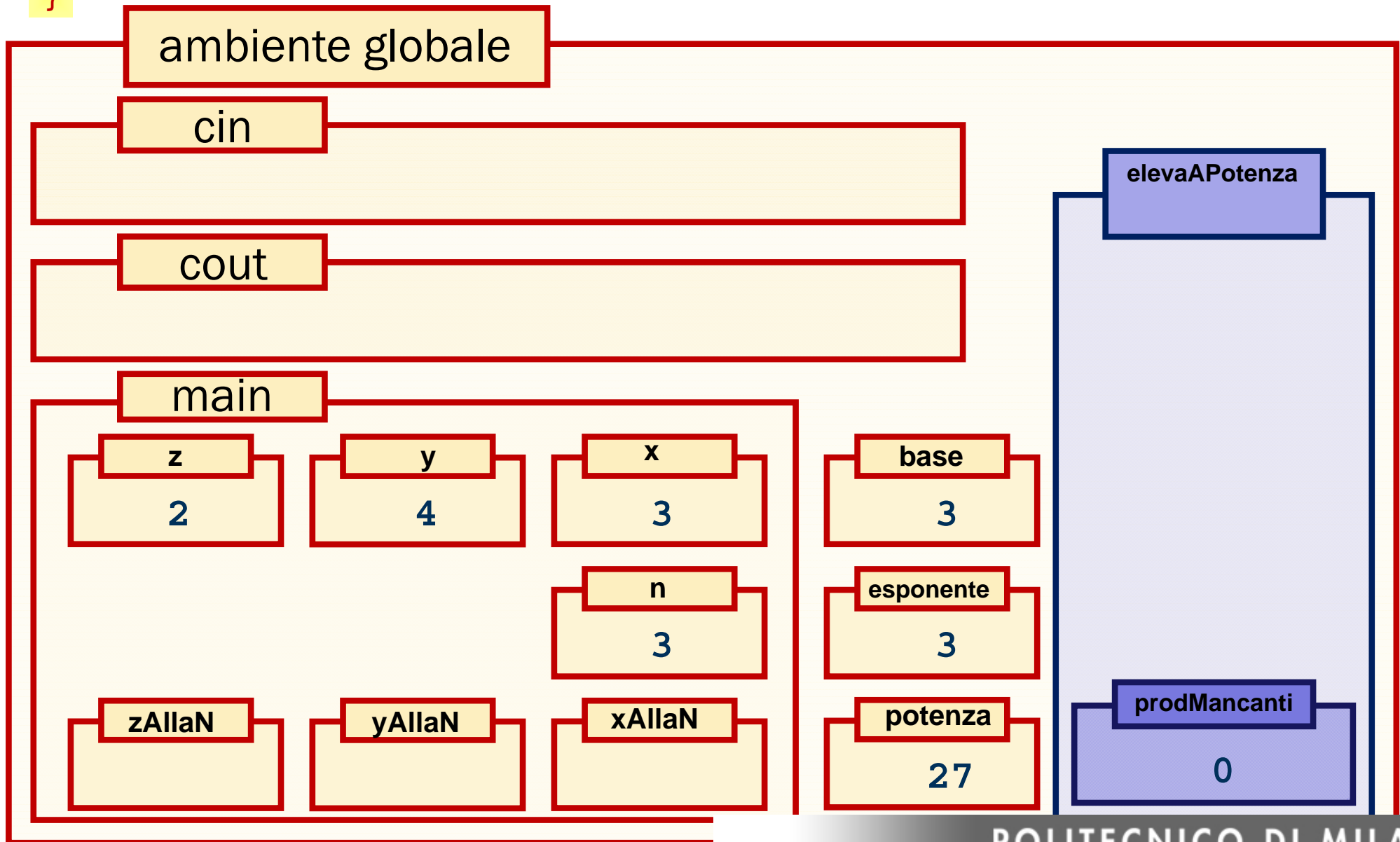
0

```
int prodMancanti; // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
```




```
potenza = 1;  
for (prodMancanti = esponente; prodMancanti >  
    potenza *= base;  
}
```

SOTTOPROBLEMA

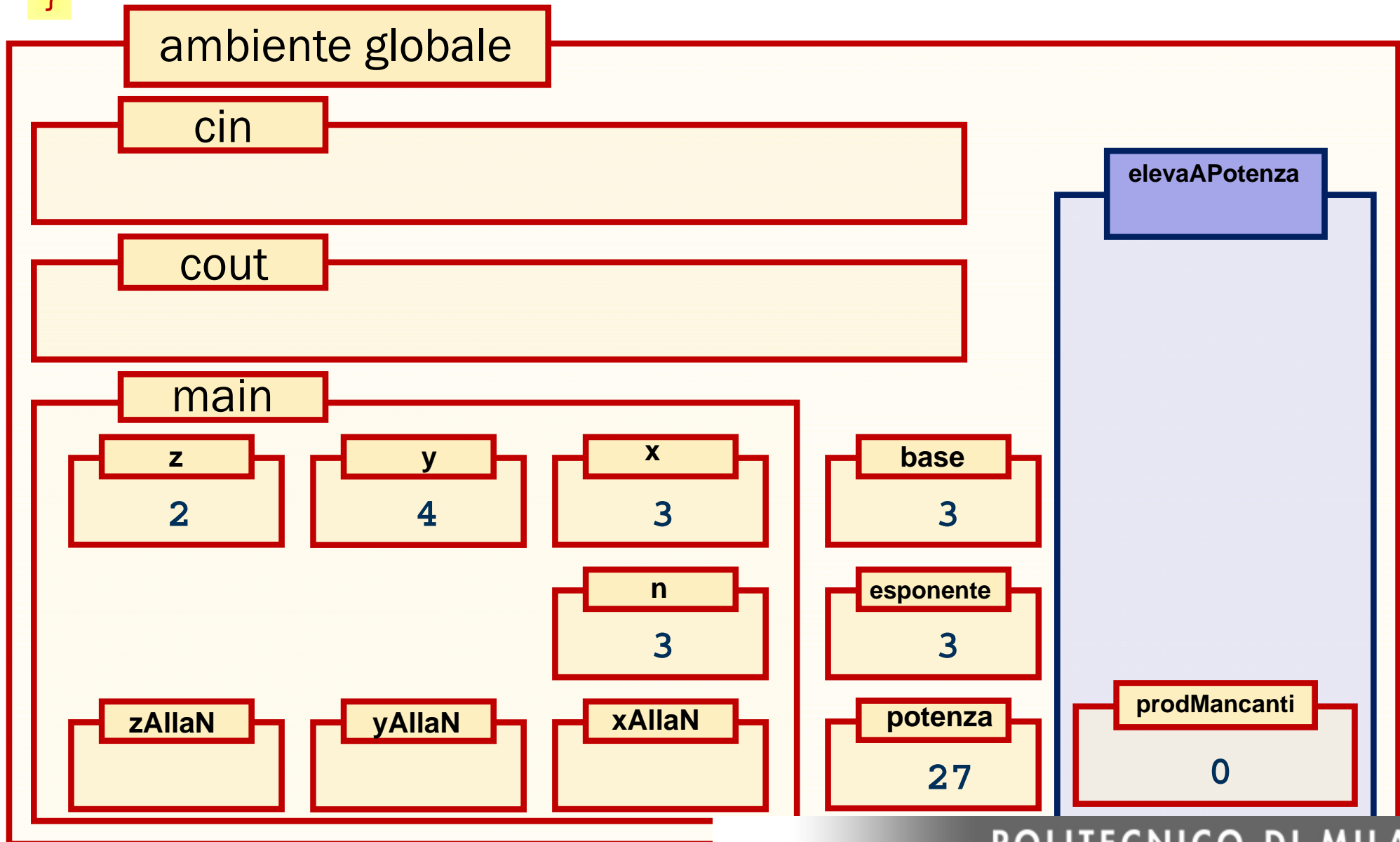


```
potenza = 1;
```

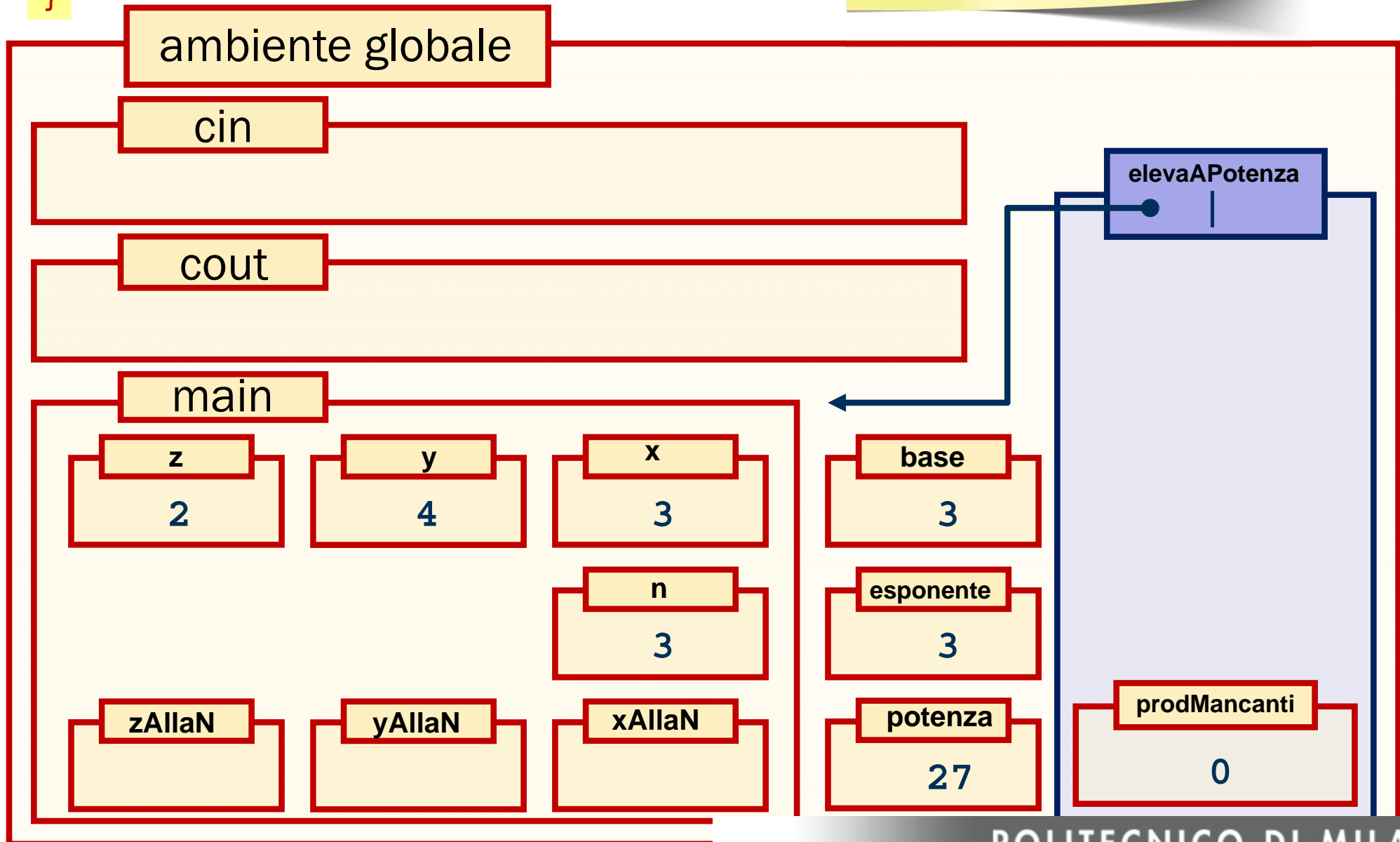
```
for (prodMancanti = esponente; prodMancanti >  
    potenza *= base;
```

```
}
```

FUNZIONE CHIAMANTE



```
potenza = 1;  
for (prodMancanti = esponente; prodMancanti >  
    potenza *= base;  
}
```

INFORMAZIONI DI
RITORNO

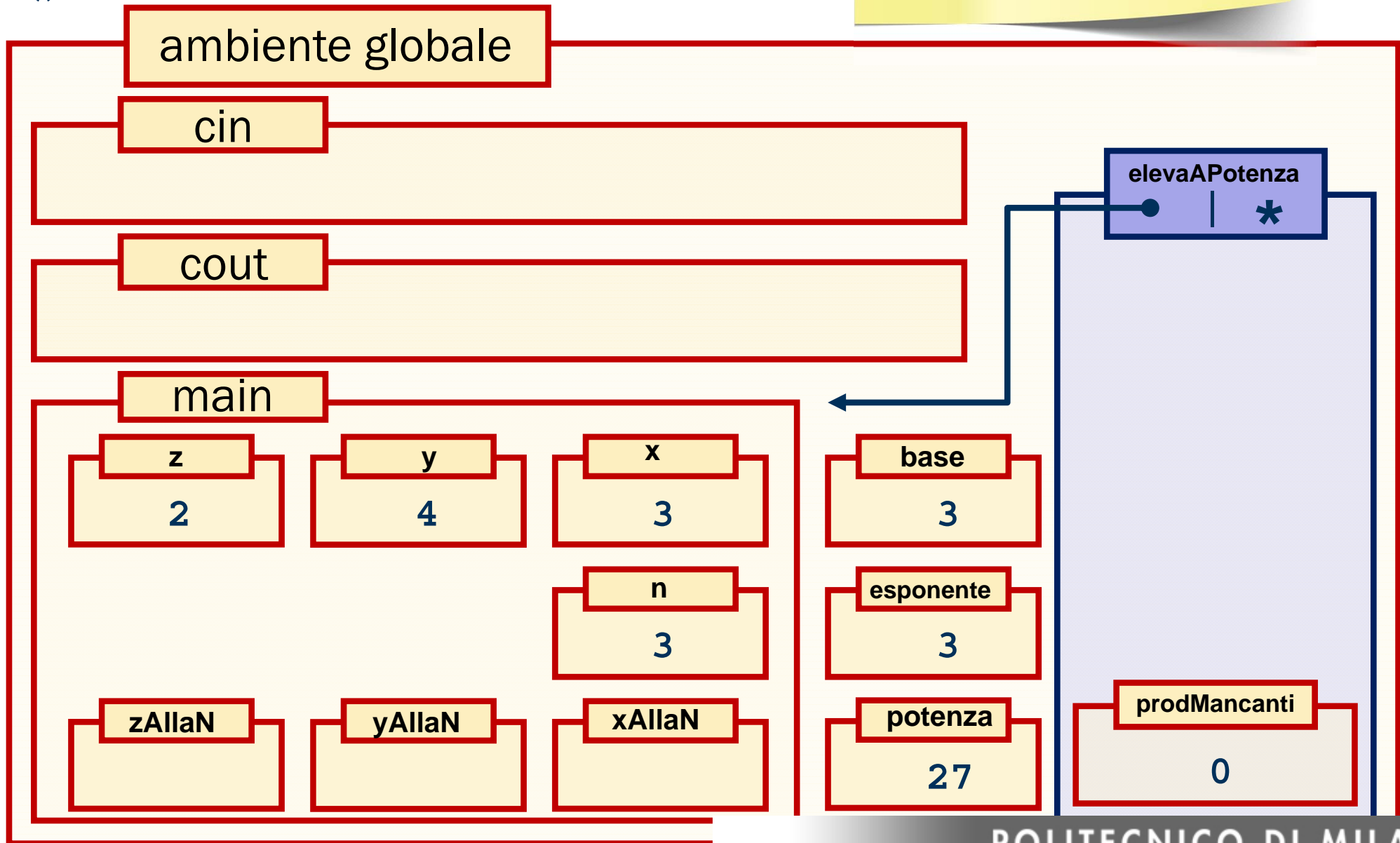
```
//calcola x elevato a n, con risultato in xAllaN
```

```
base = x; esponente = n;
```

```
elevaAPotenza();
```

```
*xAllaN = potenza
```

INFORMAZIONI DI
RITORNO

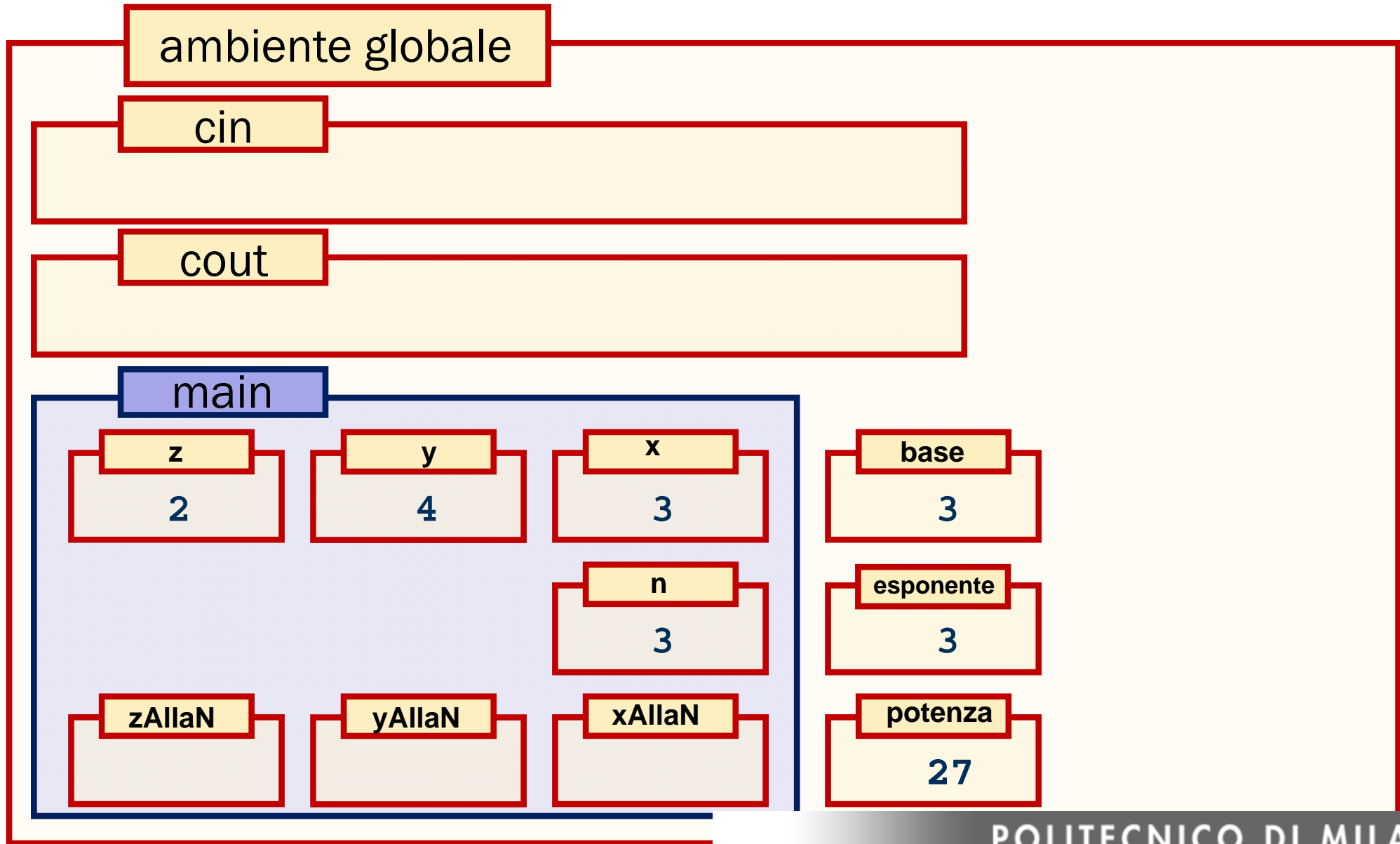


```
//calcola x elevato a n, con risultato in xAllaN
```

```
base = x; esponente = n;
```

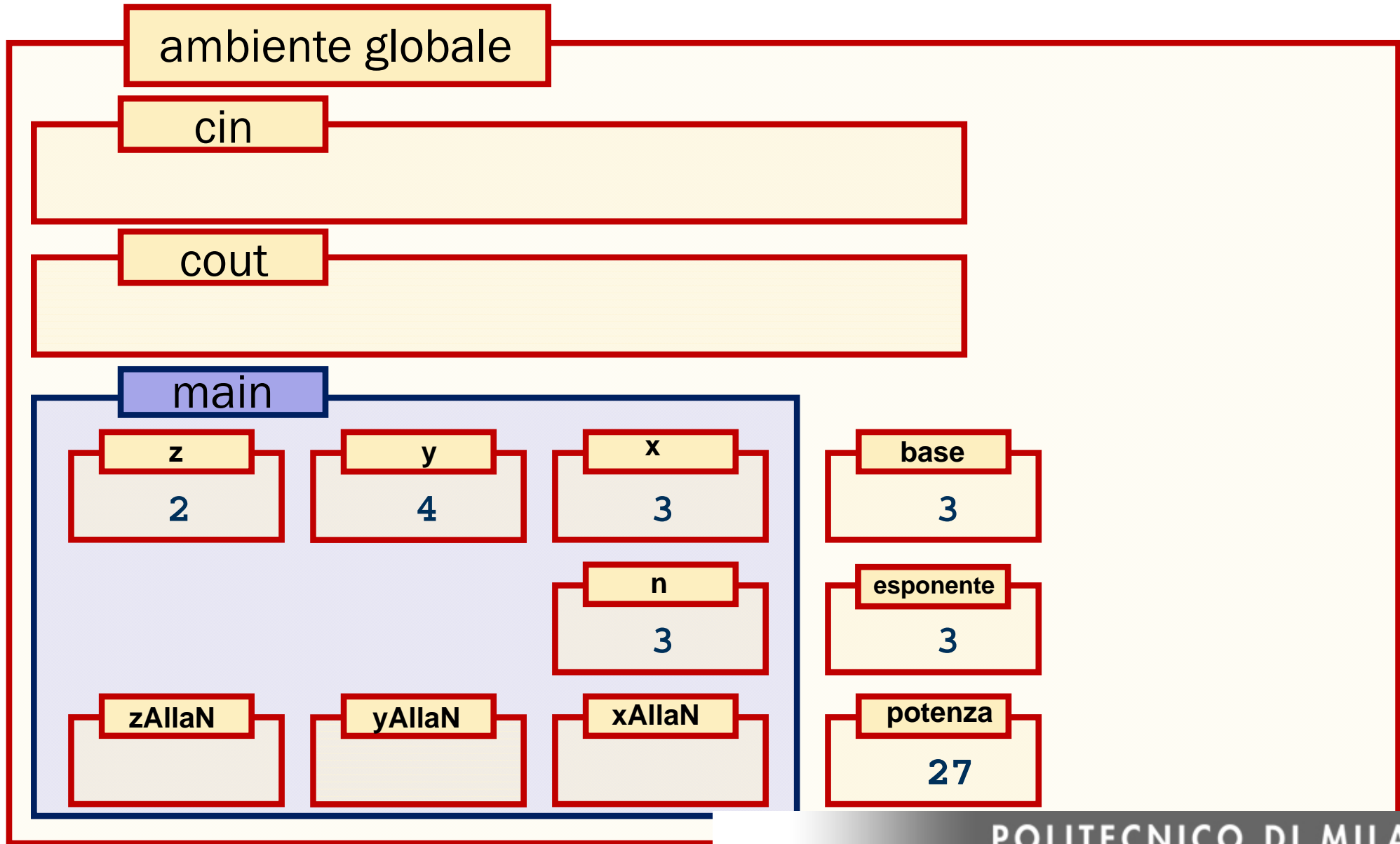
```
elevaAPotenza();
```

```
xAllaN = potenza
```



```
//calcola x elevato a n, con risultato in xAllaN  
base = x; esponente = n;  
elevaAPotenza();  
xAllaN = potenza
```

NELL'AMBIENTE DI MAIN



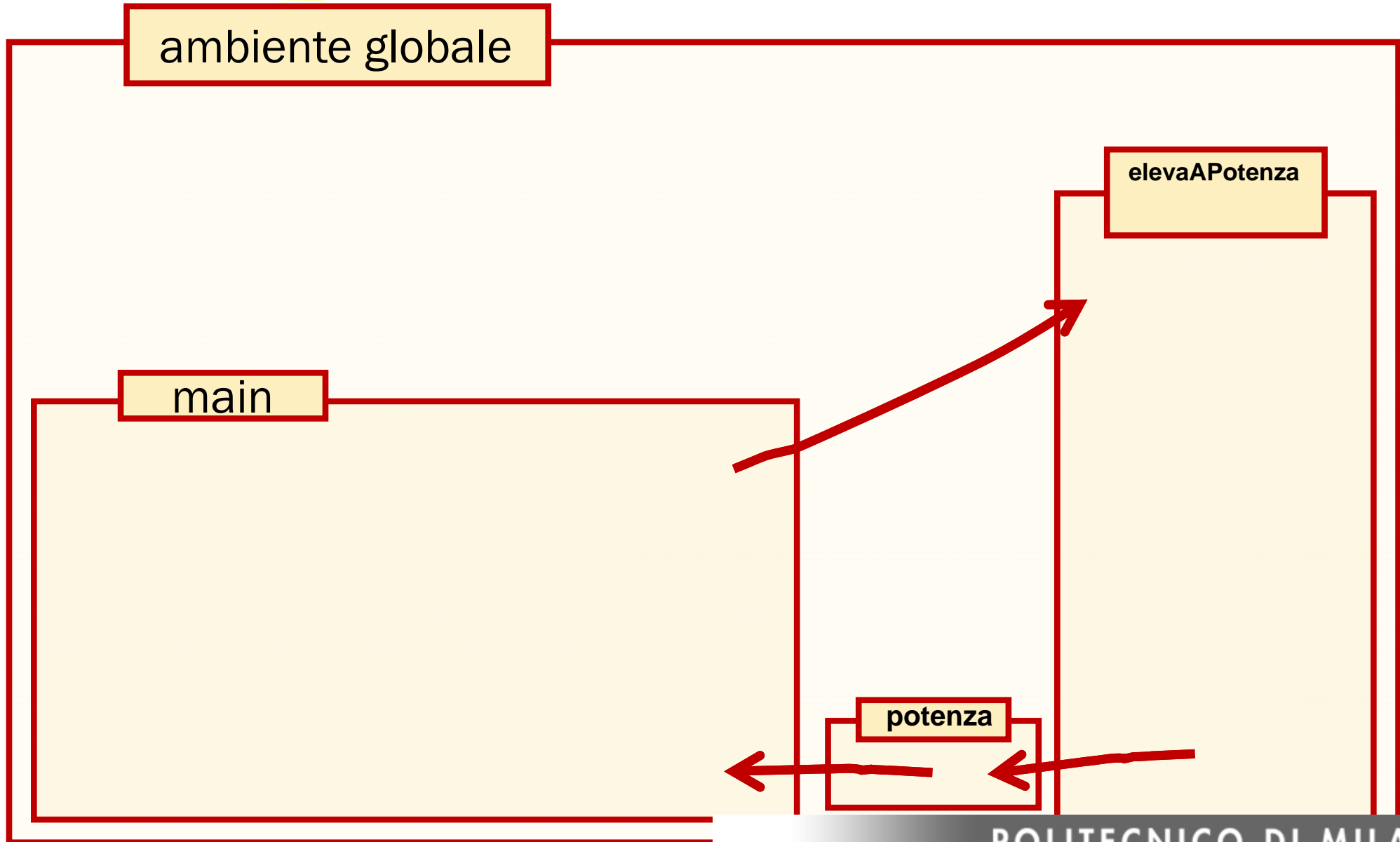
```
//calcola x elevato a n, con risultato in xAllaN
```

```
base = x; esponente = n;
```

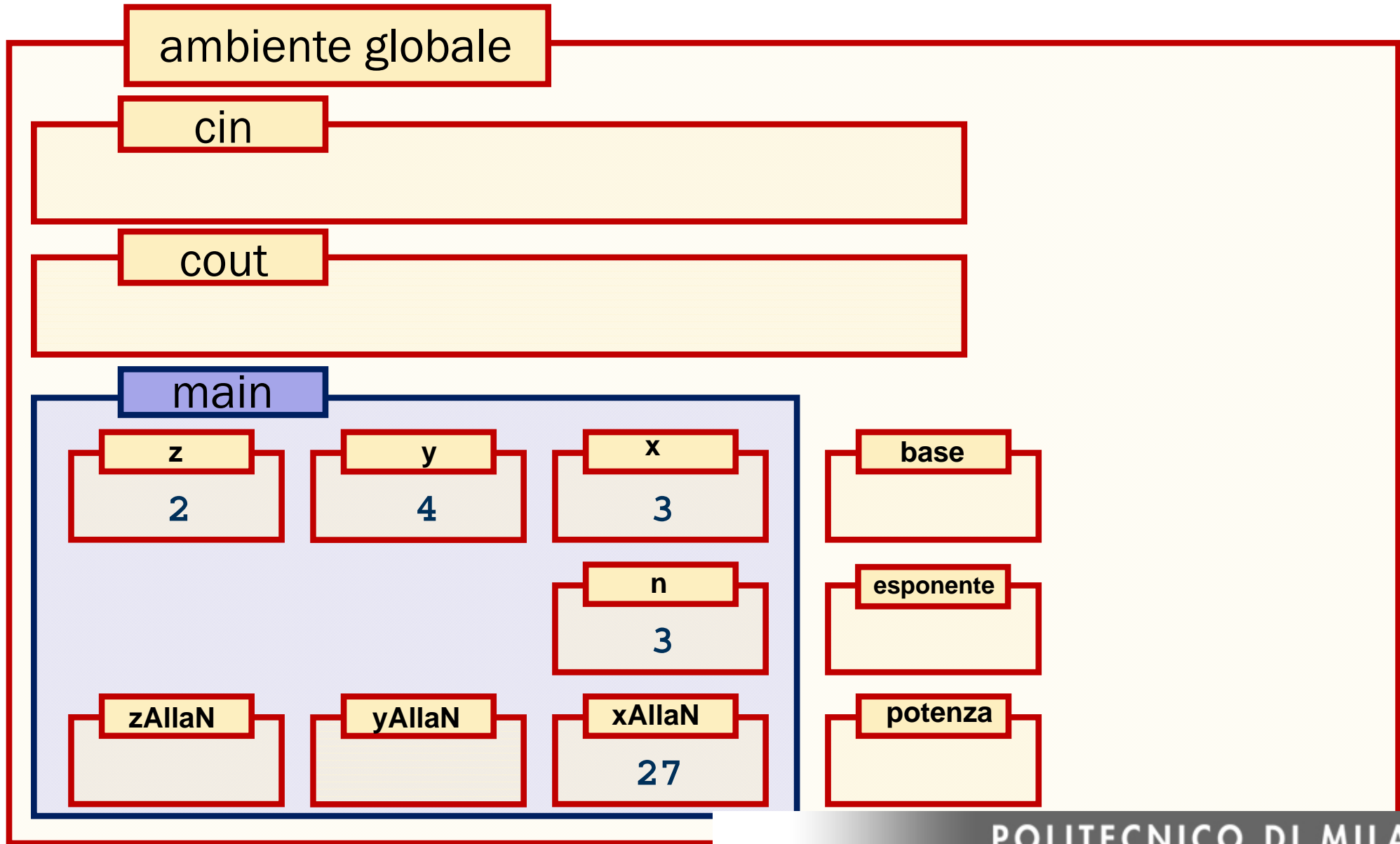
```
elevaAPotenza();
```

```
xAllaN = potenza
```

INTERFACCIA




```
//calcola y elevato a n, con risultato in yAllaN  
base = y; esponente = n;  
elevaAPotenza();  
yAllaN = potenza
```

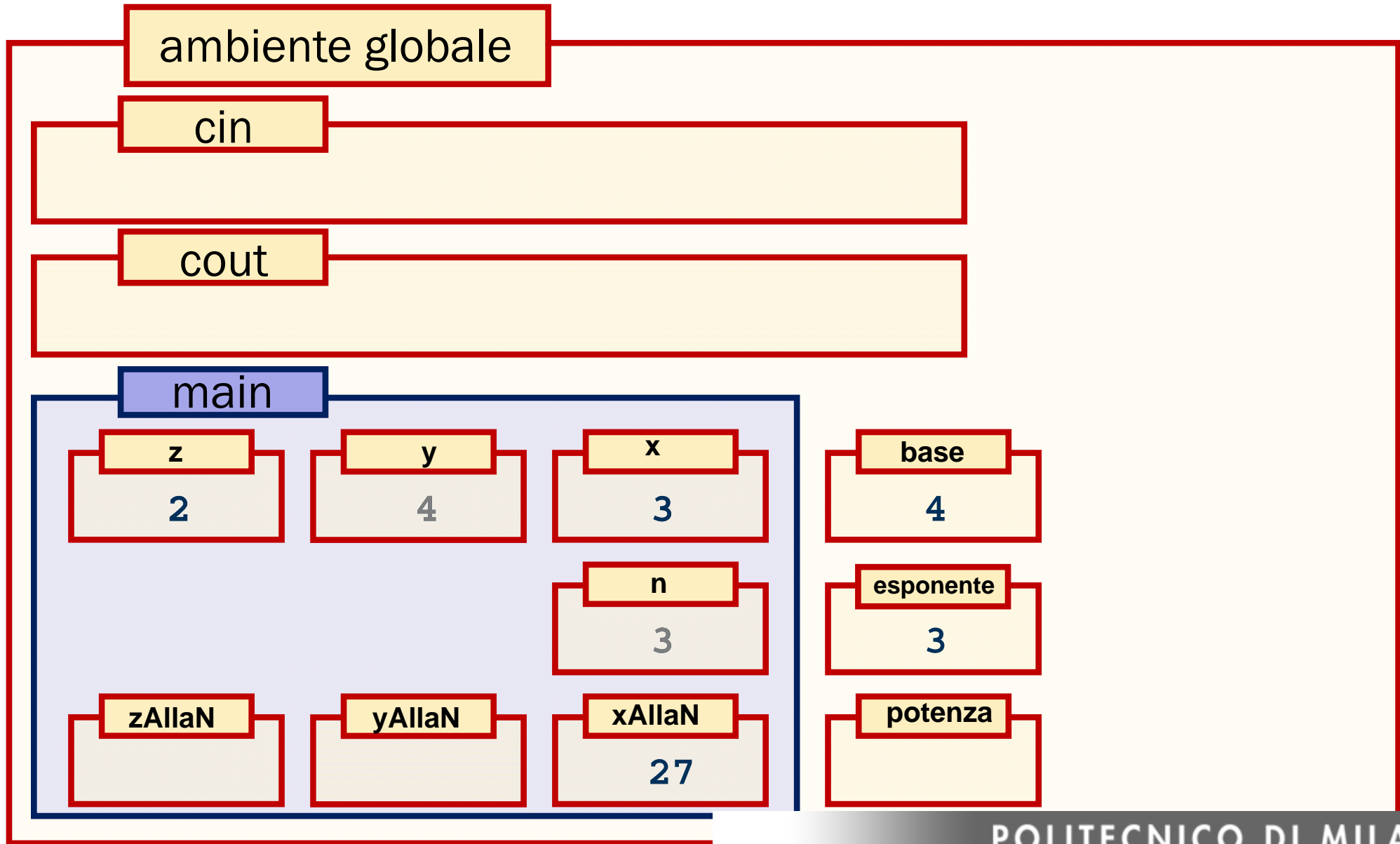


```
//calcola y elevato a n, con risultato in yAllaN
```

```
base = y; esponente = n;
```

```
elevaAPotenza();
```

```
yAllaN = potenza
```

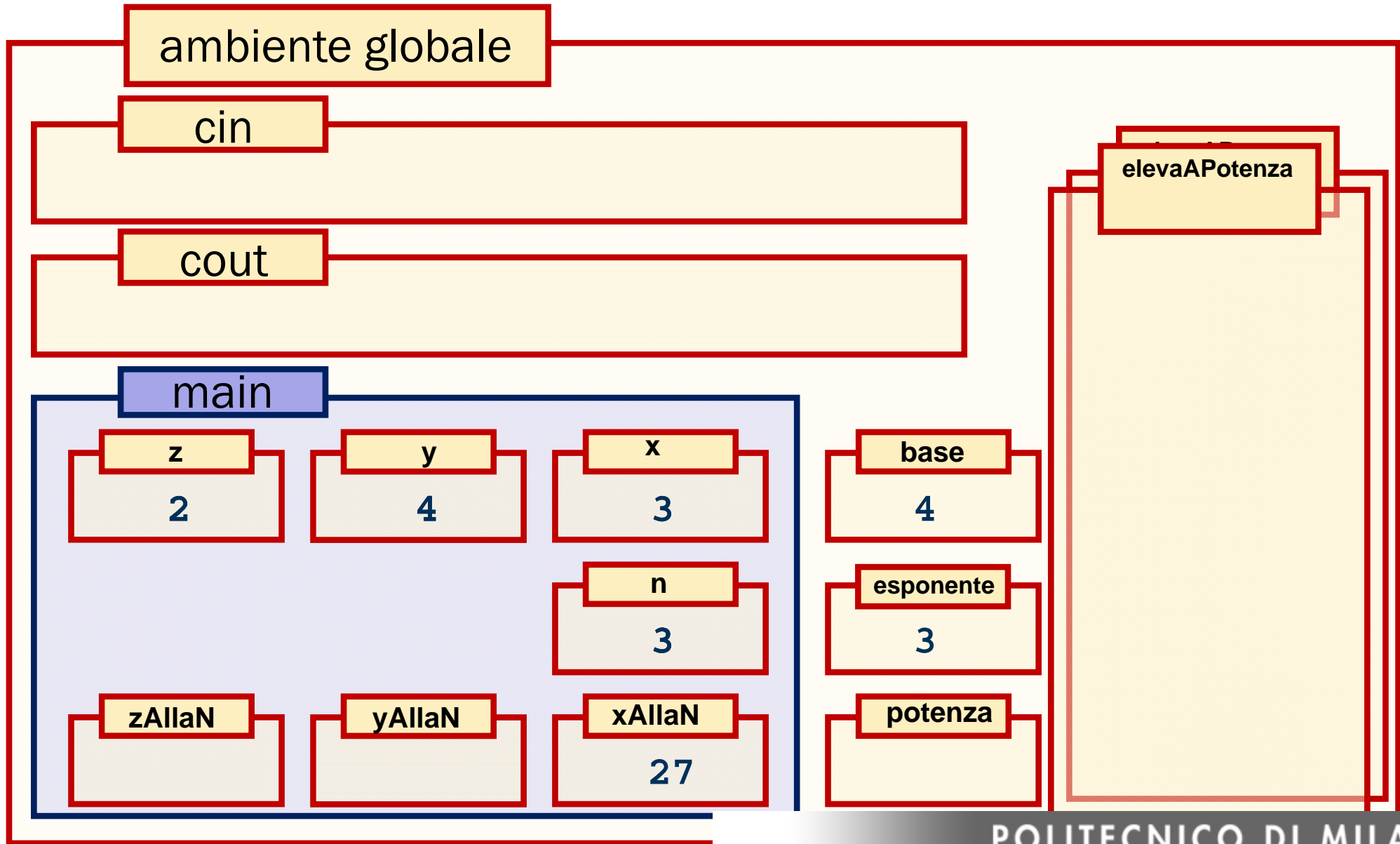


```
//calcola y elevato a n, con risultato in yAllaN
```

```
base = y; esponente = n;
```

```
elevaAPotenza();
```

```
yAllaN = potenza
```



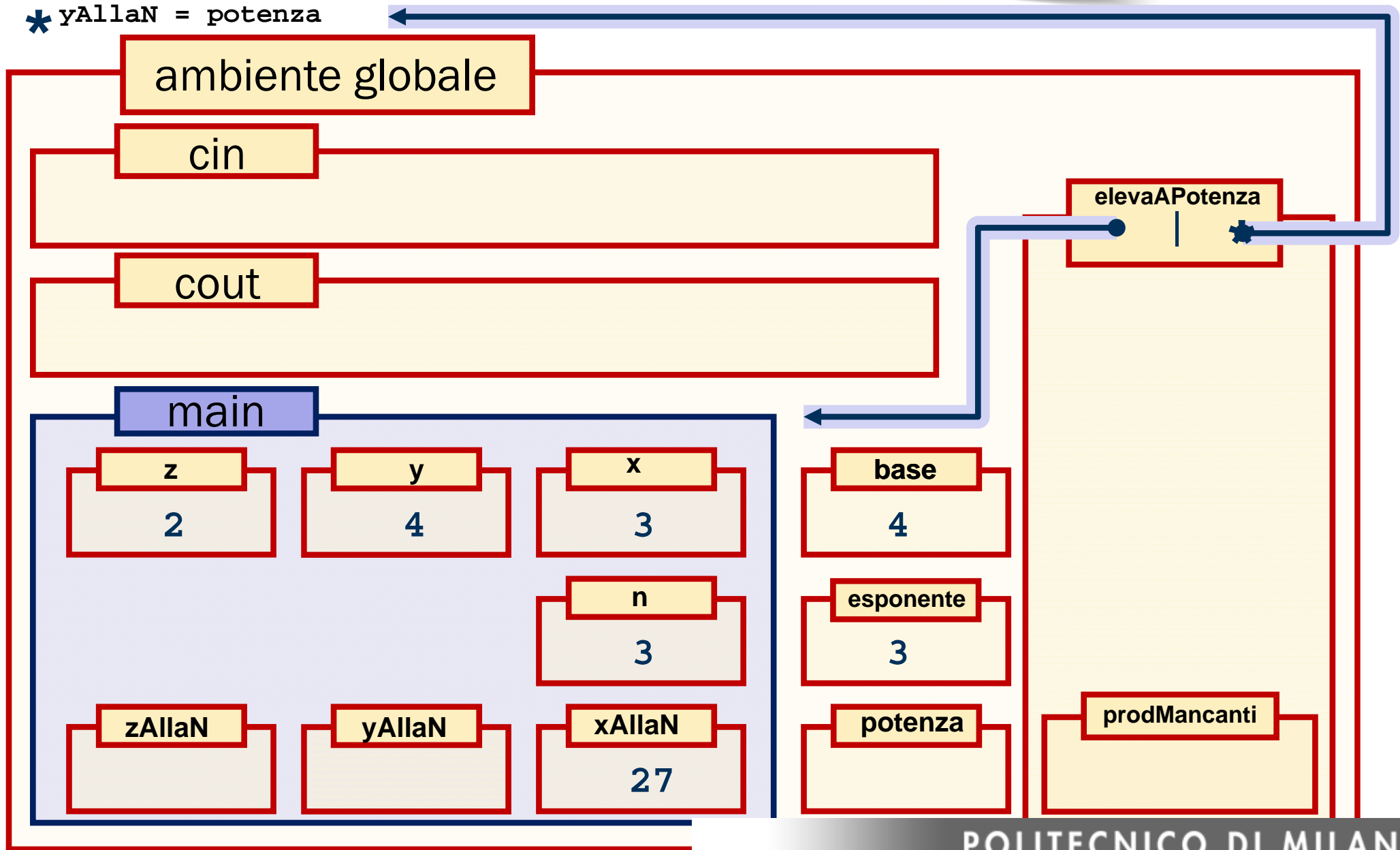
```
//calcola y elevato a n, con risultato in yAllaN
```

```
base = y; esponente = n;
```

```
elevaAPotenza();
```

```
*yAllaN = potenza
```

PUNTATORE

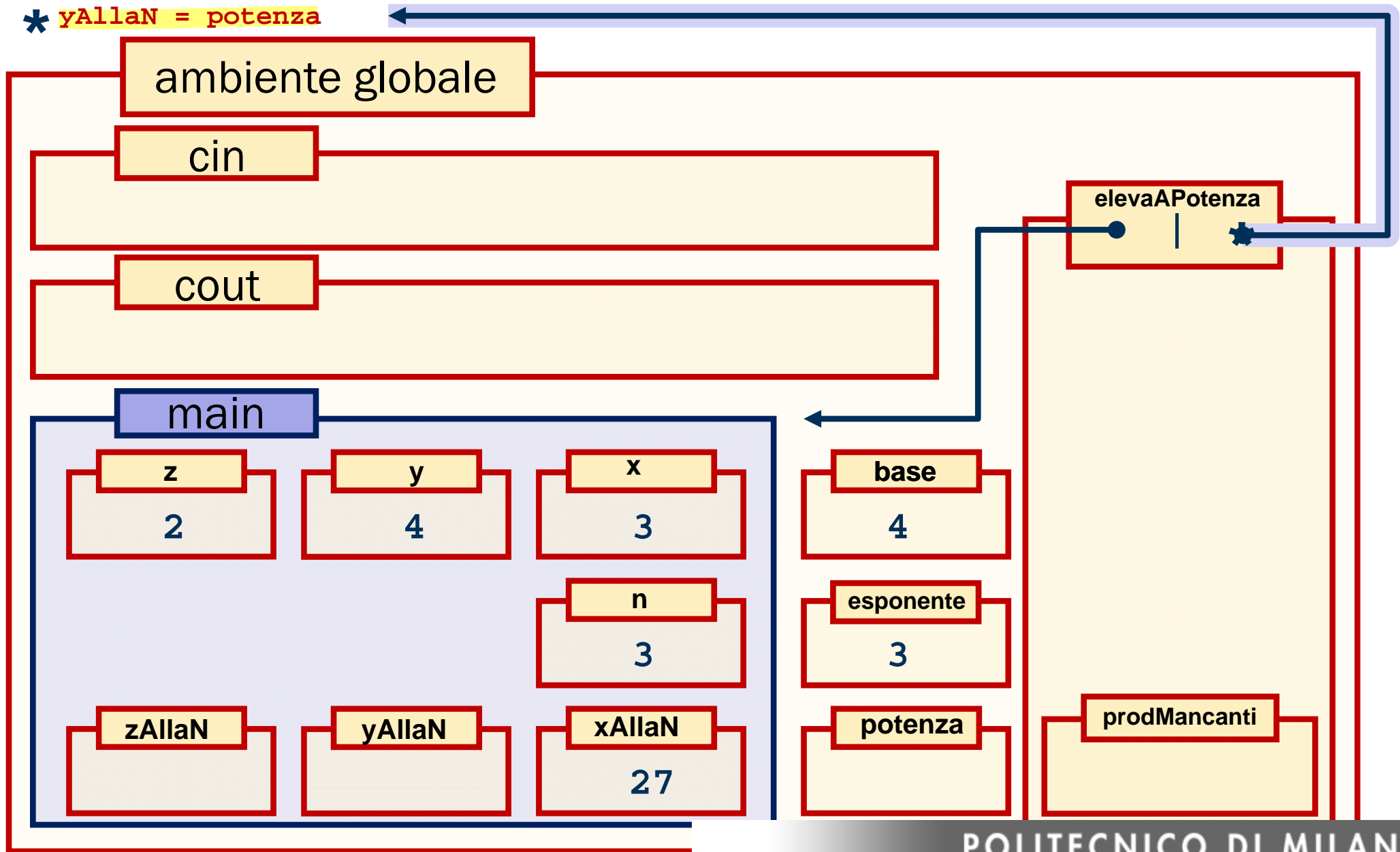


```
//calcola y elevato a n, con risultato in yAllaN
```

```
base = y; esponente = n;
```

```
elevaAPotenza();
```

```
* yAllaN = potenza
```

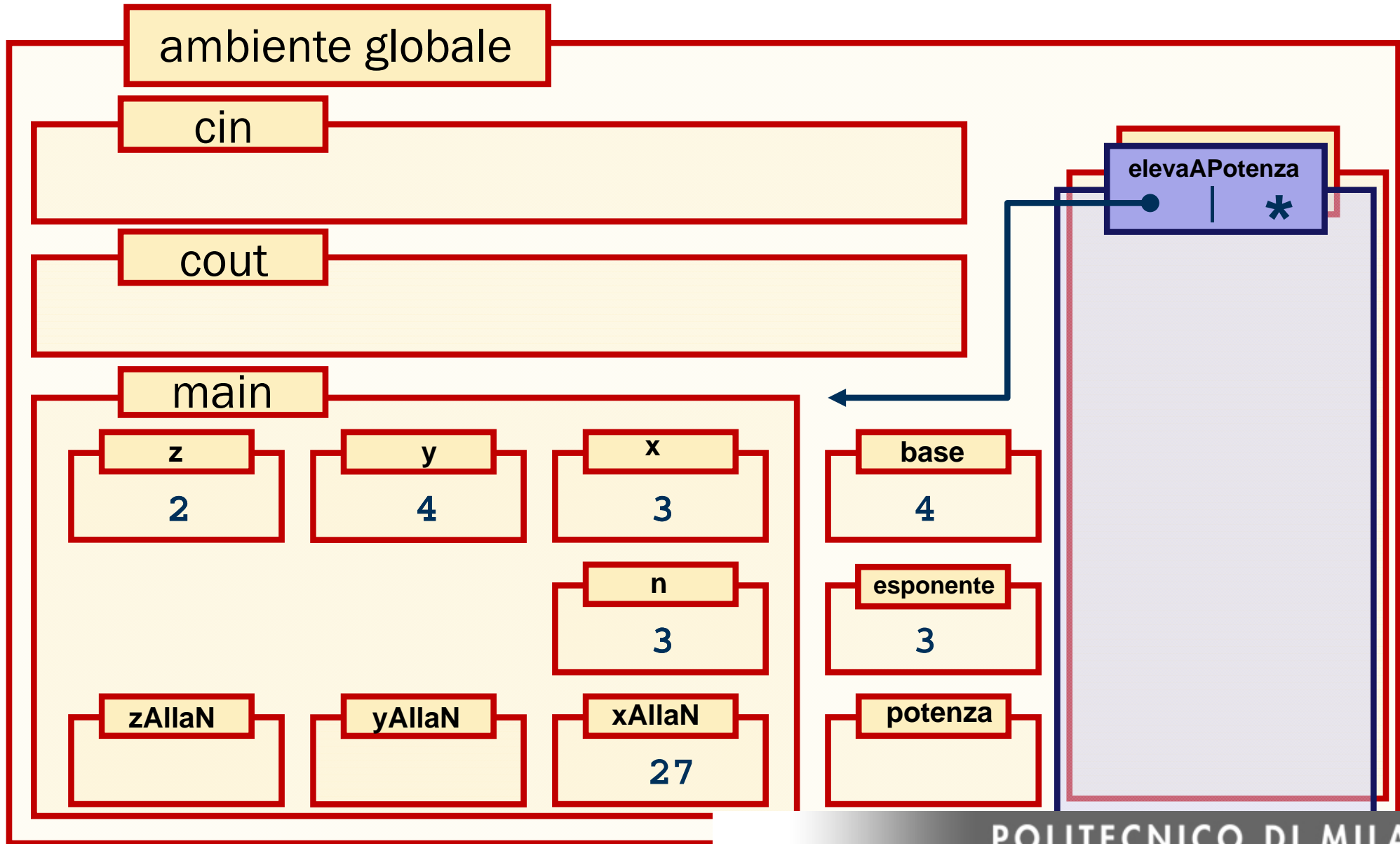


```
void elevaAPotenza()
```

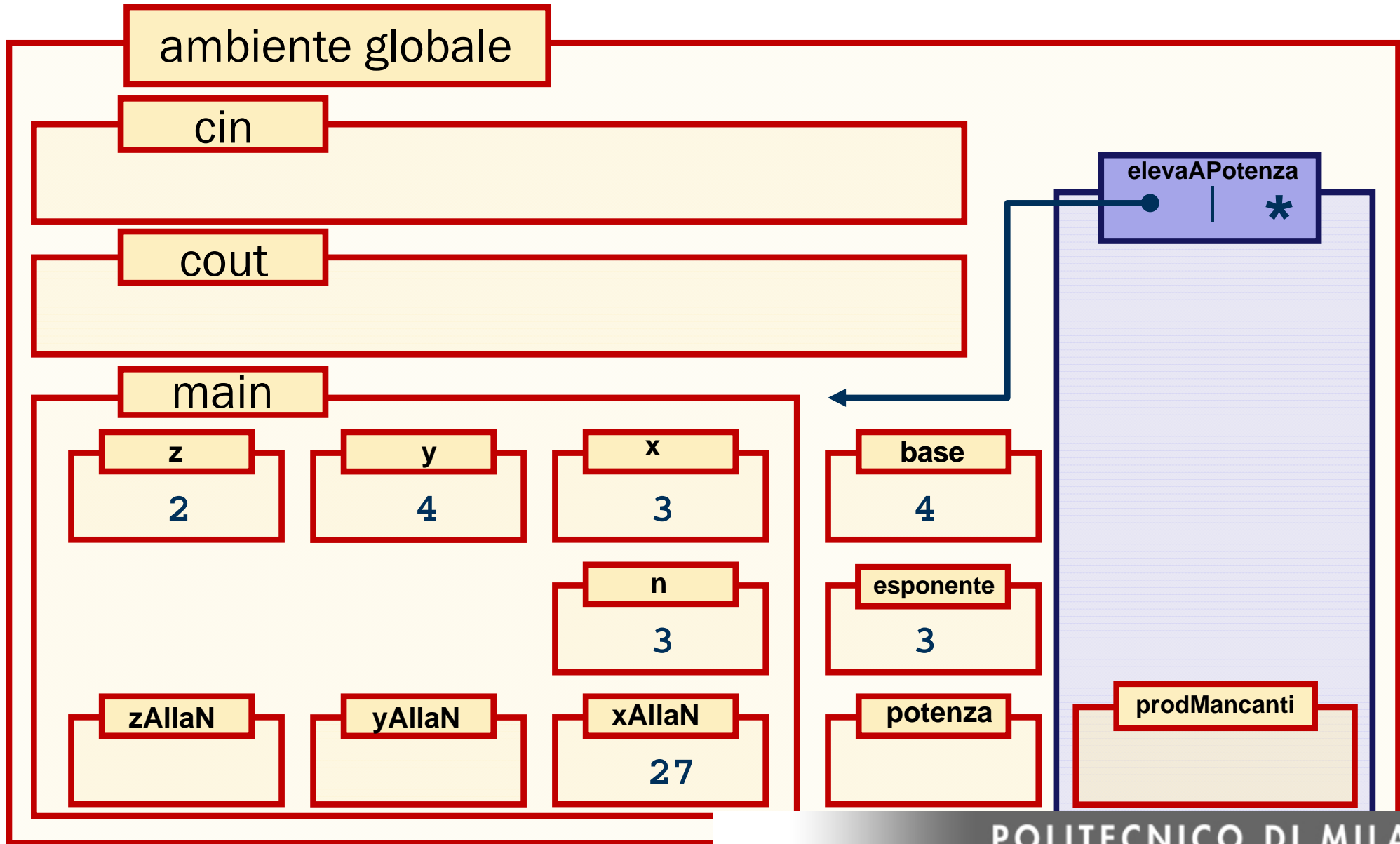
```
{ //versione con esponente positivo
```

```
  int prodMancanti;      // variabile locale ad elevaAPotenza
```

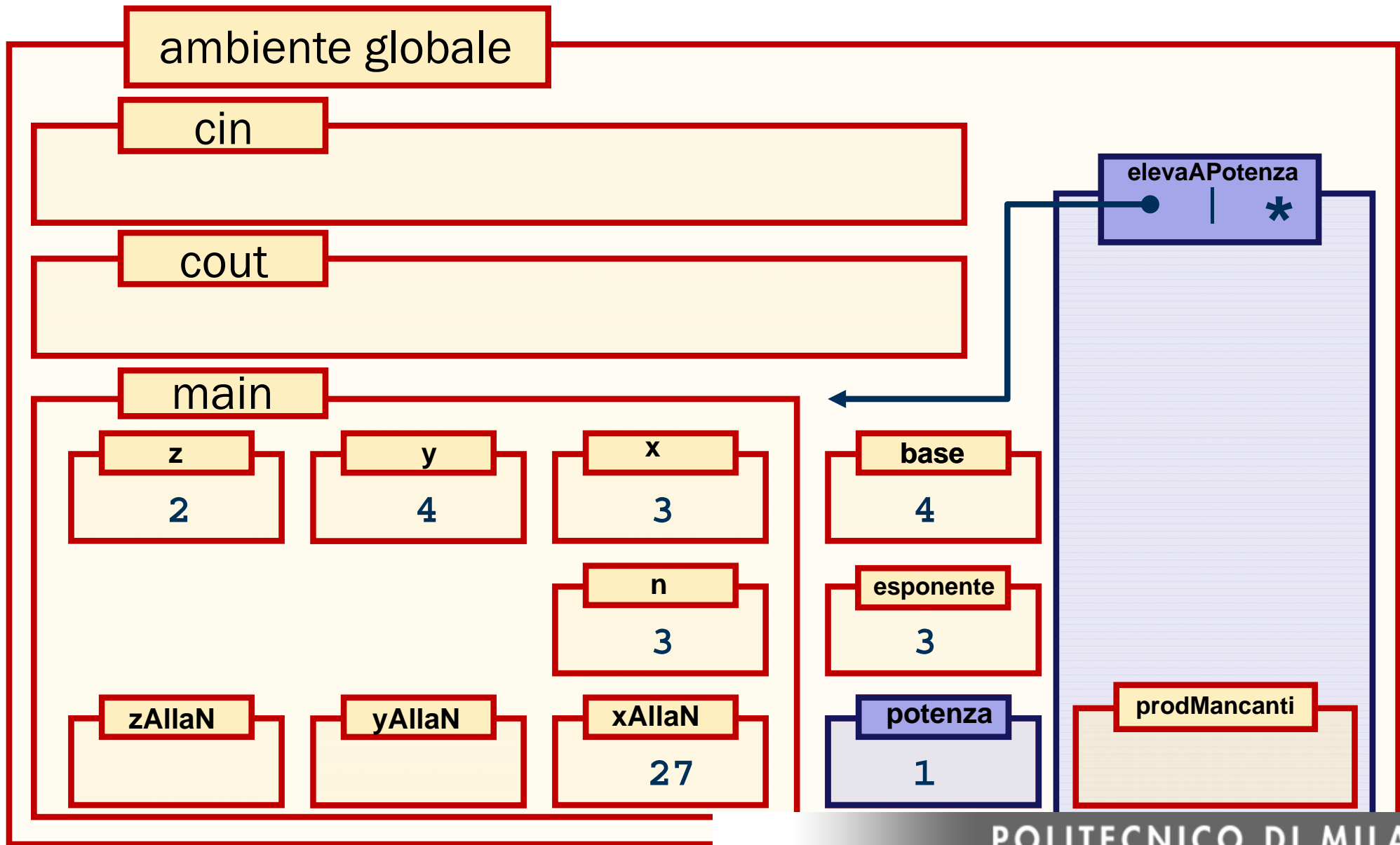
```
  potenza = 1;
```




```
void elevaAPotenza()  
{ //versione con esponente positivo  
  int prodMancanti;    // variabile locale ad elevaAPotenza  
  potenza = 1;
```

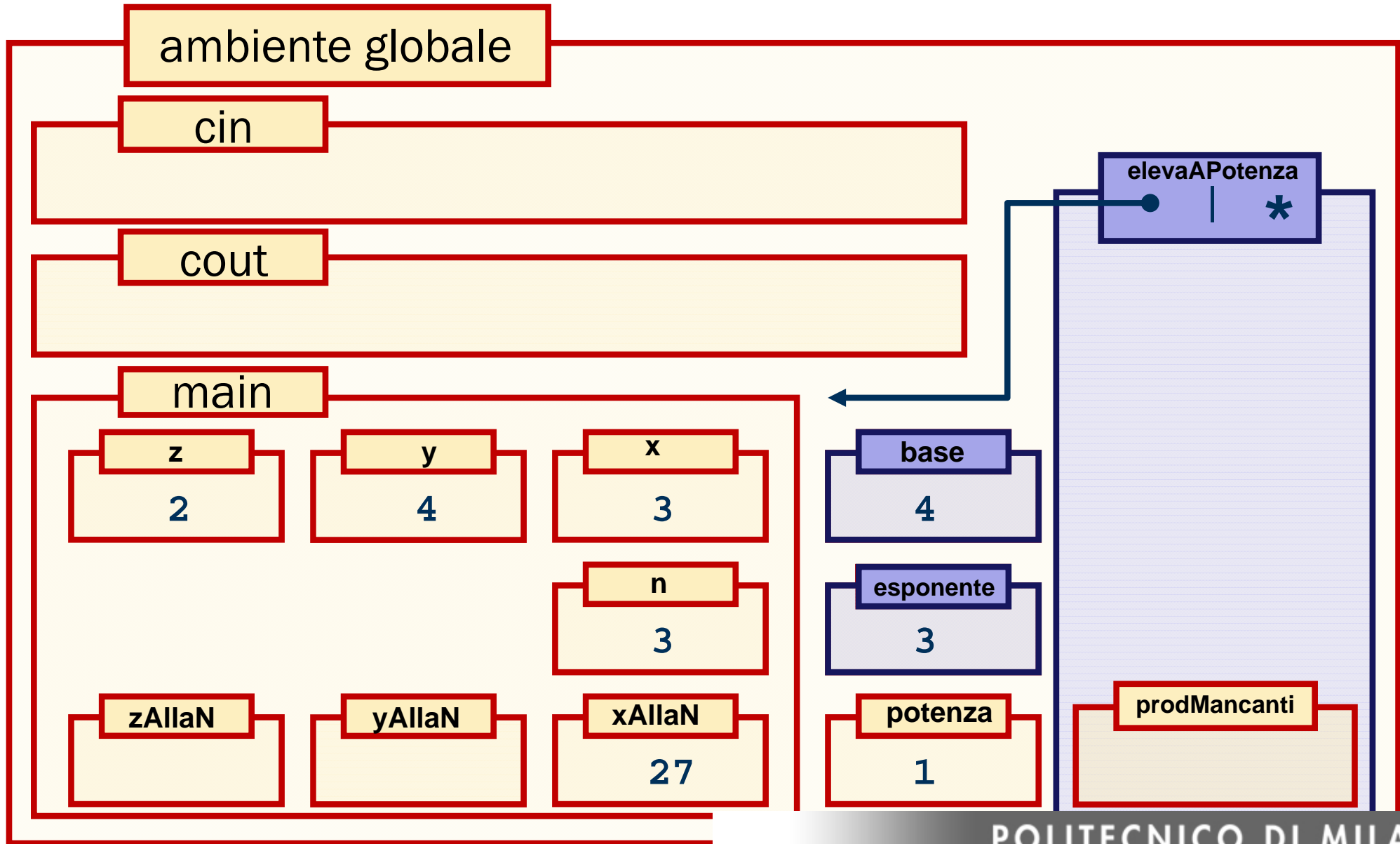



```
{ //versione con esponente positivo
  int prodMancanti;      // variabile locale ad elevaAPotenza
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```



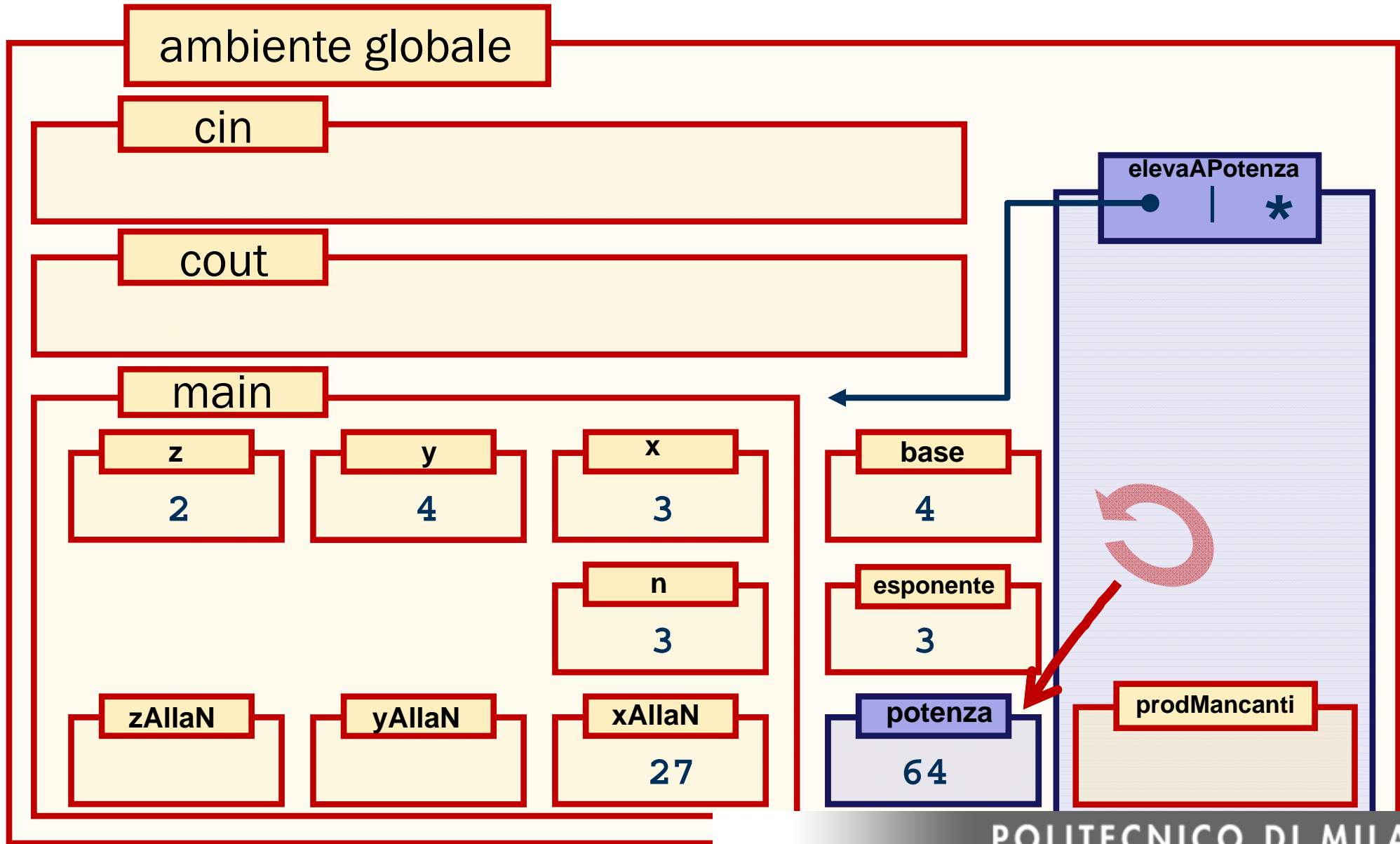
```
int prodMancanti;    // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```

```
    potenza *= base;
```



```
int prodMancanti;    // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```

```
    potenza *= base;
```



```
int prodMancanti;    // variabile locale ad elevaAPotenza
```

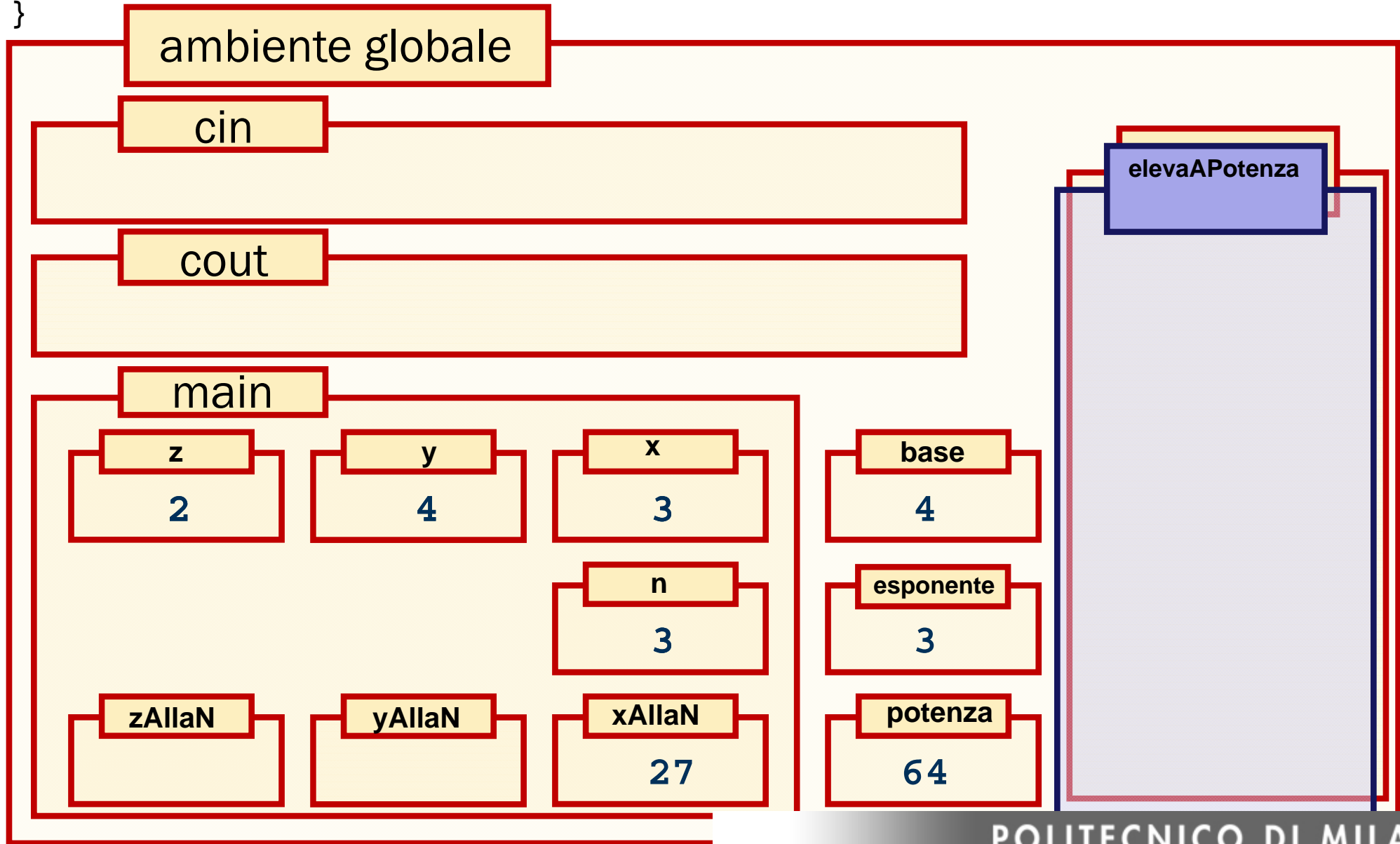
```
potenza = 1;
```

```
for (prodMancanti = esponente; prodMancanti >
```

```
    potenza *= base;
```

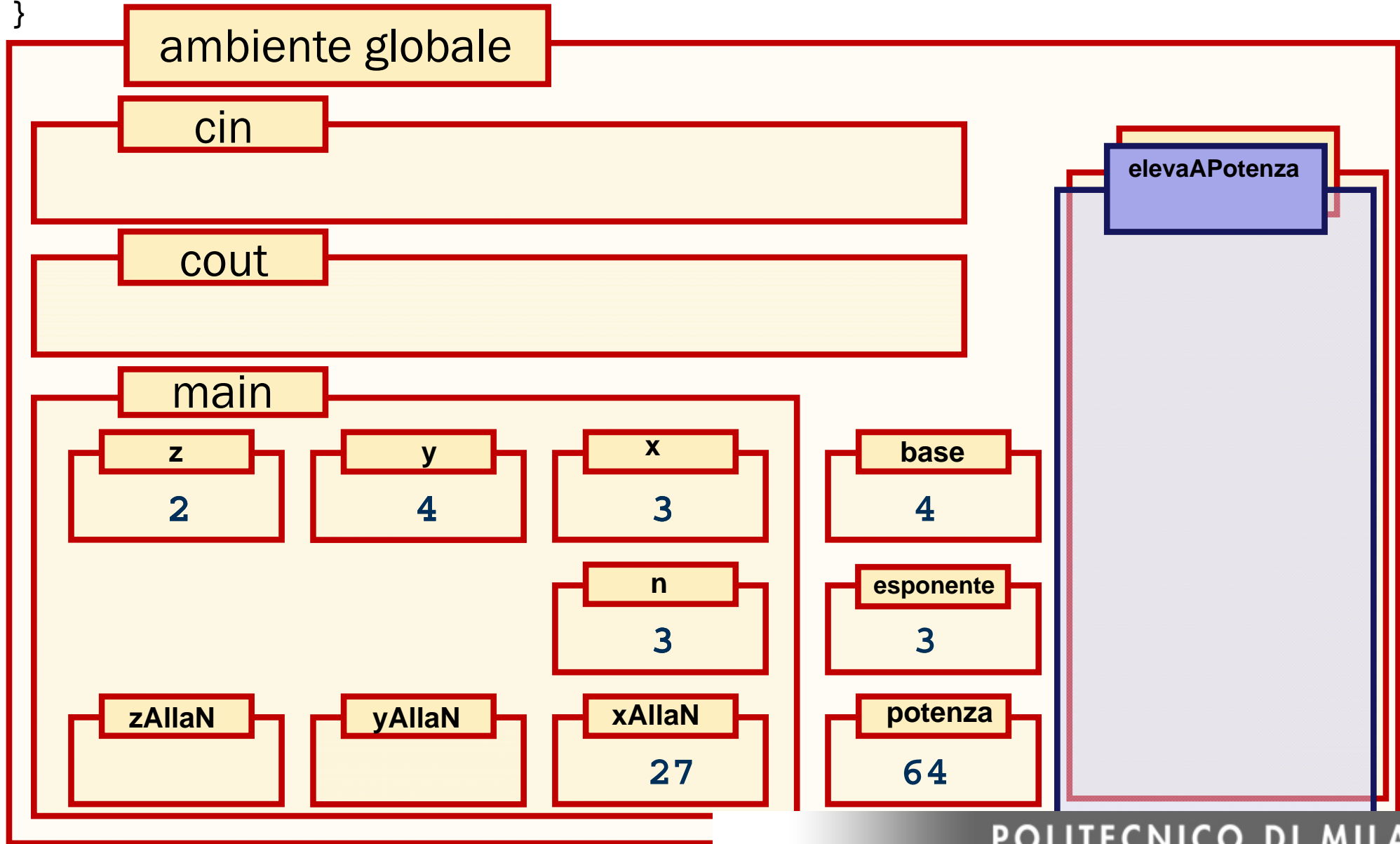
```
}
```

SECONDA COPIA

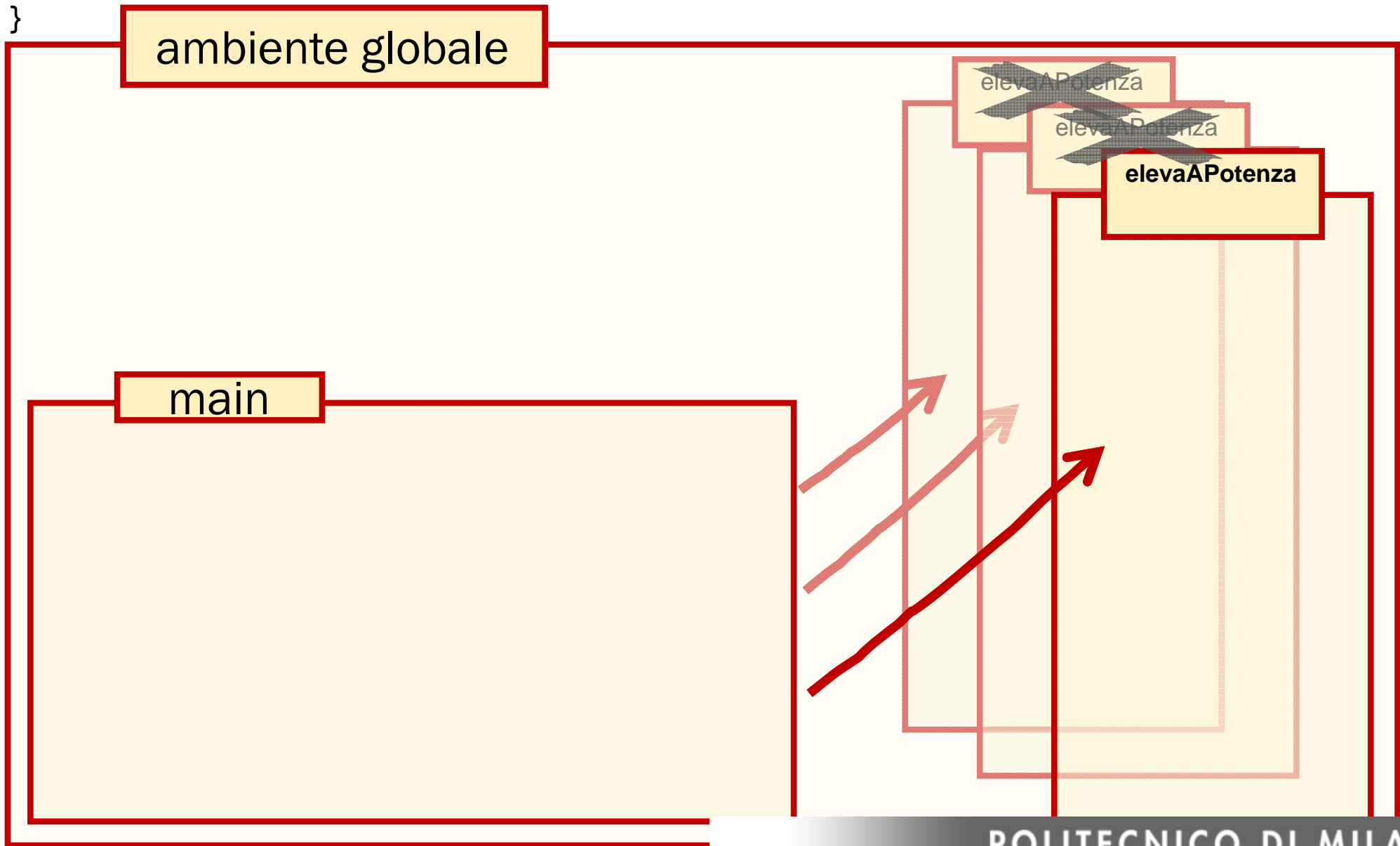


```
int prodMancanti;    // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 1; prodMancanti /= 2)
    potenza *= base;
}
```

FUNZIONI RICORSIVE




```
int prodMancanti;    // variabile locale ad elevaAPotenza
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
    potenza *= base;
}
```

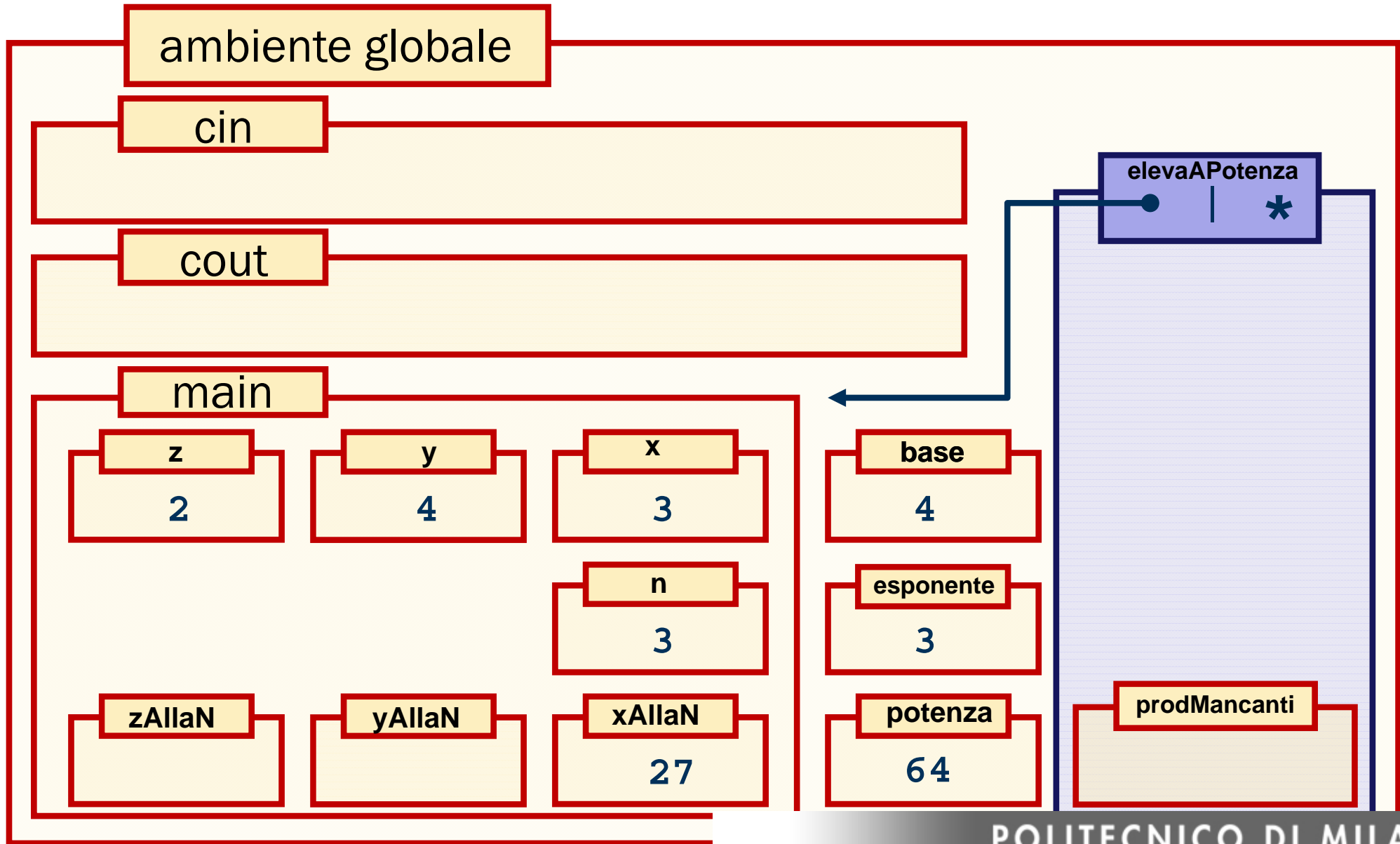


```
potenza = 1;
```

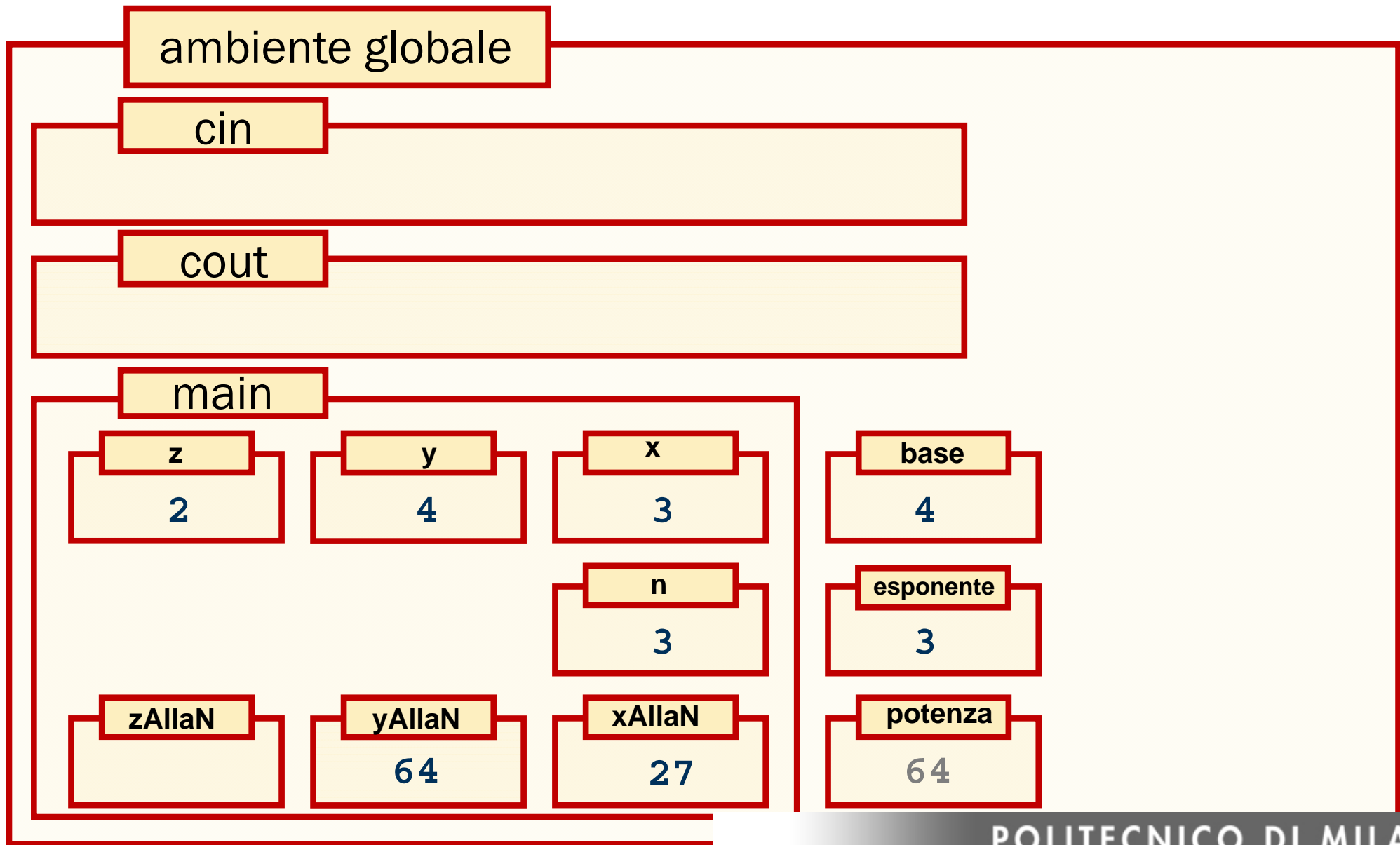
```
for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
```

```
    potenza *= base;
```

```
}
```




```
base = y; esponente = n;  
elevaAPotenza();  
yAllaN = potenza;  
//calcola z elevato a n, con risultato in zAllaN
```



```
//calcola z elevato a n, con risultato in zAllaN  
base = z; esponente = n;  
elevaAPotenza();  
zAllaN = potenza;
```

