POLITECNICO DI MILANO

INFORMATICA

Funzioni ricorsive

Esponente positivo

$$base^{esponente}$$

$$1 * \underbrace{base * base * base * \ldots * base}_{esponente \text{ volte}}$$

specifica del problema

$base^{esponente}$

se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$

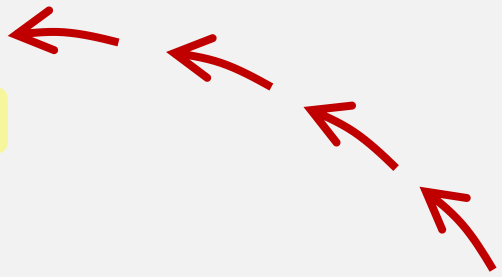
$base^{esponente}$

se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$



$base^{esponente}$

se esponente = 1

$base^{esponente}$ = base

se esponente > 1

$base^{esponente}$ = base * $base^{esponente-1}$

b * b * b * be-3

$base^{esponente}$

se $esponente = 1$

$base^{esponente} = base$

se $esponente > 1$

$base^{esponente} = base * base^{esponente-1}$

b * b * b * b*...* b

$base^{esponente}$

se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$

b * b * b * b*...* b

$base^{esponente}$

```
int prodMancanti;// variabile locale alla funzione elevaAPotenza
  int potenza;              // variabile necessaria per il calcolo
                            // del valore di ritorno
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
    potenza *= base;
  return potenza;
}

void main()
{ int x, y, z, n,           //dati su cui operare
```

b * b * b * b* . . . * b

$base^{esponente}$

se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$

```
int elevaAPotenza(int base, int esponente) // funzione con valore di ritorno
{ //versione per esponente positivo
    int prodMancanti;     // variabile locale alla funzione elevaAPotenza
    int potenza;          // variabile per il calcolo del valore di ritorno
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
      potenza *= base;
    return potenza;
}
```

$base^{esponente}$

se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$

```
int elevaAPotenza(int base, int esponente) // funzione con valore di ritorno
{ //versione con esponente positivo
    int prodMancanti;     // variabile locale alla funzione elevaAPotenza
    int potenza;          // variabile per il calcolo del valore di ritorno
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
      potenza *= base;
    return potenza;
}
```

Definizione ricorsiva

$base^{esponente}$

se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$

```
int elevaAPotenza(int base, int esponente) // funzione con valore di ritorno
{ //versione con esponente positivo
    int prodMancanti;     // variabile locale alla funzione elevaAPotenza
    int potenza;          // variabile per il calcolo del valore di ritorno
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
      potenza *= base;
    return potenza;
}
```

Definizione ricorsiva

Operazione di fattoriale

n!

se n = 0,1
n! = 1

se n > 1
*n! = n * (n – 1)!*

n * (n – 1) * (n – 2) * ... * 1 !

Definizione ricorsiva

Operazione di fattoriale

n!

$$\prod_{i=1}^{n} i$$

Produttoria

```
n * (n - 1) * (n - 2) * ... * 1
```

Funzioni ricorsive

Problemi formulati ricorsivamente

Funzioni ricorsive

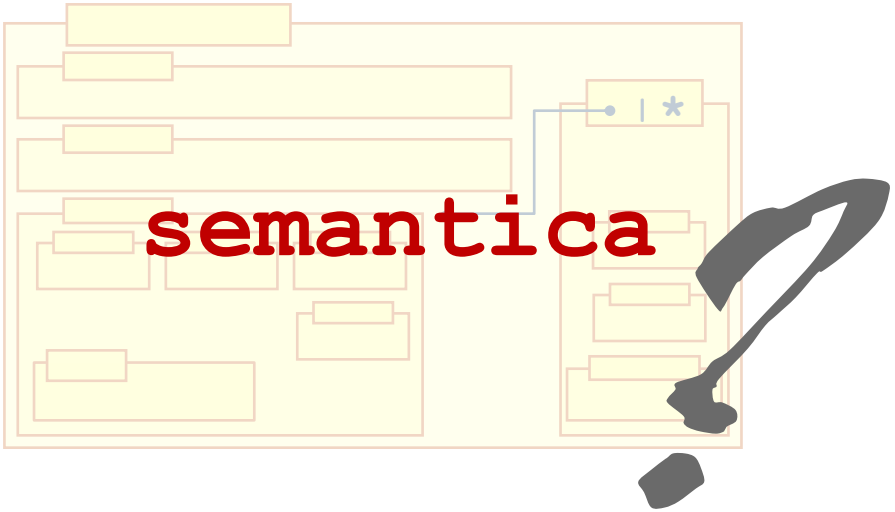
```
int elevaAPotenza(int base, int esponente)
{ //versione con esponente positivo
    int prodMancanti;
    int potenza;
    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
      potenza *= base;
    return potenza;
}
```

se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$



se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$

```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$

```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

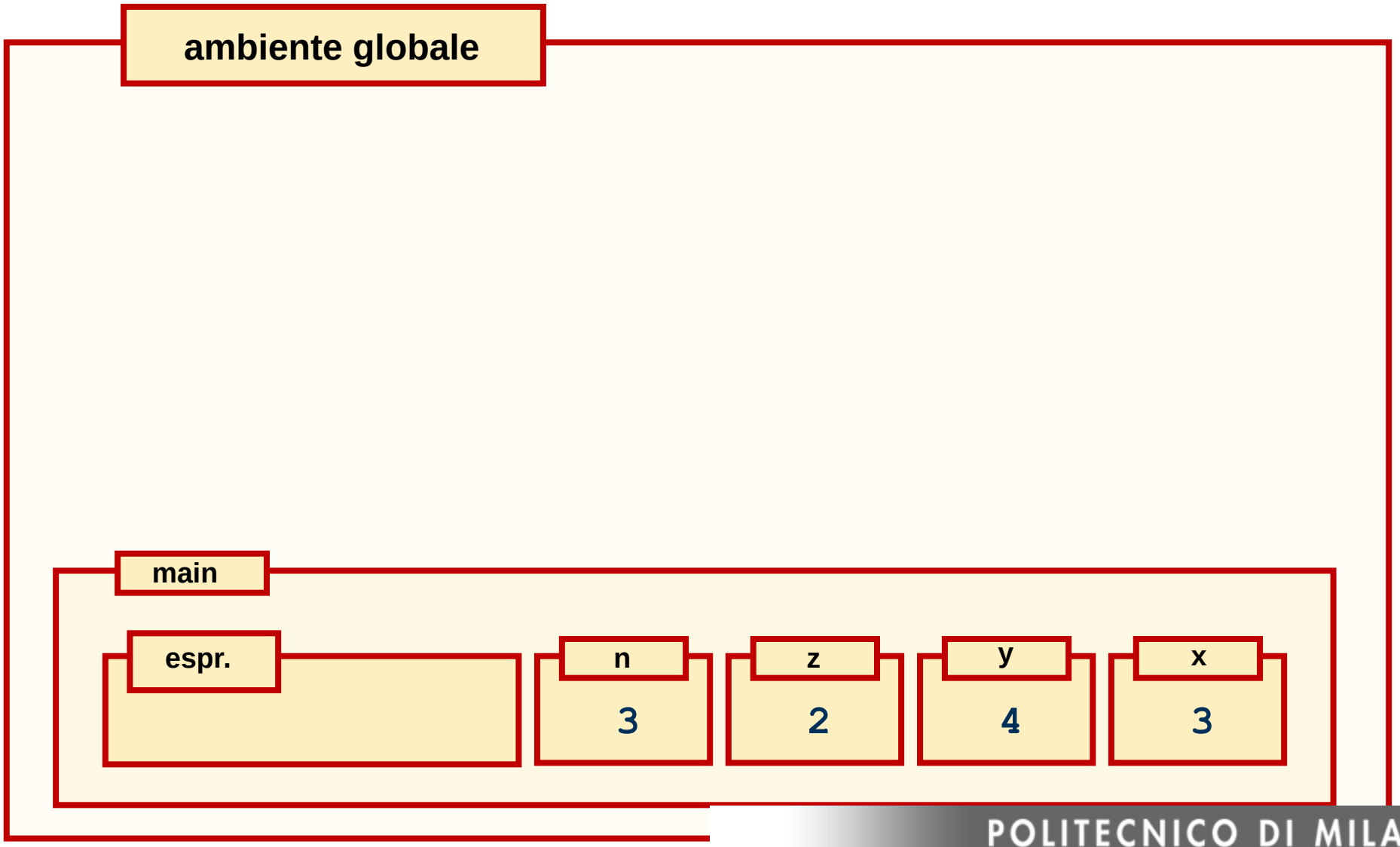
```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * La funzione viene definita ricorsivamente
 */
#include <iostream.h>

int elevaAPotenza(int base, int esponente)   // versione ricorsiva
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}

void main()
{ int x, y, z, n,                            //dati su cui operare
  " Presenta le funzionalità del programma"
  " Leggi i dati e verifica che rispondano alle specifiche"
  if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
  ...
}
```

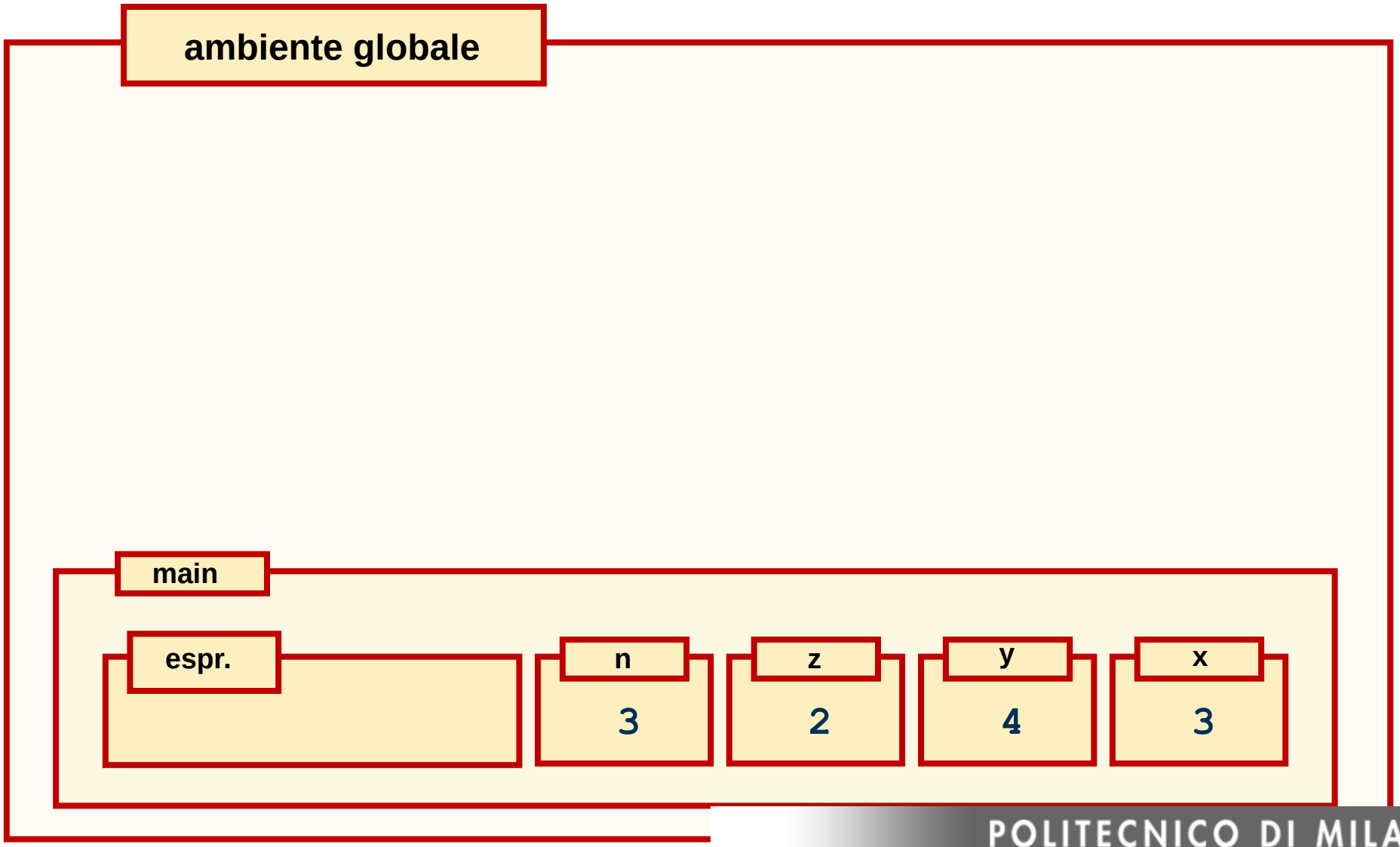
Modello ad ambiente

ambiente globale

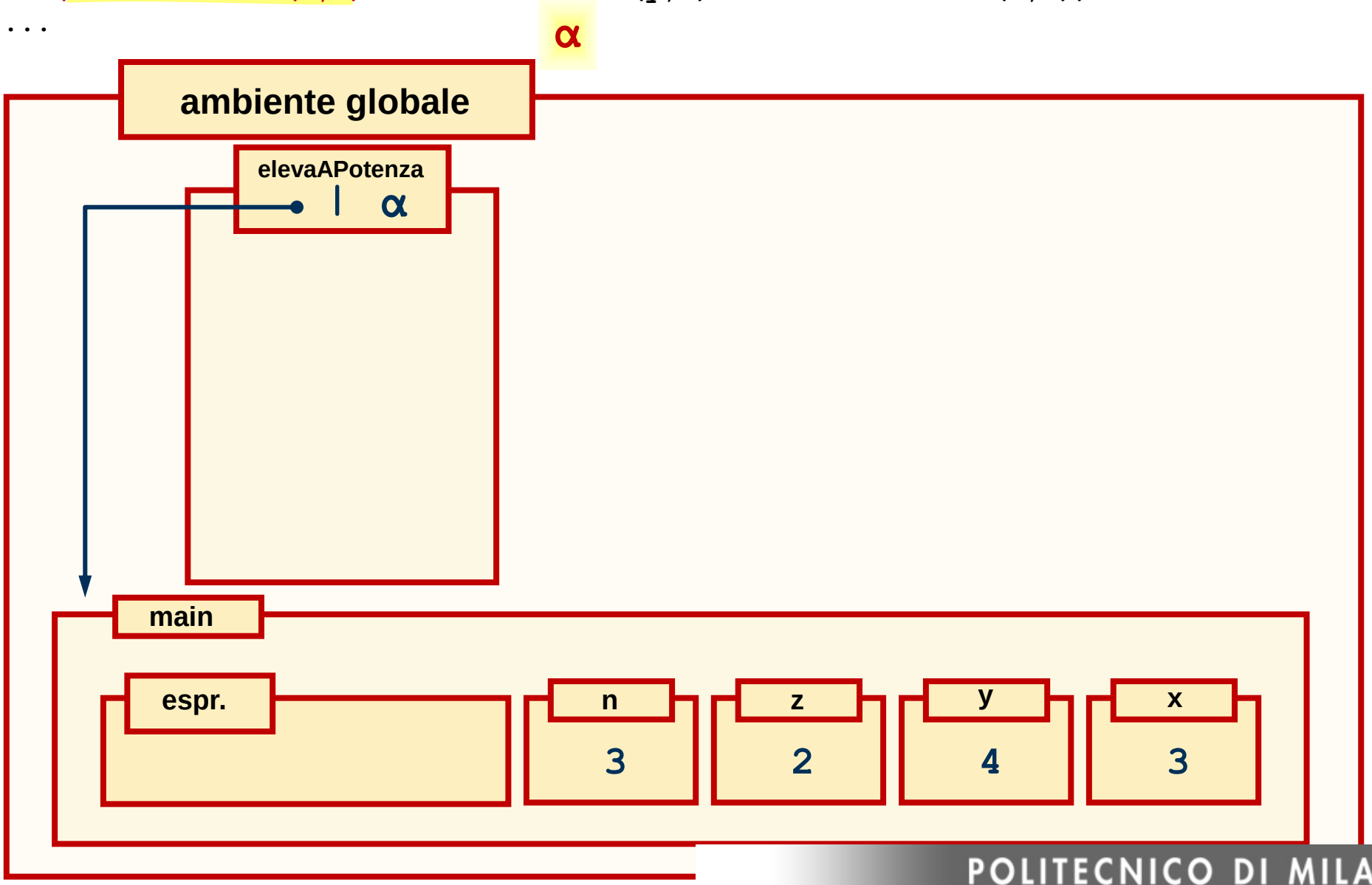
```
void main()
{ int x, y, z, n,                    // dati su cui operare
  "Presenta le funzionalità del programma"
```



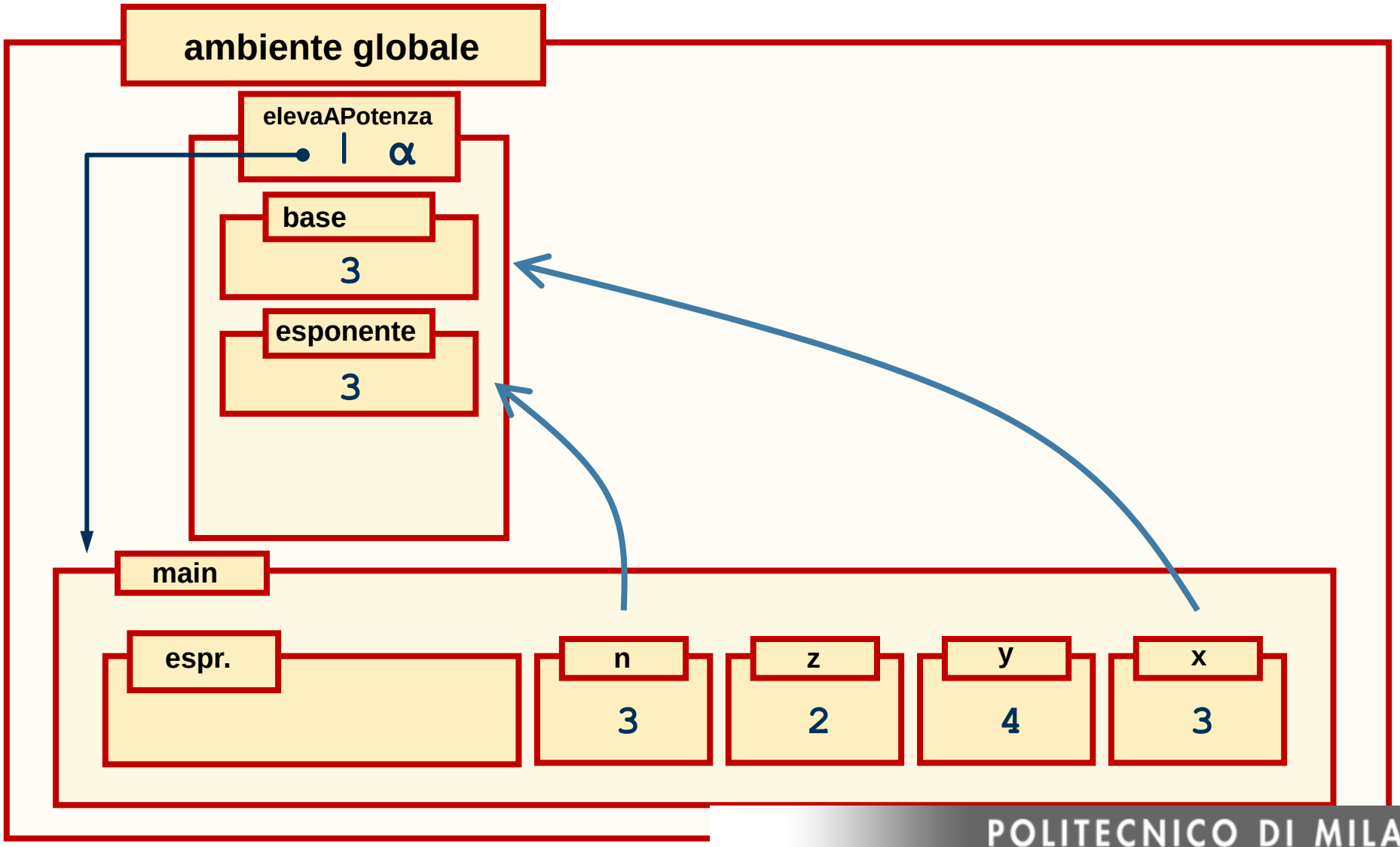
```
"Leggi i dati e verifica che rispondano alle specifiche"
   if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
   …
```



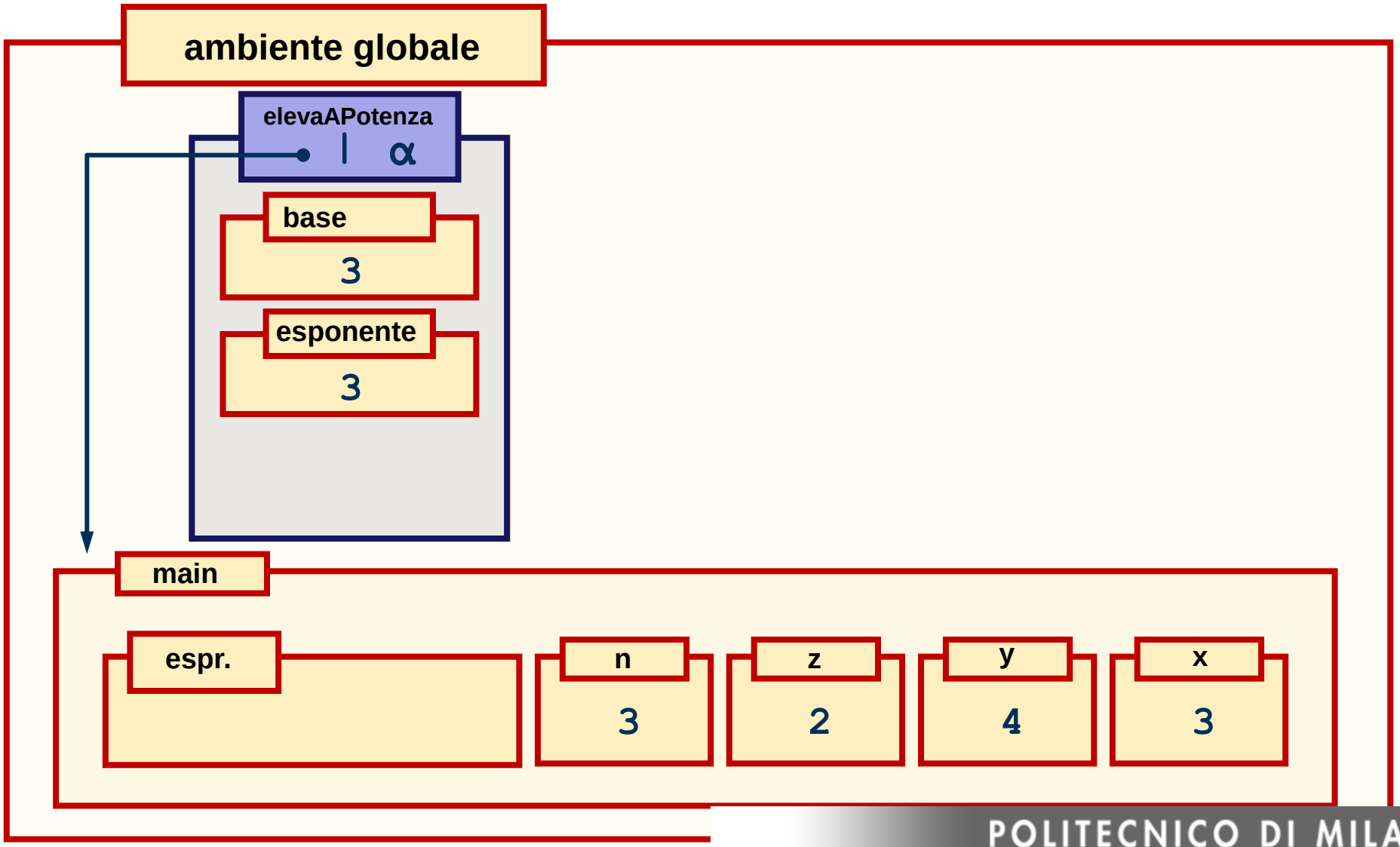
```
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
```



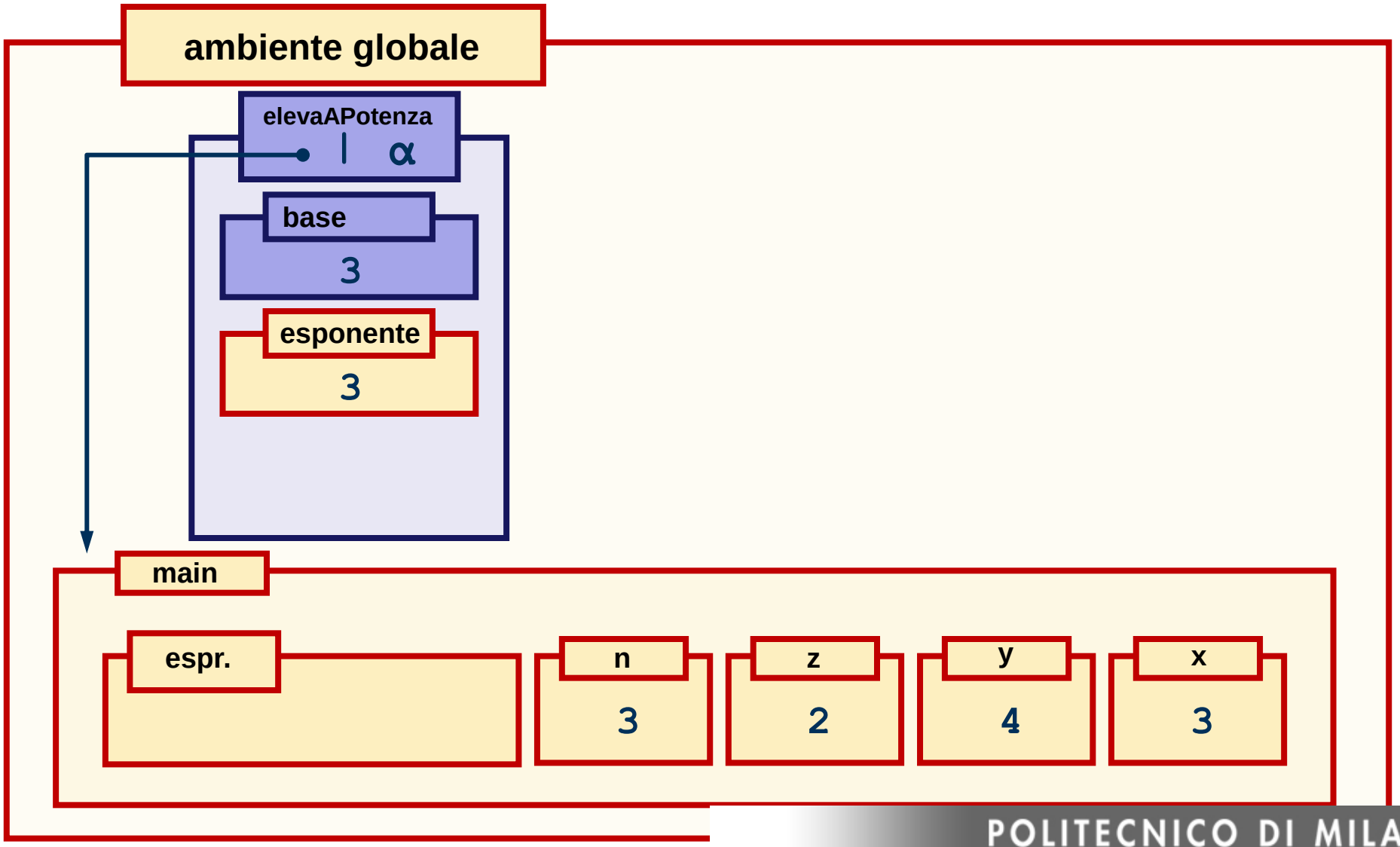
```
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
```



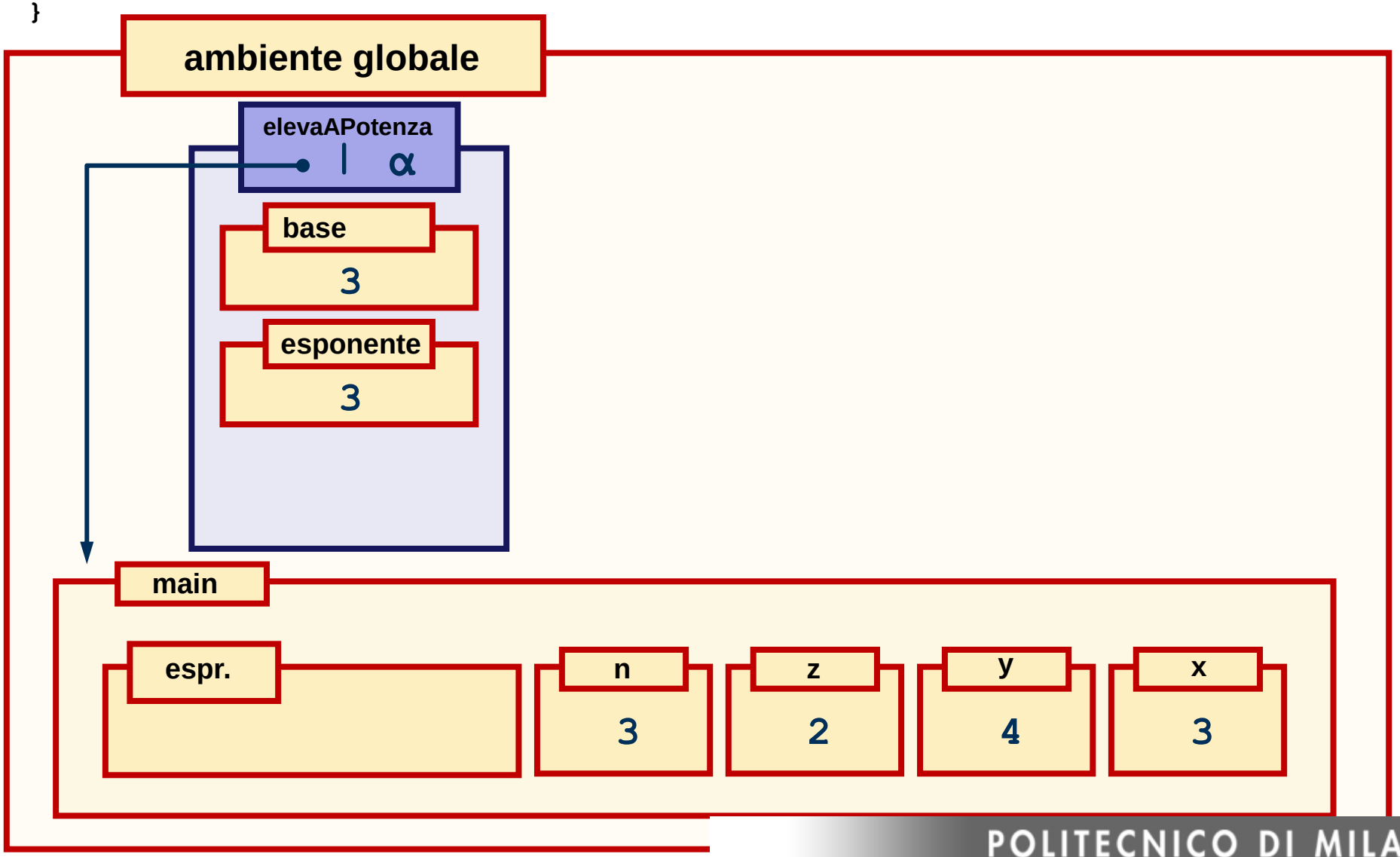
```
int elevaAPotenza(int base, int esponente)   // versione ricorsiva
{ if (esponente == 1)
    return base;
```



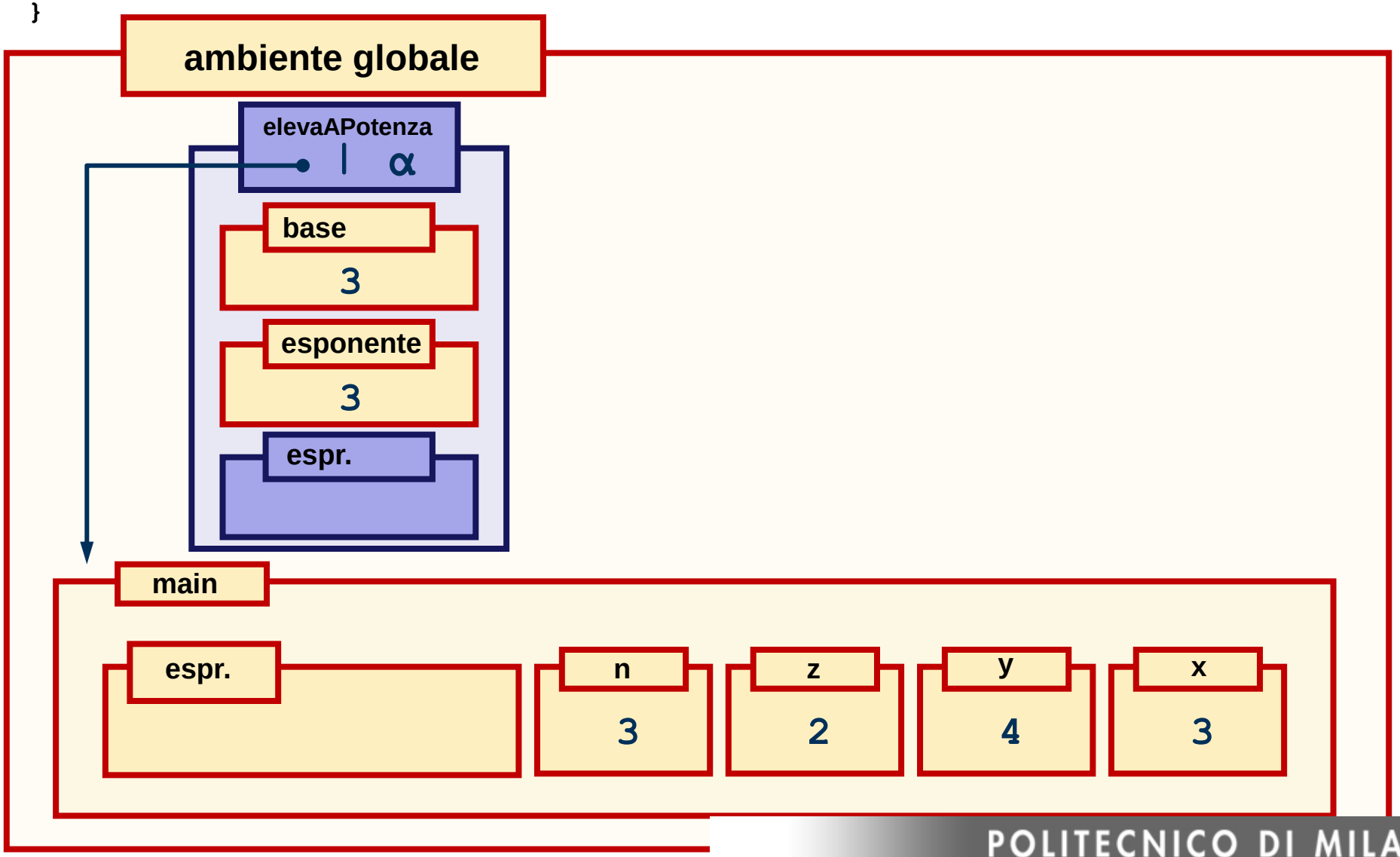
```
int elevaAPotenza(int base, int esponente)   // versione ricorsiva
{ if (esponente == 1)
    return base;
  if (esponente > 1)
```



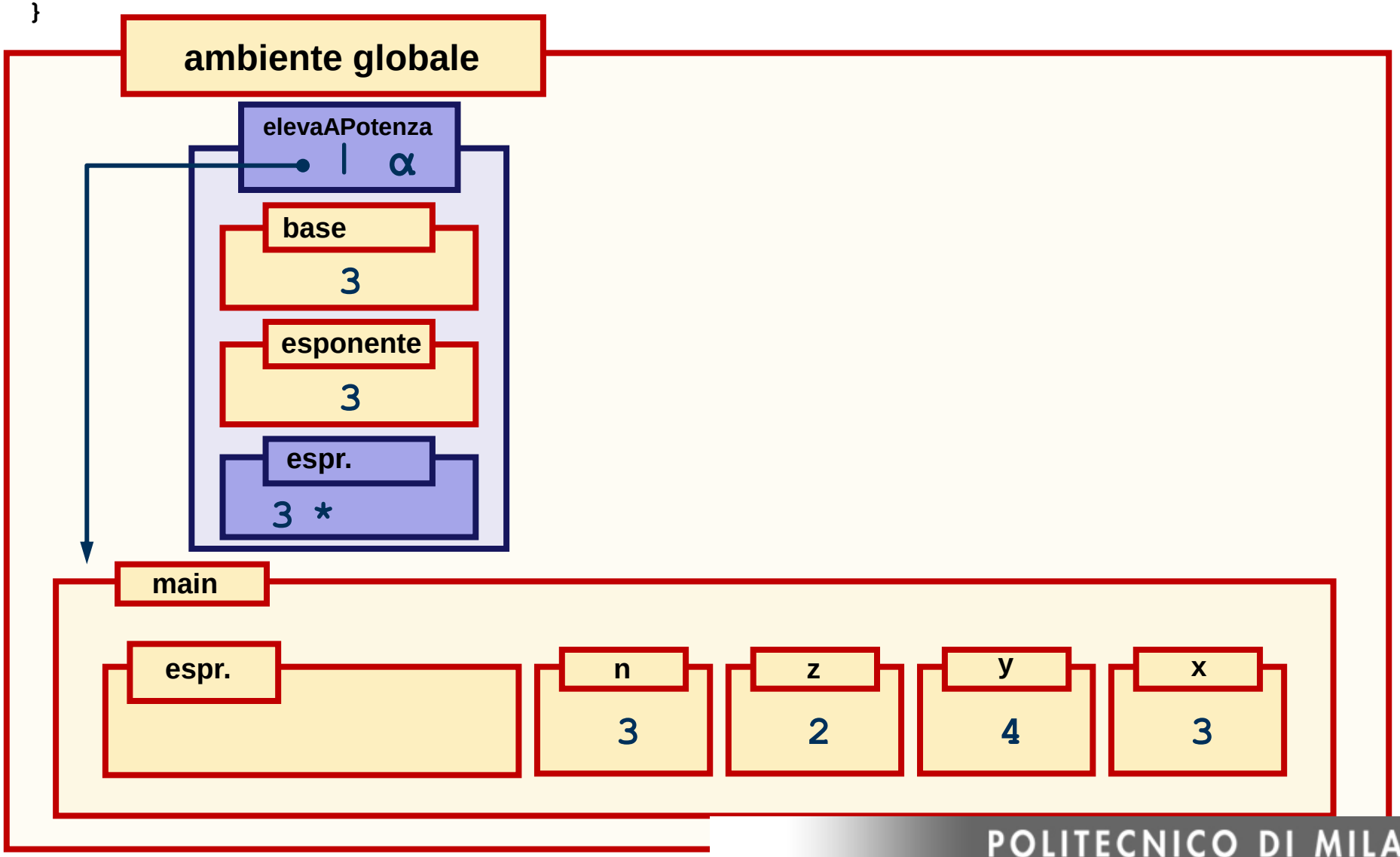
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

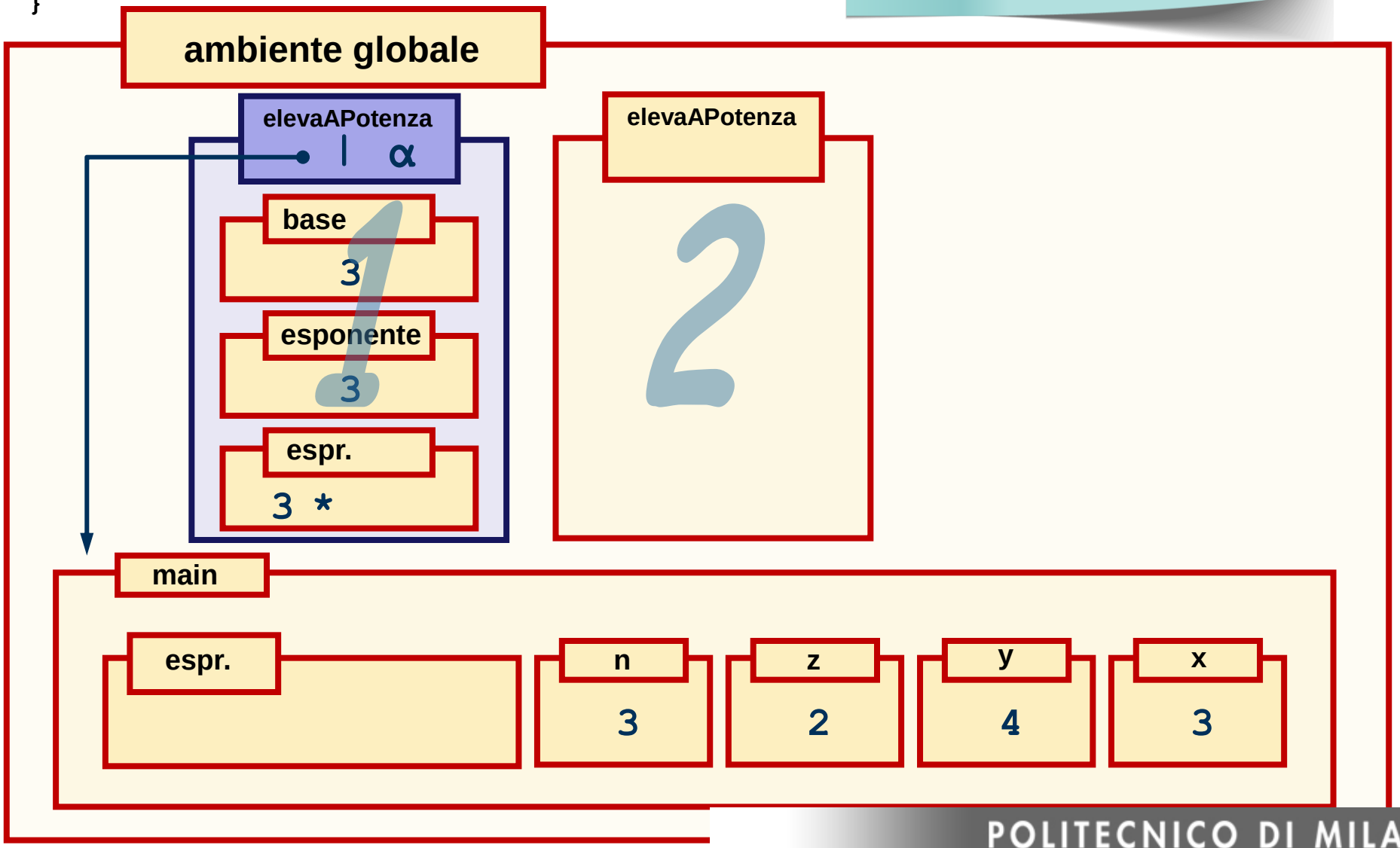


```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

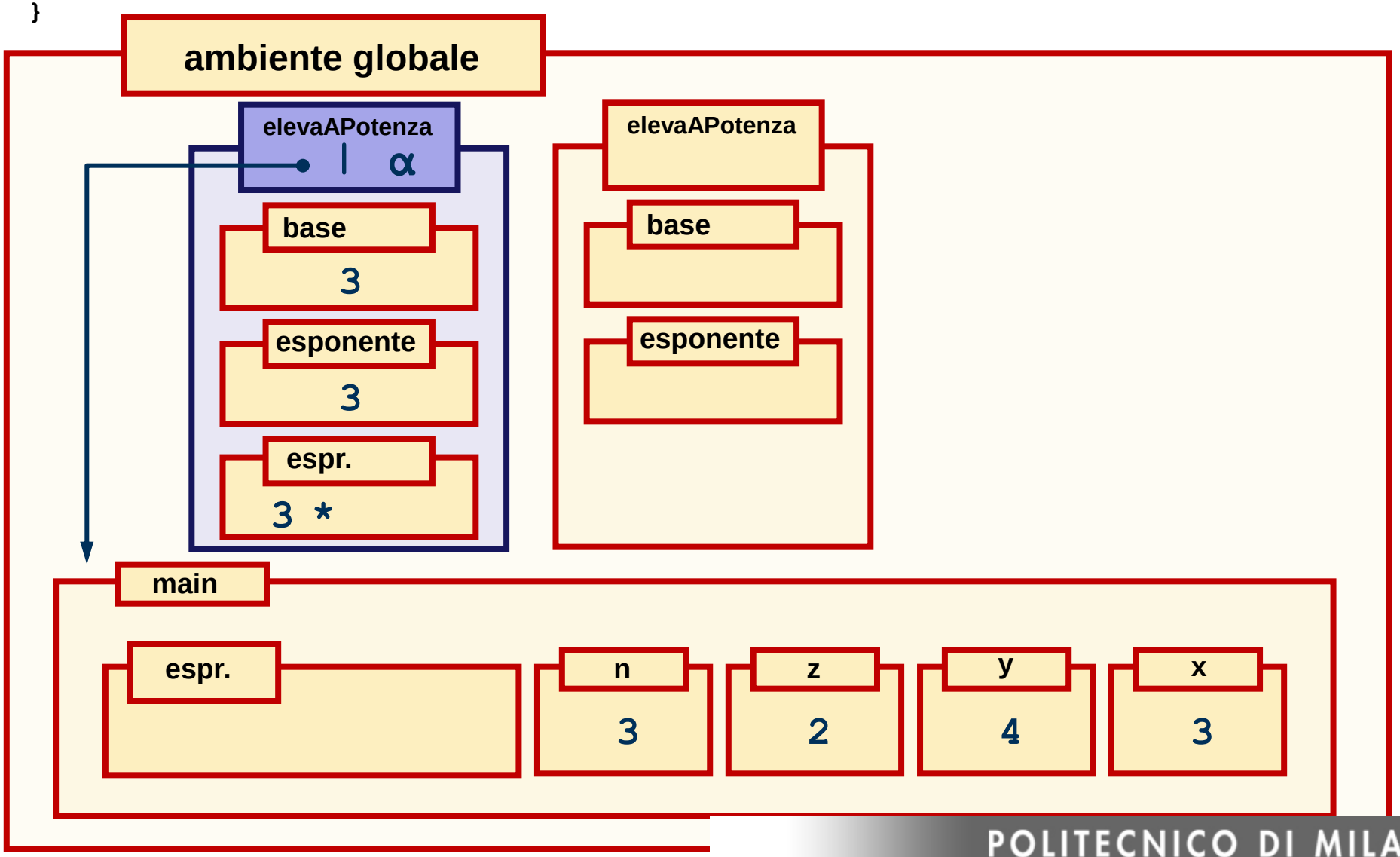


```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1
}
```

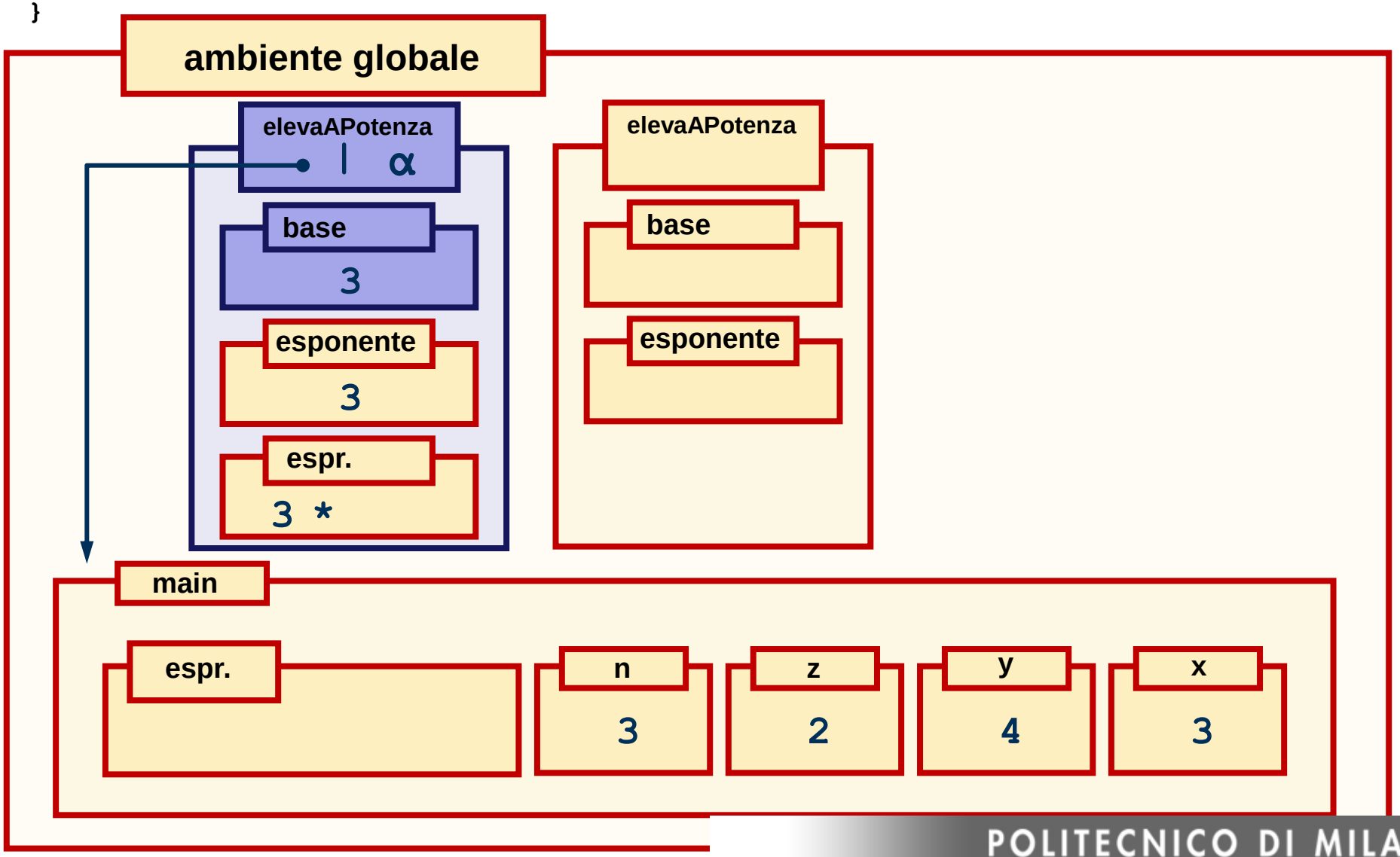
Più copie contemporanee



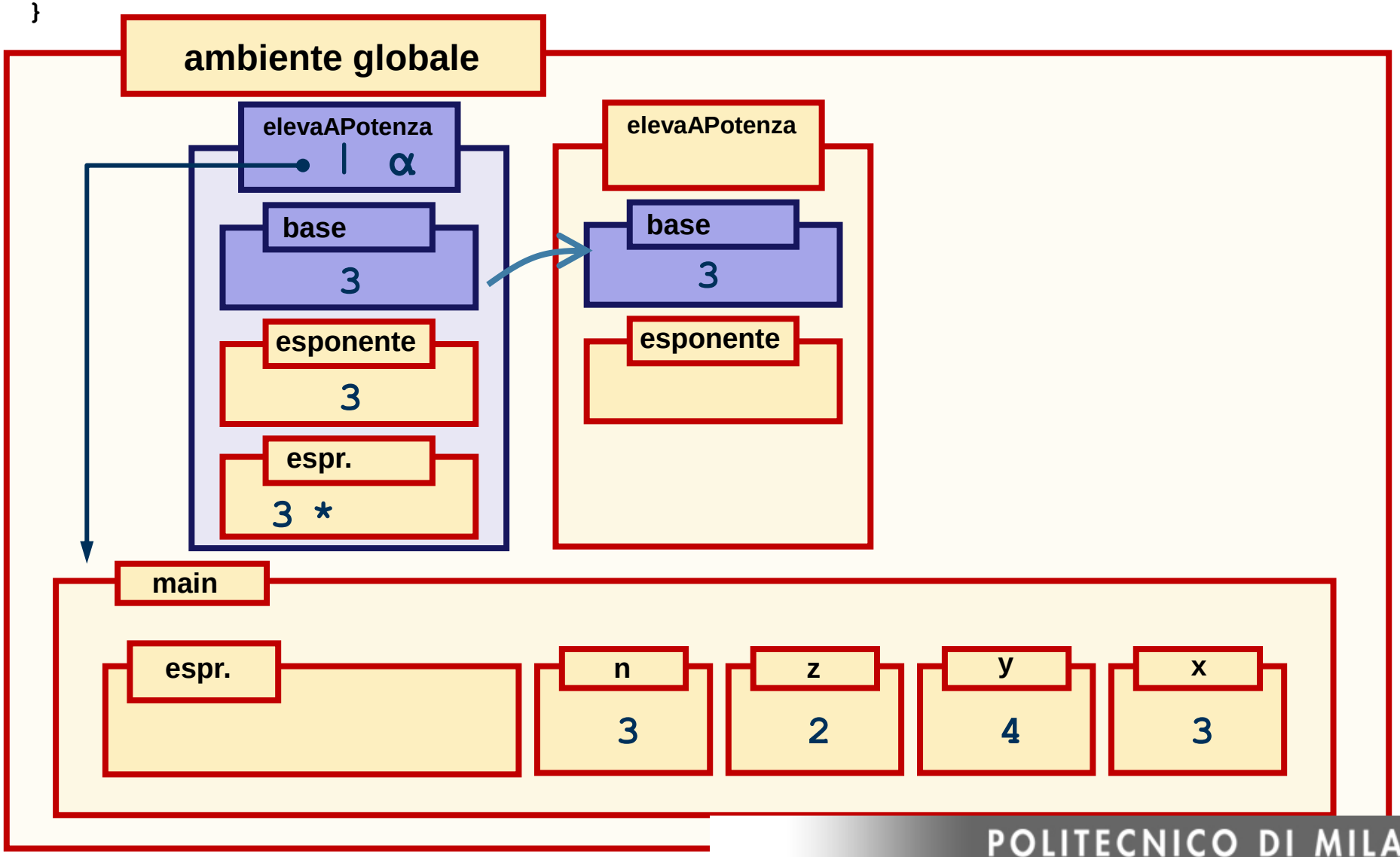
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



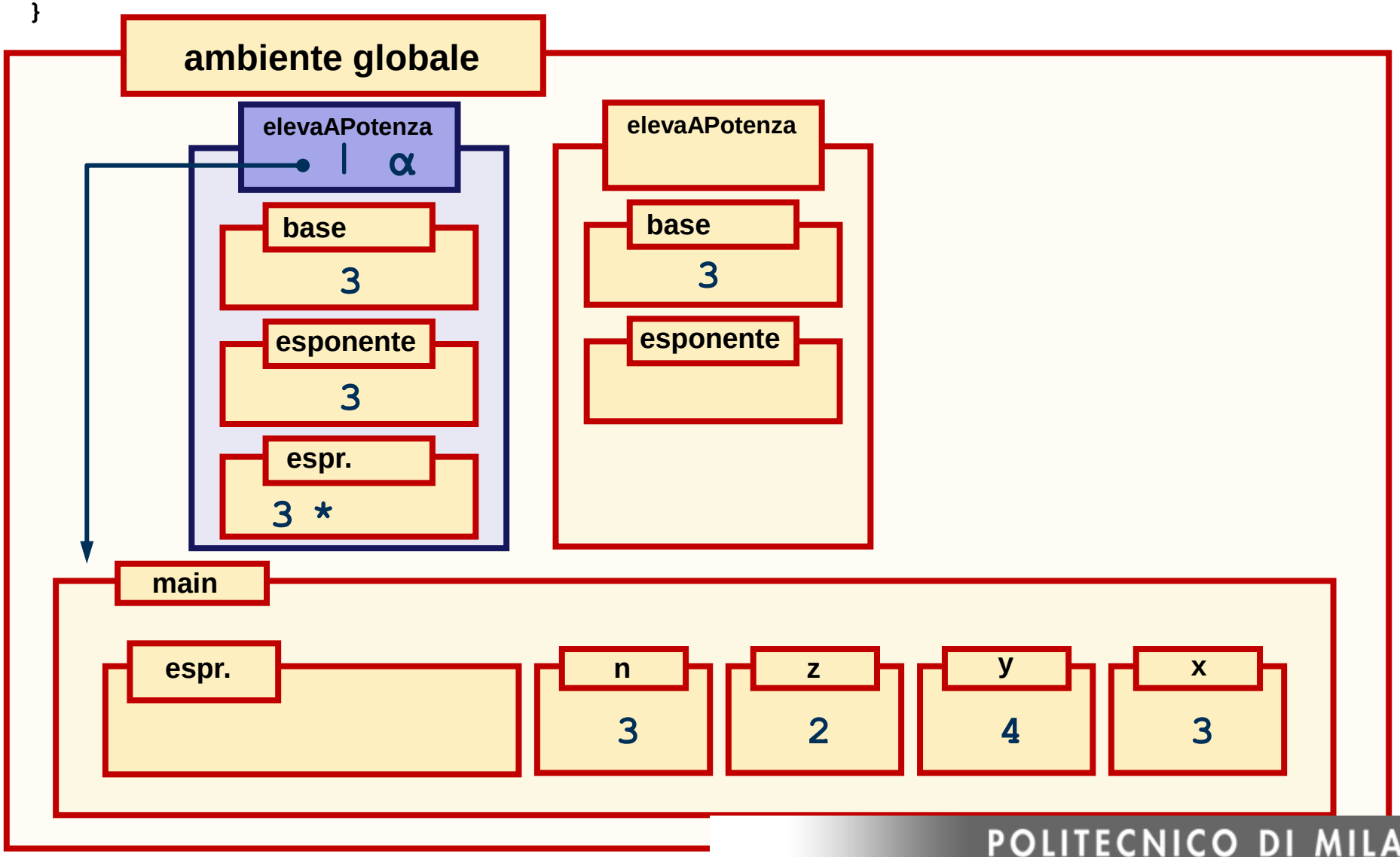
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



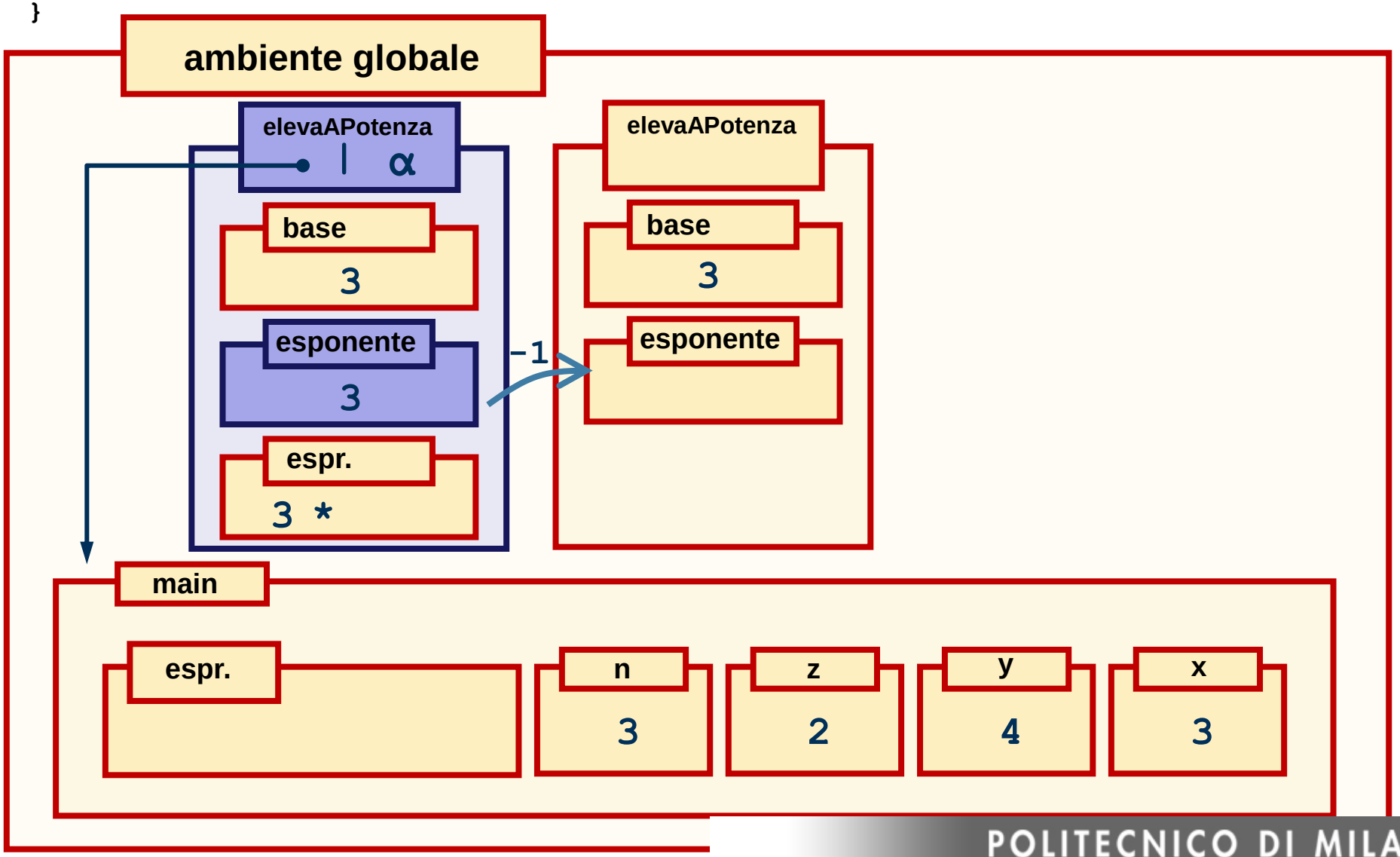
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



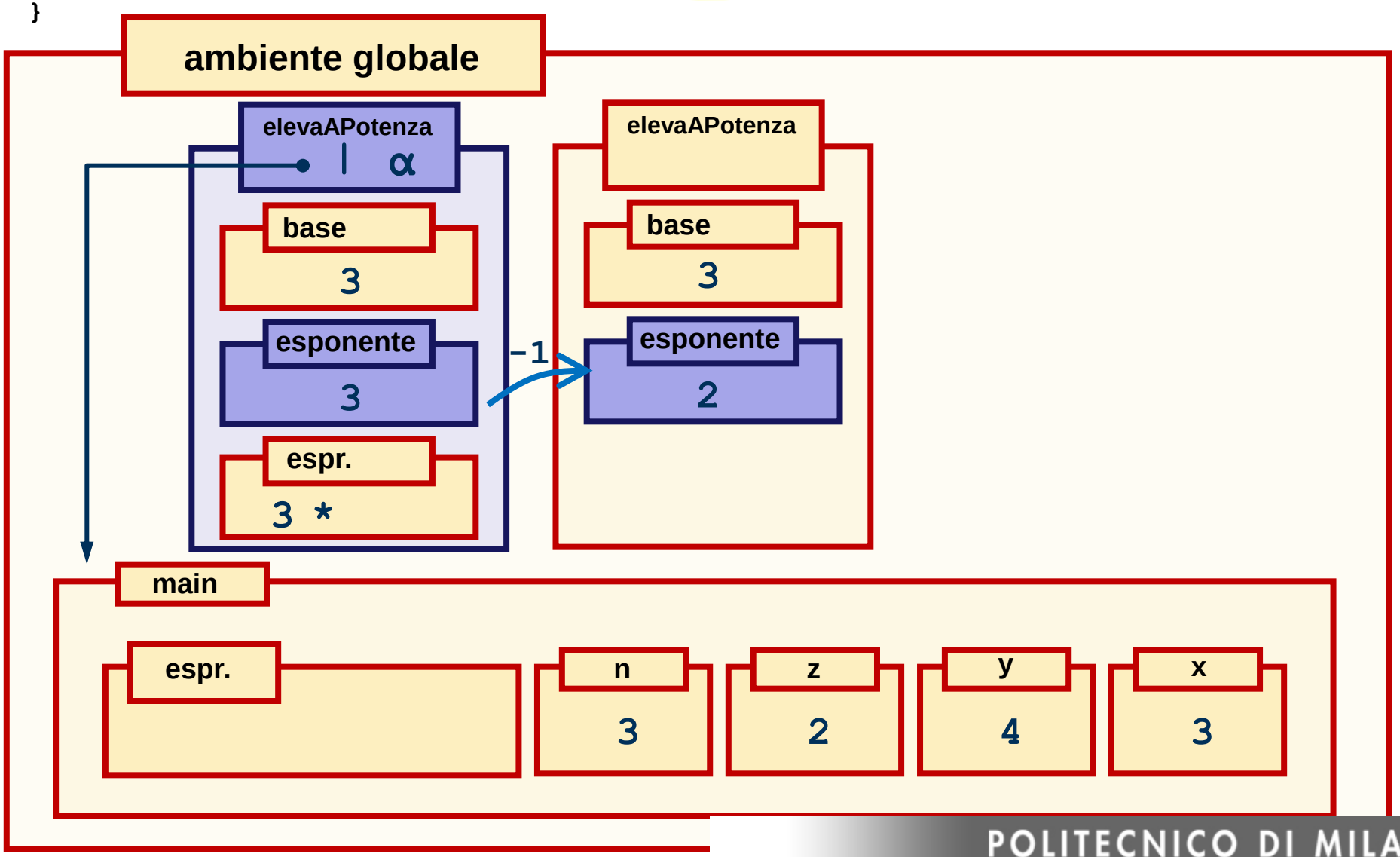
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



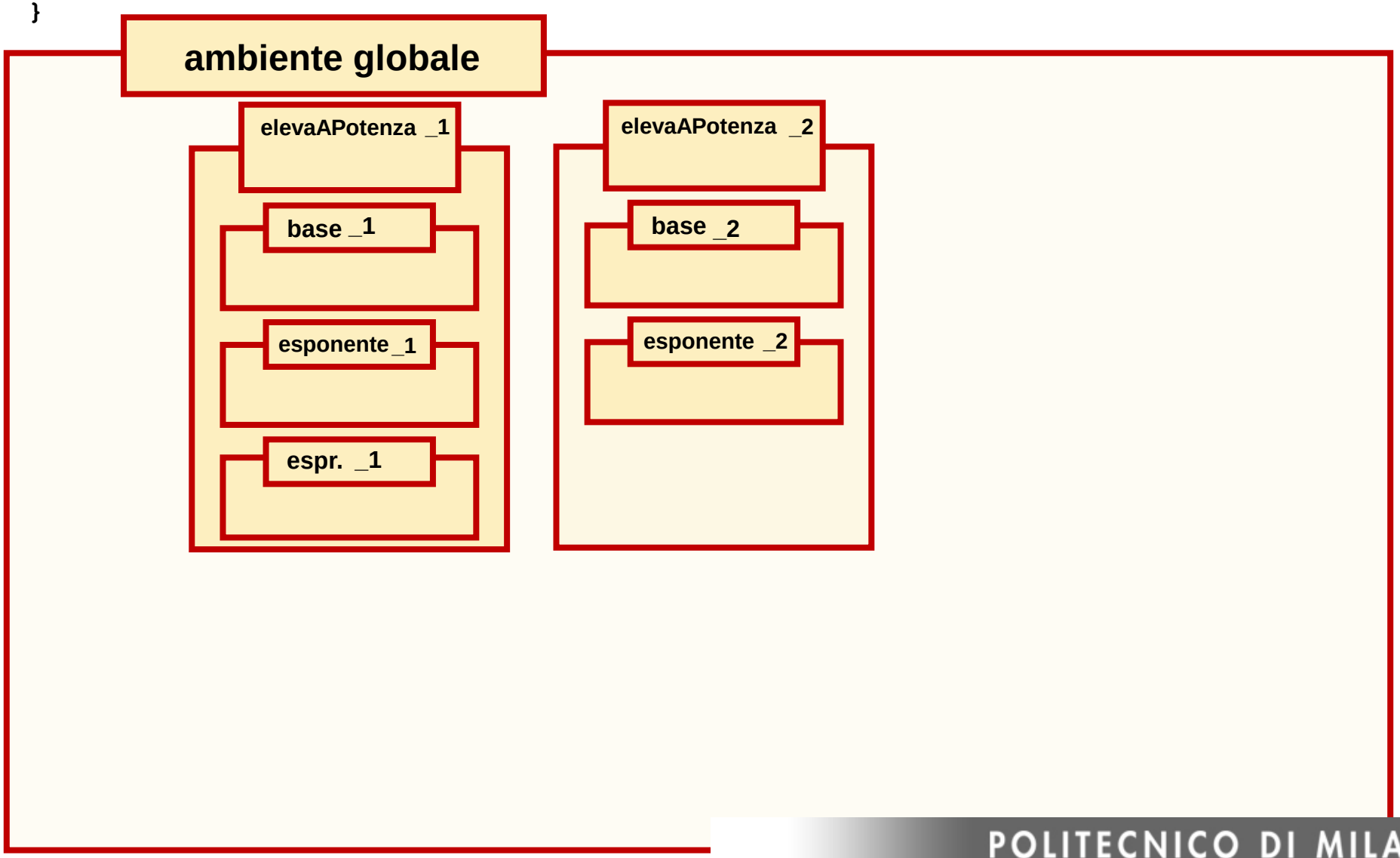
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



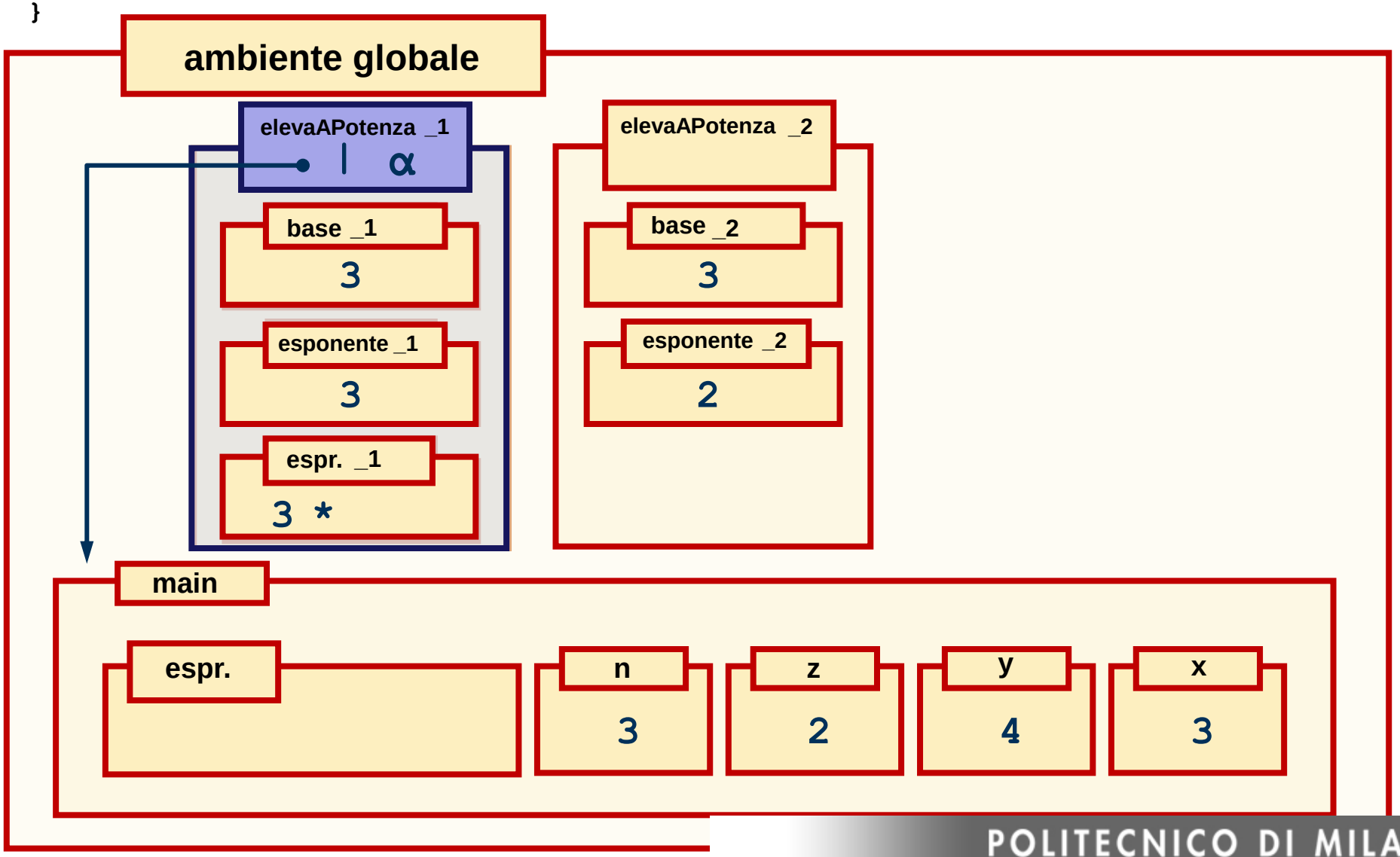
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



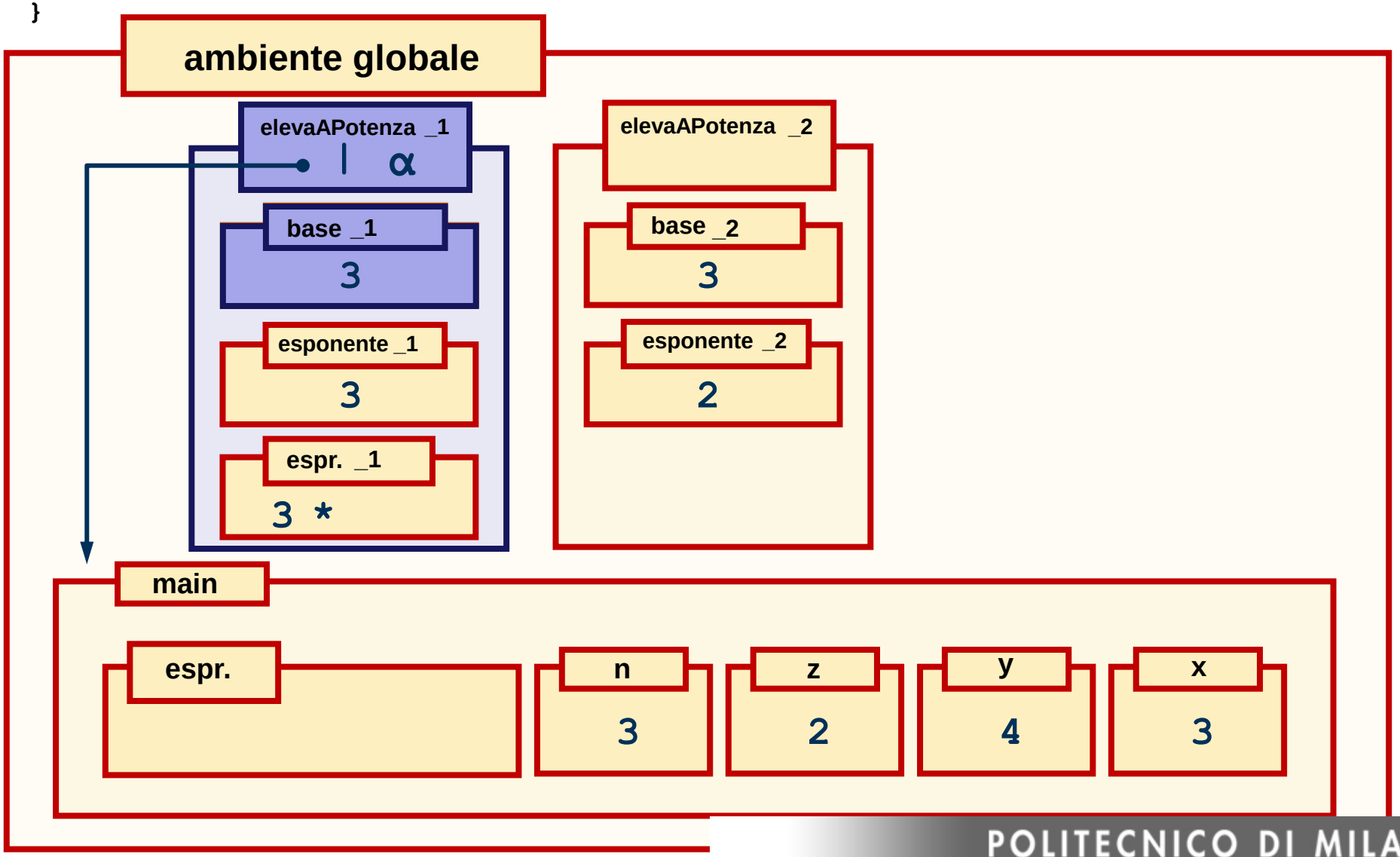
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



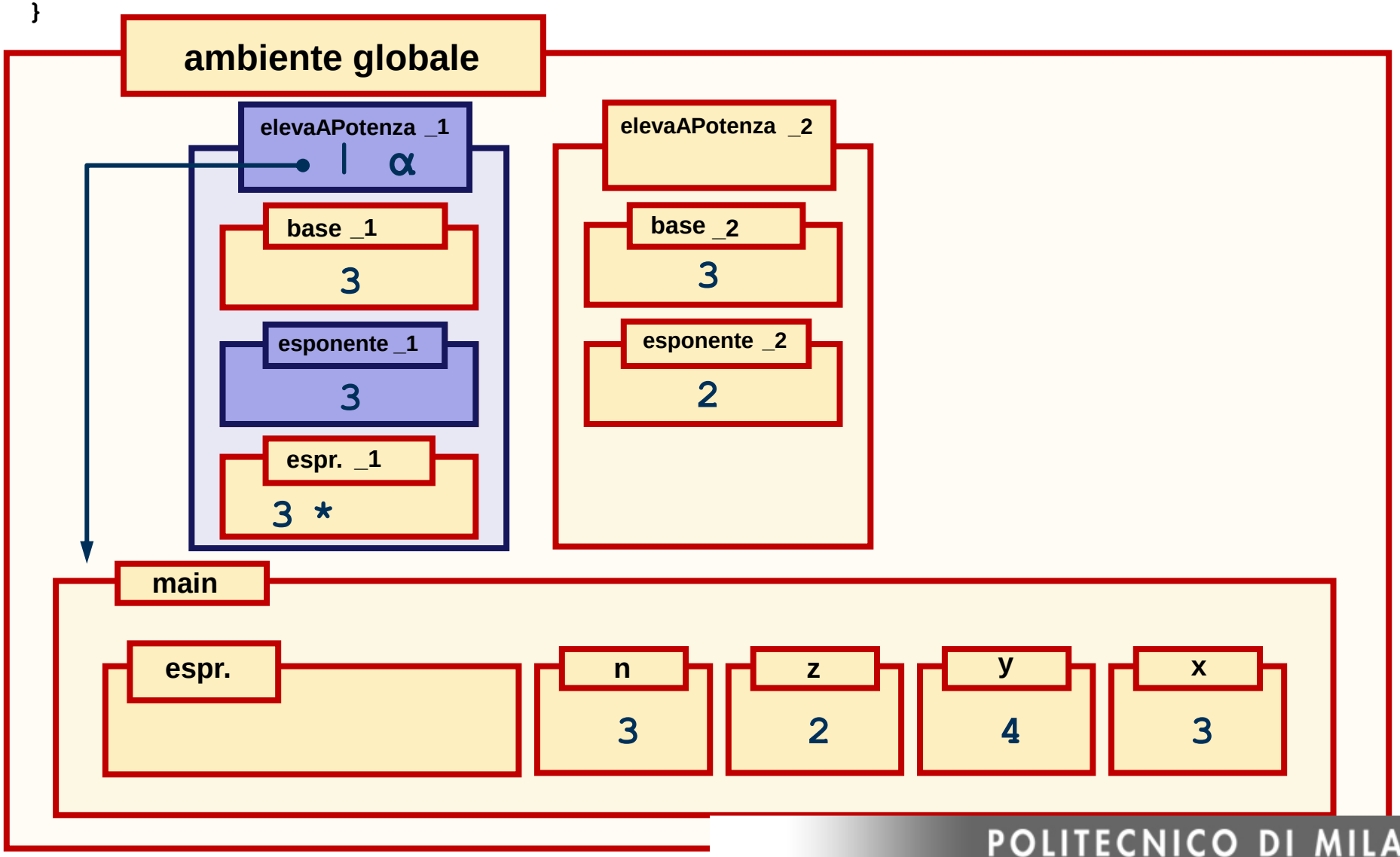
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



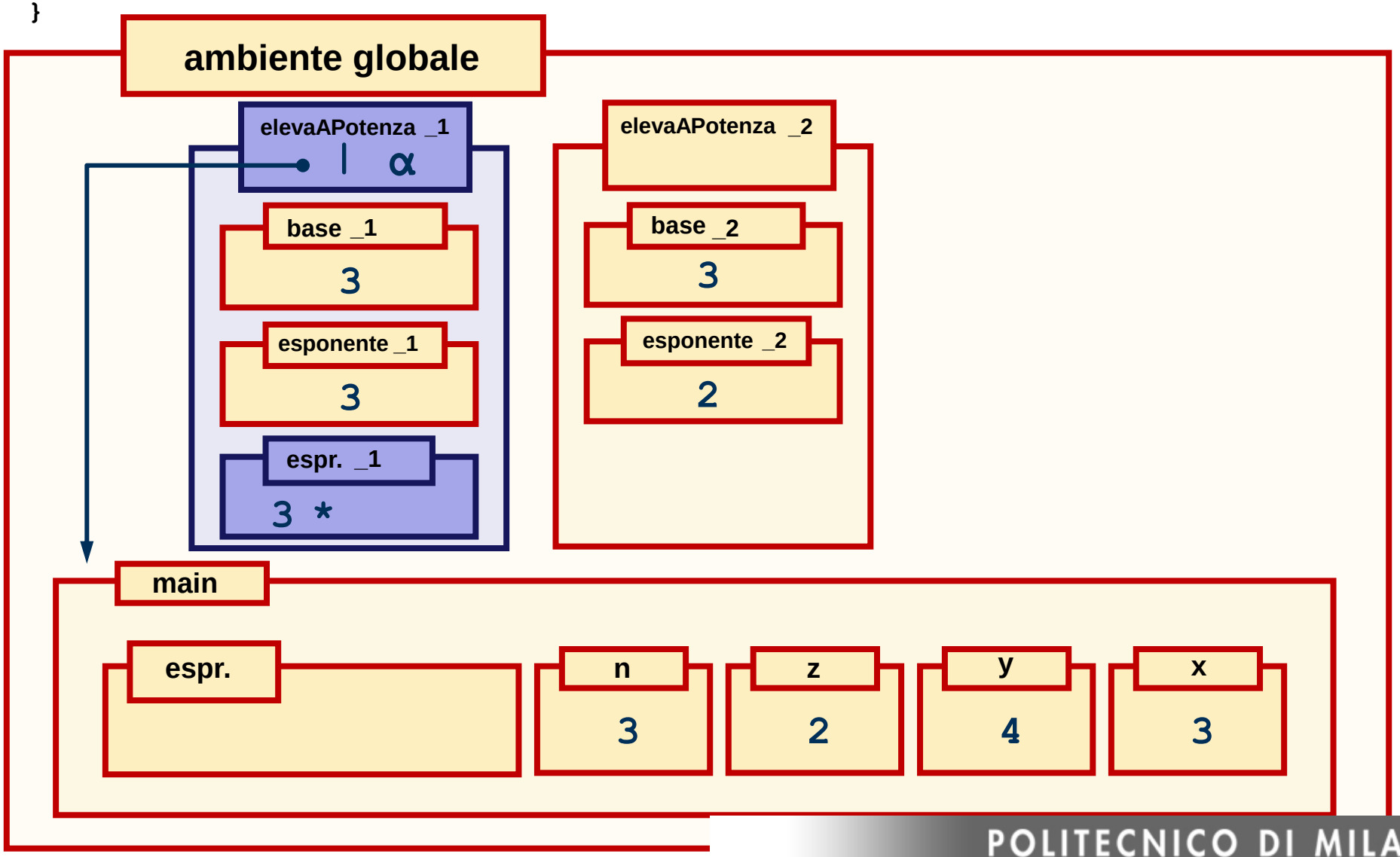
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



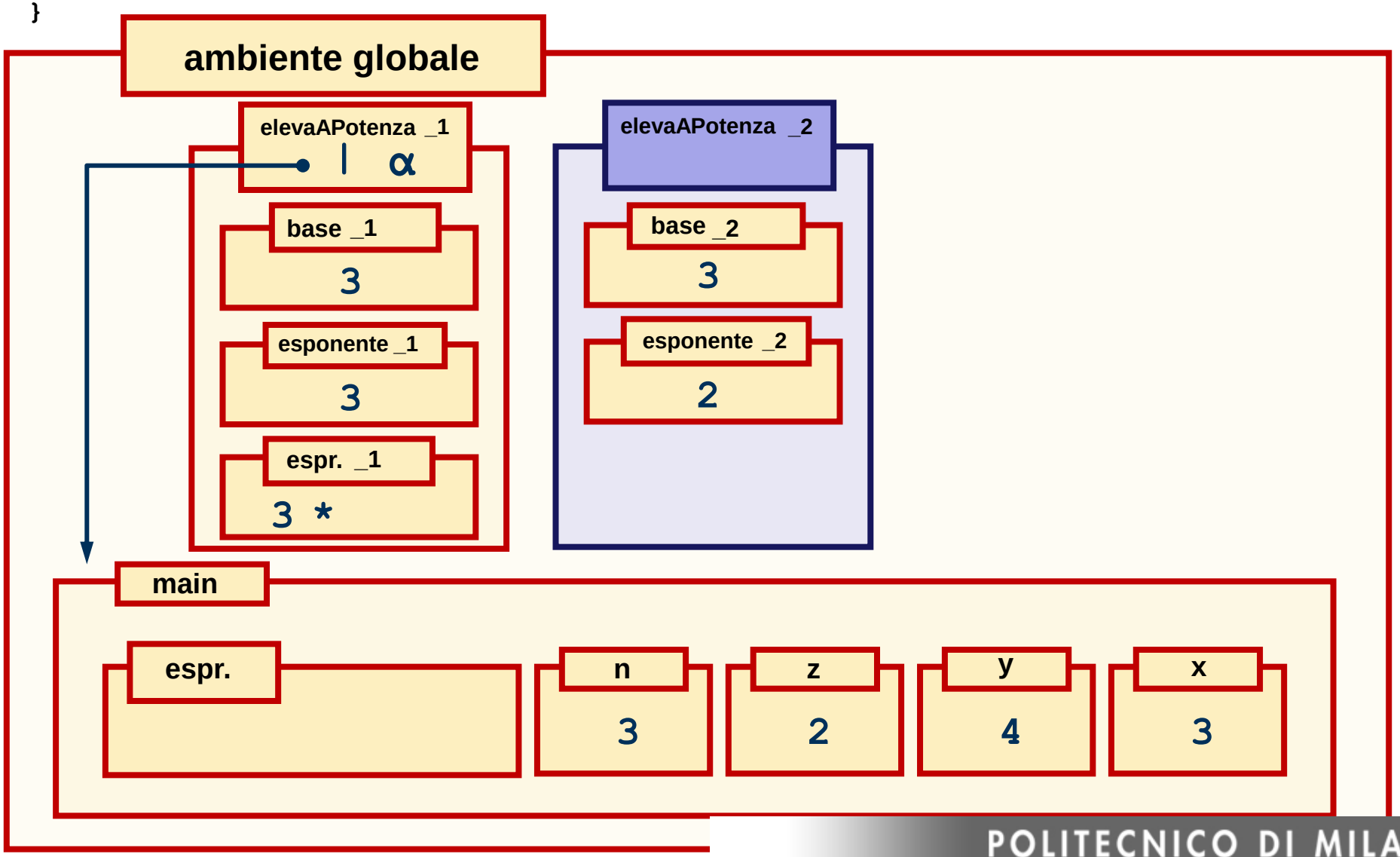
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



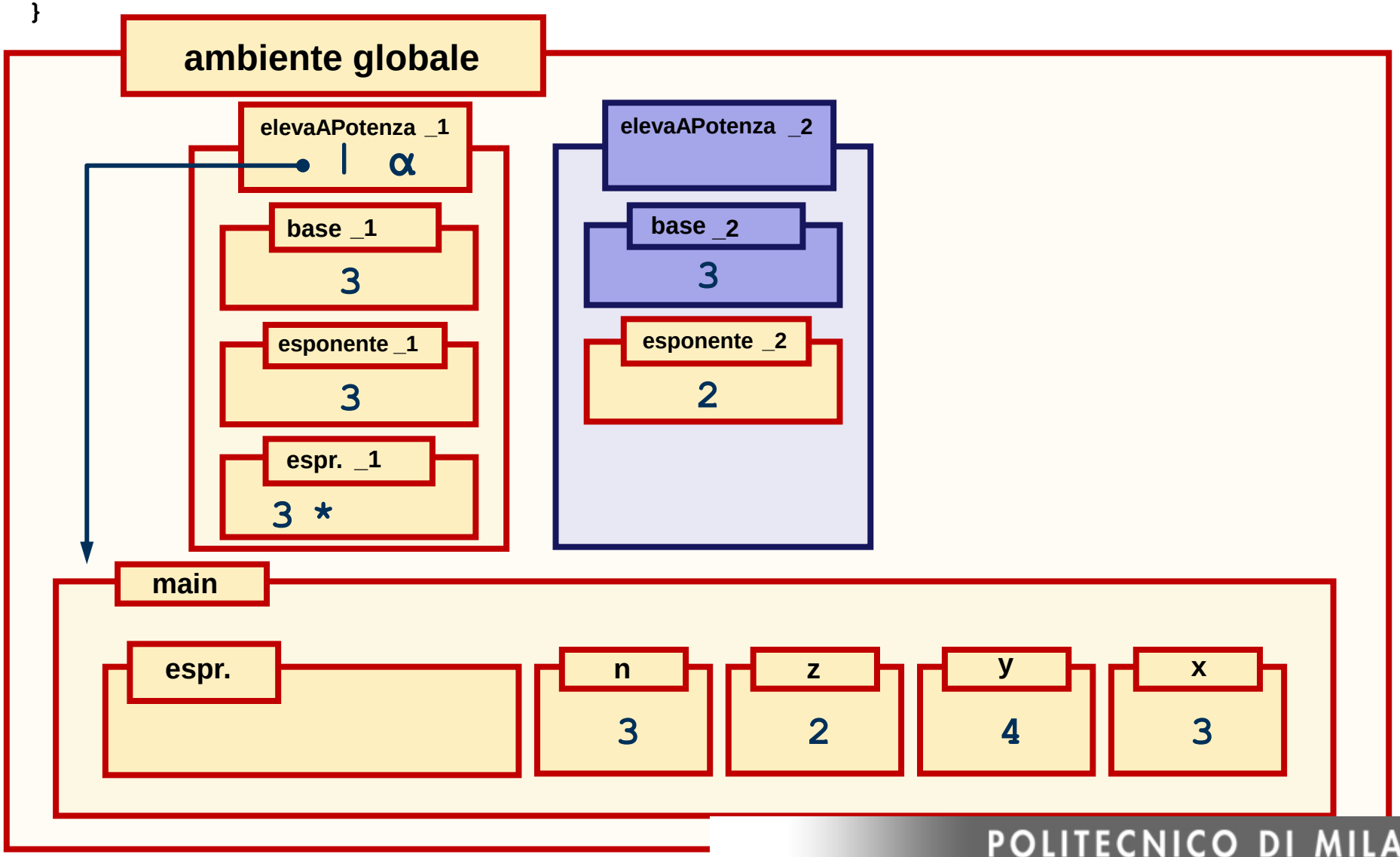
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



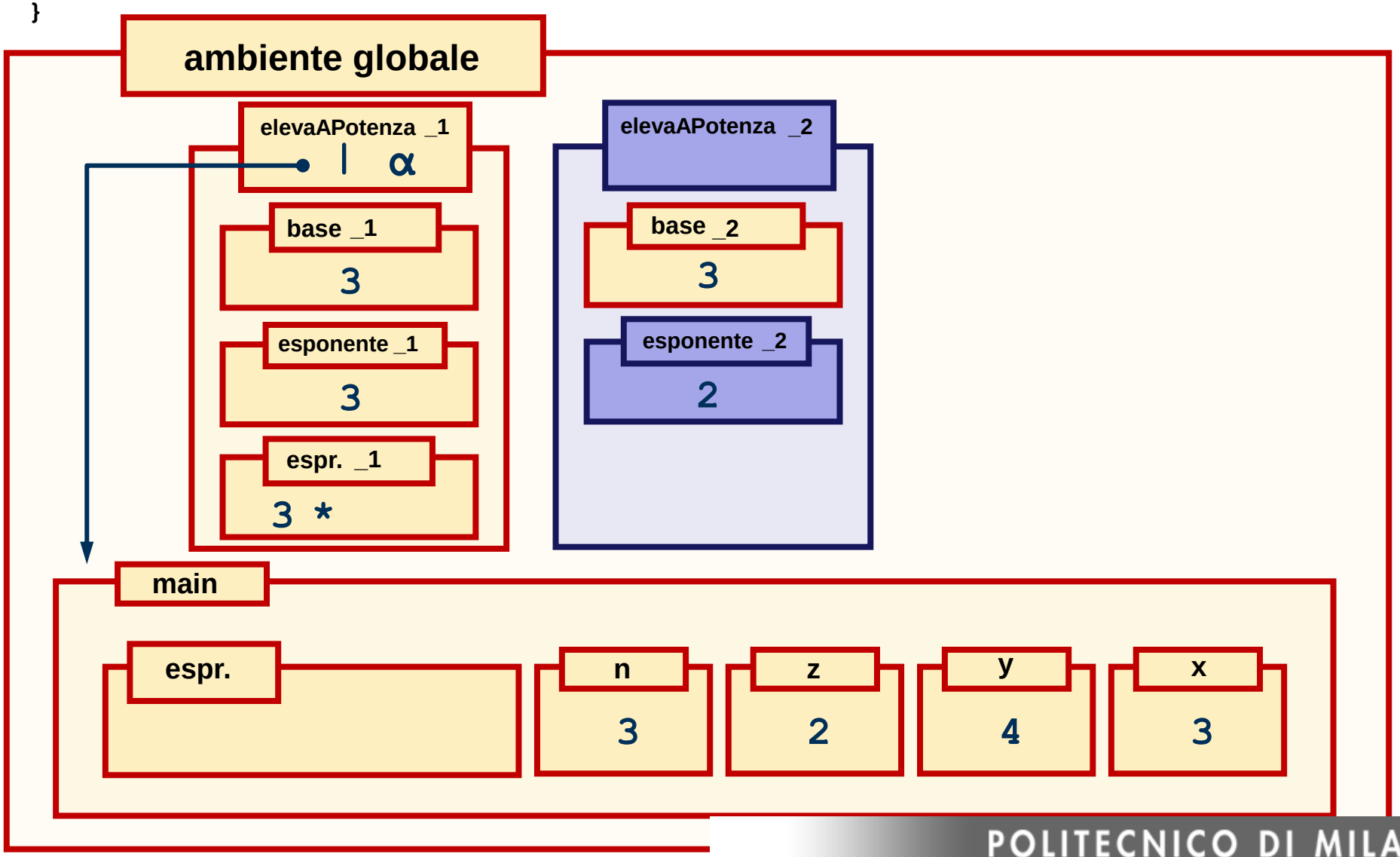
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



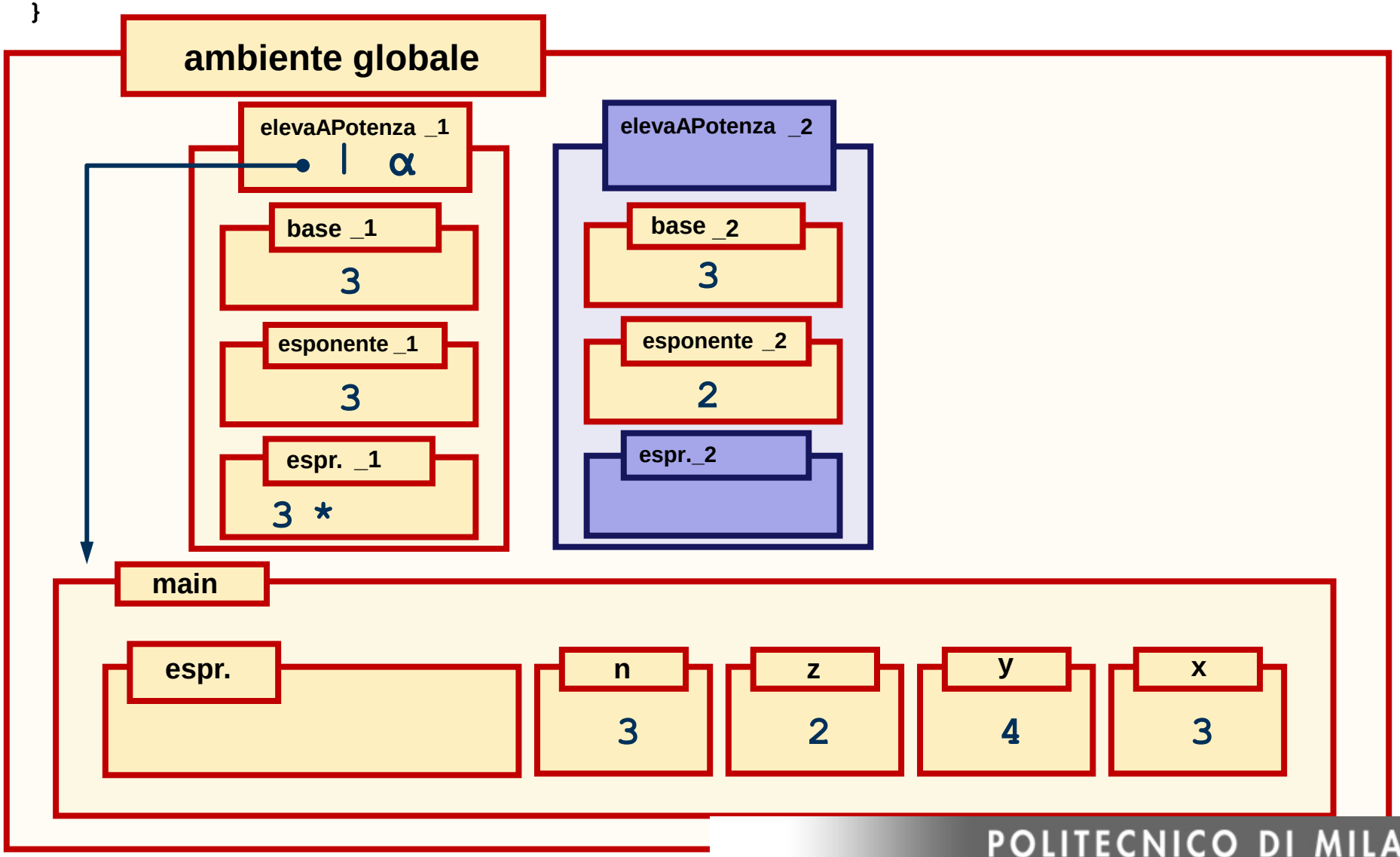
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



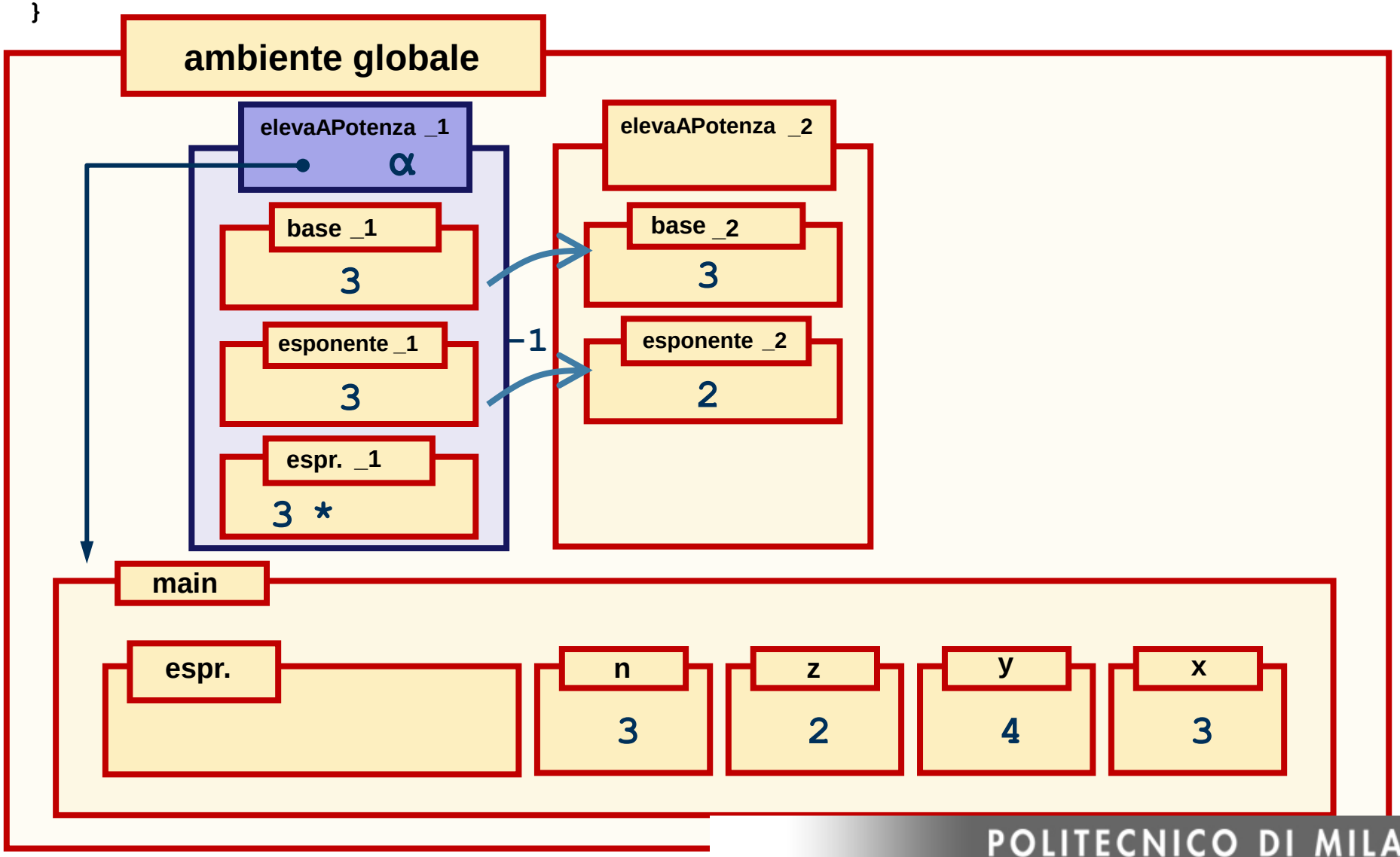
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



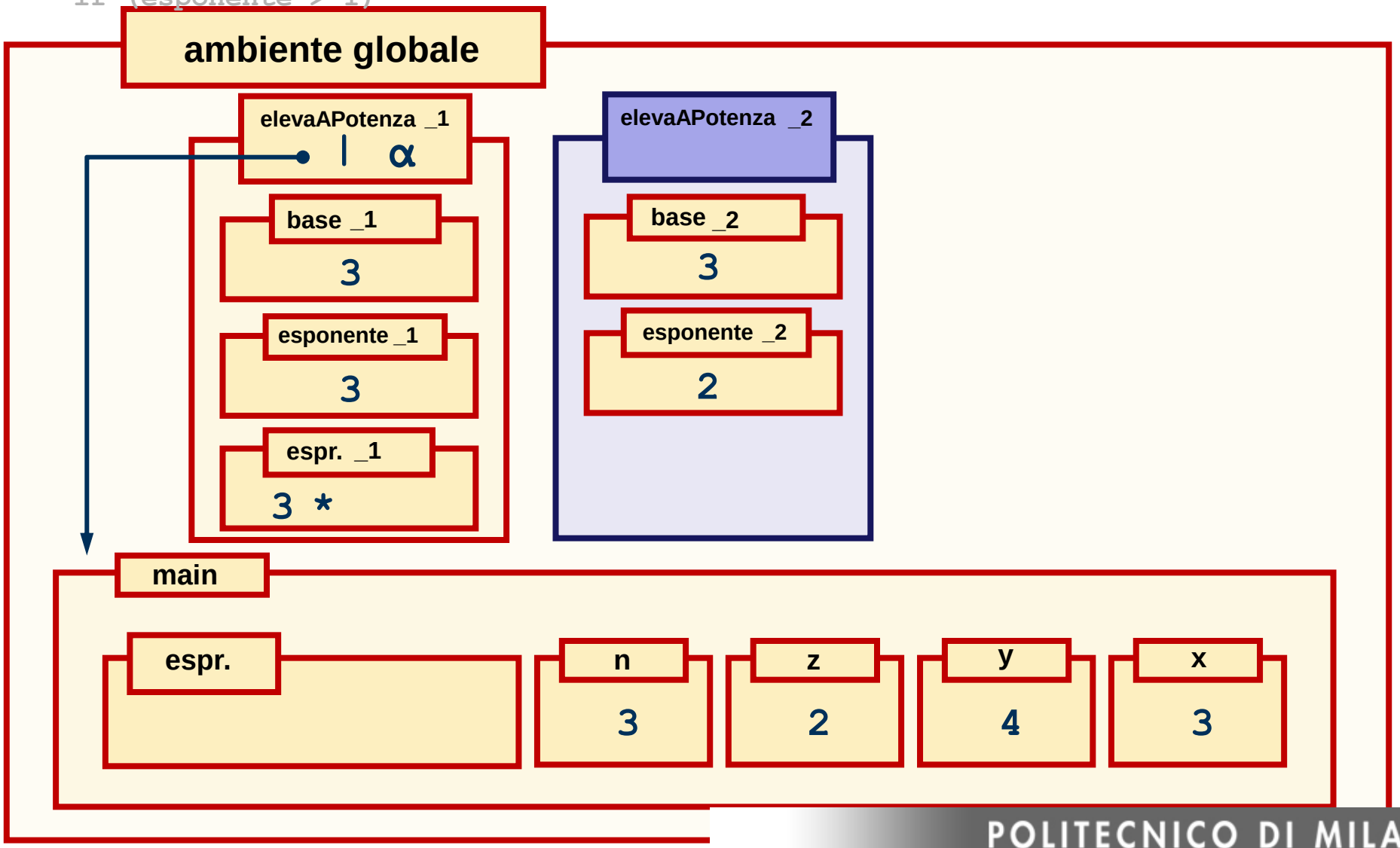
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



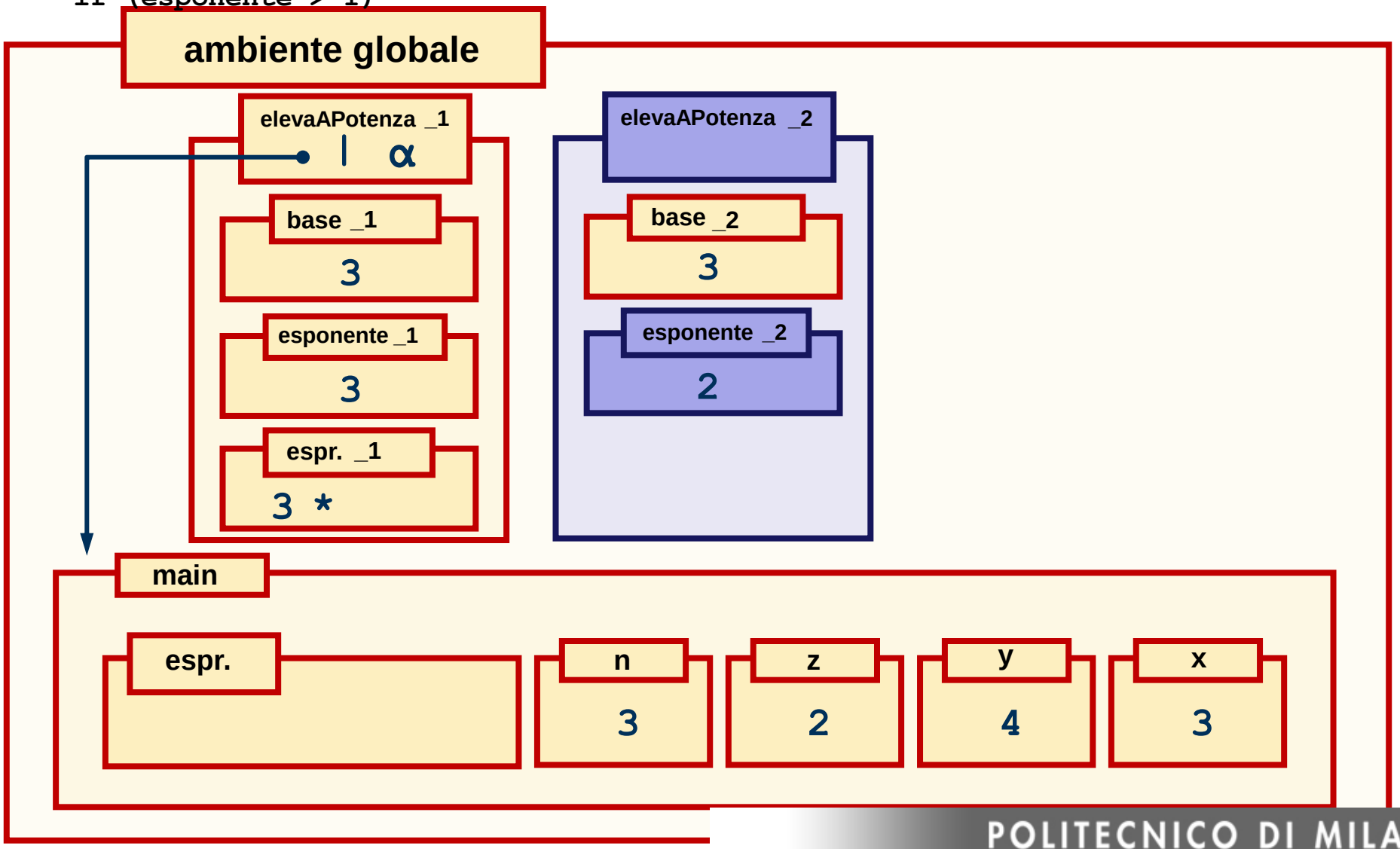
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



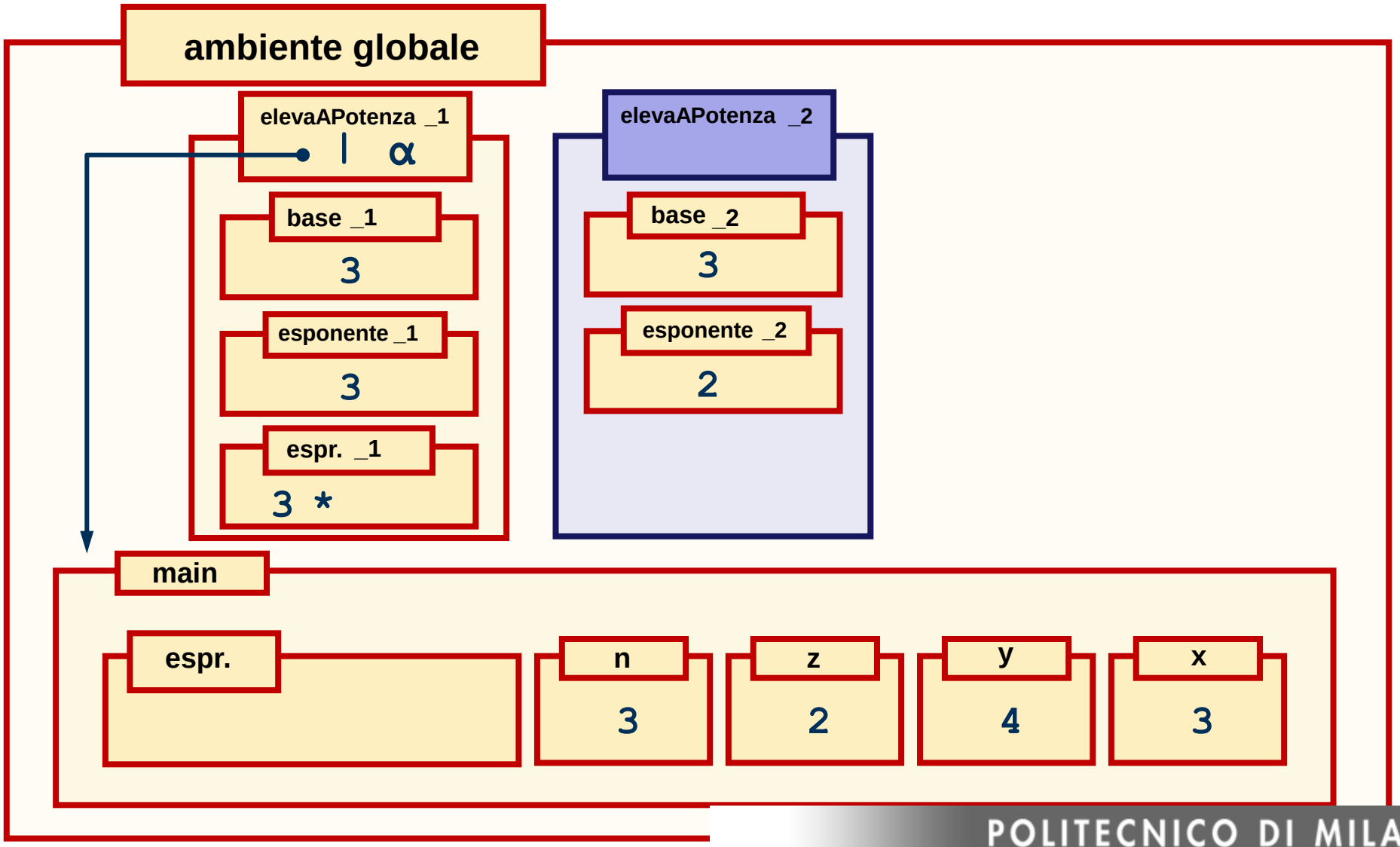
```
int elevaAPotenza(int base, int esponente)  // versione ricorsiva
{ if (esponente == 1)
    return base;
  if (esponente > 1)
```



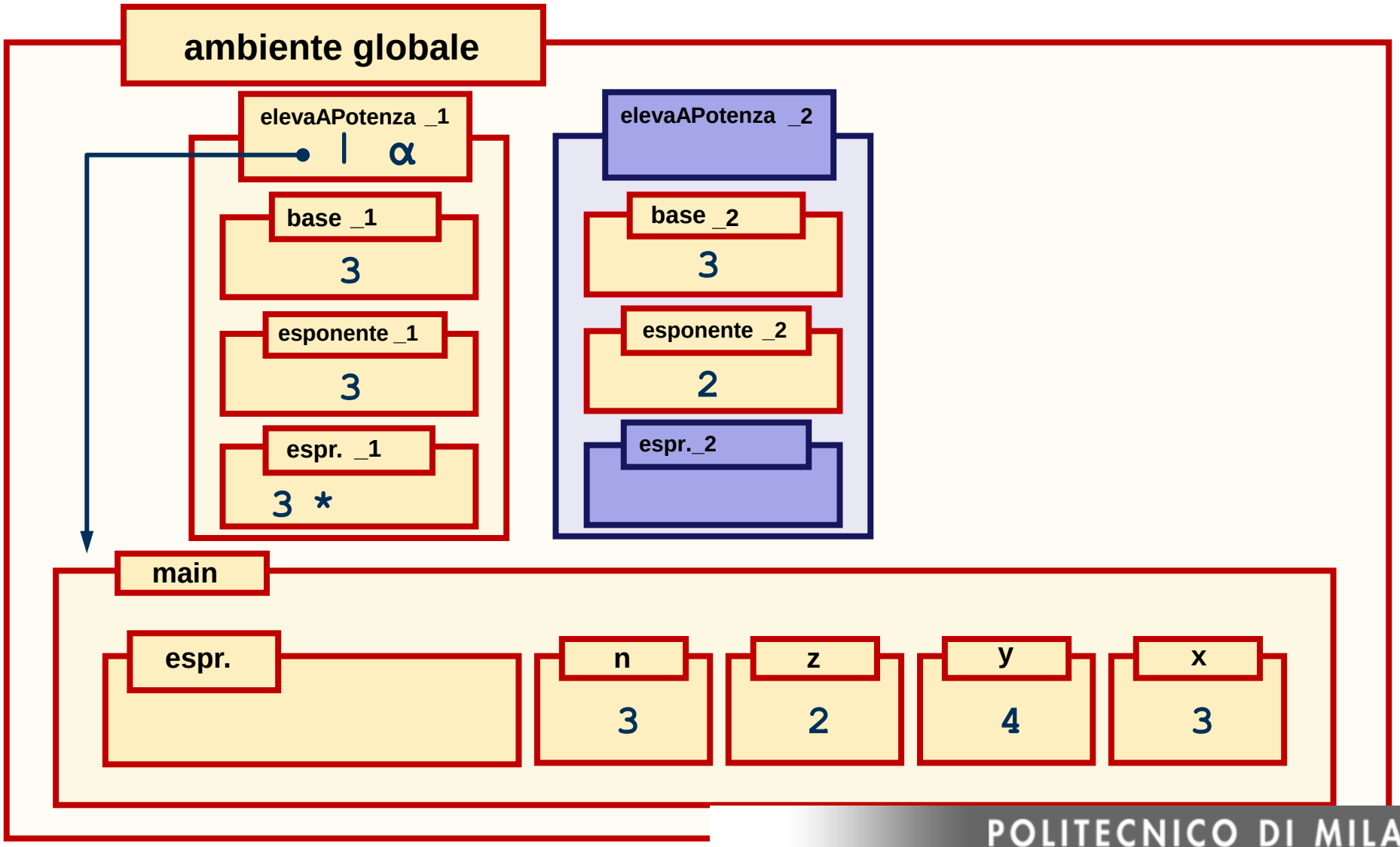
```
int elevaAPotenza(int base, int
{ if (esponente == 1)
    return base;
  if (esponente > 1)
```



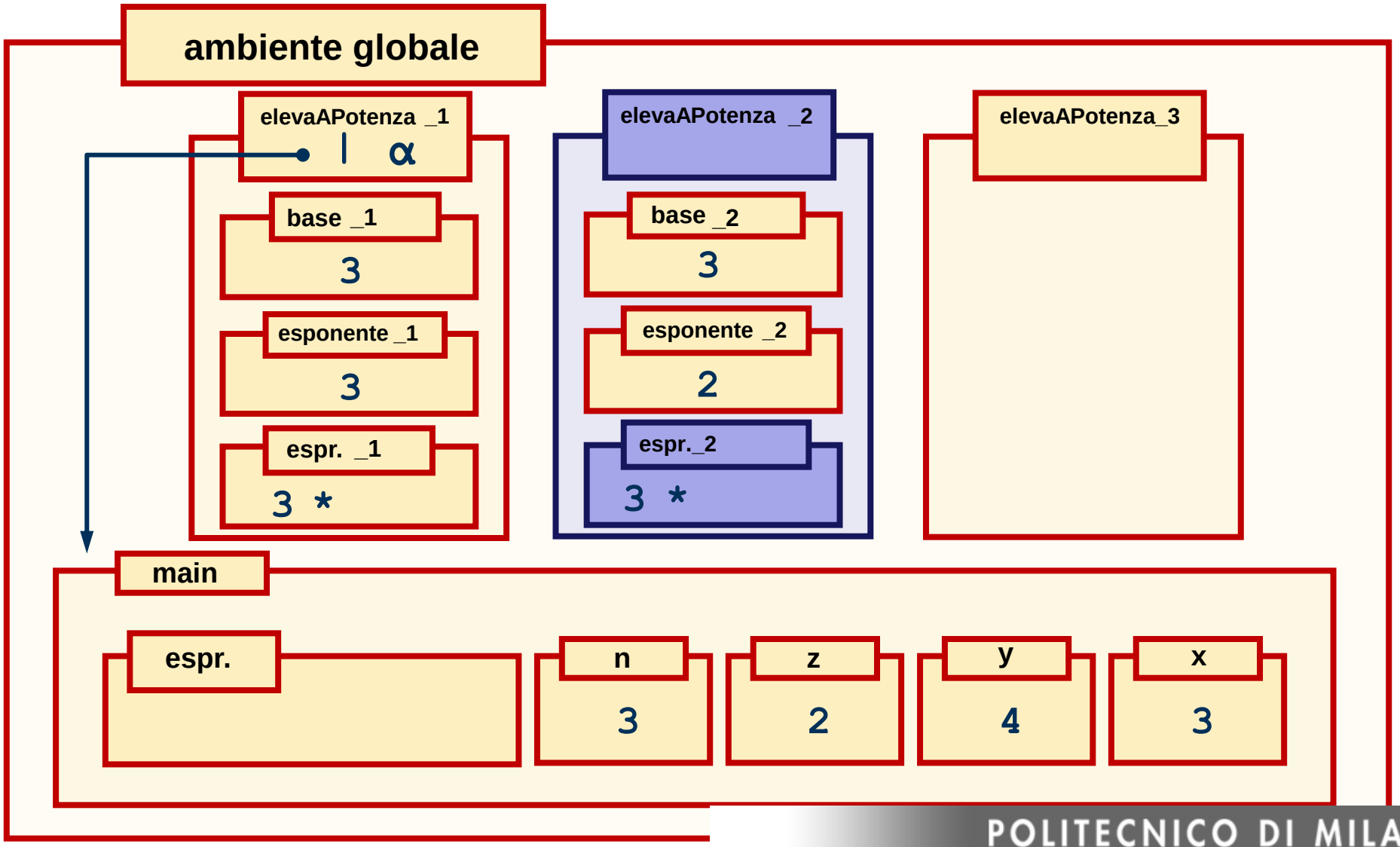
```
if (esponente > 1)
  return base * elevaAPotenza(base, esponente-1);
}
```



```
if (esponente > 1)
  return base * elevaAPotenza(base, esponente-1);
}
```

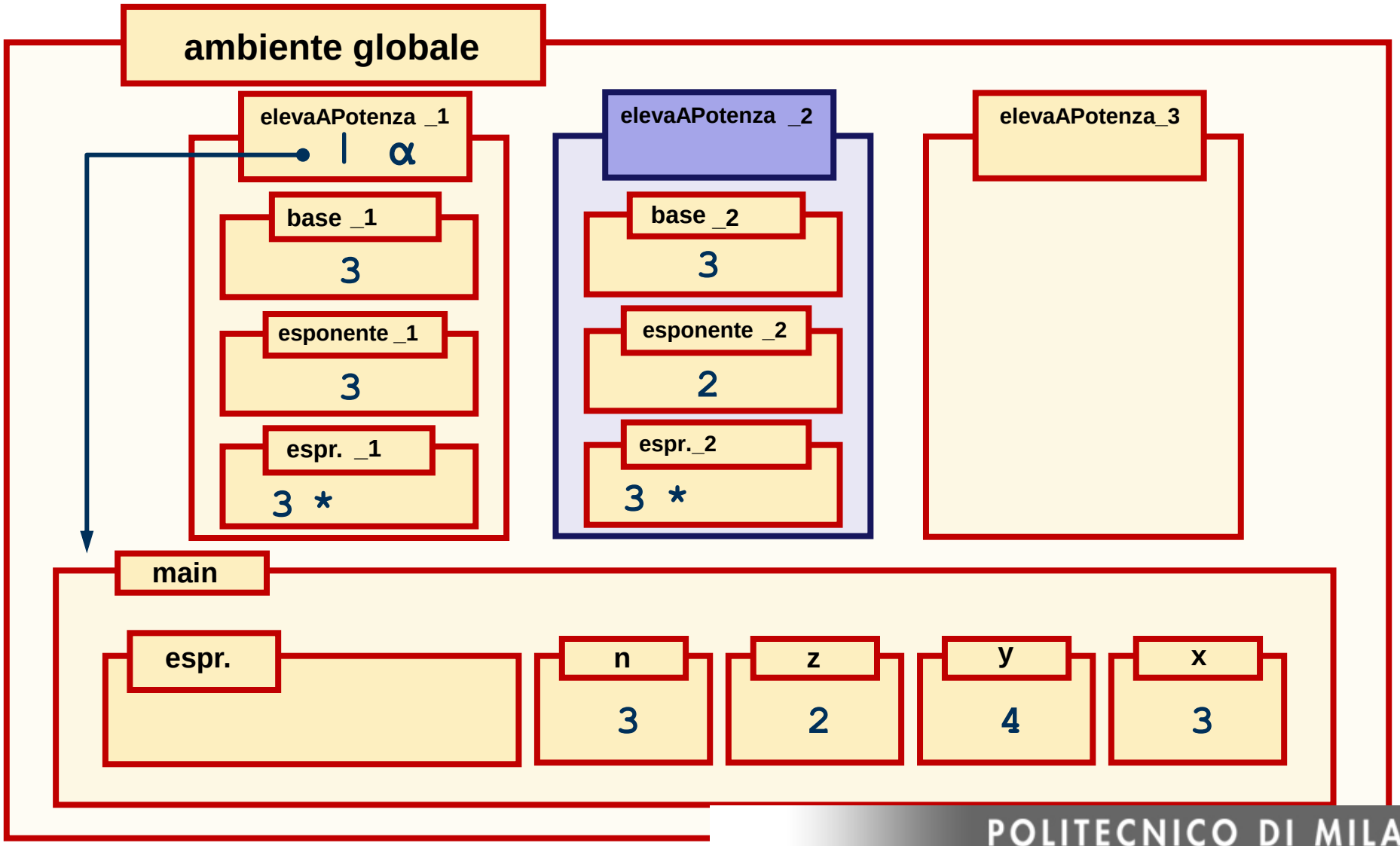


```
if (esponente > 1)
  return base * elevaAPotenza(base, esponente-1);
}
```

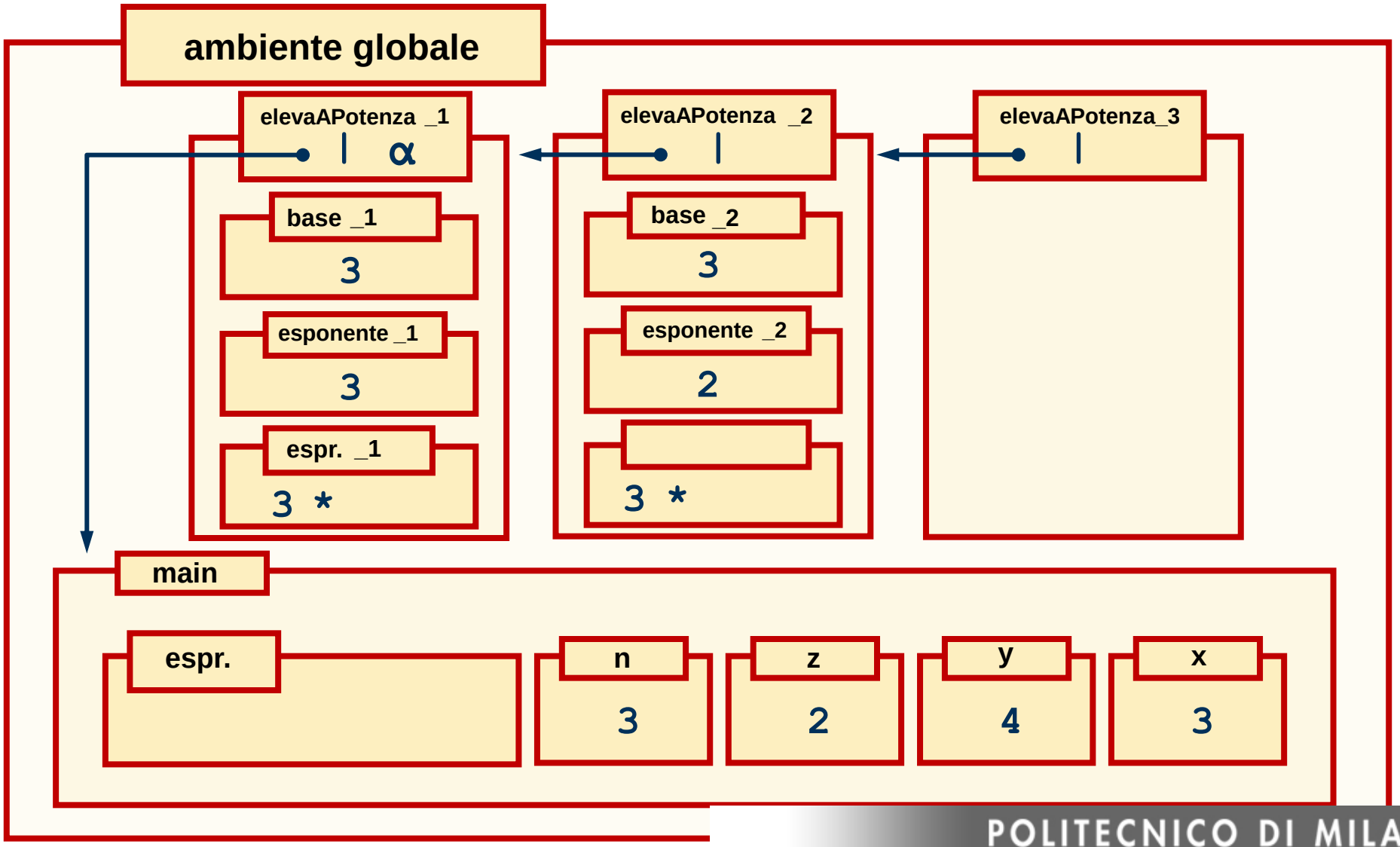


```
if (esponente > 1)
  return base * elevaAPotenza(base, esp
}
```

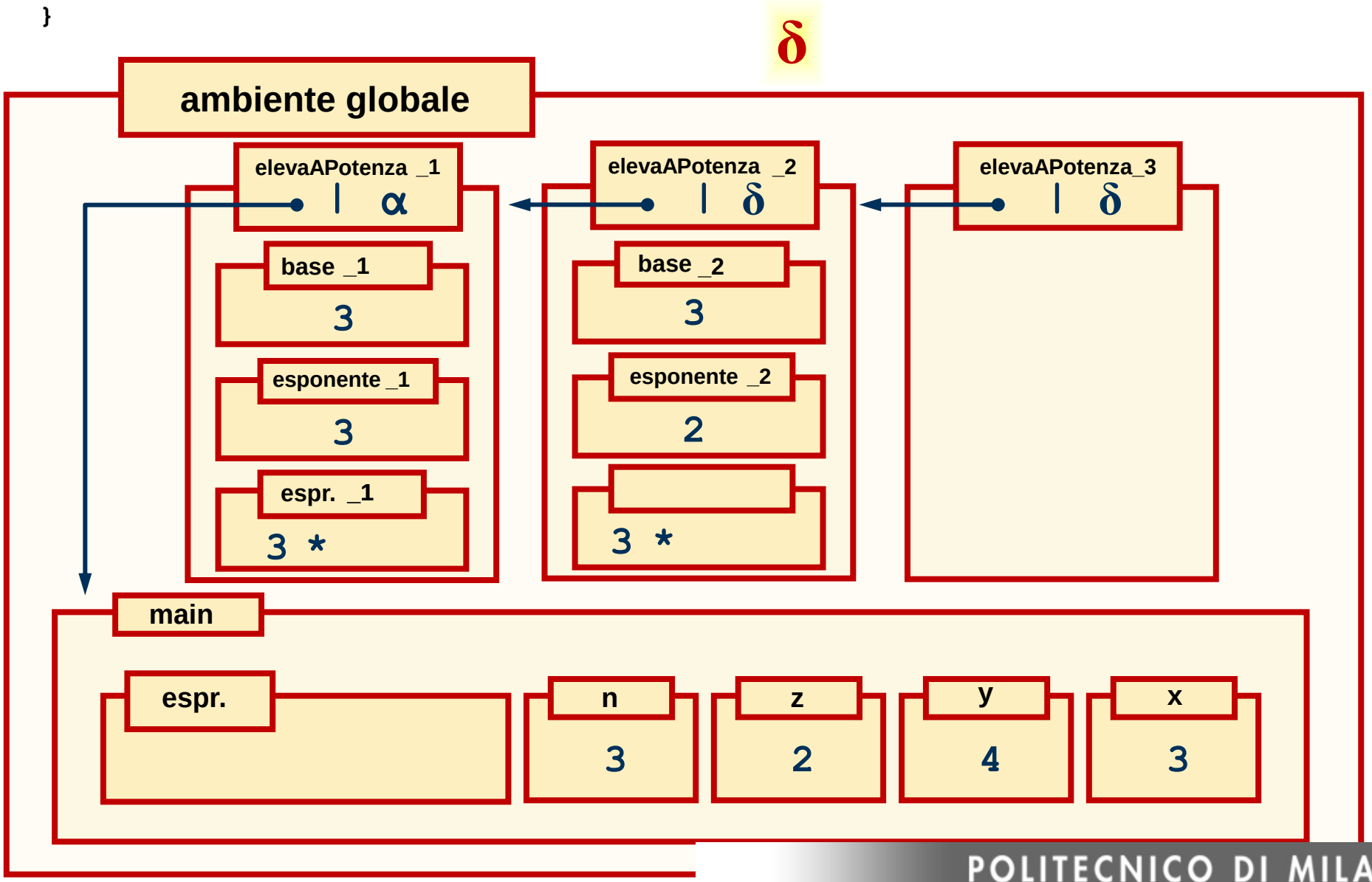
Informazioni di ritorno



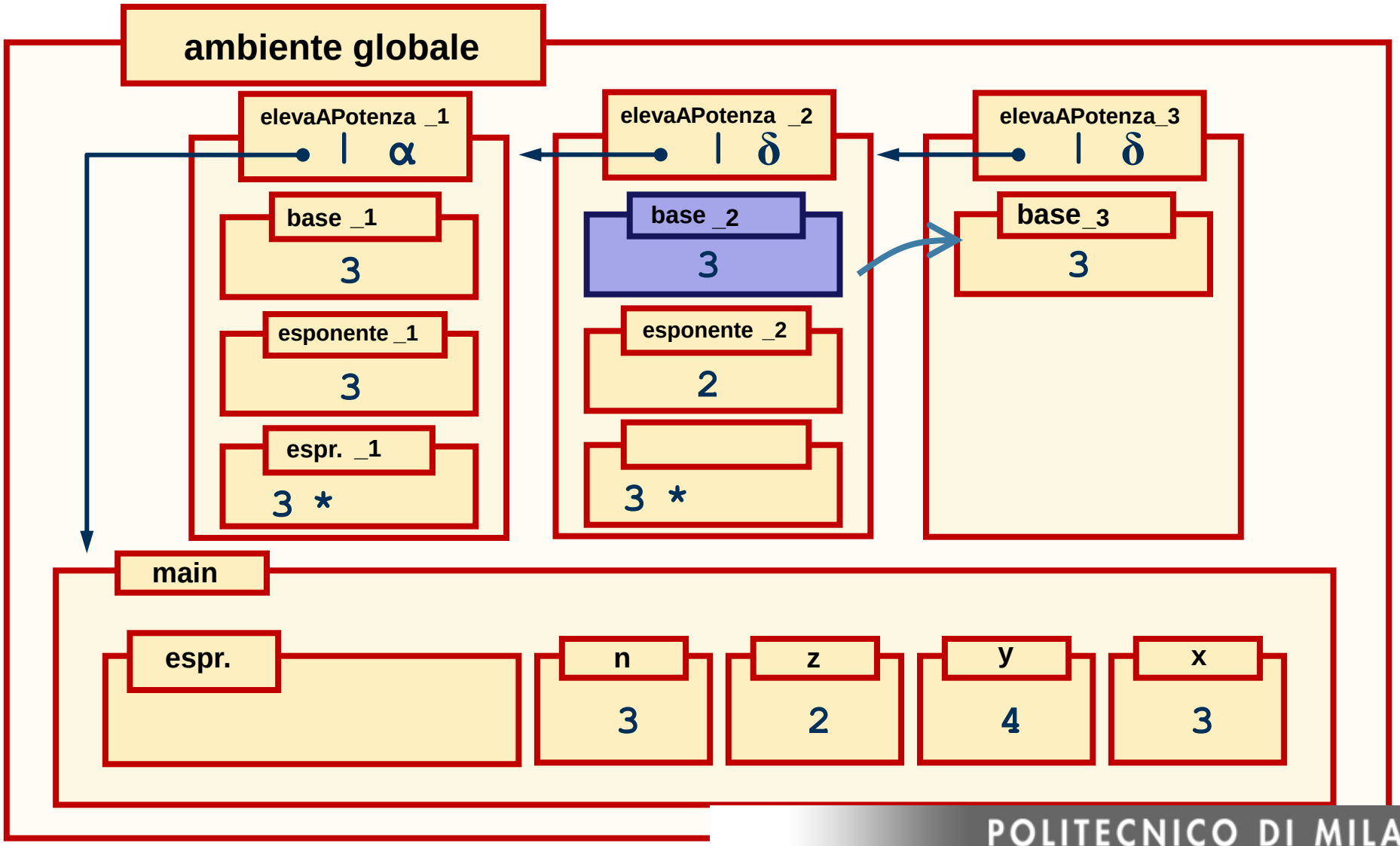
```
if (esponente > 1)
  return base * elevaAPotenza(base, esponente-1);
}
```



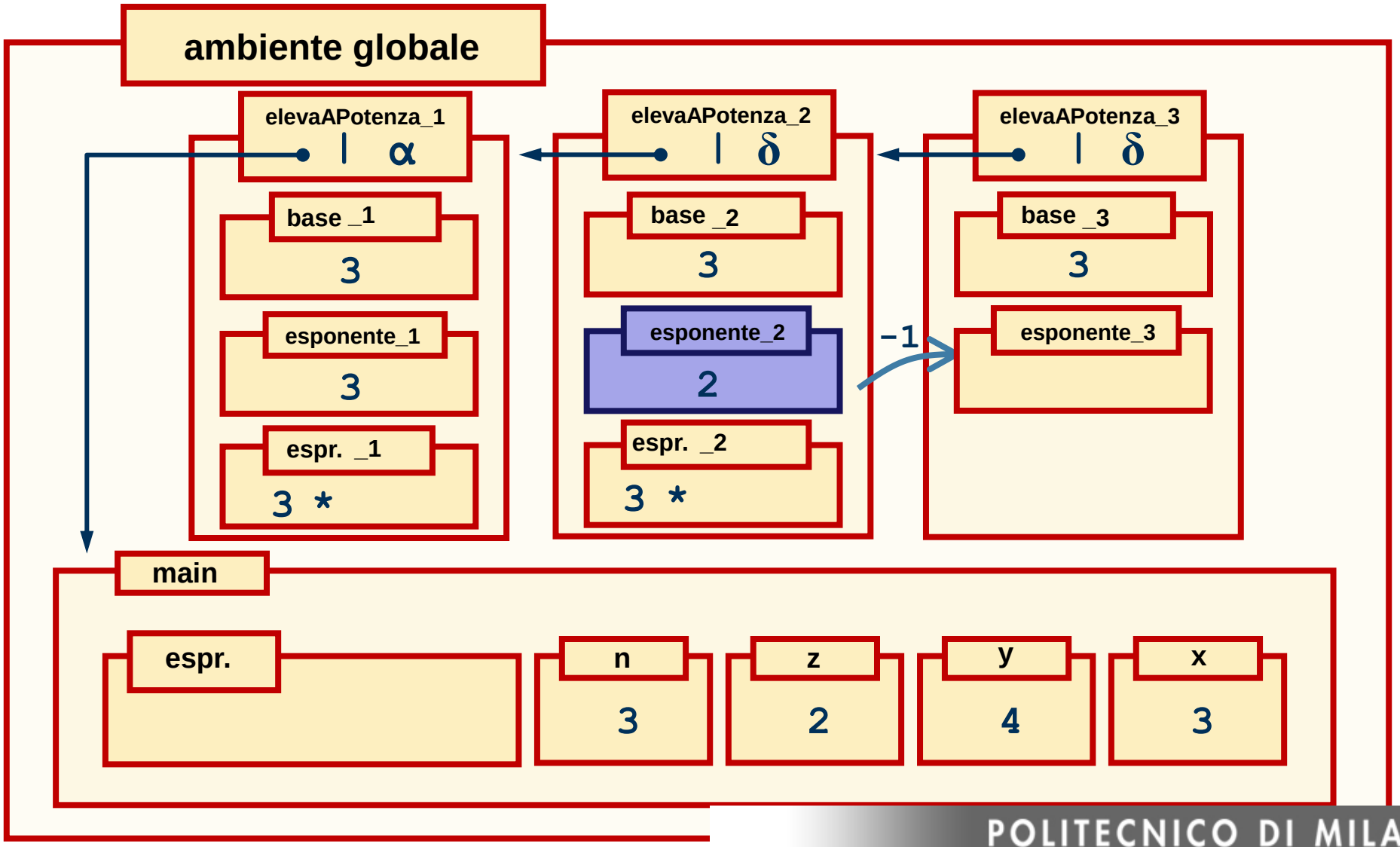
```
if (esponente > 1)
  return base * elevaAPotenza(base, esponente-1);
}
```



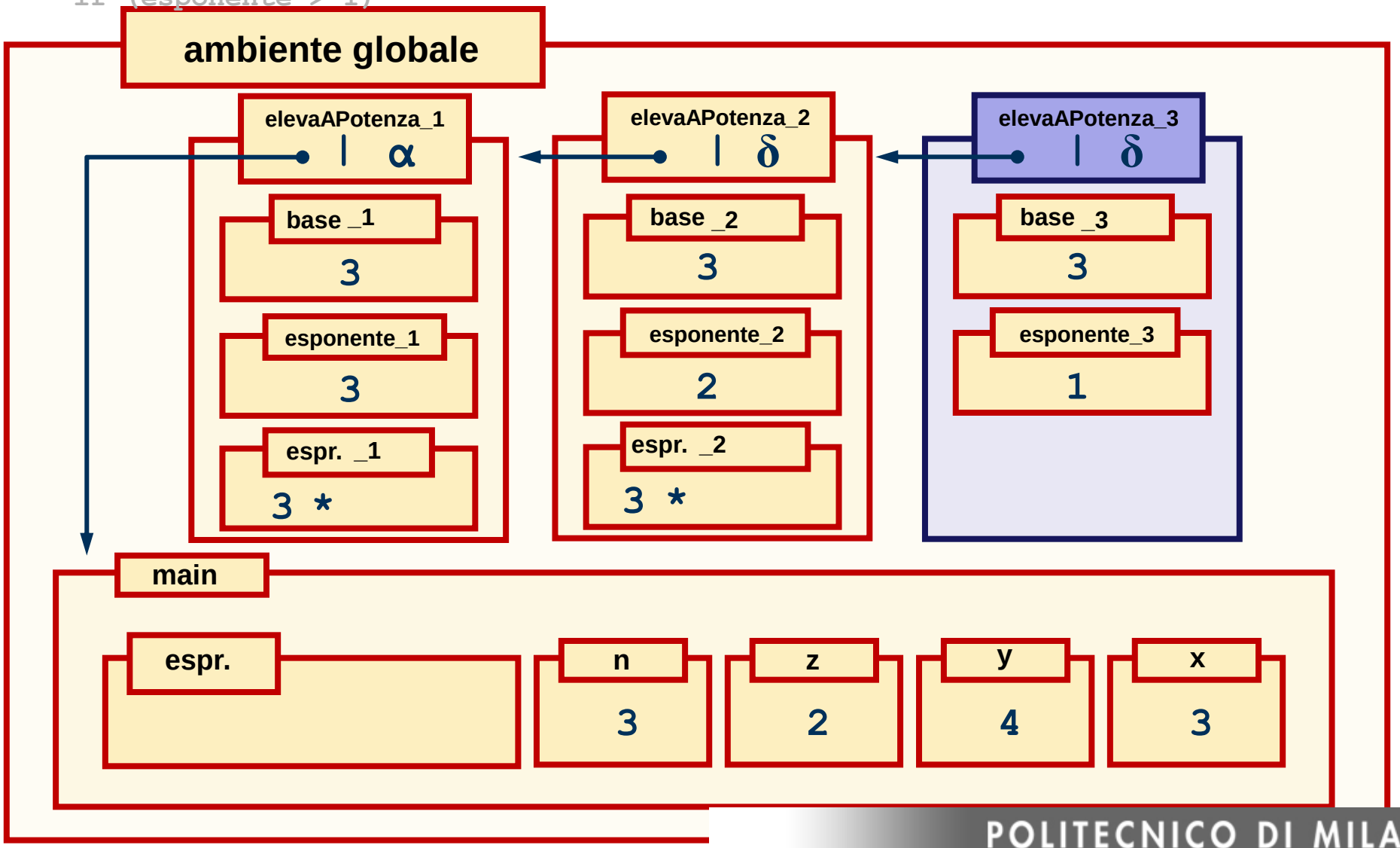
```
if (esponente > 1)
  return base * elevaAPotenza(base, esponente-1);
}
```



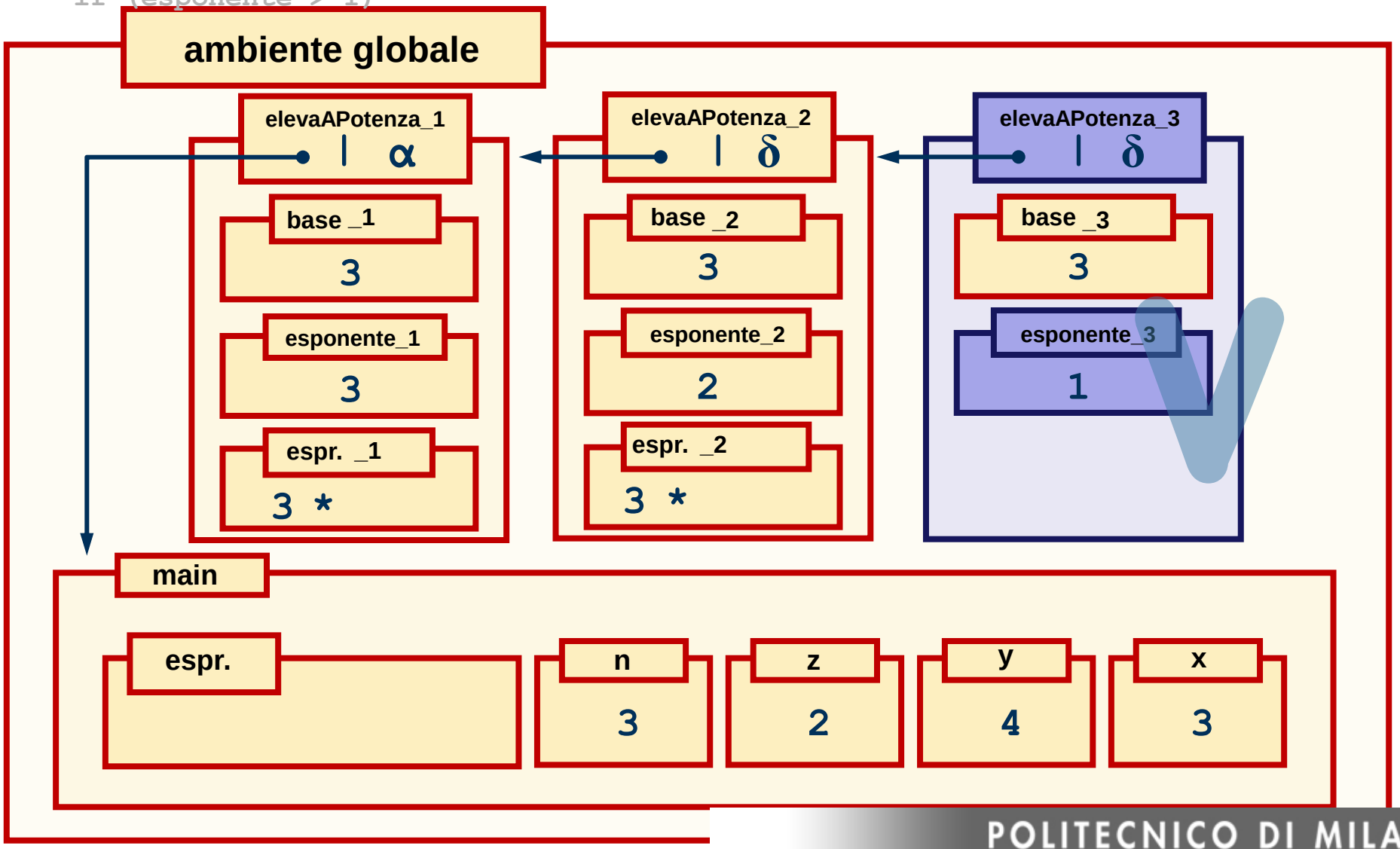
```
if (esponente > 1)
  return base * elevaAPotenza(base, esponente-1);
}
```



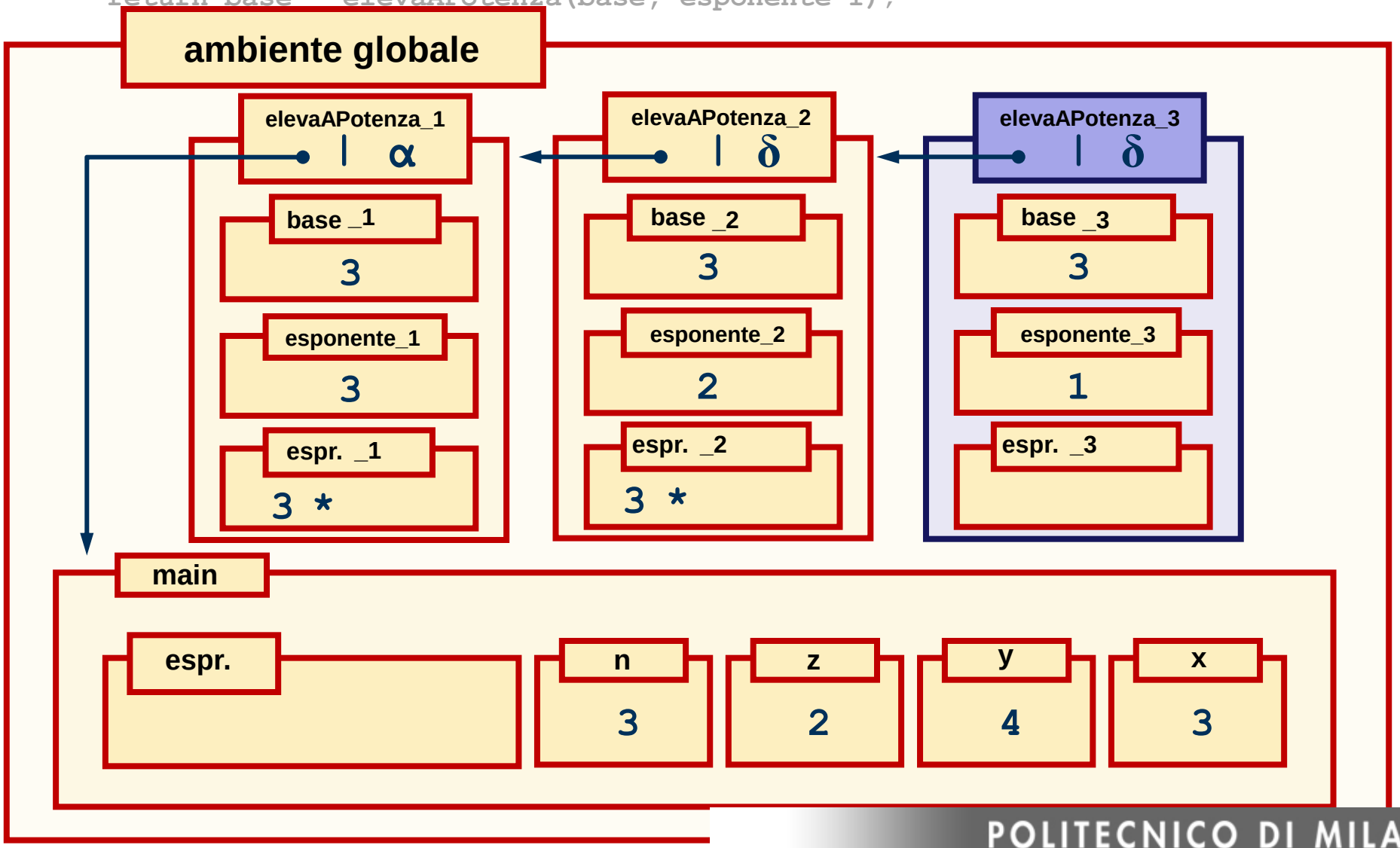
```
int elevaAPotenza(int base, int esponente)  // versione ricorsiva
{ if (esponente == 1)
    return base;
  if (esponente > 1)
```



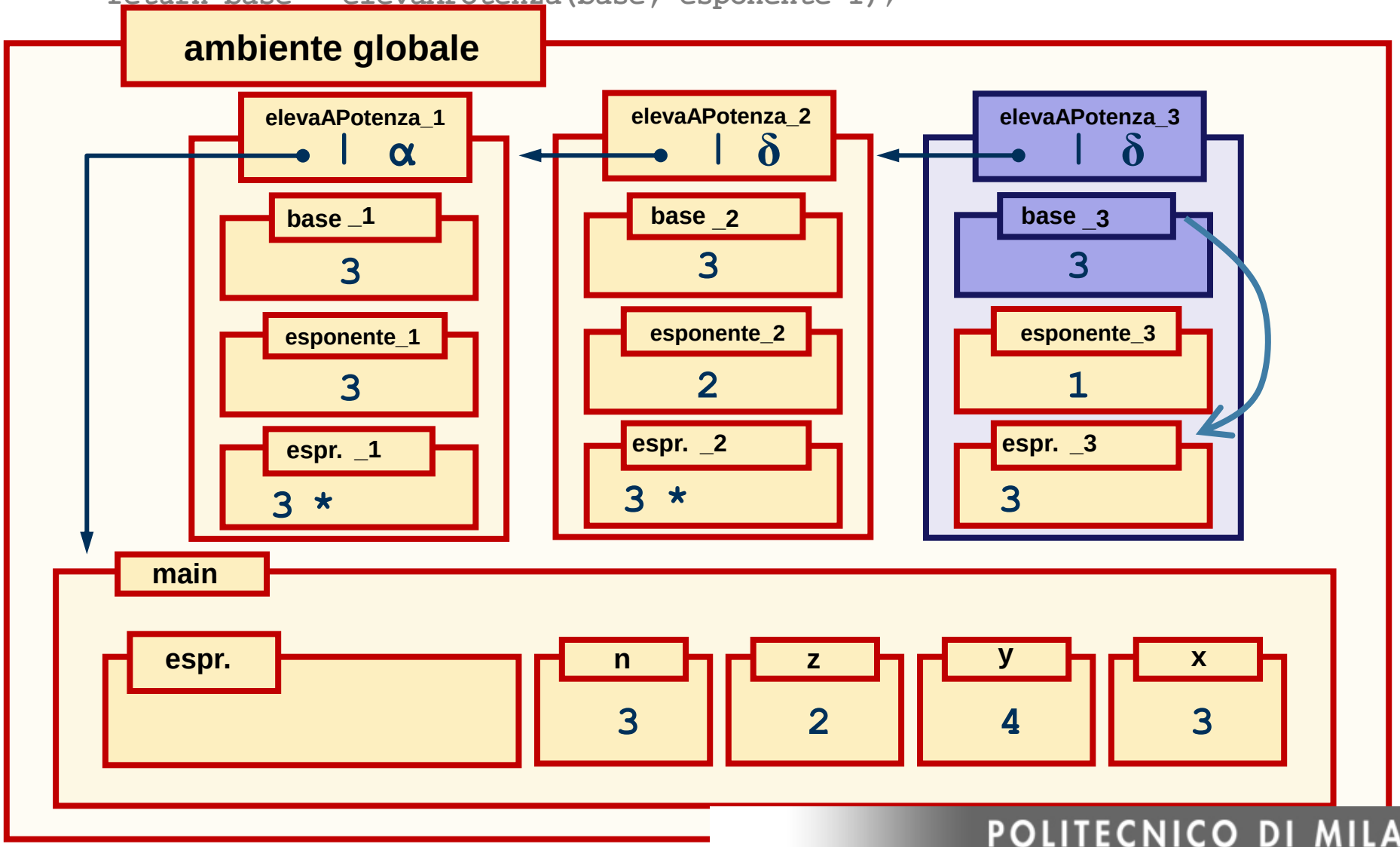
```
int elevaAPotenza(int base, int esponente)   // versione ricorsiva
{ if (esponente == 1)
    return base;
  if (esponente > 1)
```



```
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
```

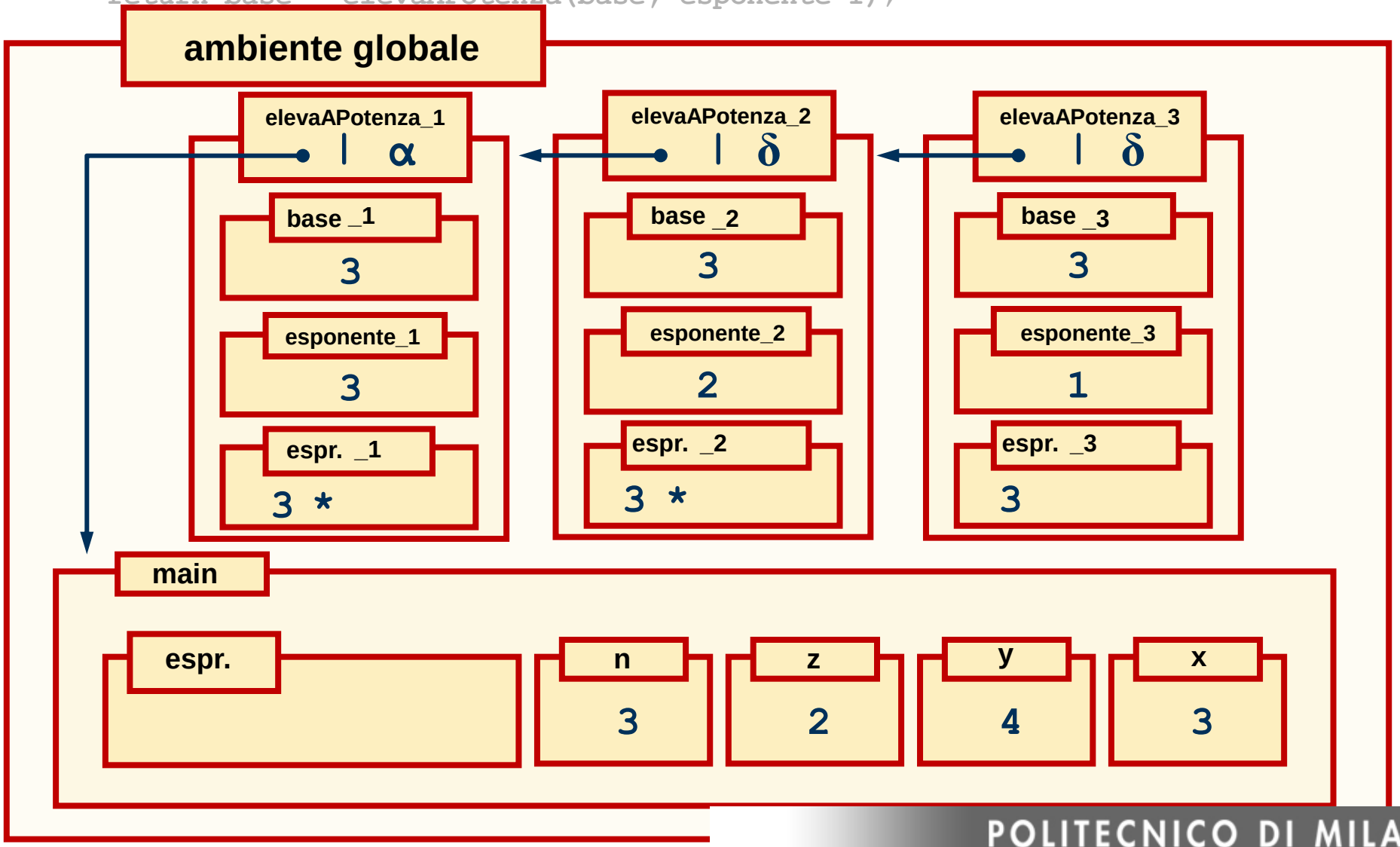


```
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
```

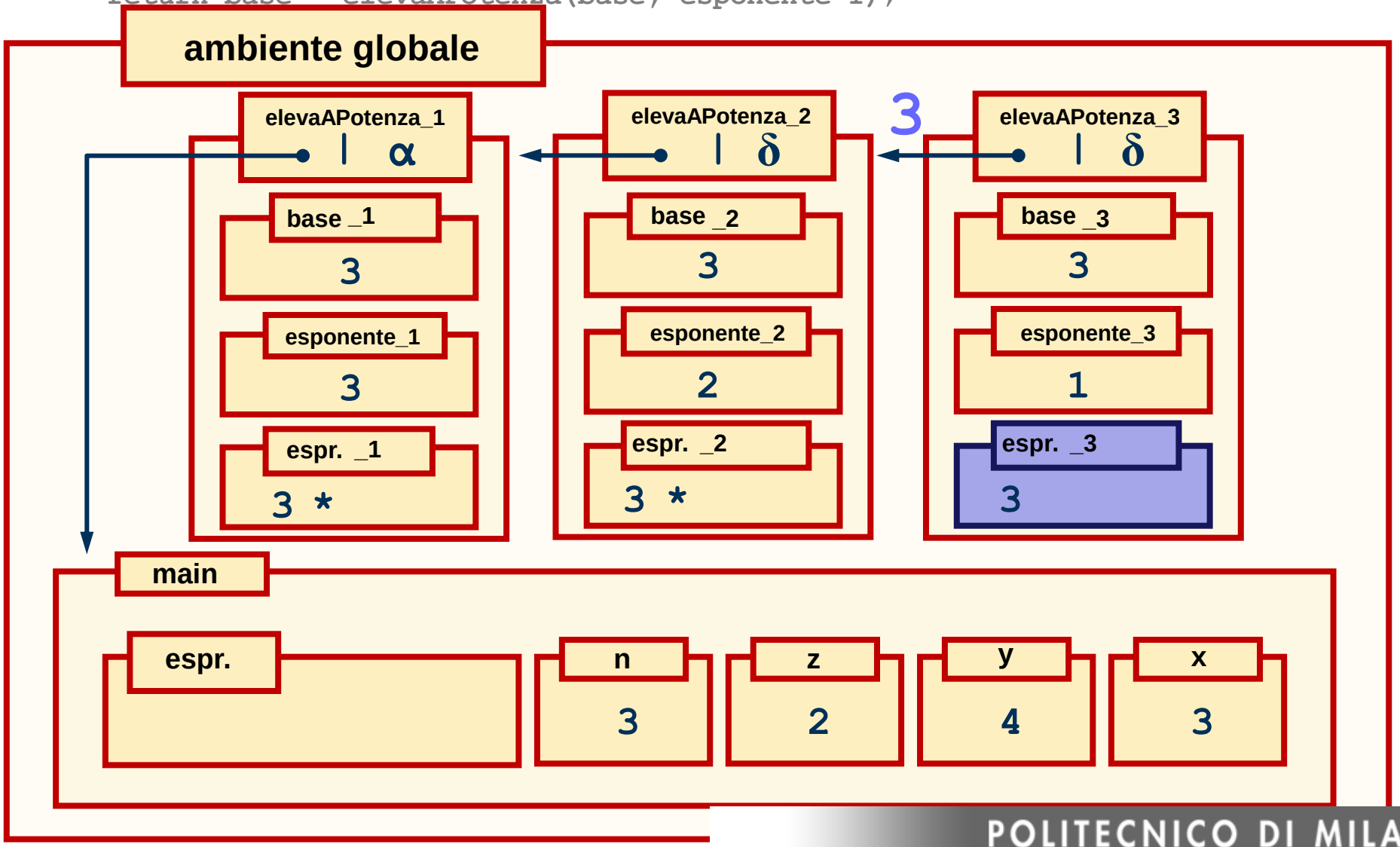


```
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
```

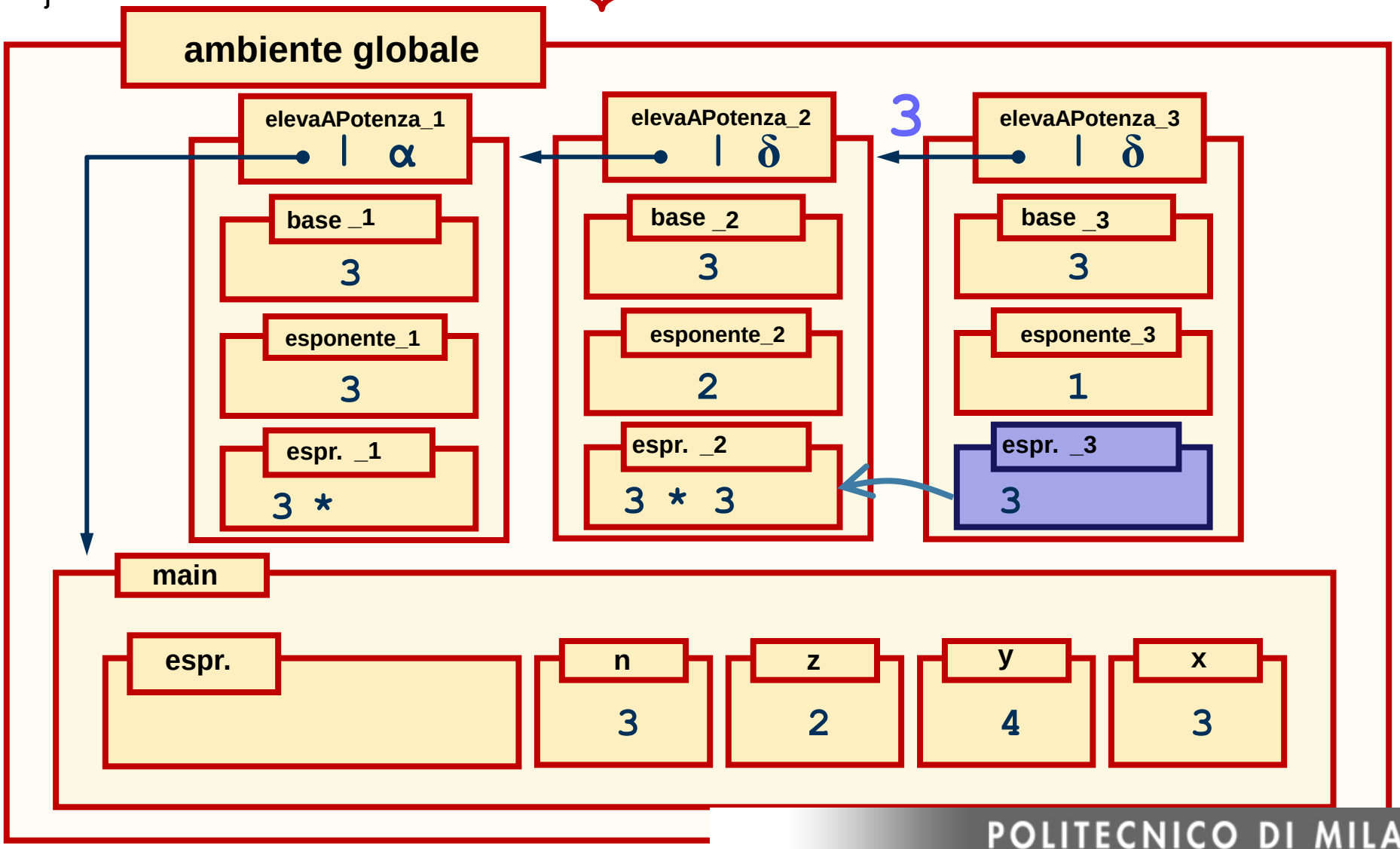
La catena ricorsiva termina



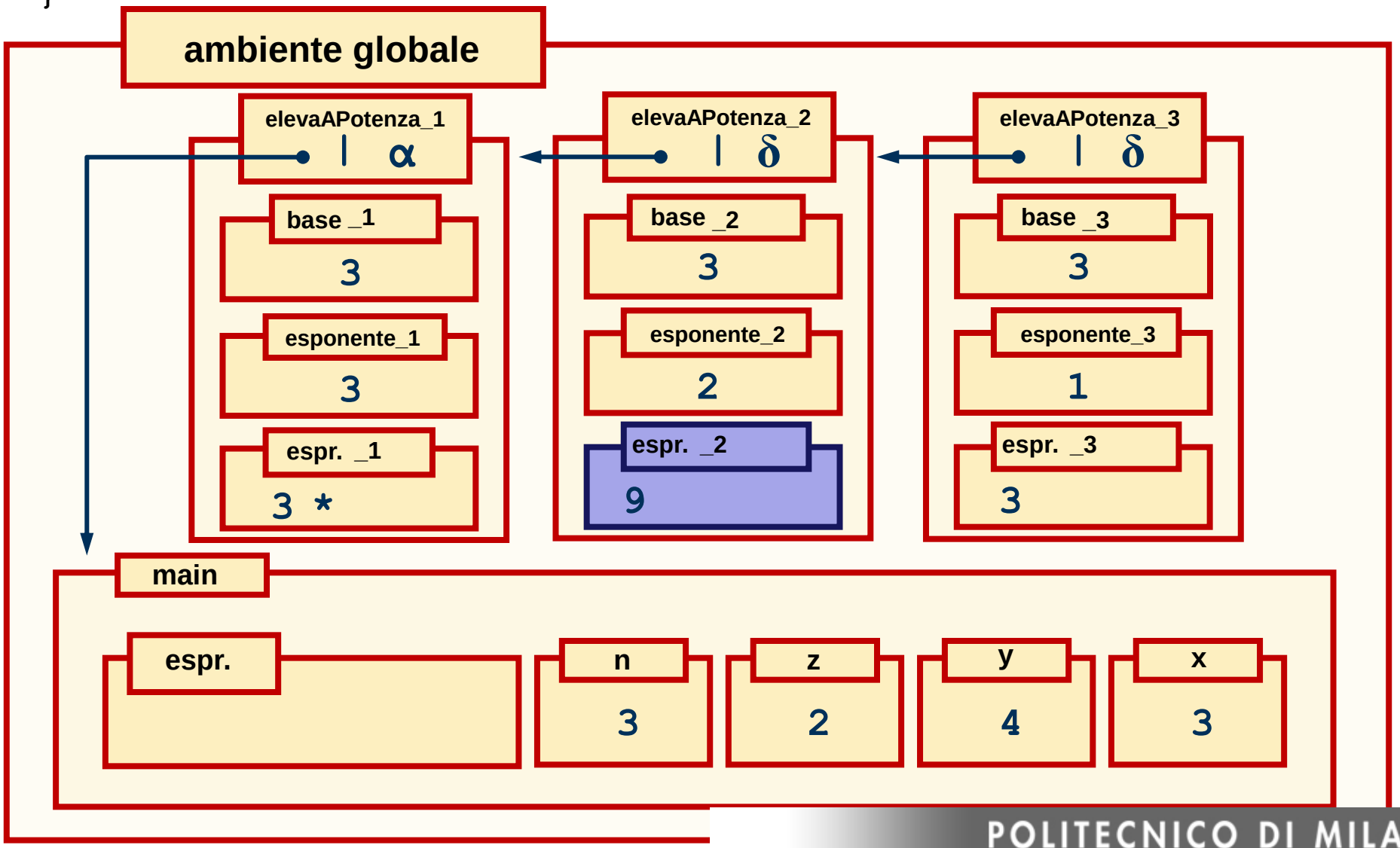
```
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
```



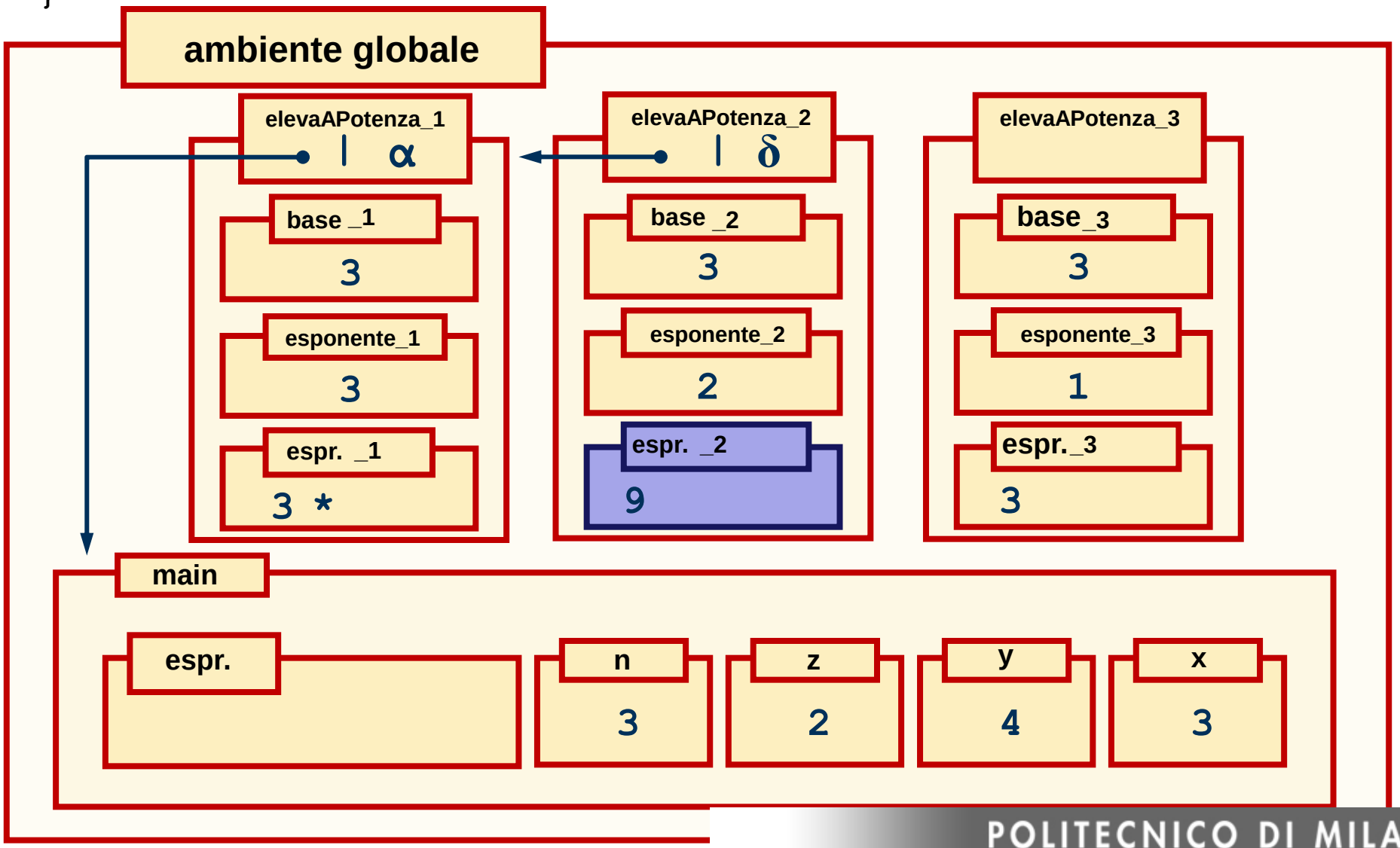
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

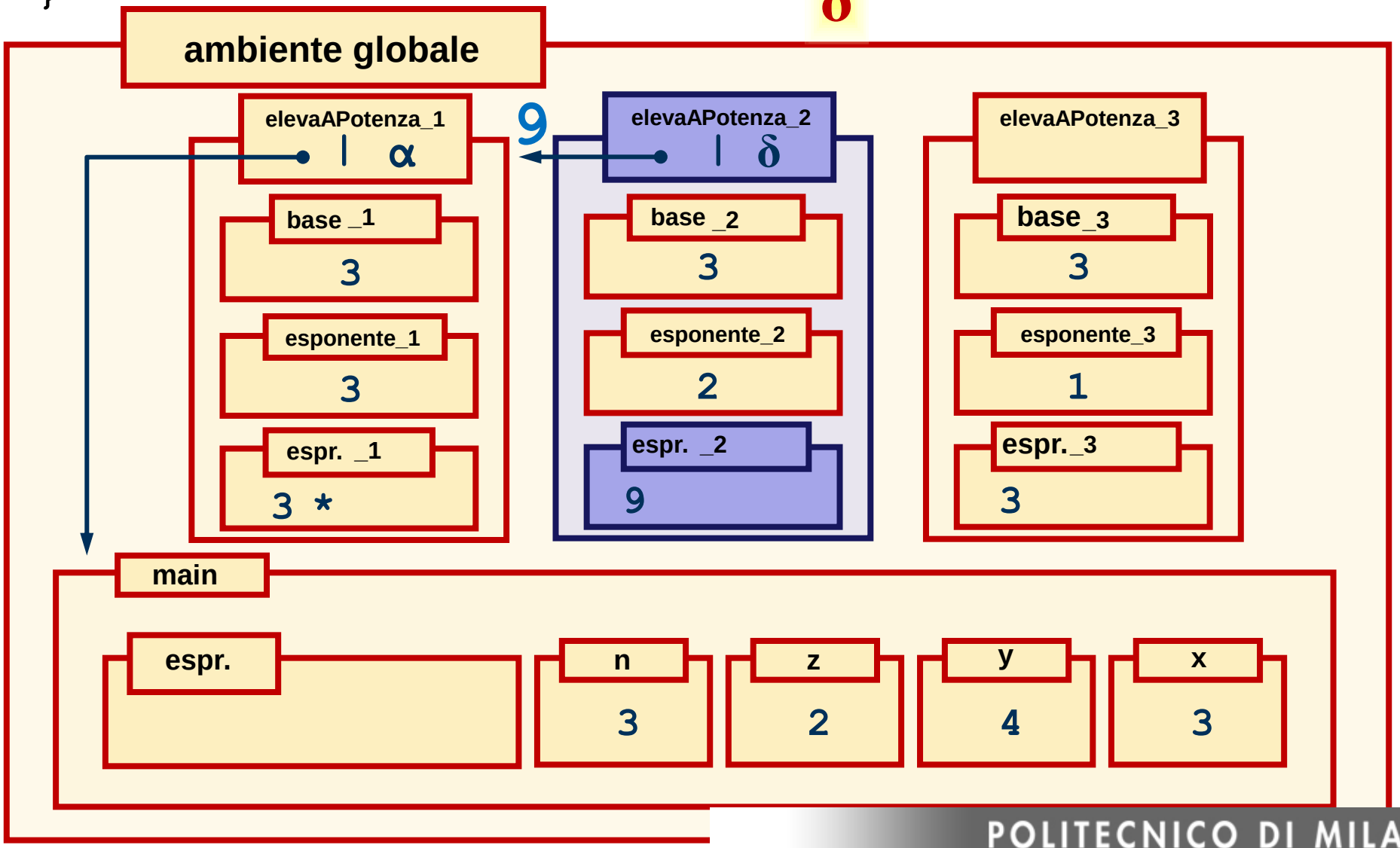


```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

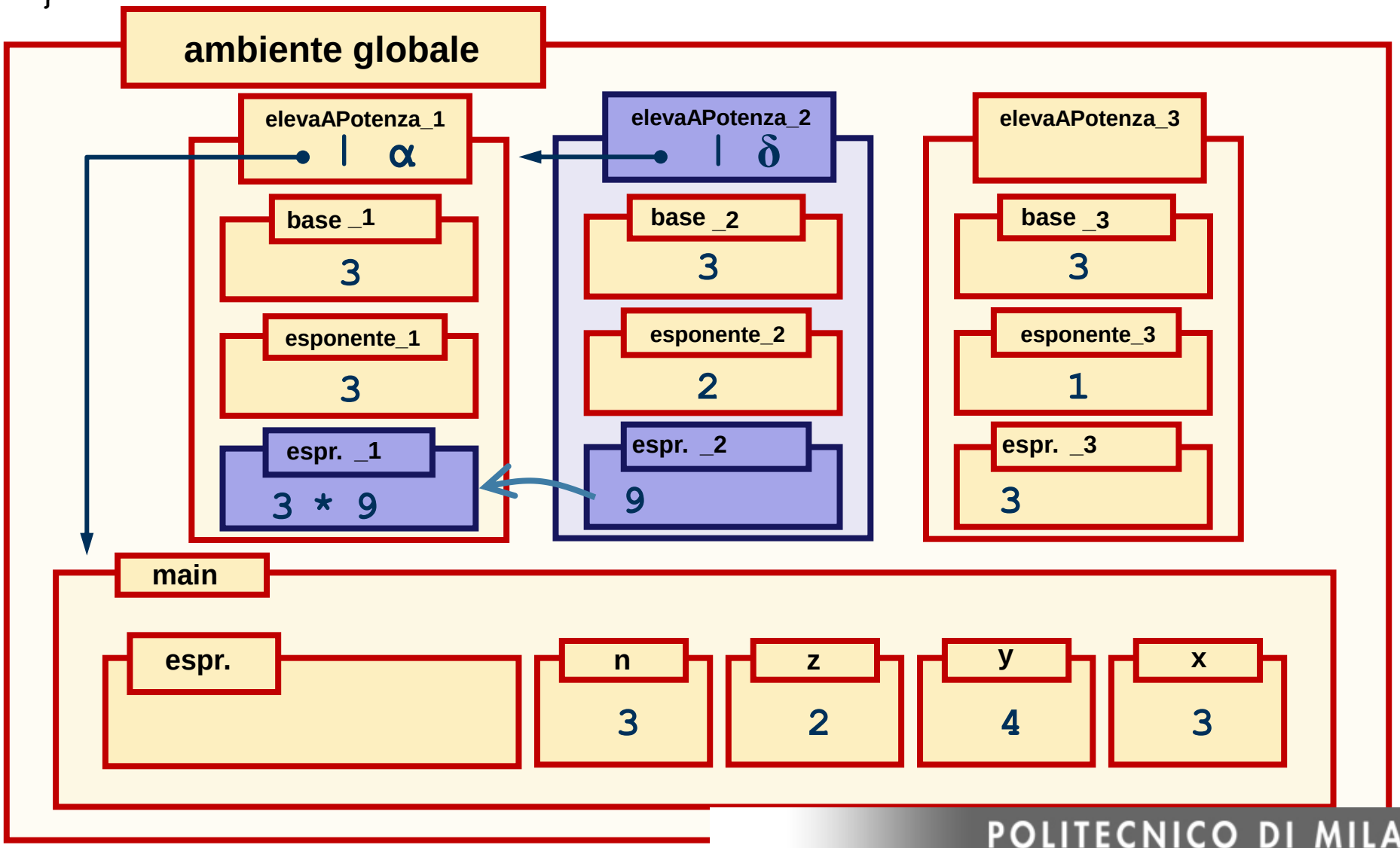


```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

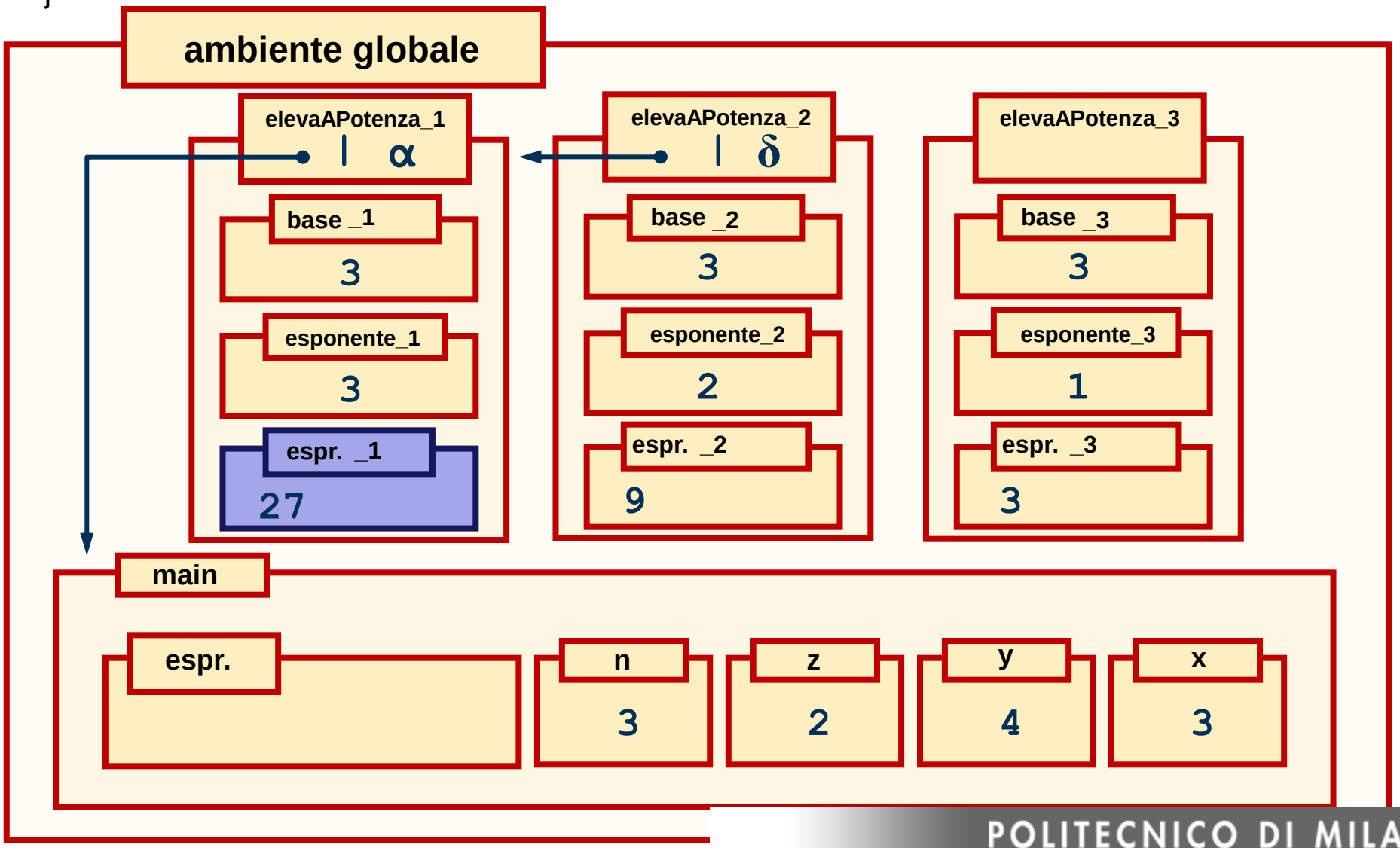
δ



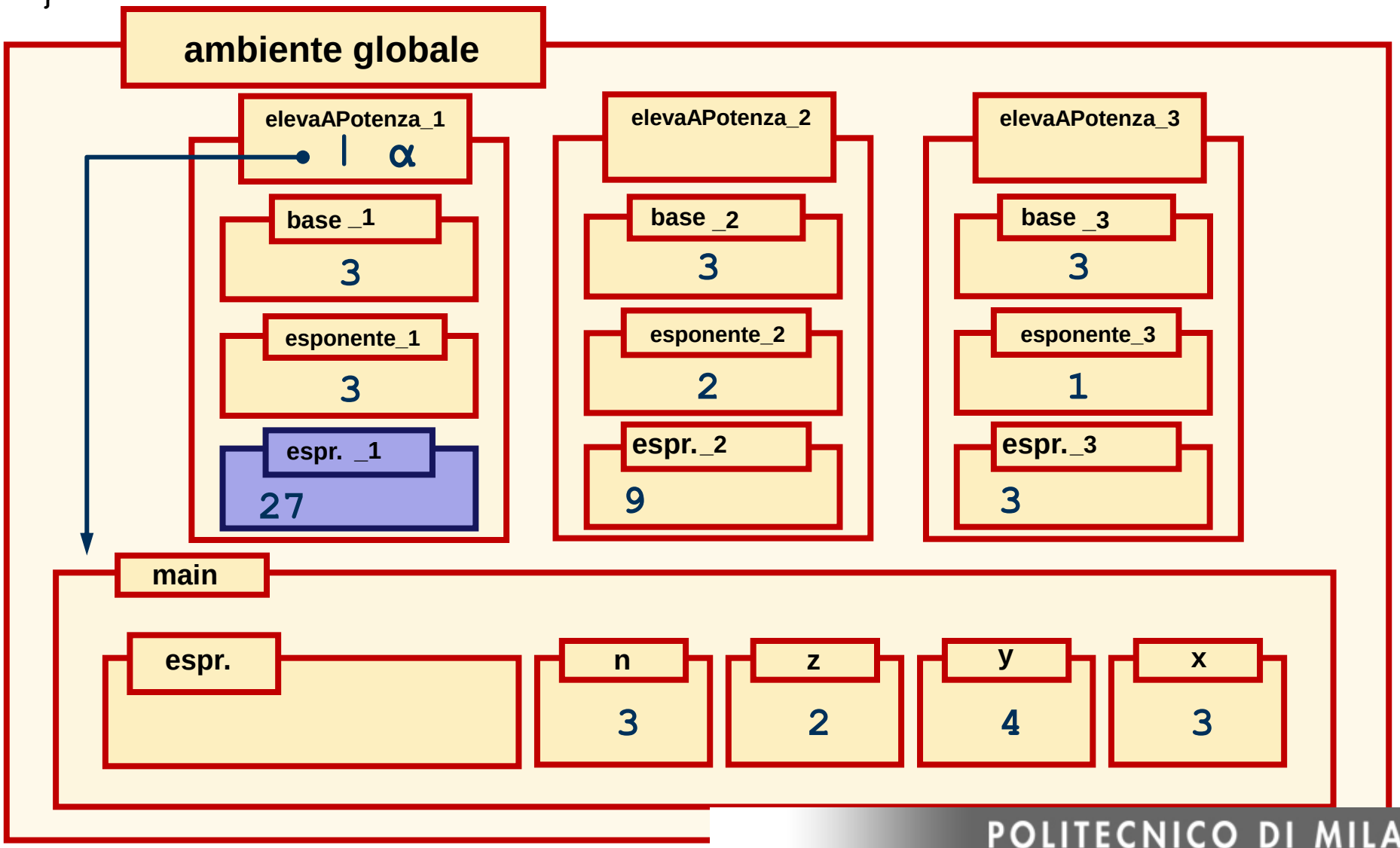
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



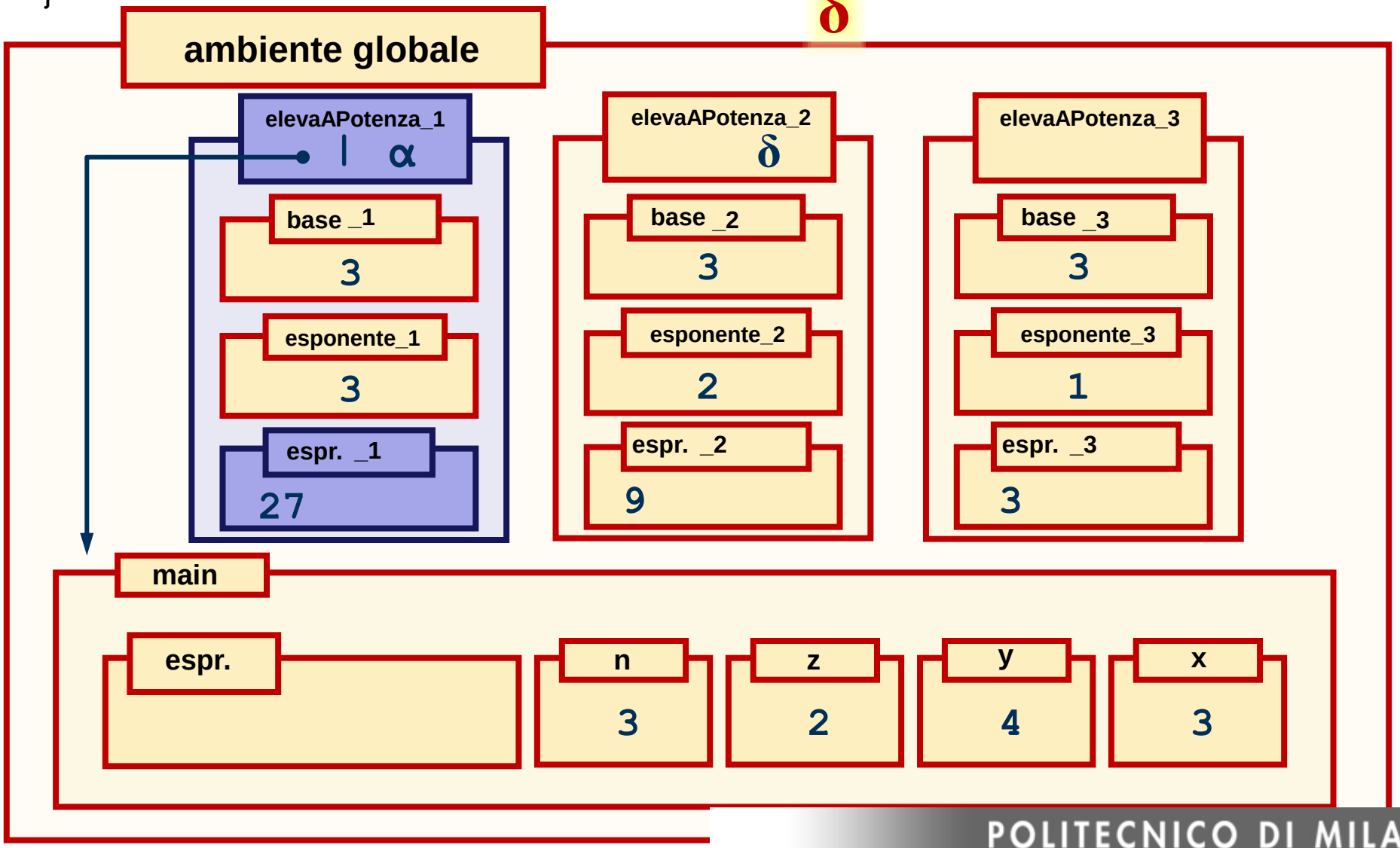
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



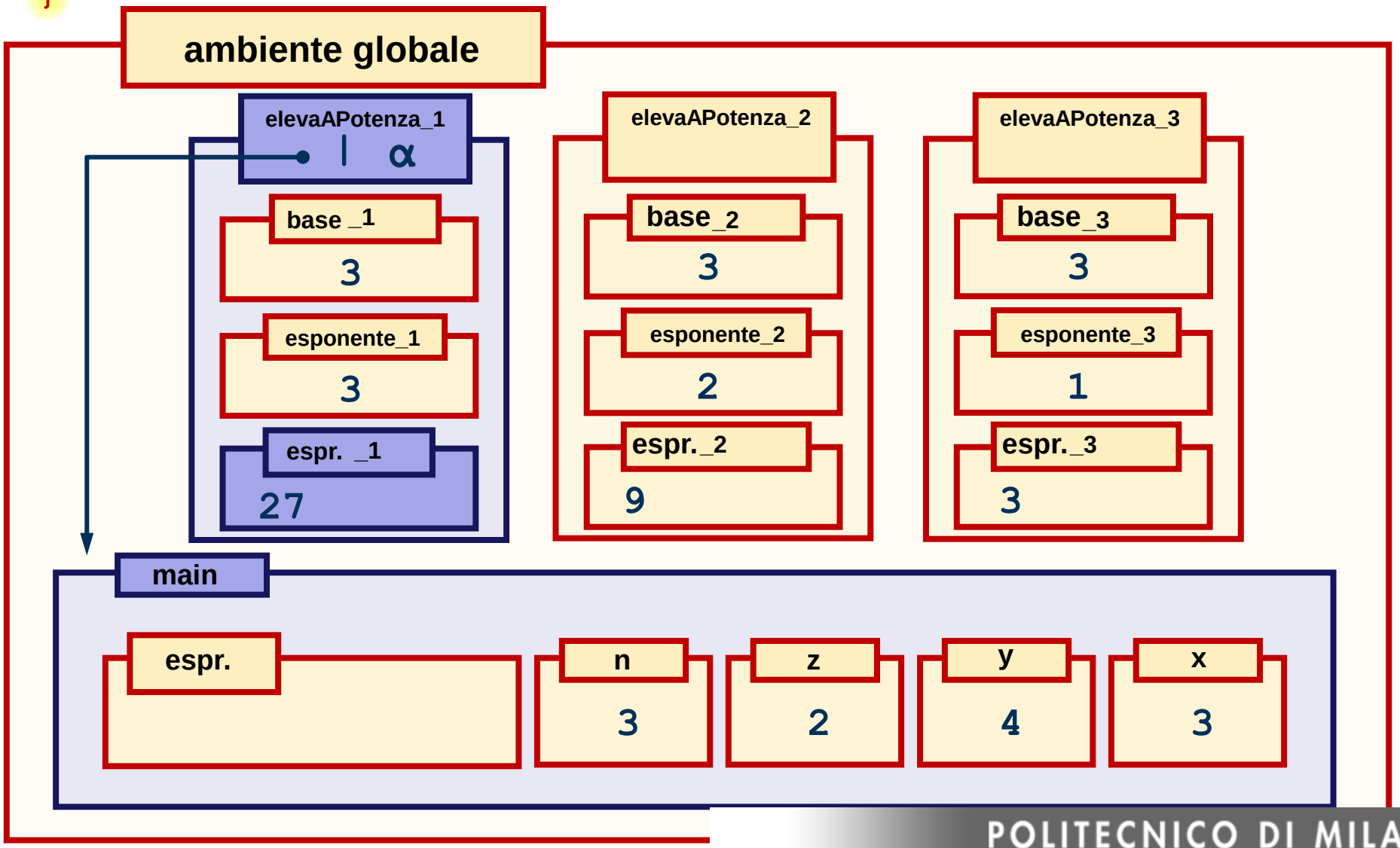
```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

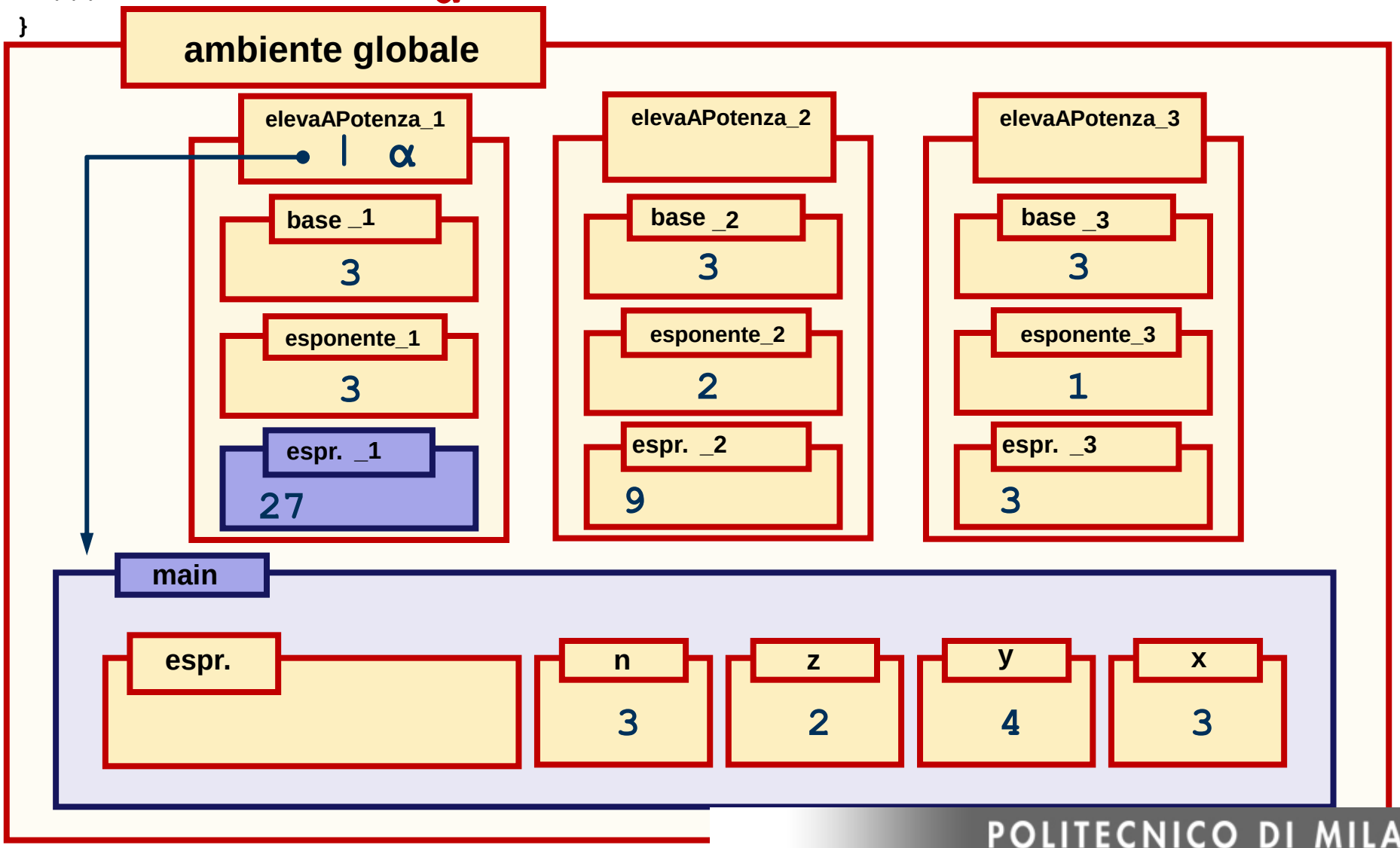


```
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

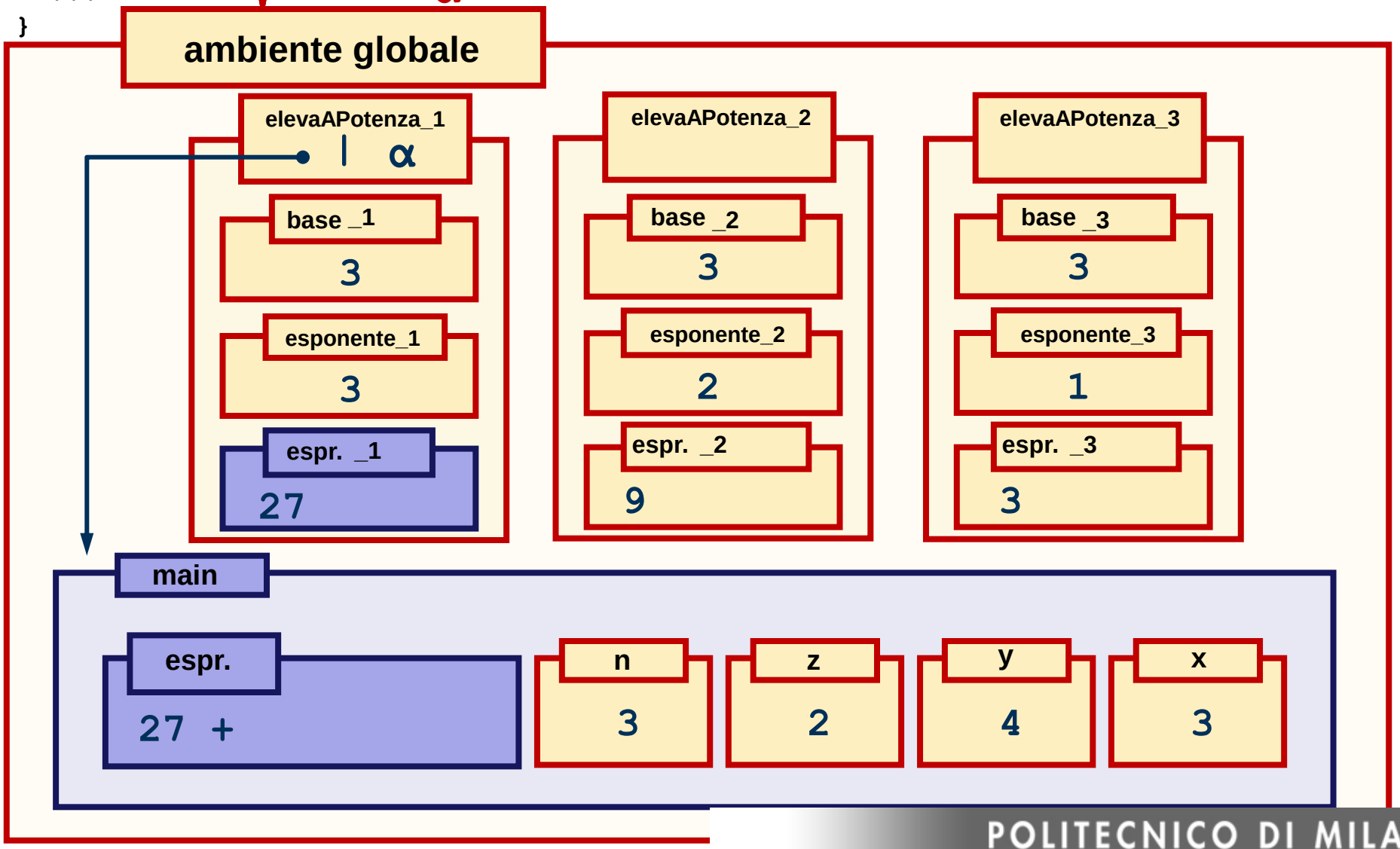


```
"Presenta le funzionalità del programma"
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
}
```

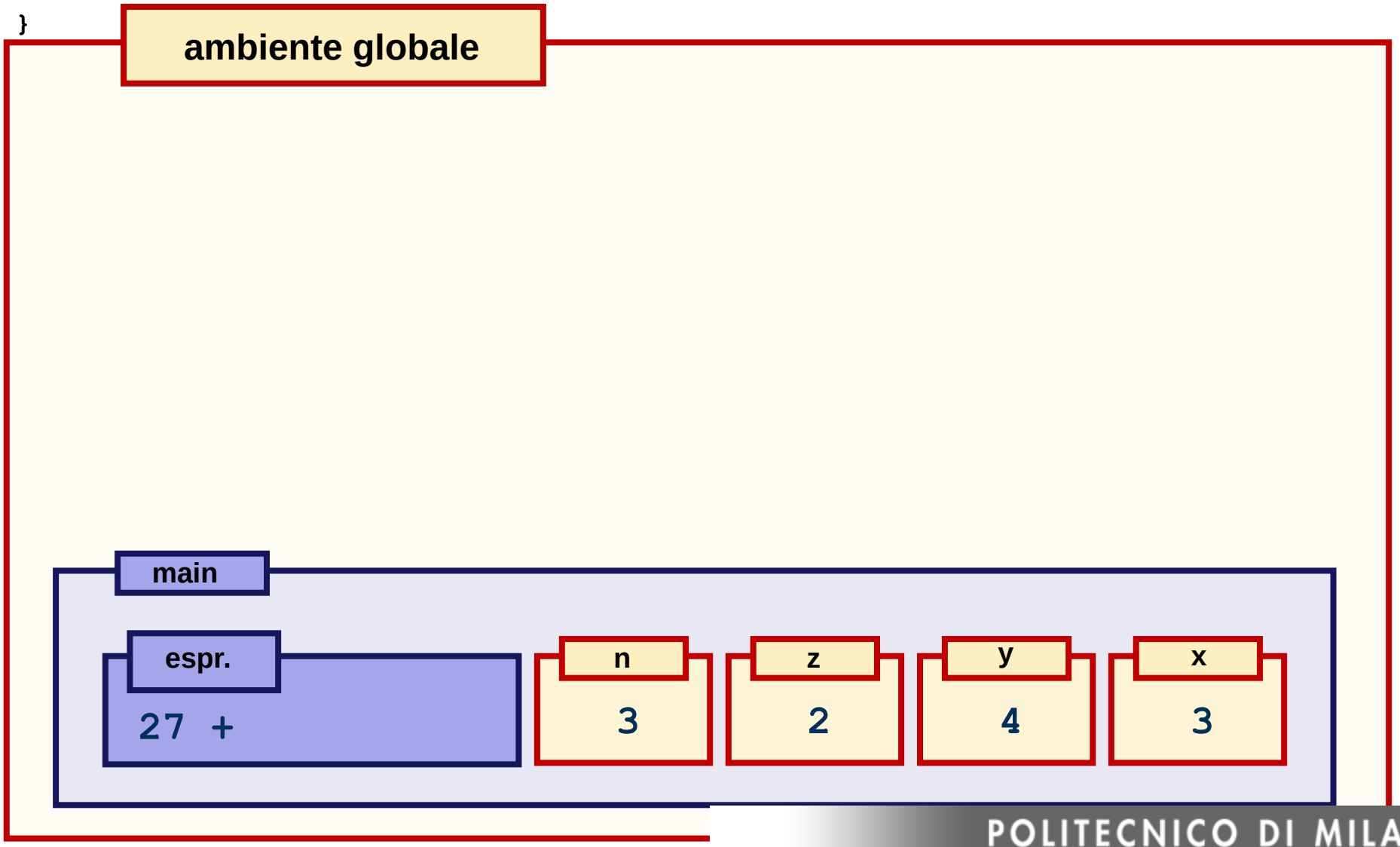
α



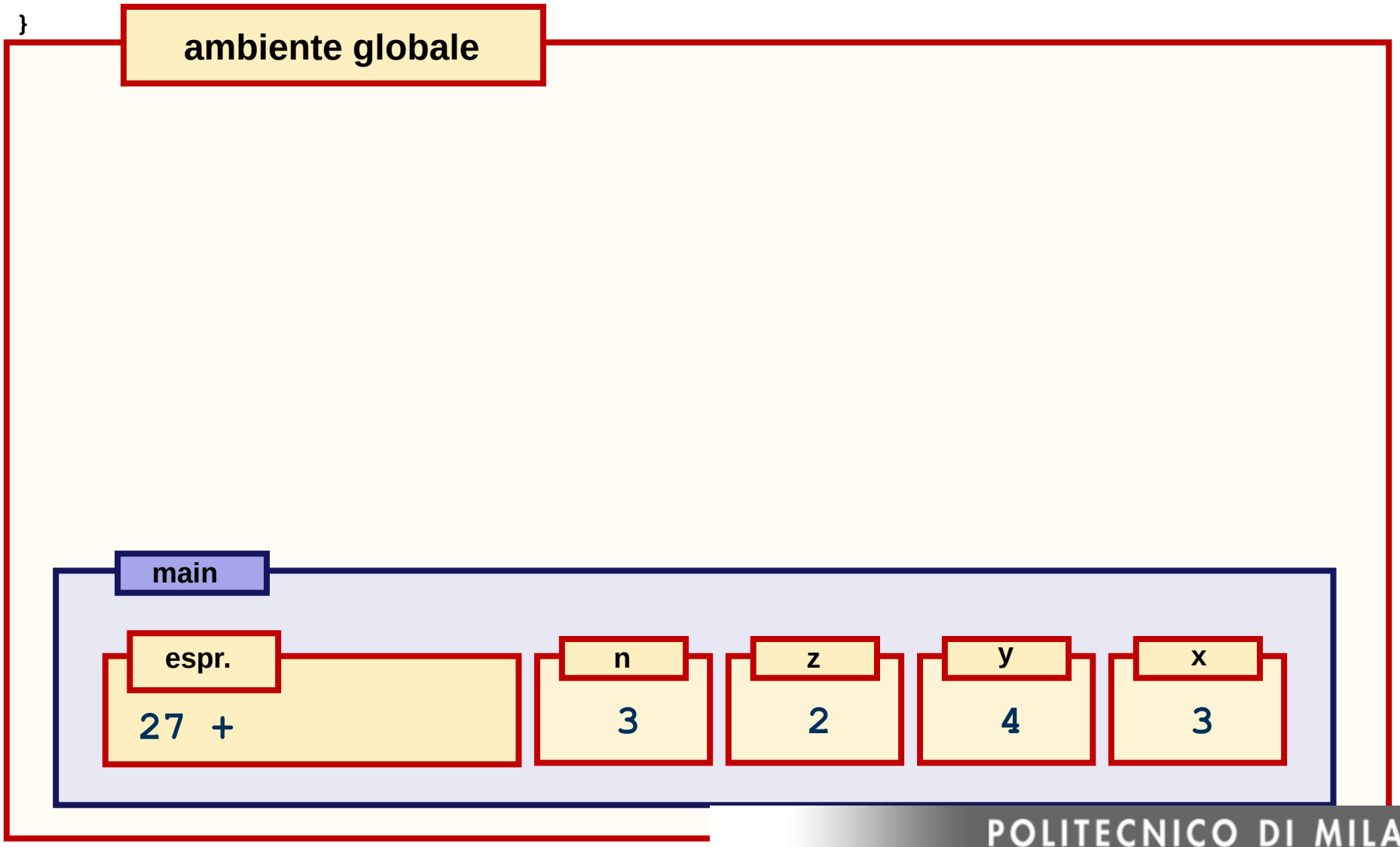
```
"Presenta le funzionalità del programma"
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
}
```



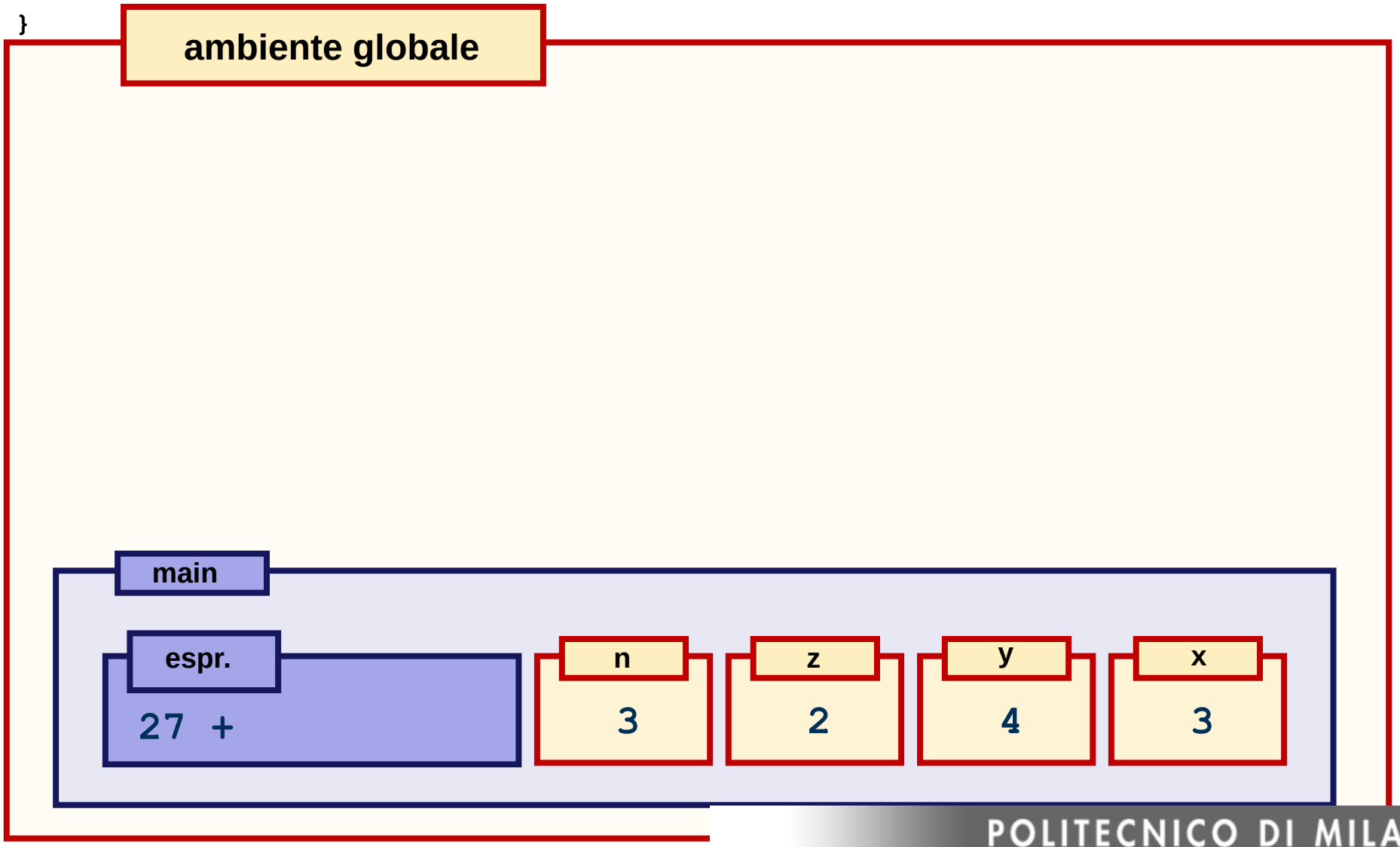
```
  "Presenta le funzionalità del programma"
  "Leggi i dati e verifica che rispondano alle specifiche"
  if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
  ...
}
```



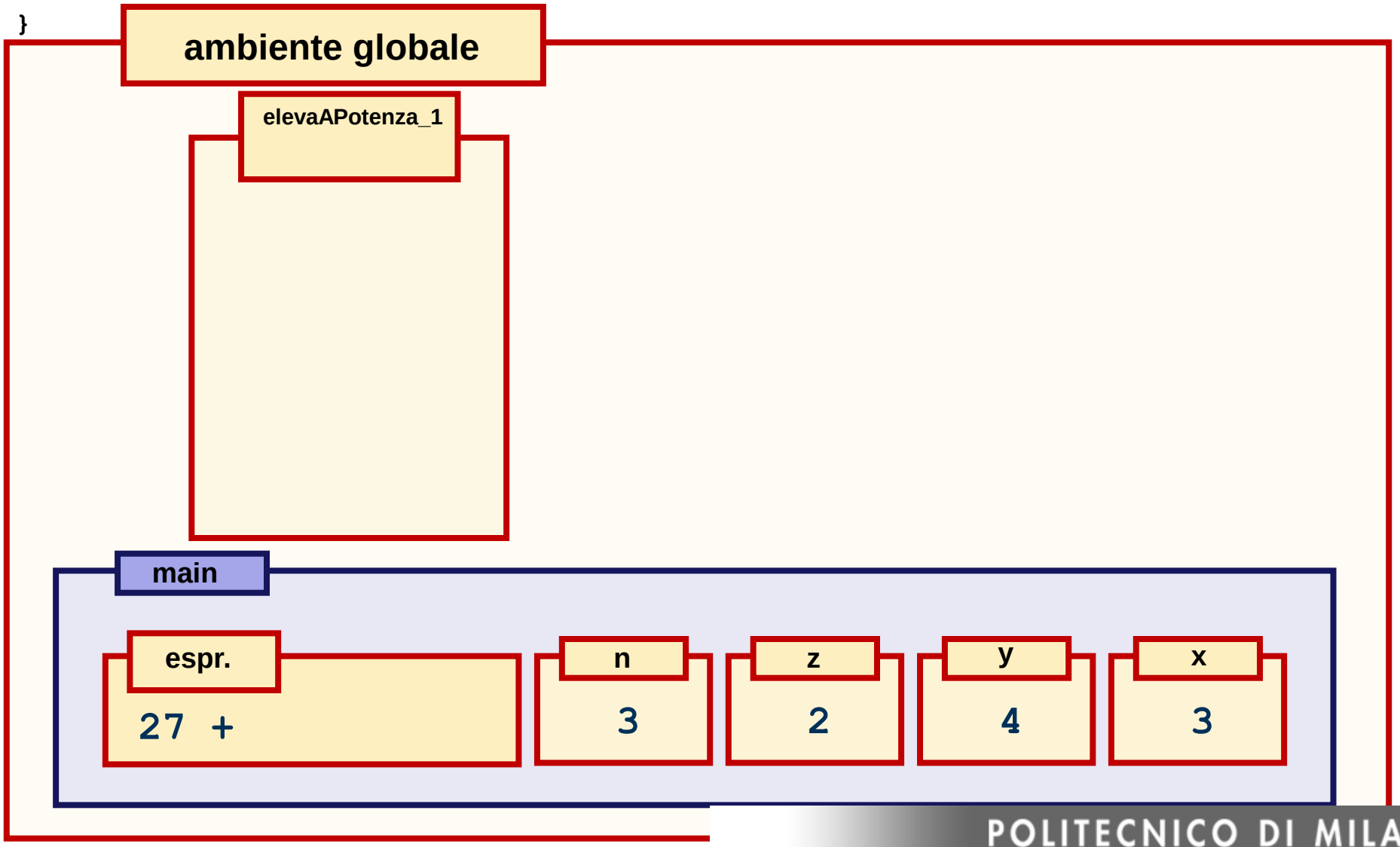
```
   "Presenta le funzionalità del programma"
   "Leggi i dati e verifica che rispondano alle specifiche"
   if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
   ...
}
```



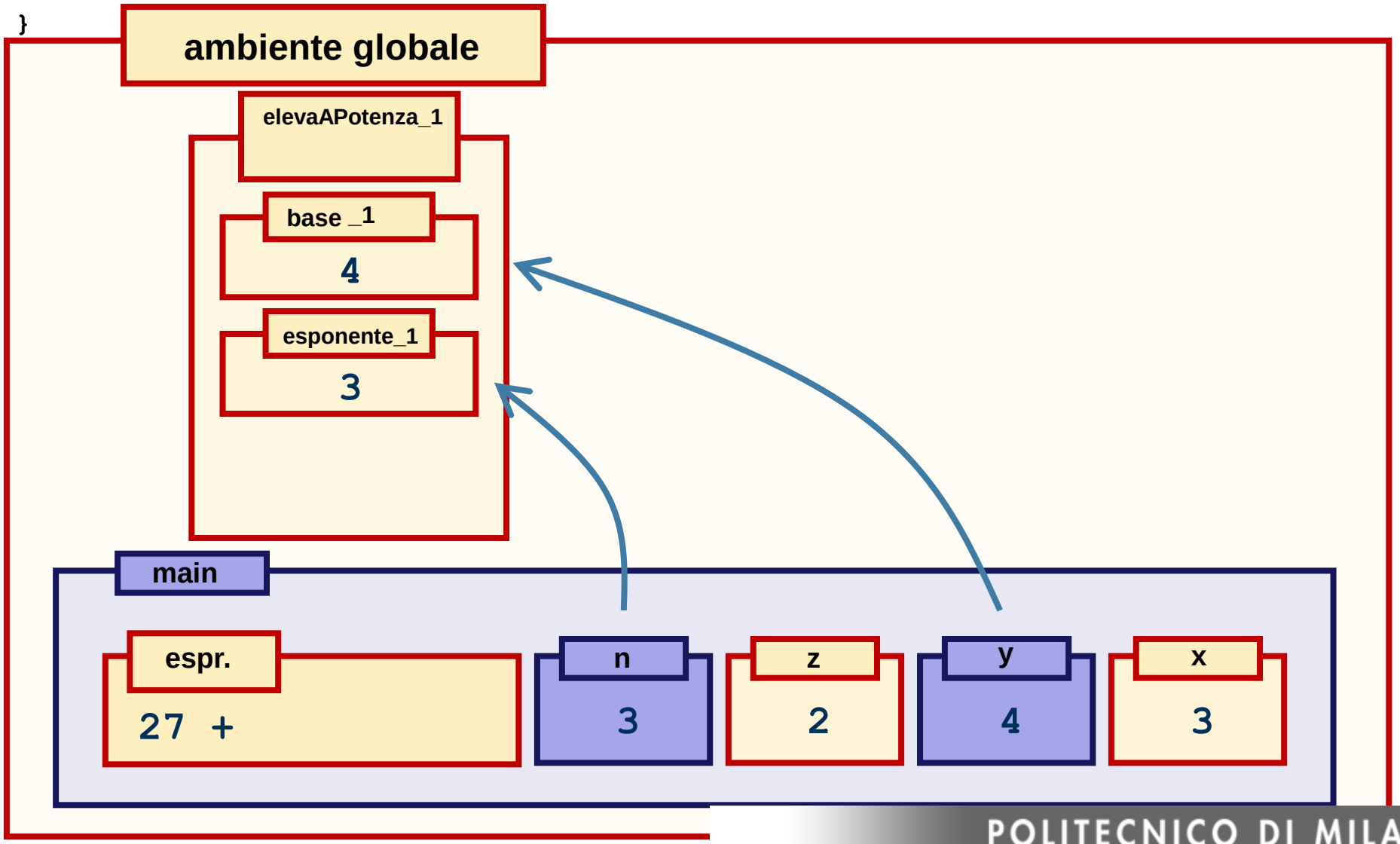
```
   "Presenta le funzionalità del programma"
   "Leggi i dati e verifica che rispondano alle specifiche"
   if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
   ...
}
```



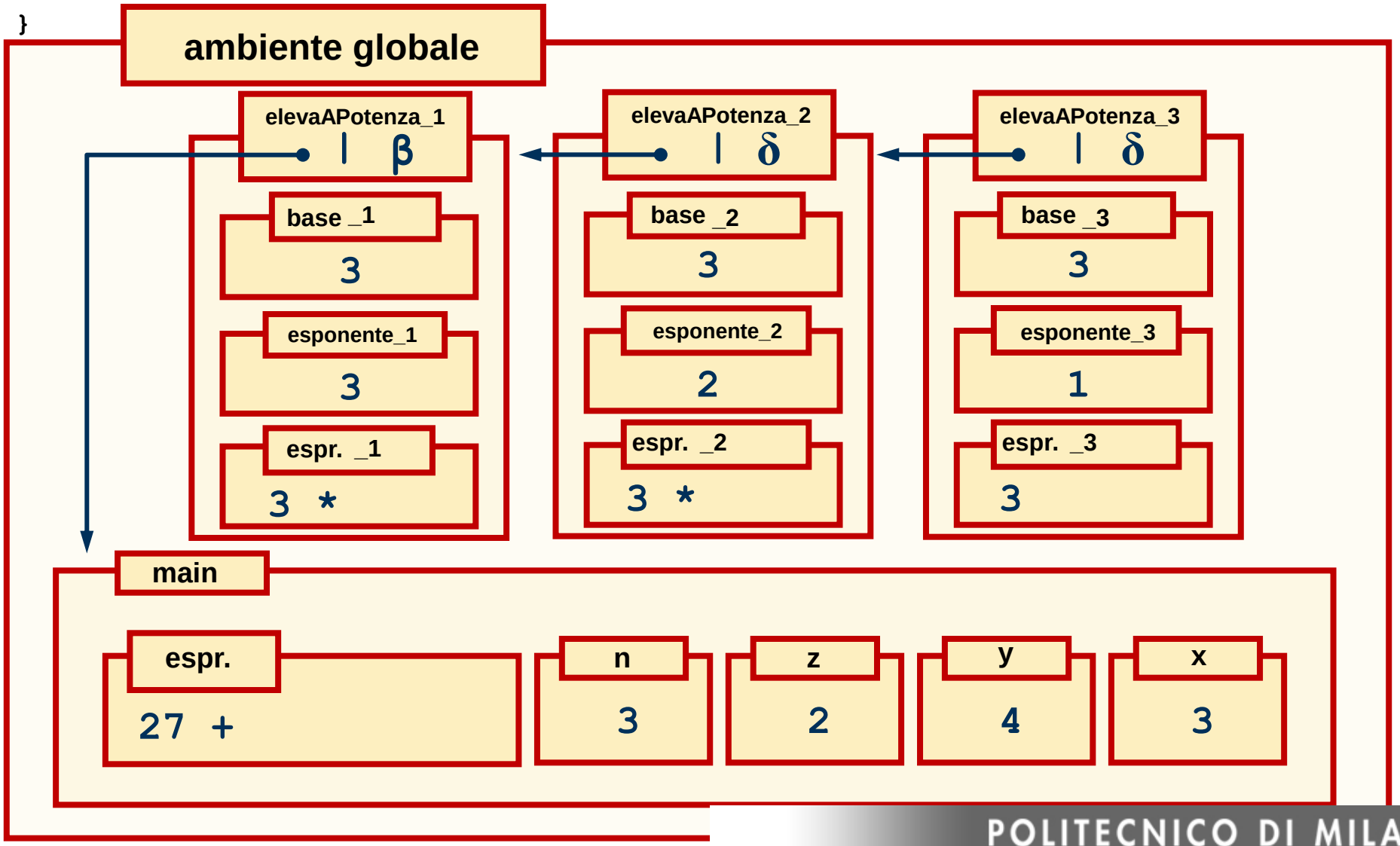
```
    "Presenta le funzionalità del programma"
    "Leggi i dati e verifica che rispondano alle specifiche"
    if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
    ...
}
```



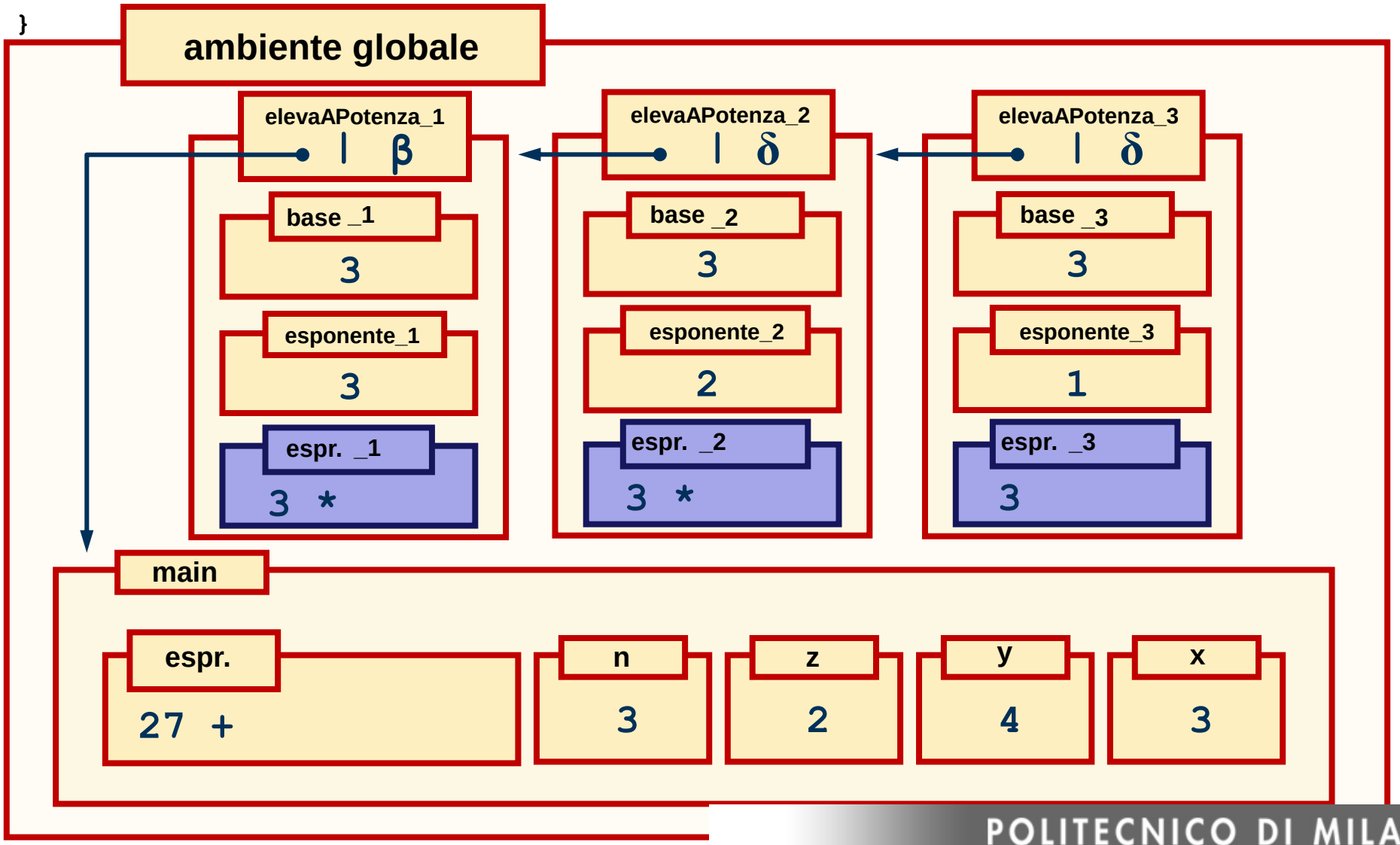
```
"Presenta le funzionalità del programma"
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
}
```



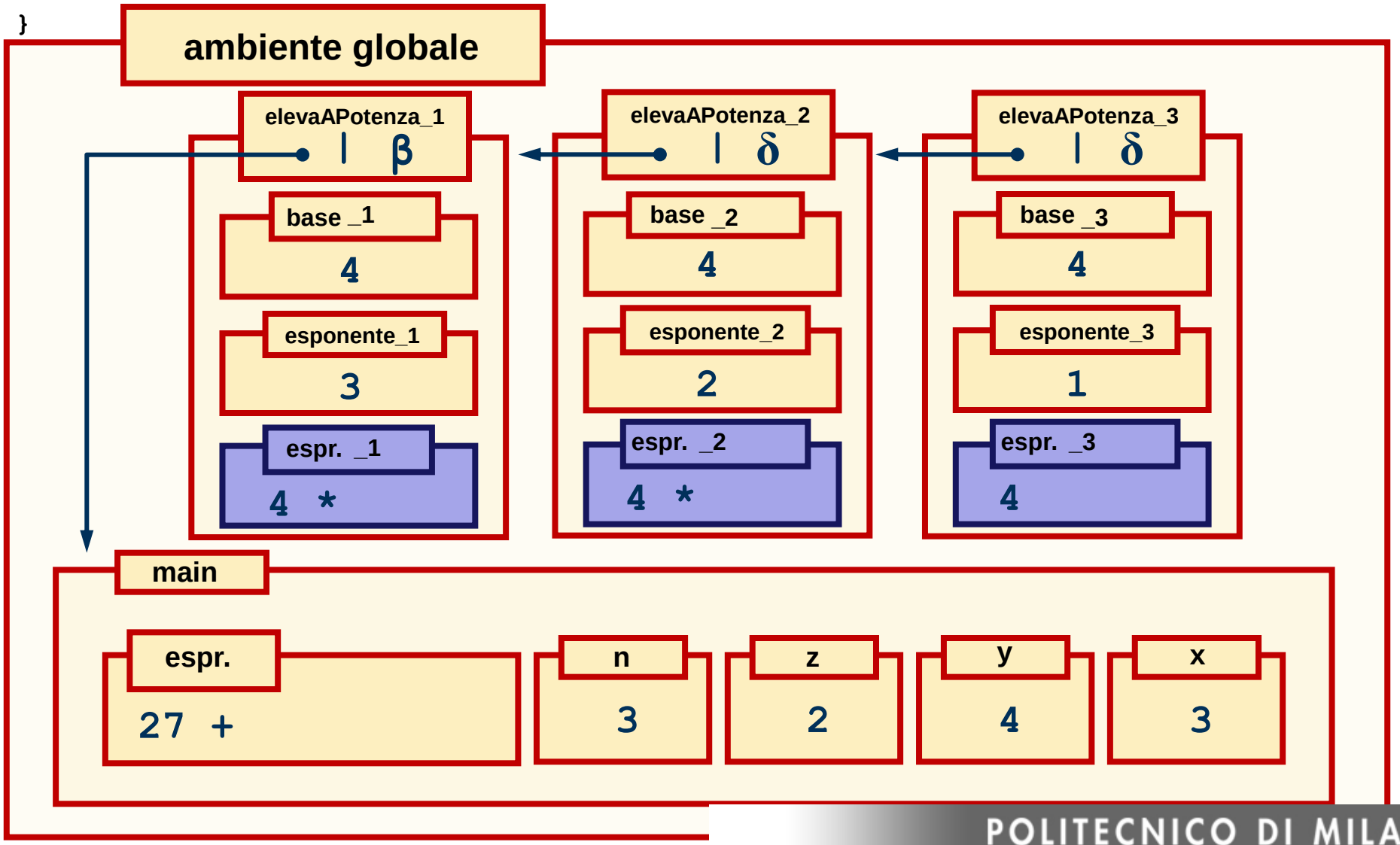
```
"Presenta le funzionalità del programma"
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
}
```



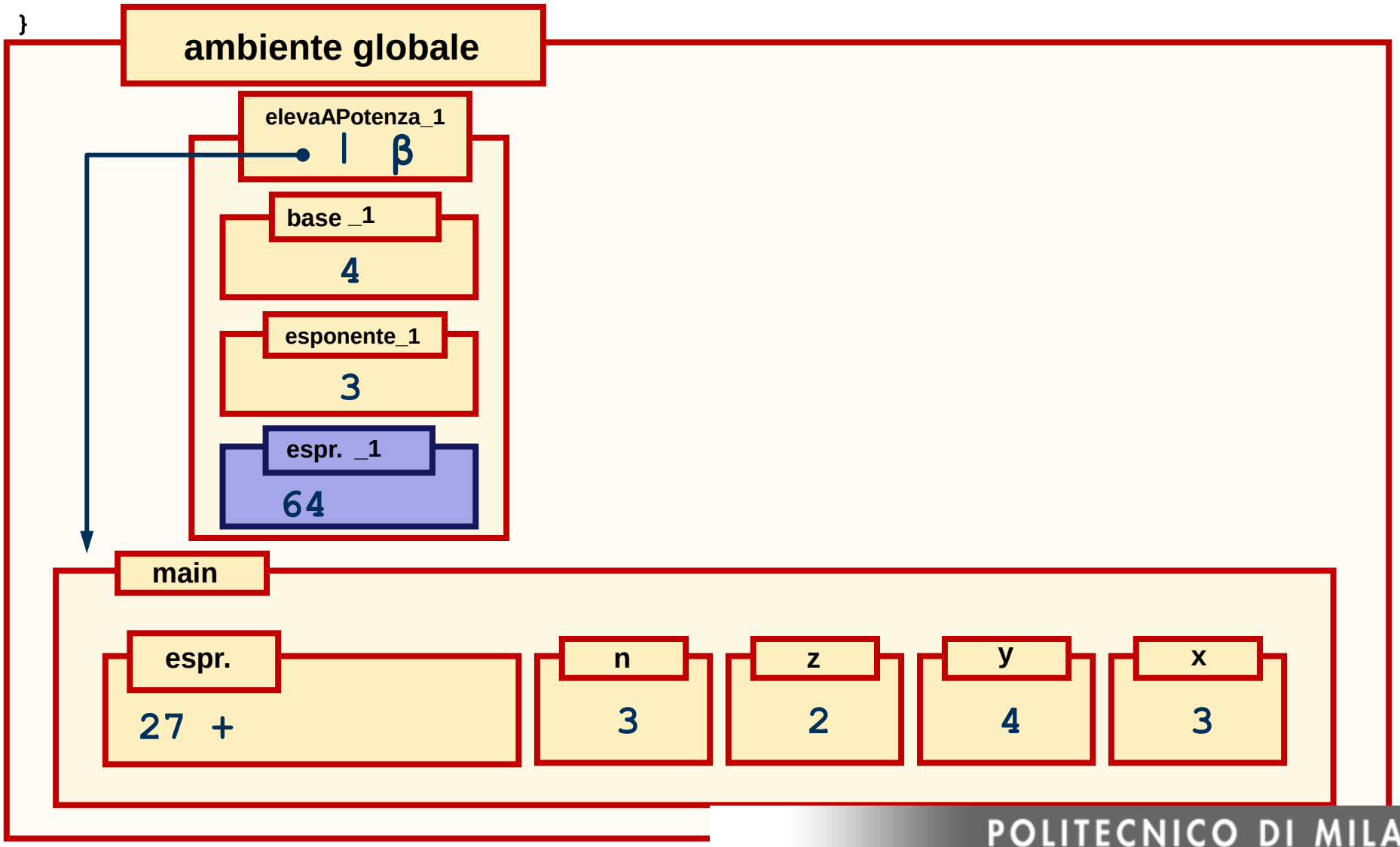
```
"Presenta le funzionalità del programma"
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
}
```



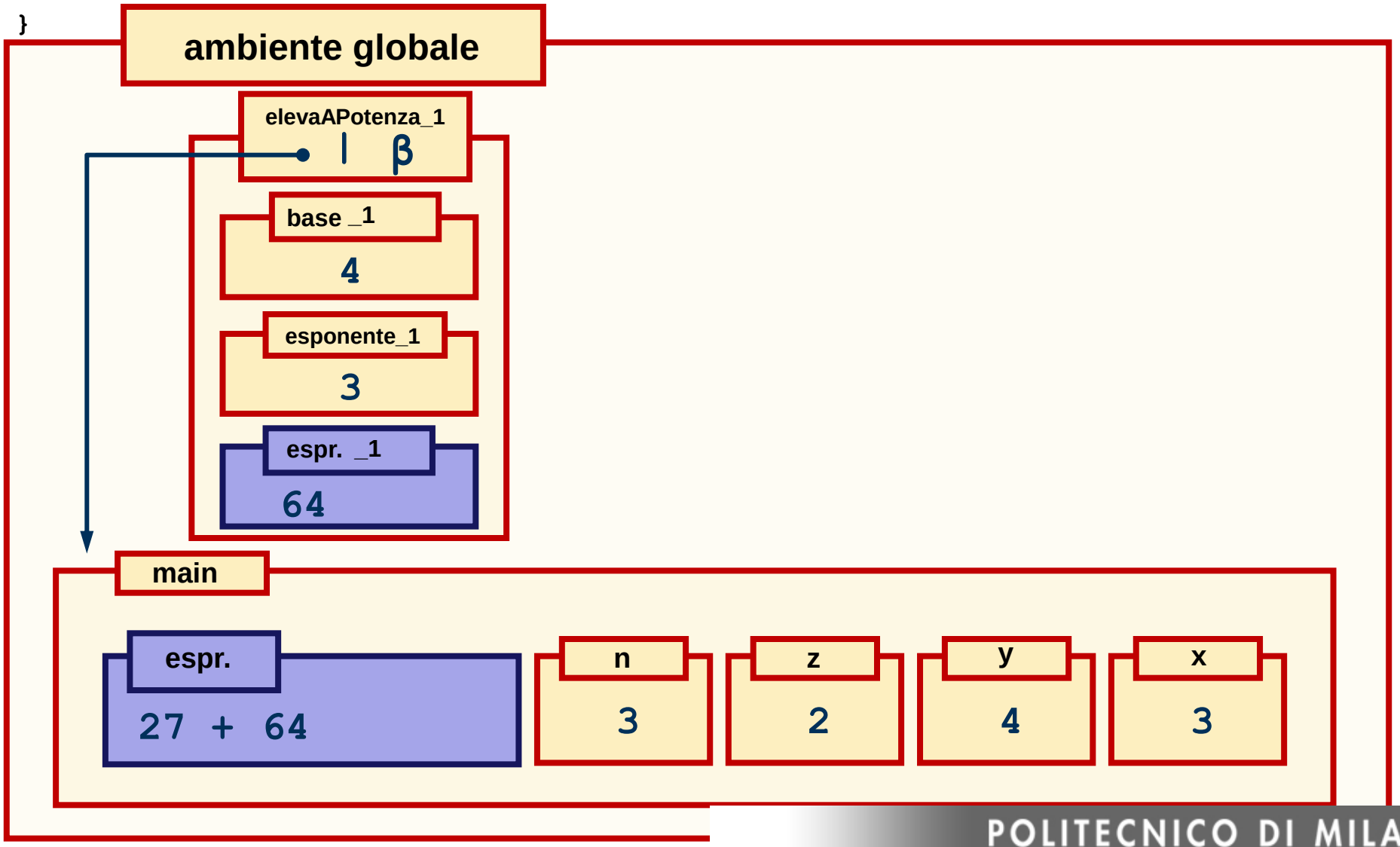
```
"Presenta le funzionalità del programma"
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
}
```



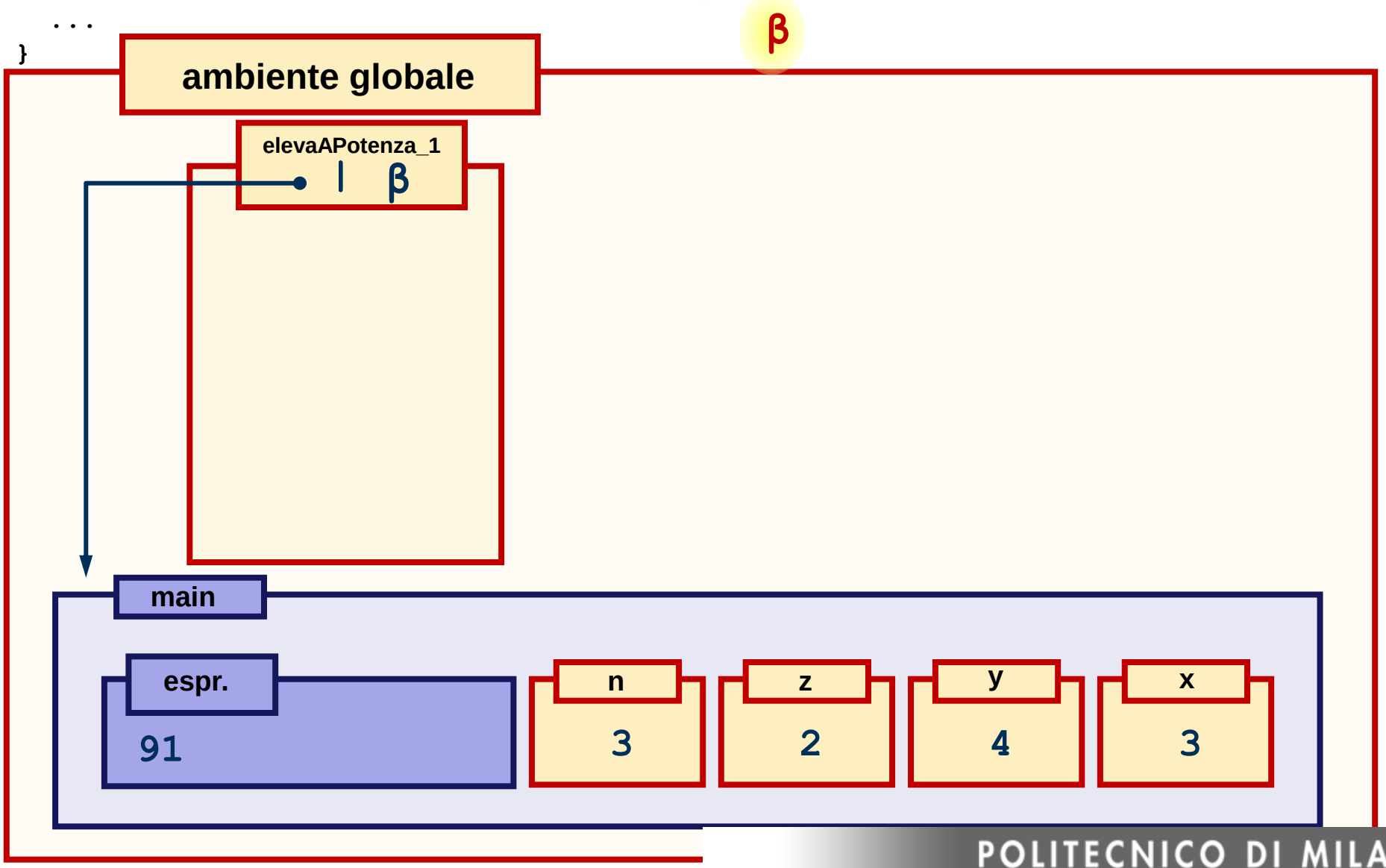
```
    "Presenta le funzionalità del programma"
    "Leggi i dati e verifica che rispondano alle specifiche"
    if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
    ...
}
```



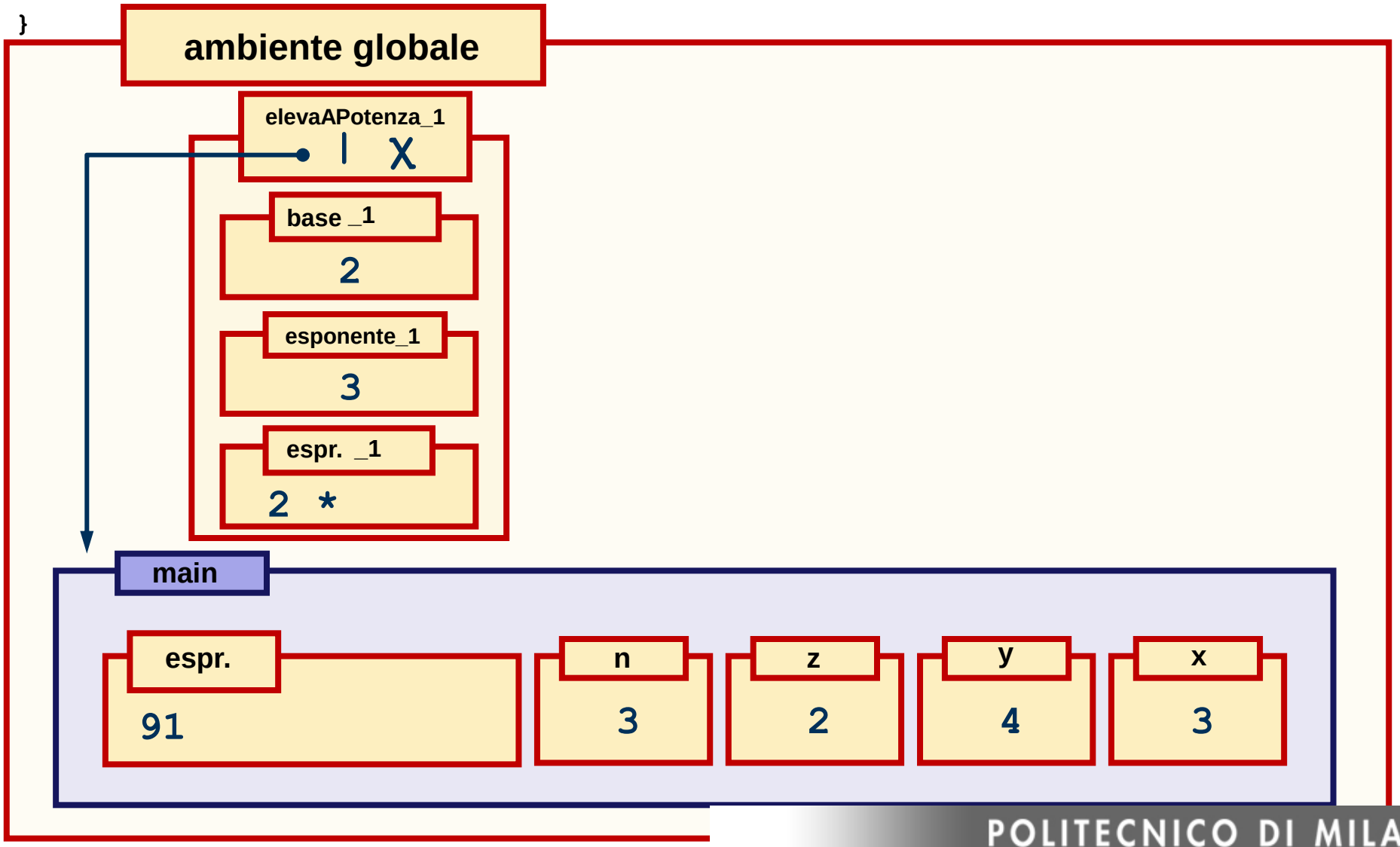
```
    "Presenta le funzionalità del programma"
    "Leggi i dati e verifica che rispondano alle specifiche"
    if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
    ...
}
```



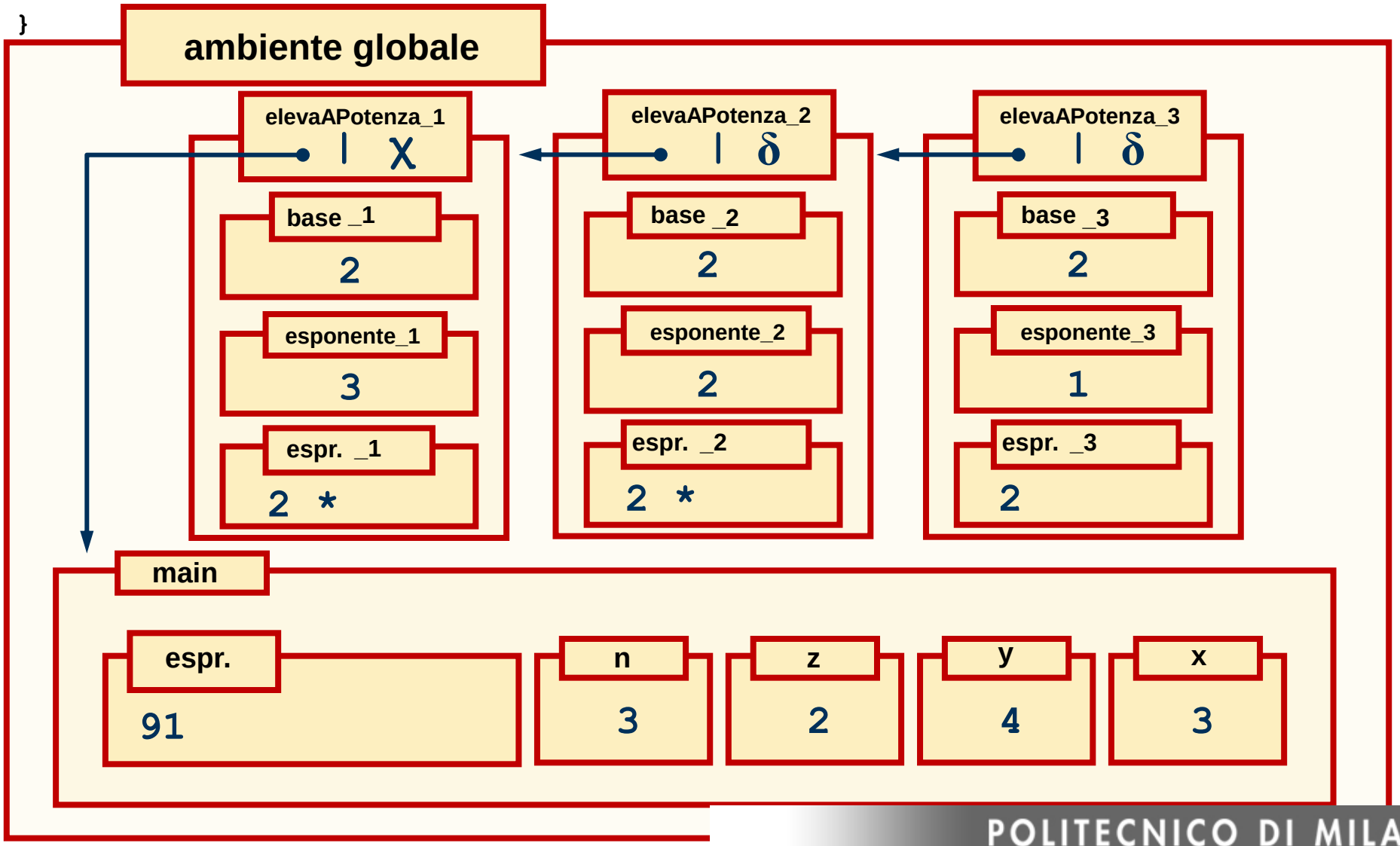
```
    "Presenta le funzionalità del programma"
    "Leggi i dati e verifica che rispondano alle specifiche"
    if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
    ...
}
```



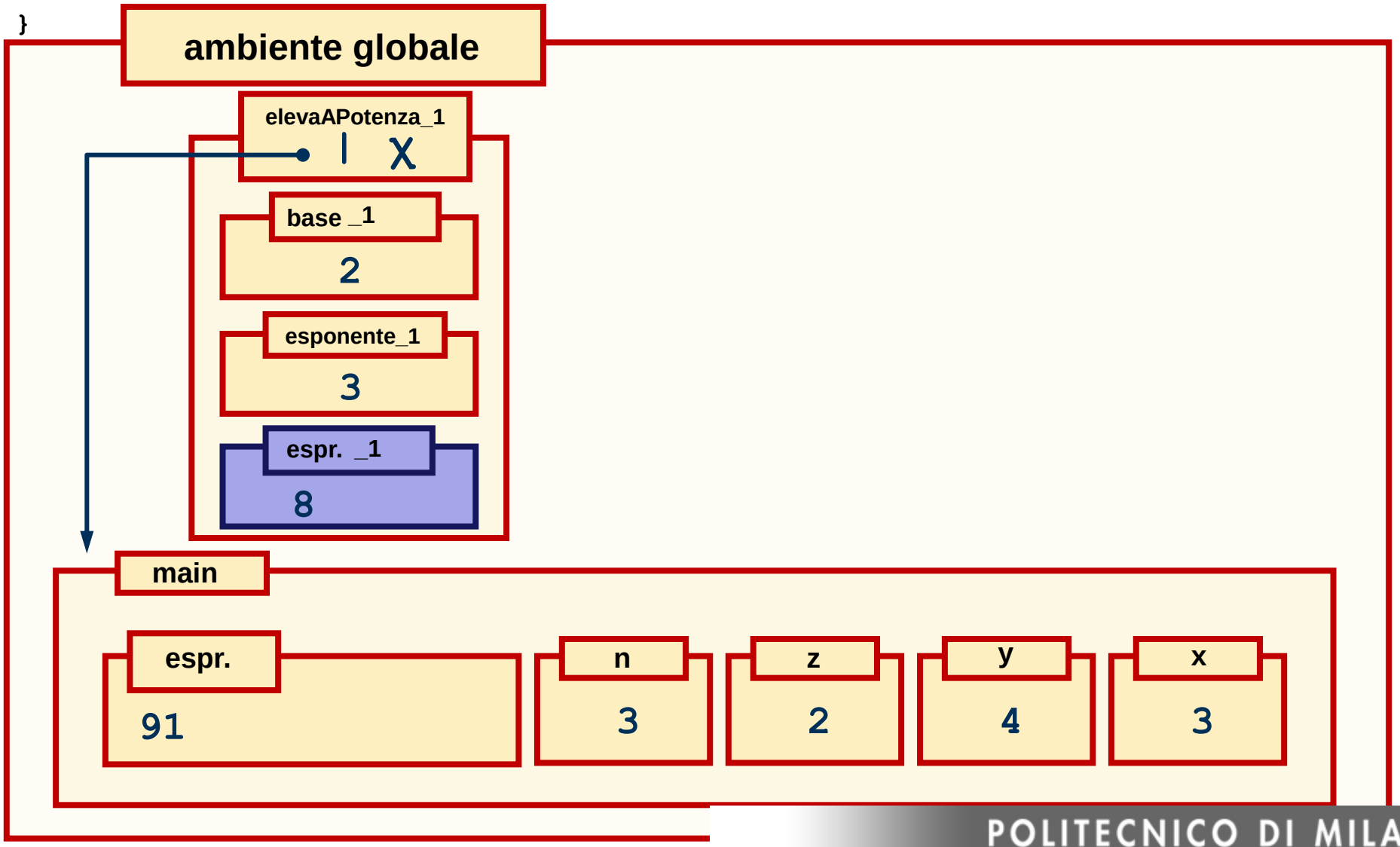
```
"Presenta le funzionalità del programma"
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
}
```



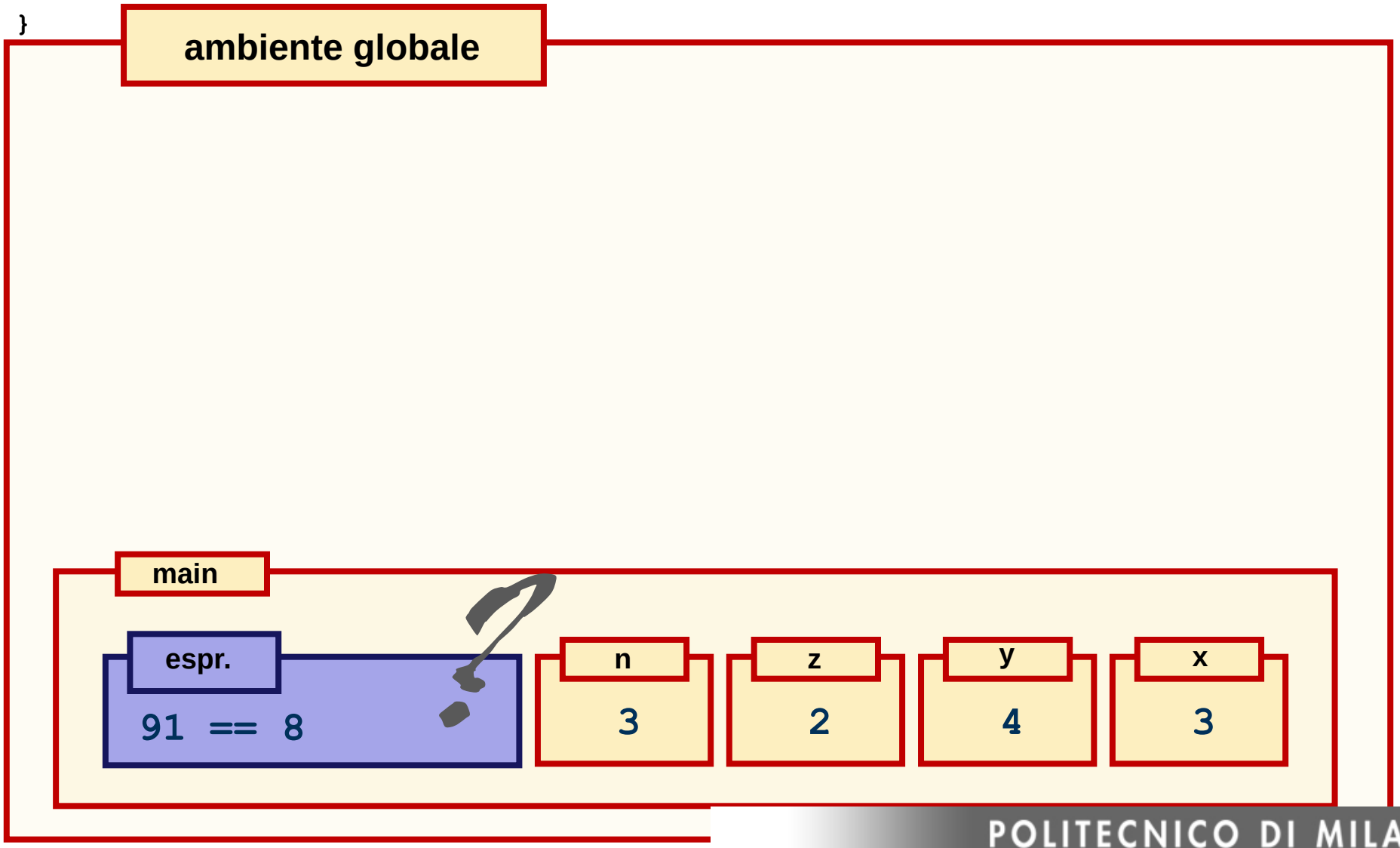
```
"Presenta le funzionalità del programma"
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
}
```



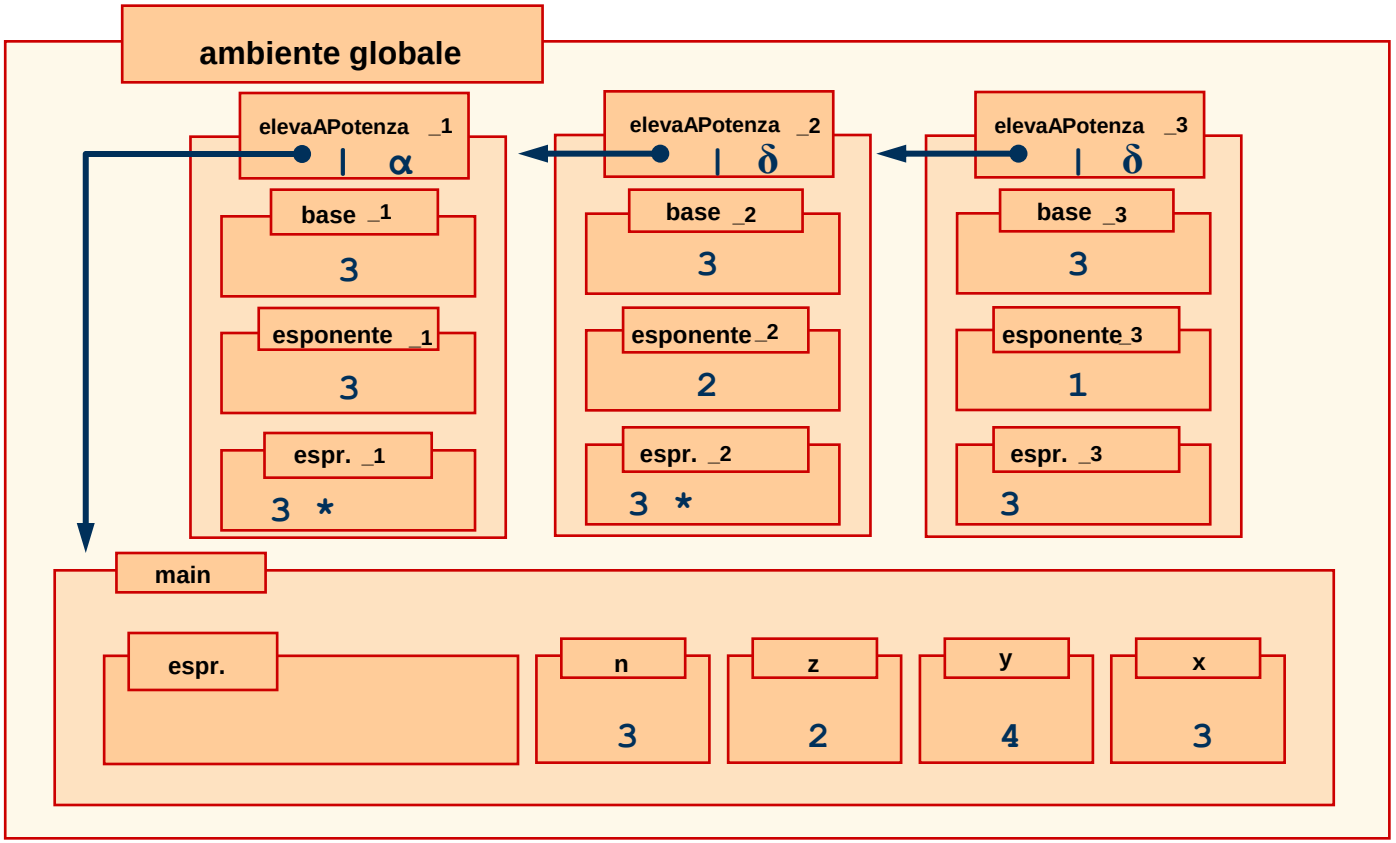
```
"Presenta le funzionalità del programma"
"Leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
...
}
```



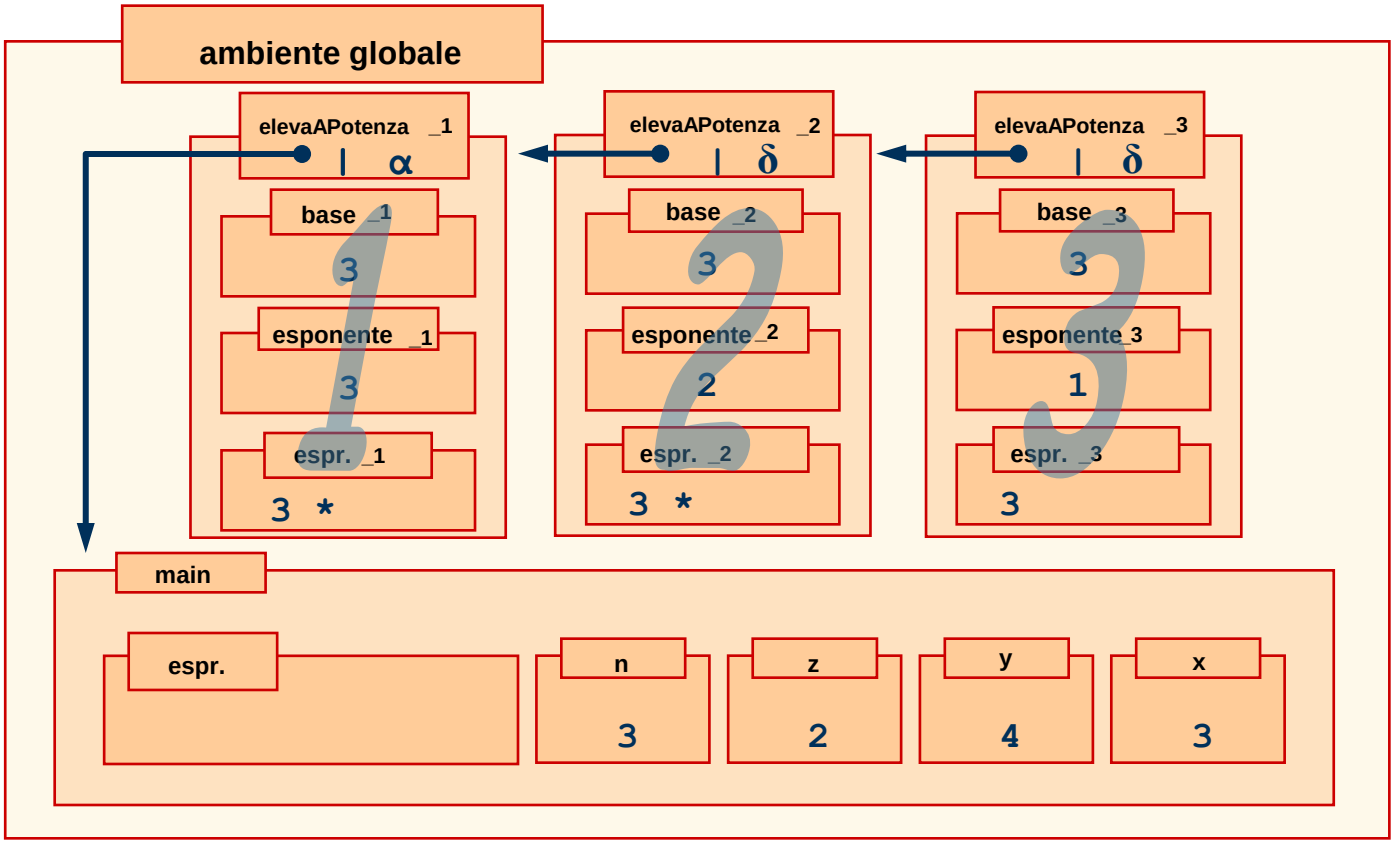
```
  "Presenta le funzionalità del programma"
  "Leggi i dati e verifica che rispondano alle specifiche"
  if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
  ...
}
```



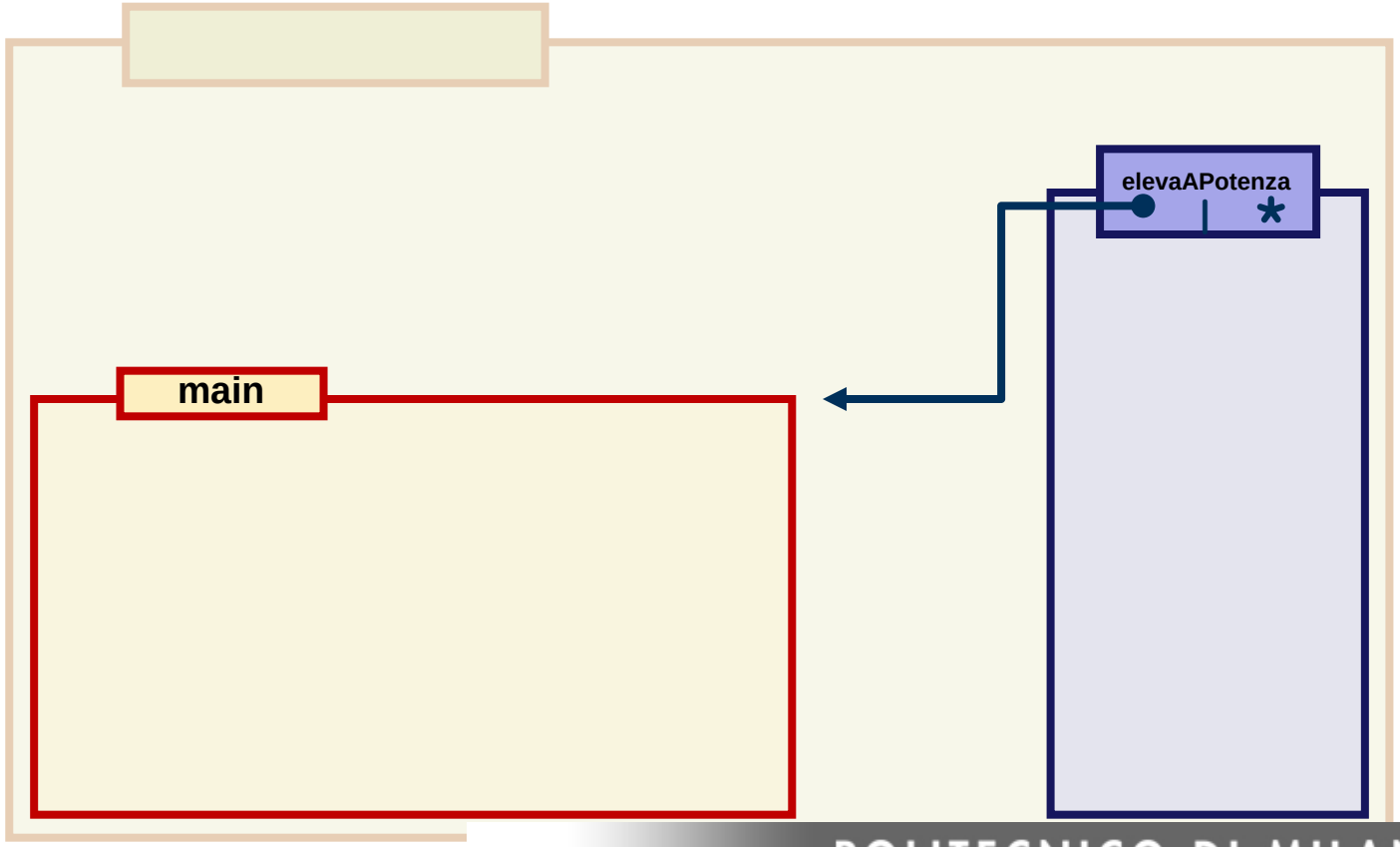
```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



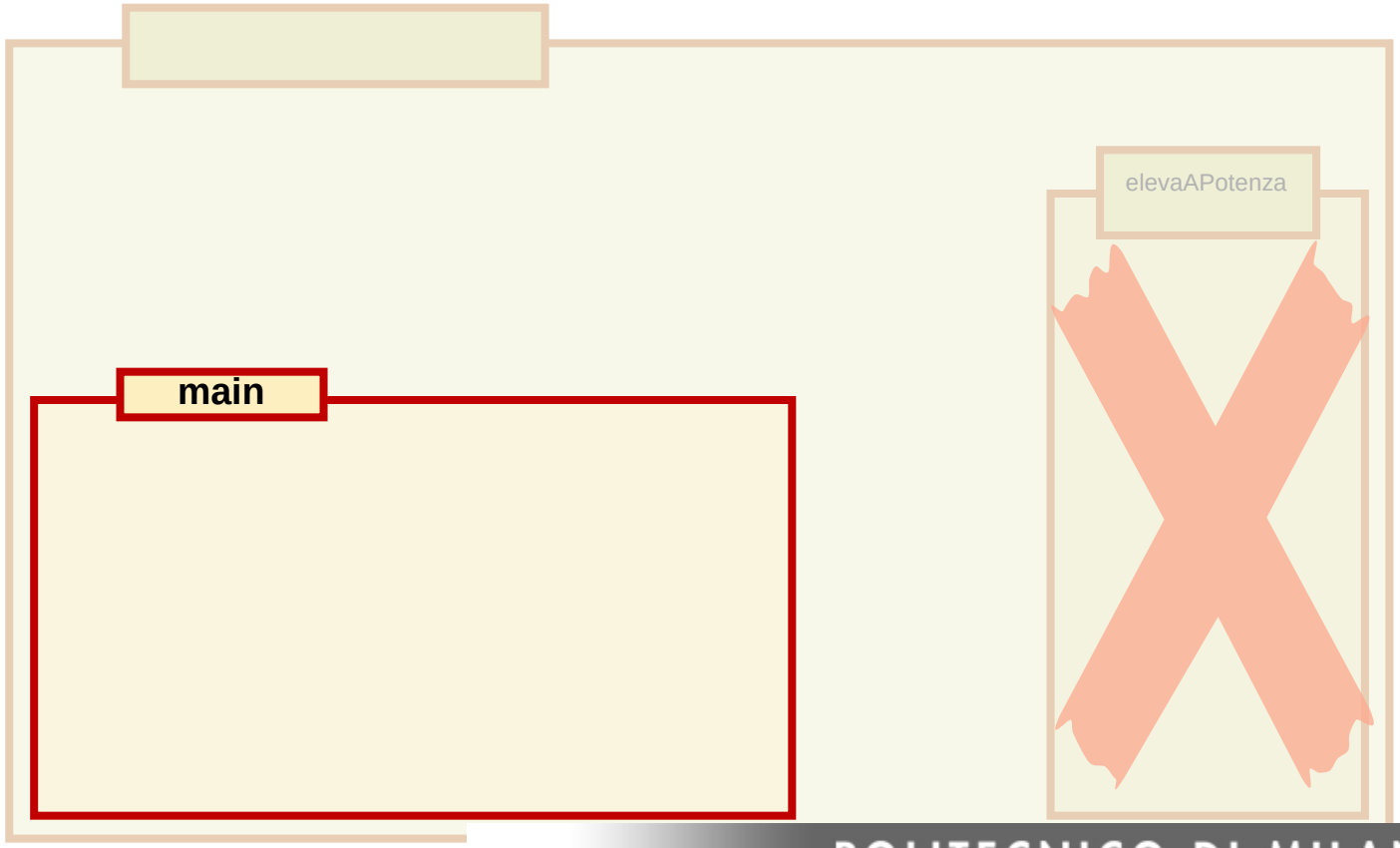
```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```



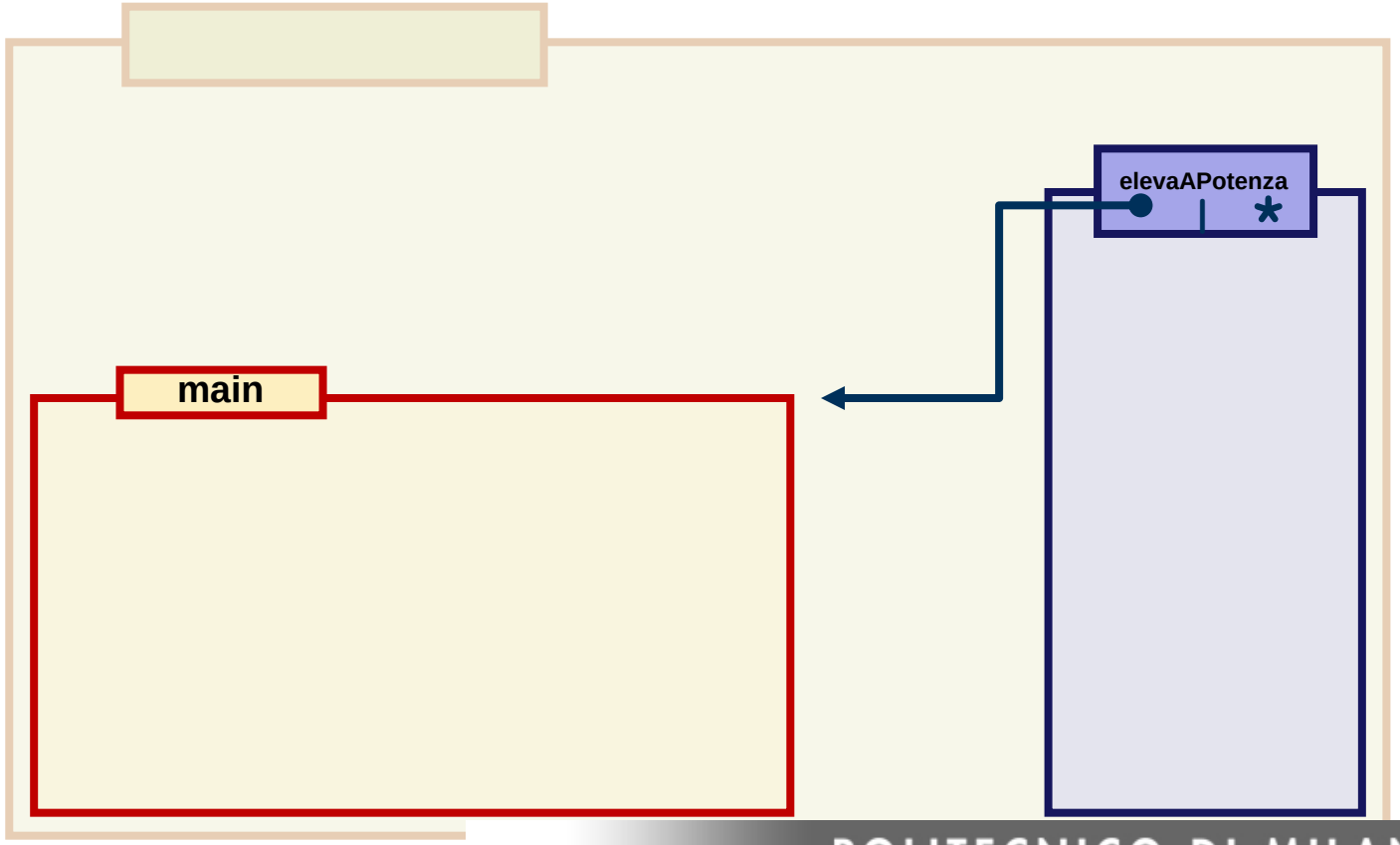
```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```



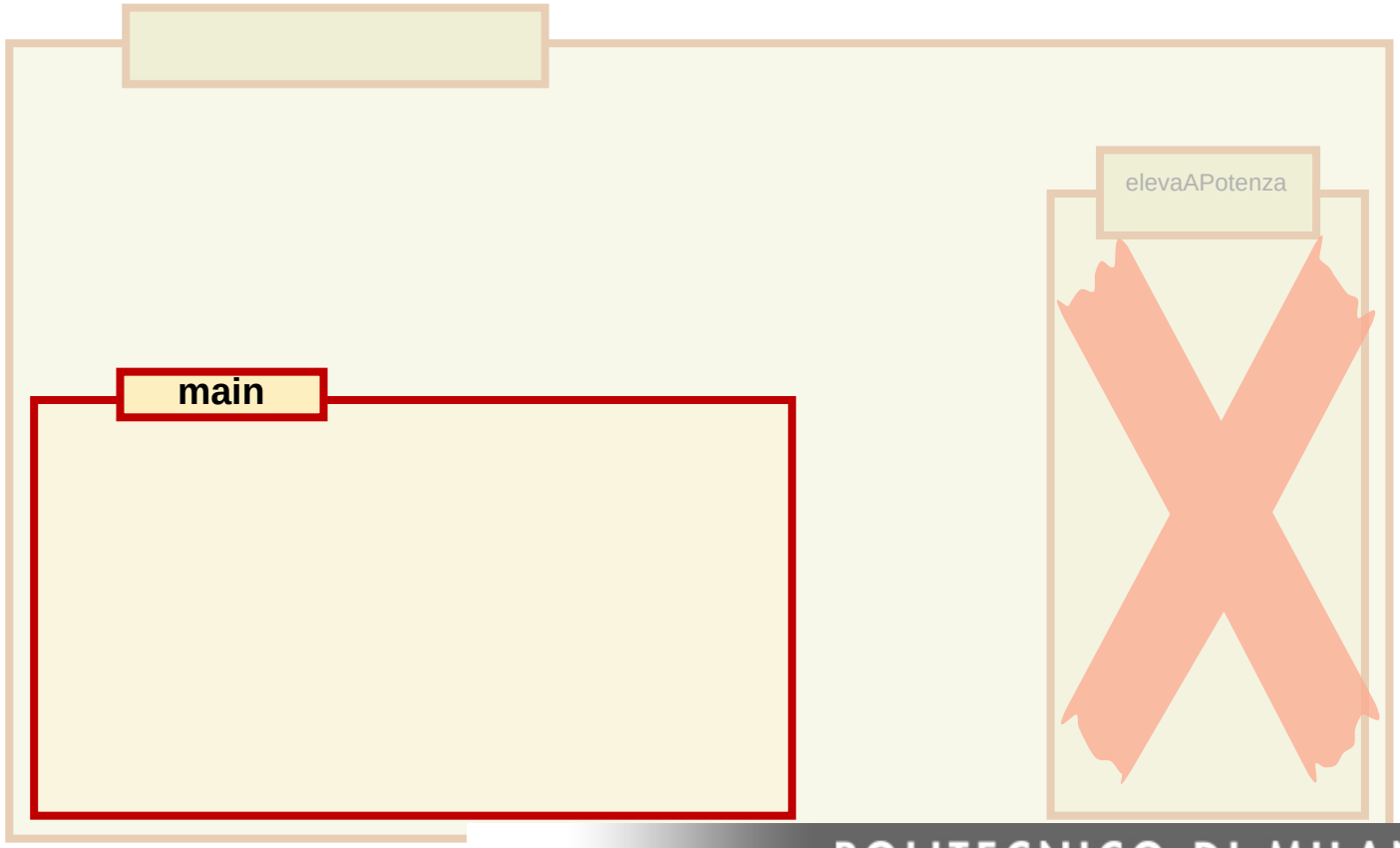
```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```



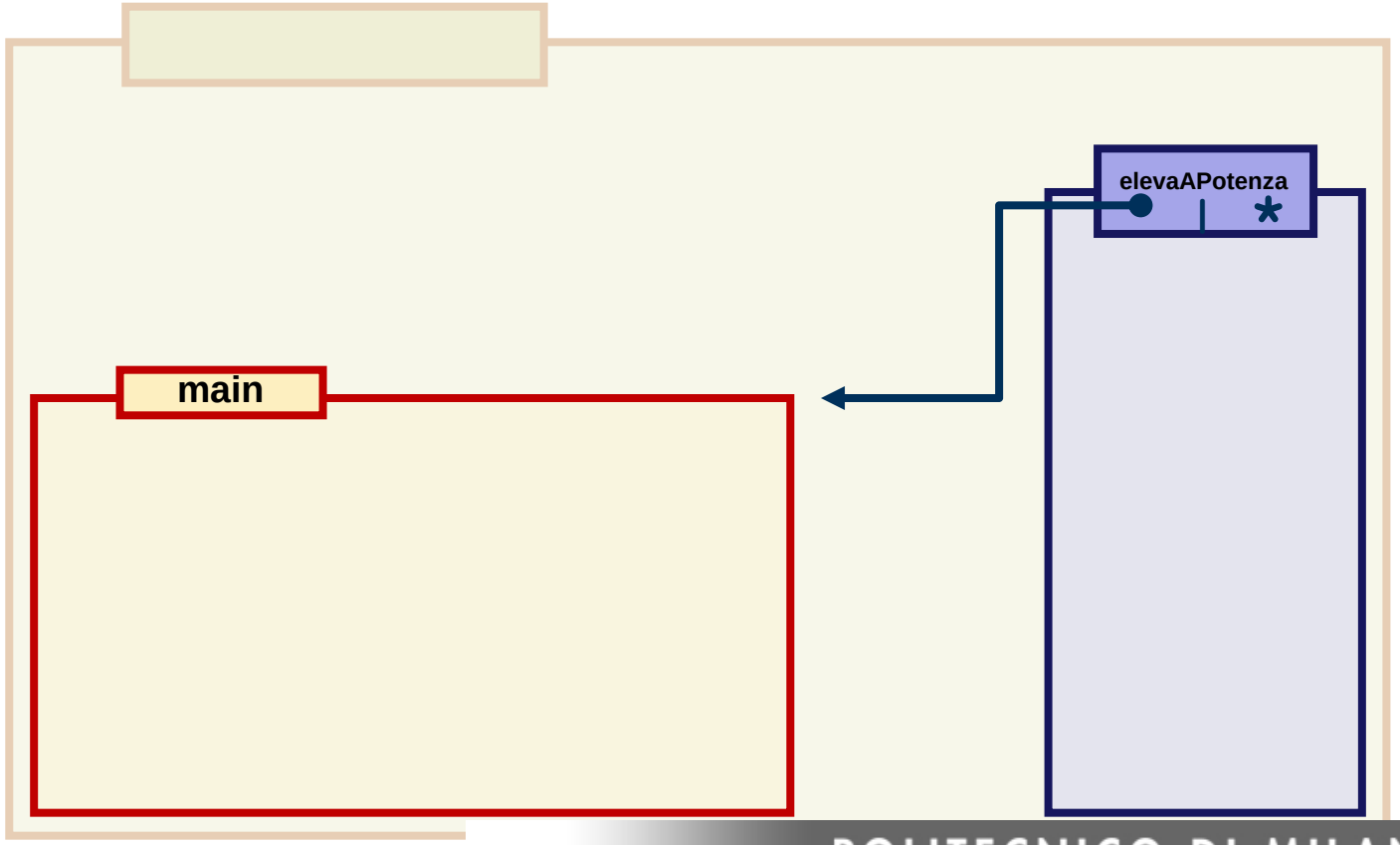
```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```



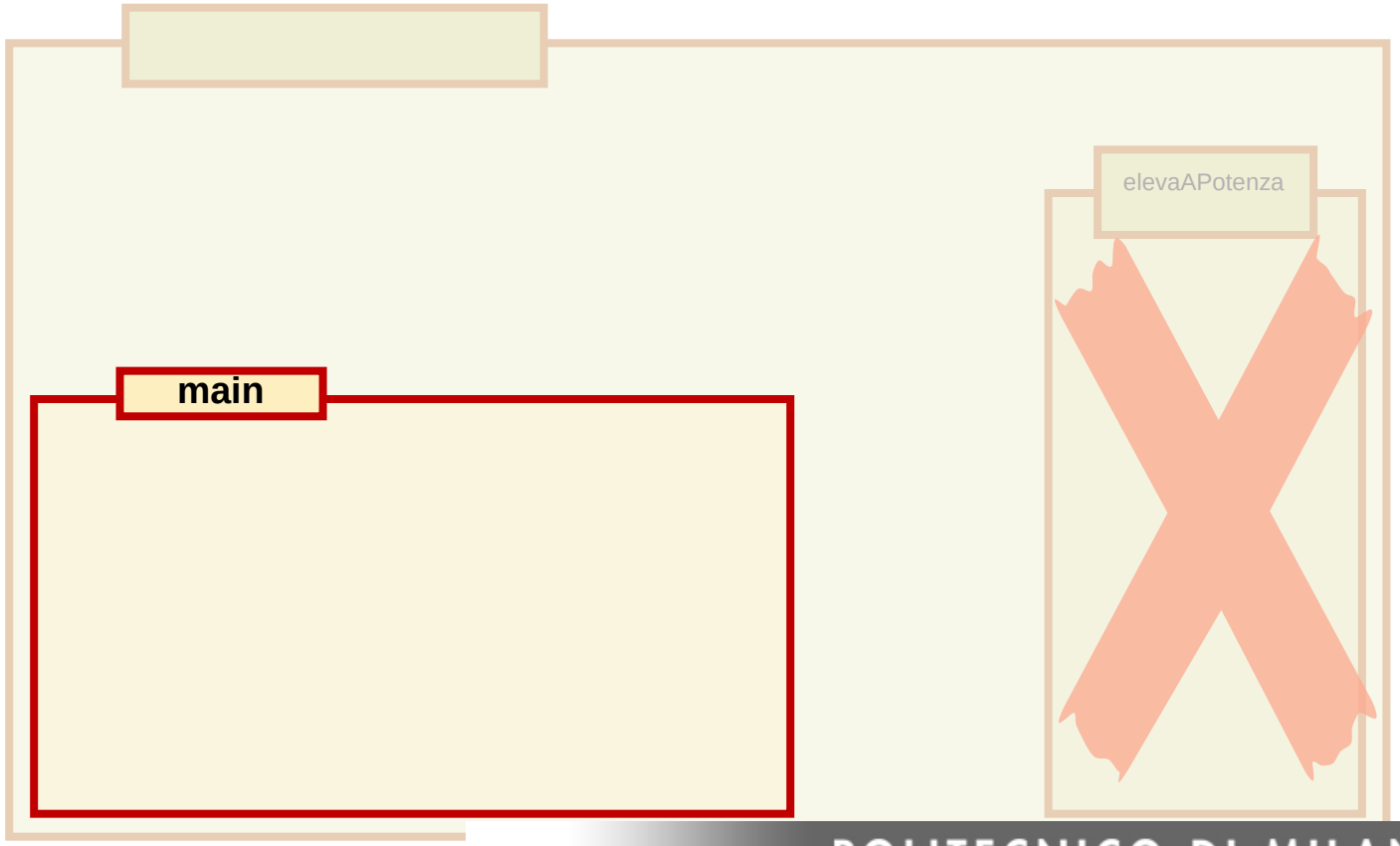
```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```



```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```

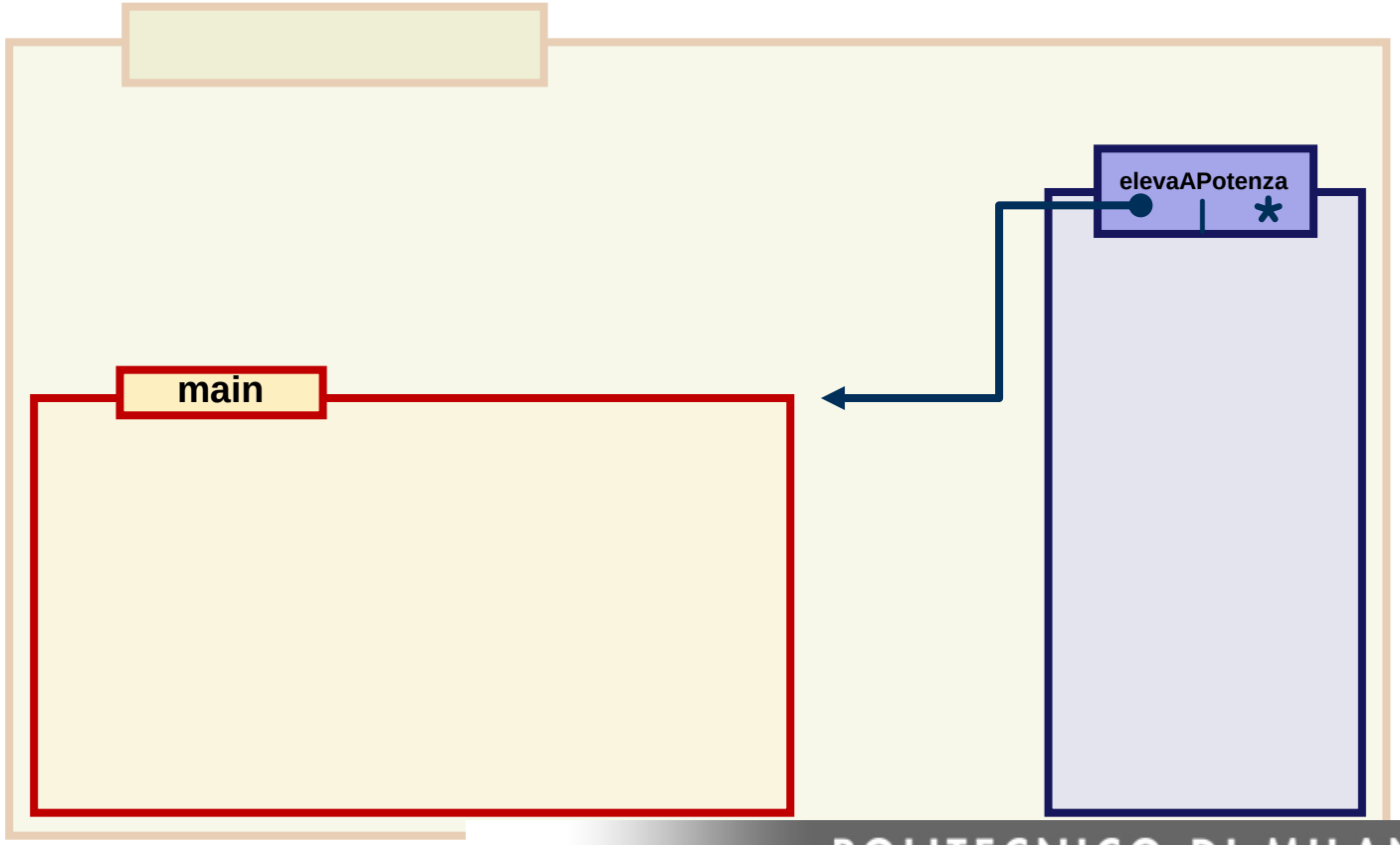


```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```



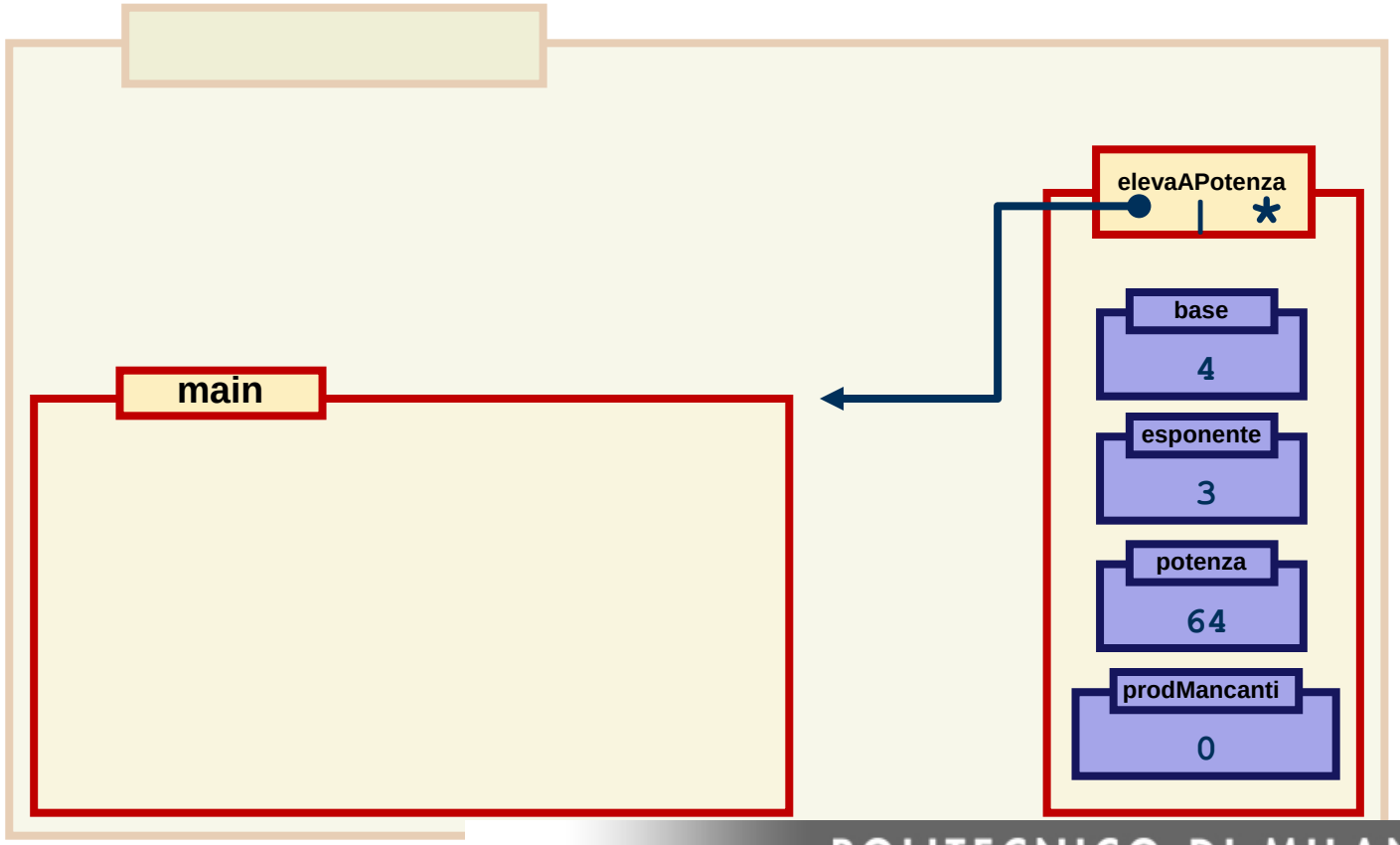
```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```

Allocazione statica



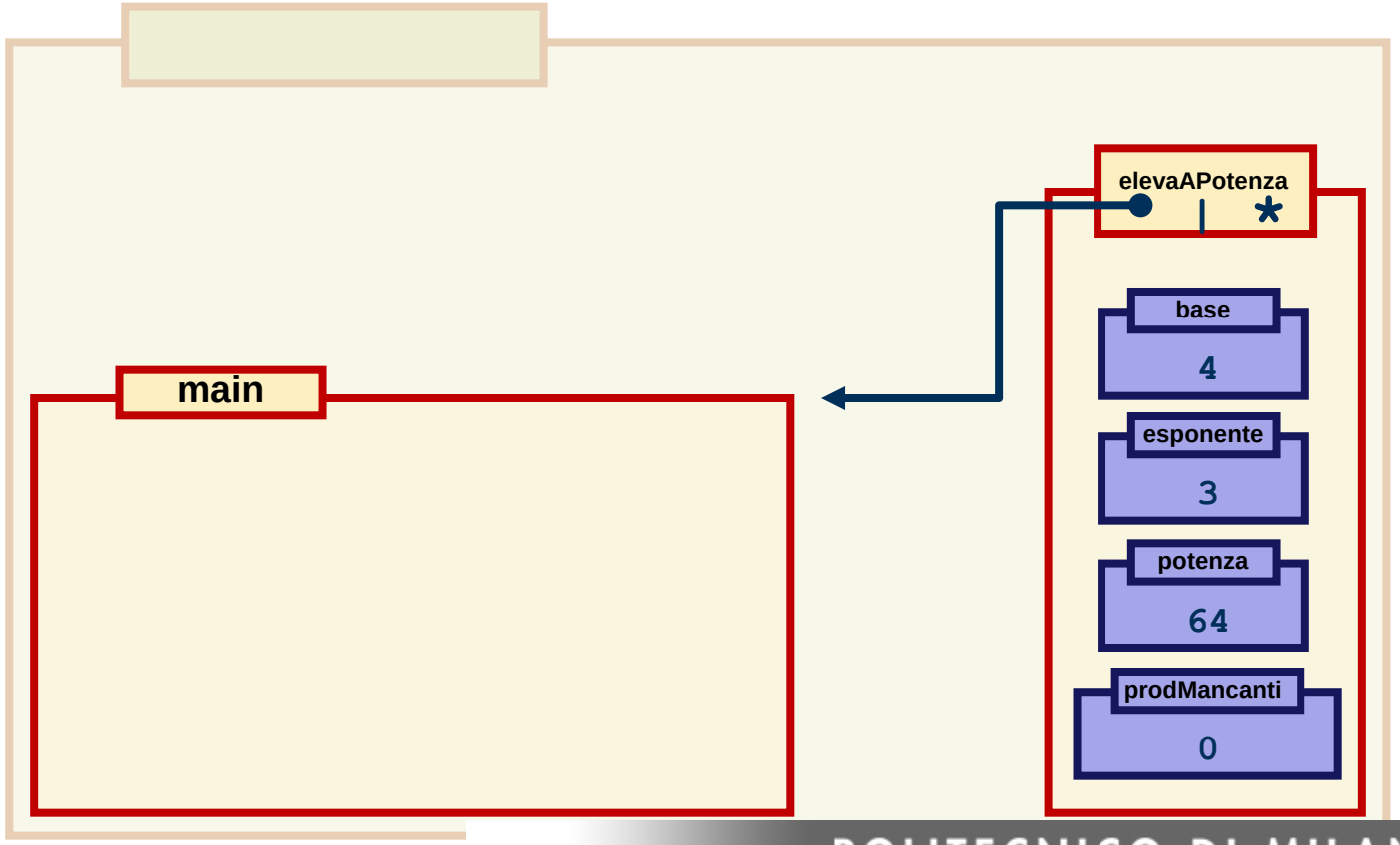
```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```

Allocazione statica

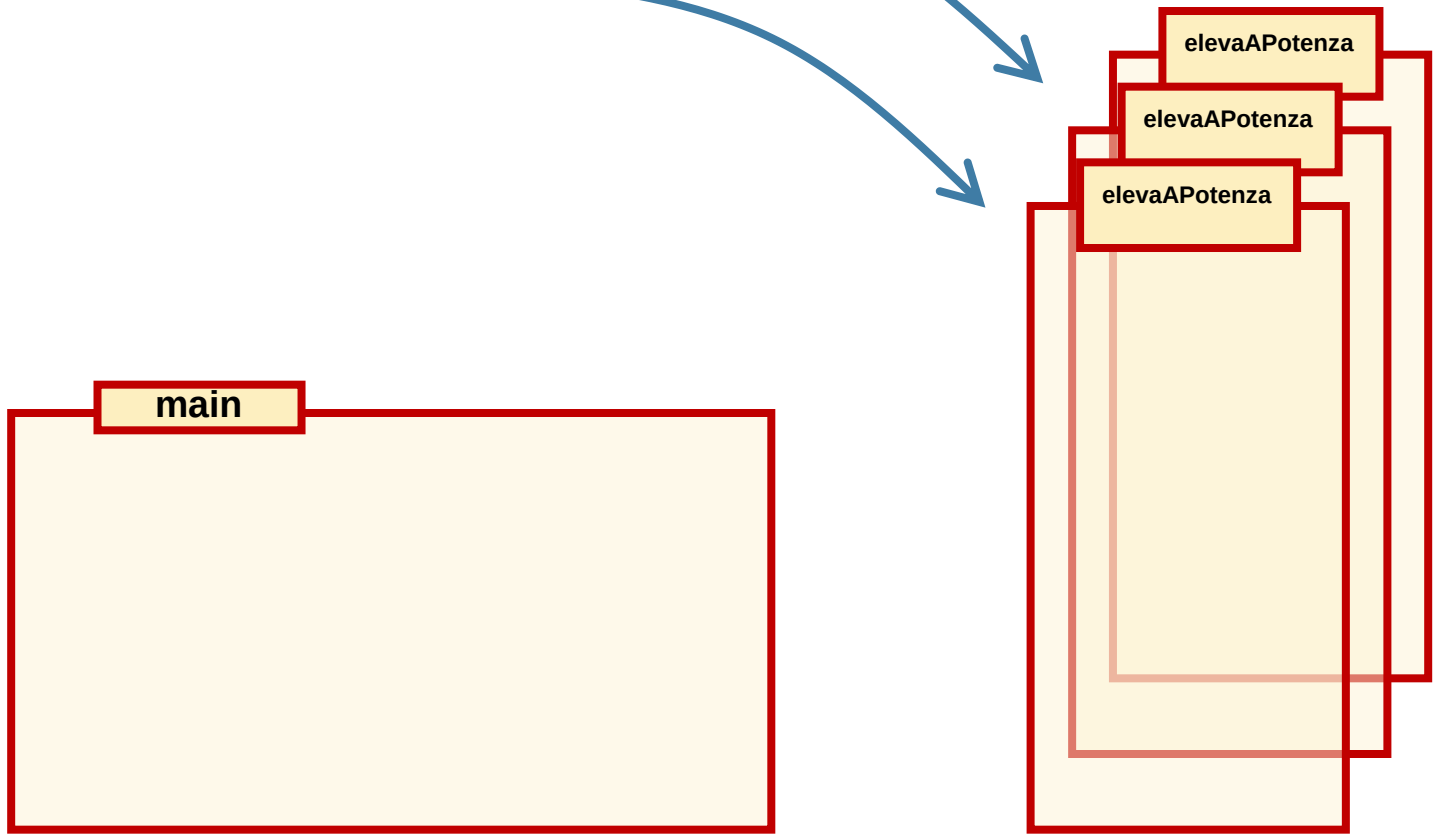


```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```

Allocazione statica



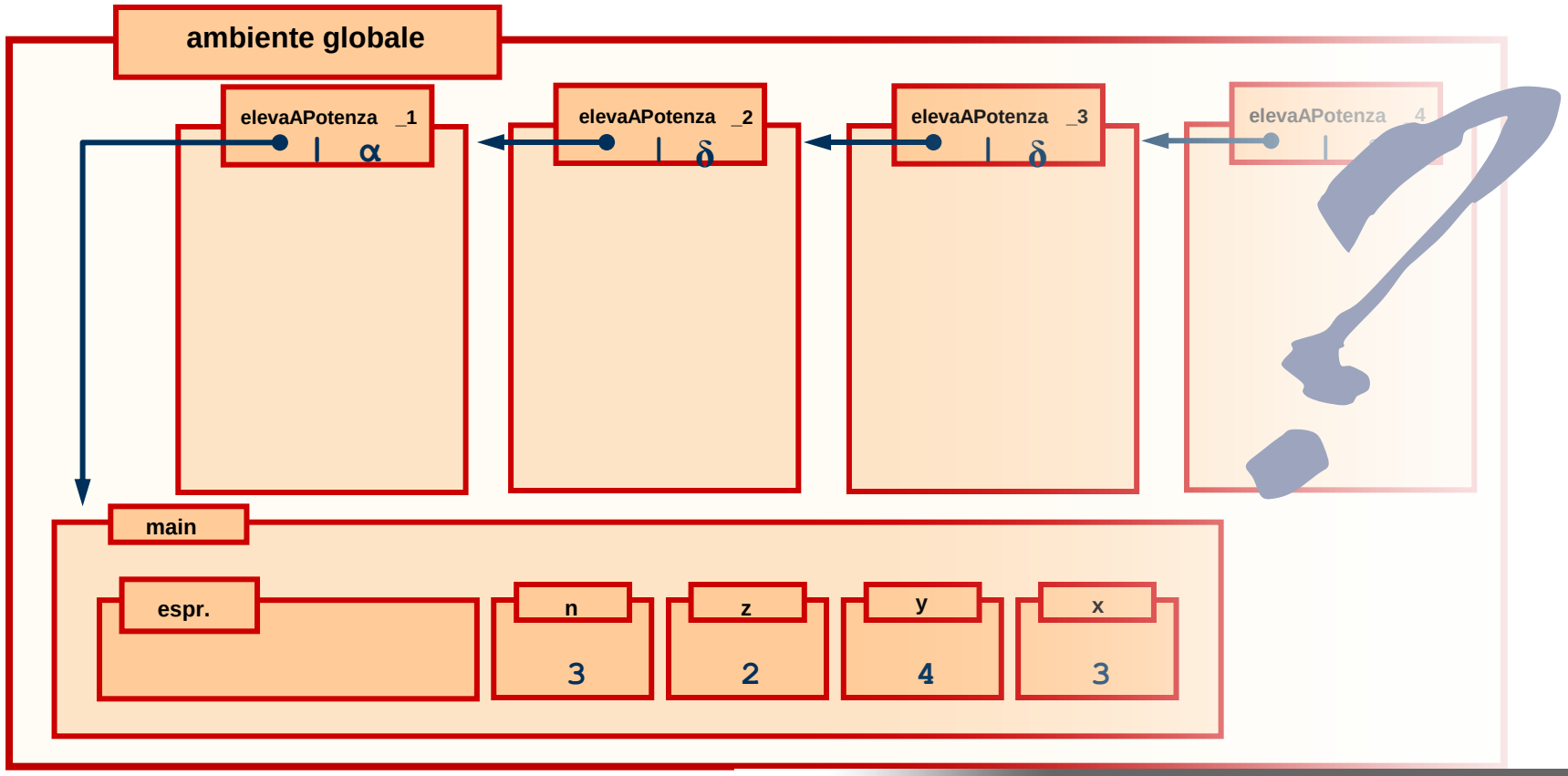
```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```



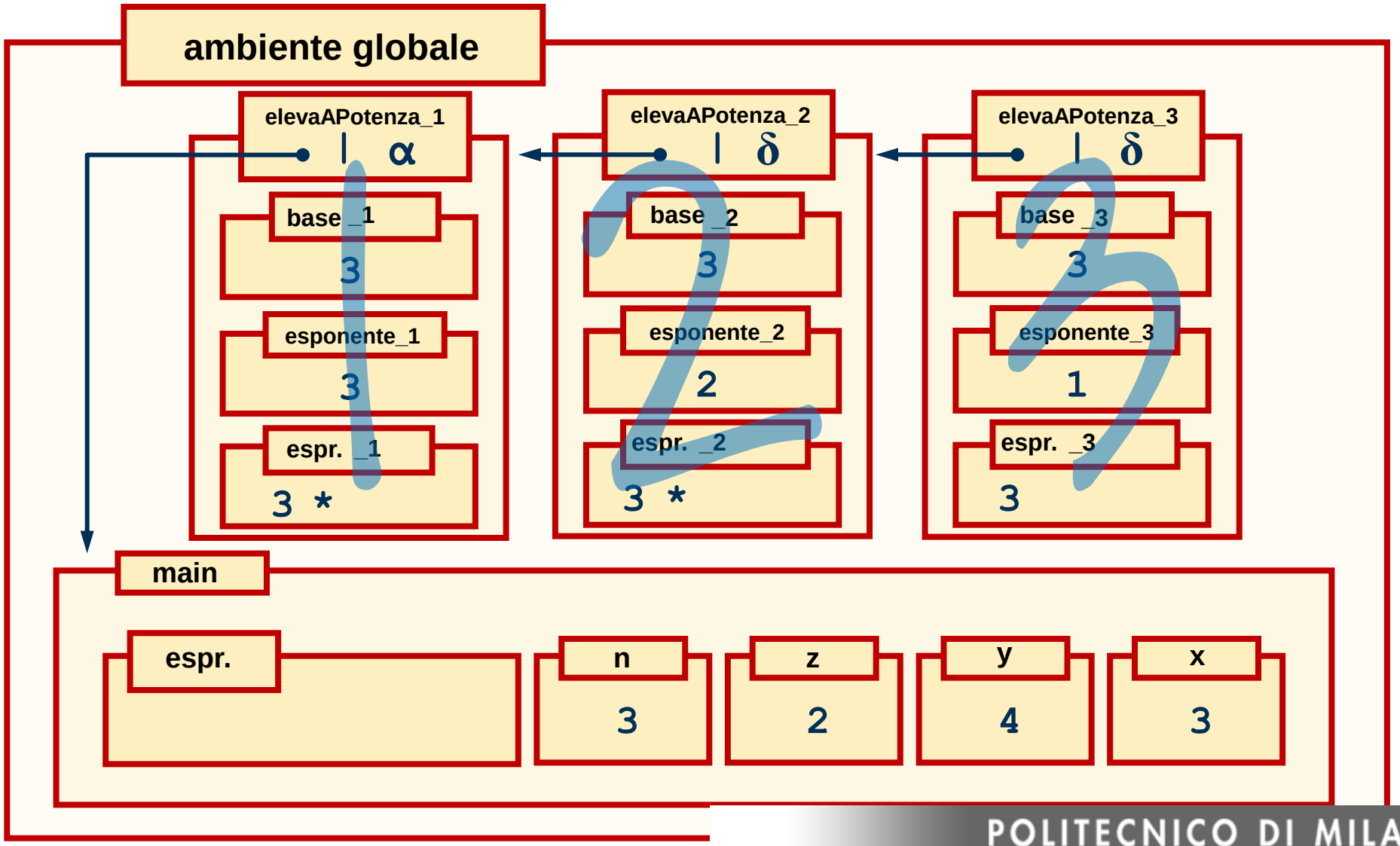
```
...
elevaAPotenza()
...
elevaAPotenza()
...
elevaAPotenza()
...
```

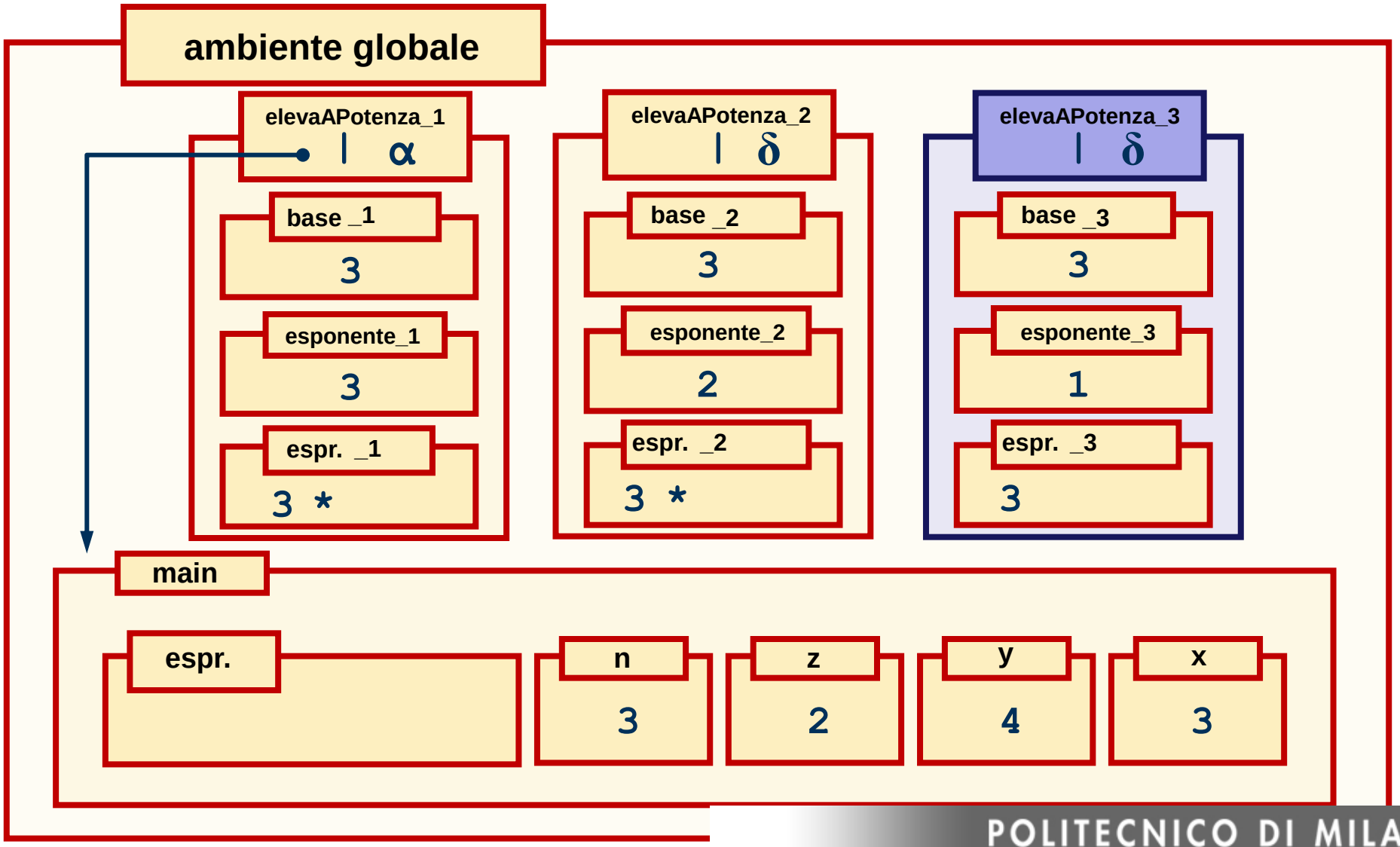
Allocazione dinamica

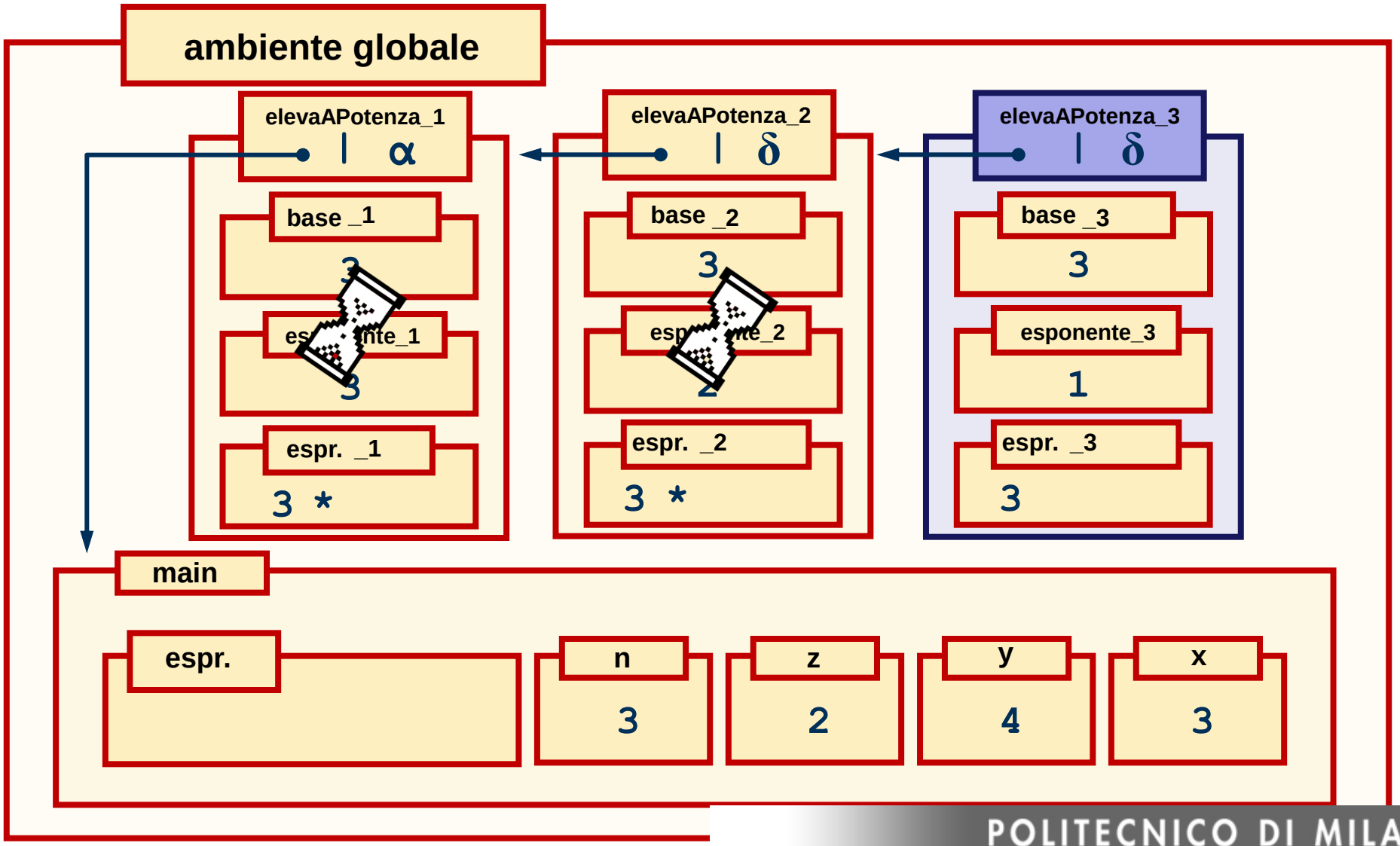
```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

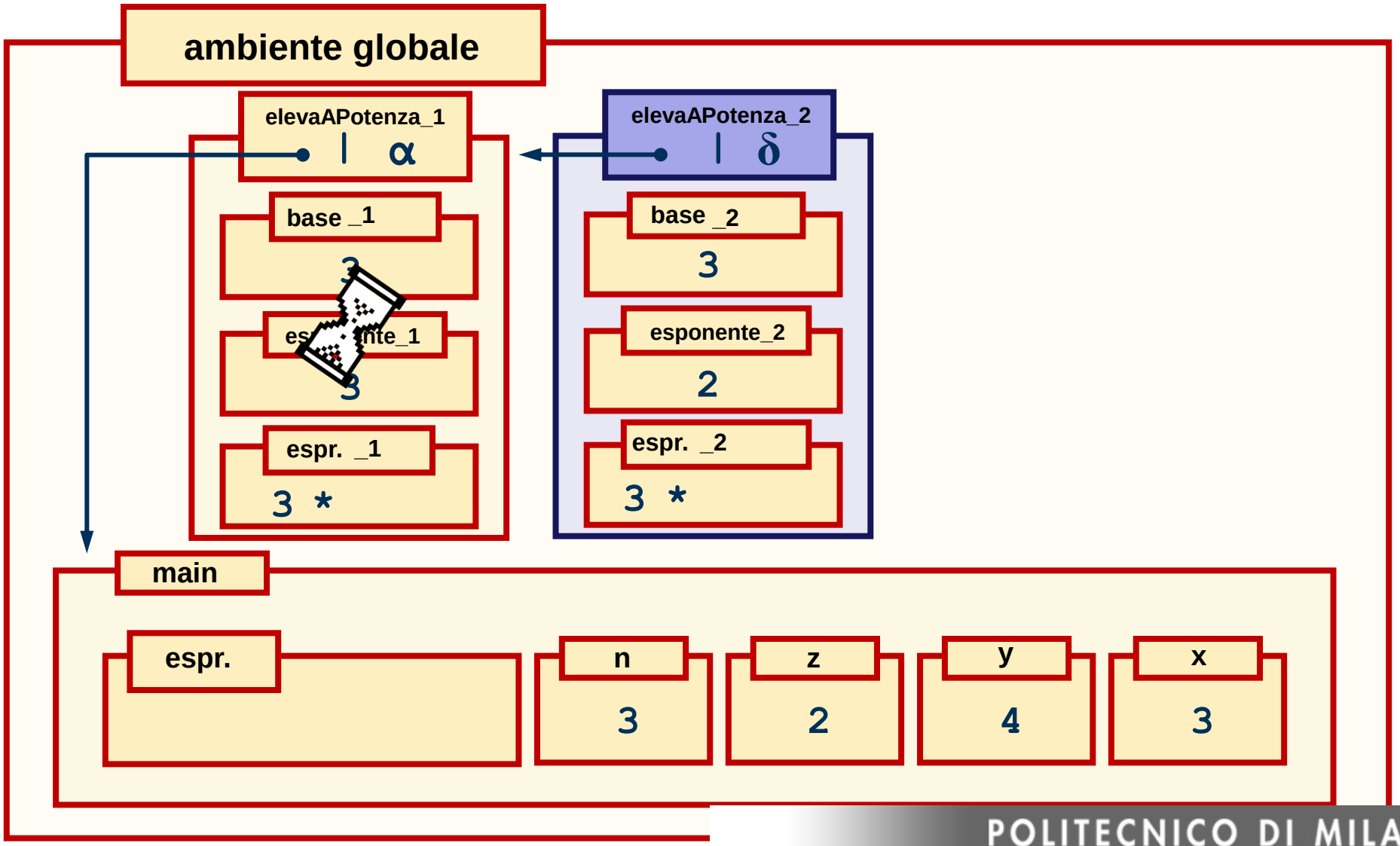


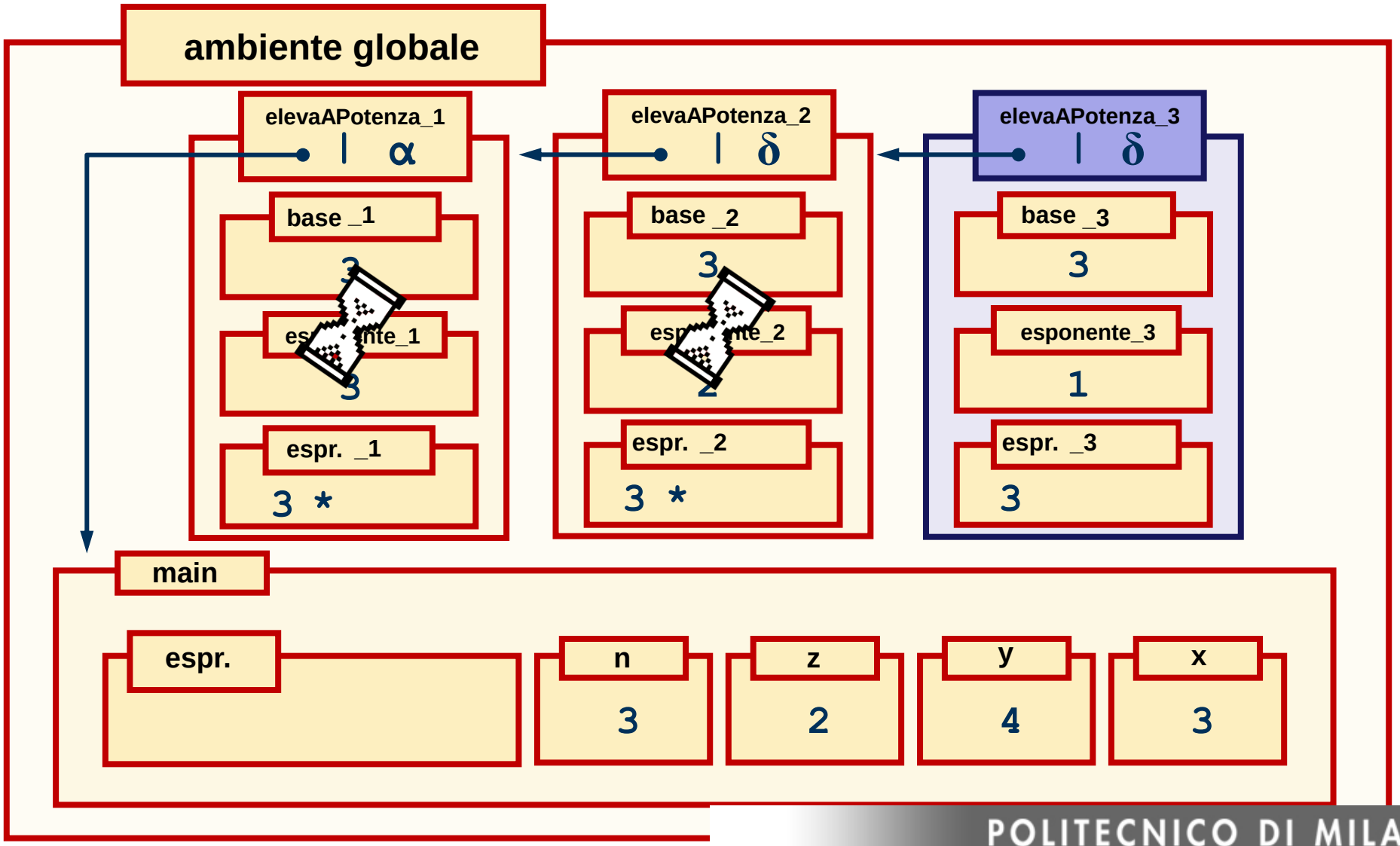
I linguaggi che ammettono funzioni ricorsive richiedono l'allocazione dinamica delle funzioni

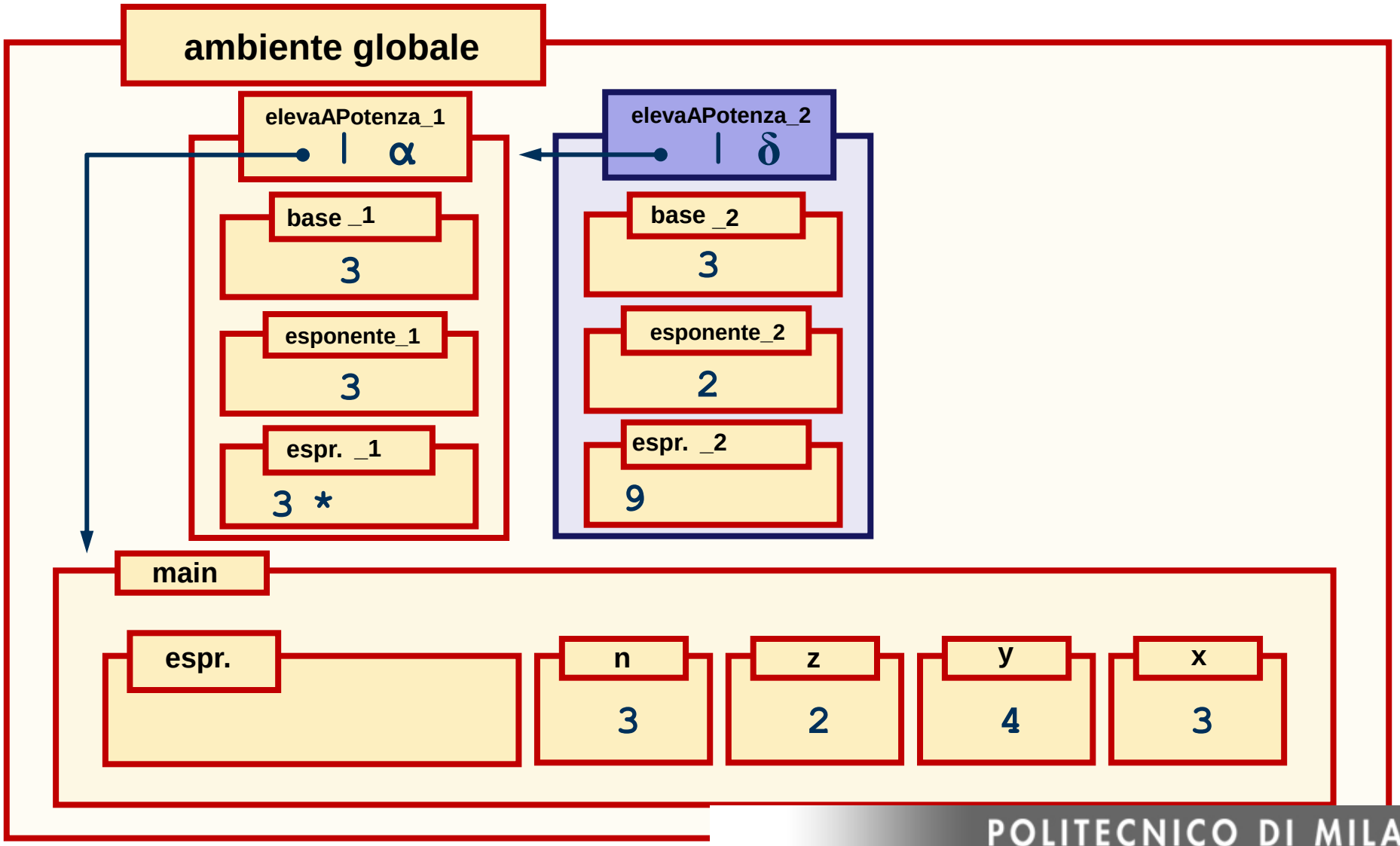

ambiente globale
elevaAPotenza_1
α
base _1
3
esponente_1
3
espr. _1
3 *
elevaAPotenza_2
δ
base _2
3
esponente_2
2
espr. _2
3 *
elevaAPotenza_3
δ
base _3
3
esponente_3
1
espr. _3
3
main
espr.
n
3
z
2
y
4
x
3

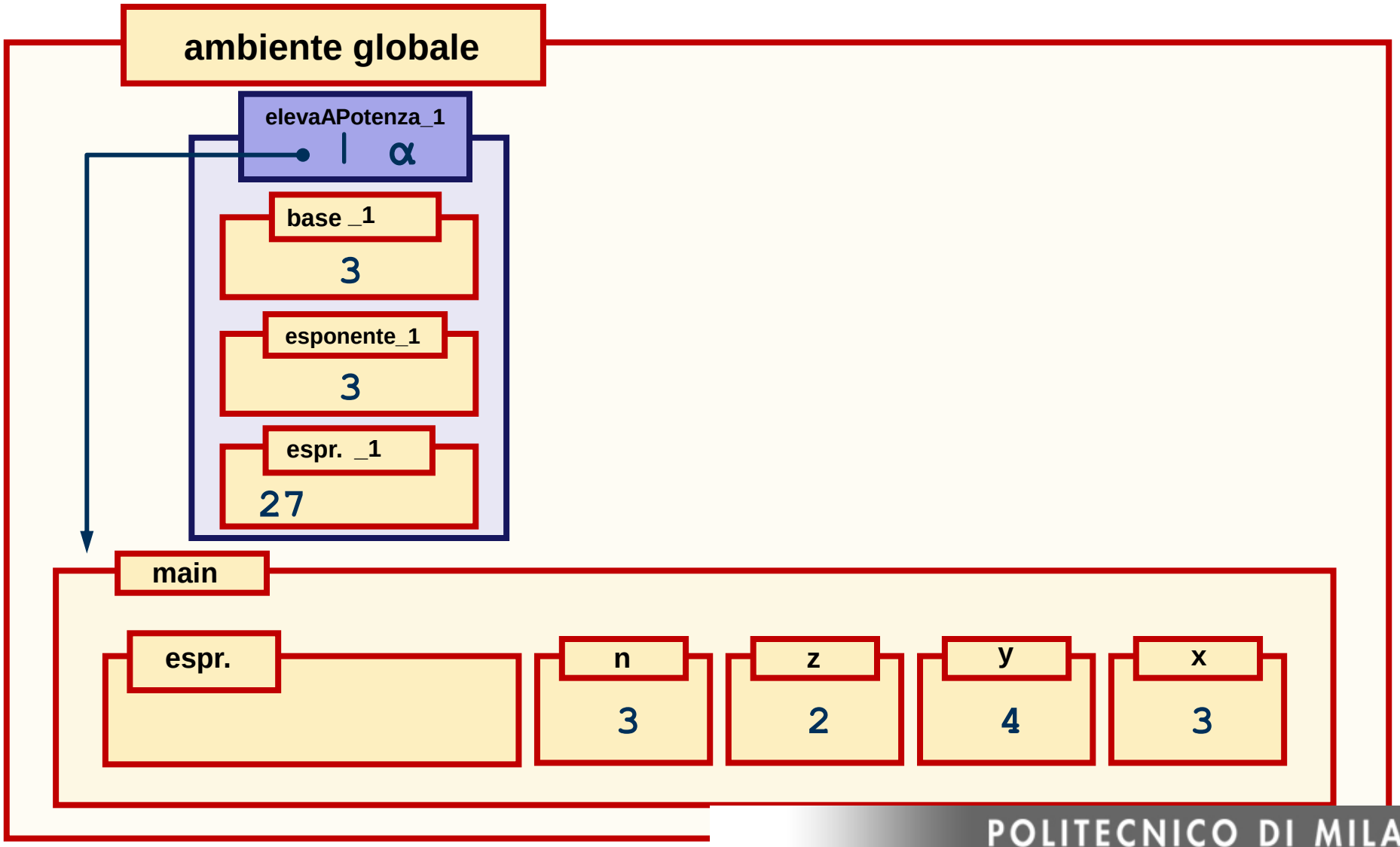

ambiente globale
elevaAPotenza_1
| α
base _1
3
esponente_1
3
espr. _1
3 *
elevaAPotenza_2
| δ
base _2
3
esponente_2
2
espr. _2
3 *
elevaAPotenza_3
| δ
base _3
3
esponente_3
1
espr. _3
3
main
espr.
n
3
z
2
y
4
x
3

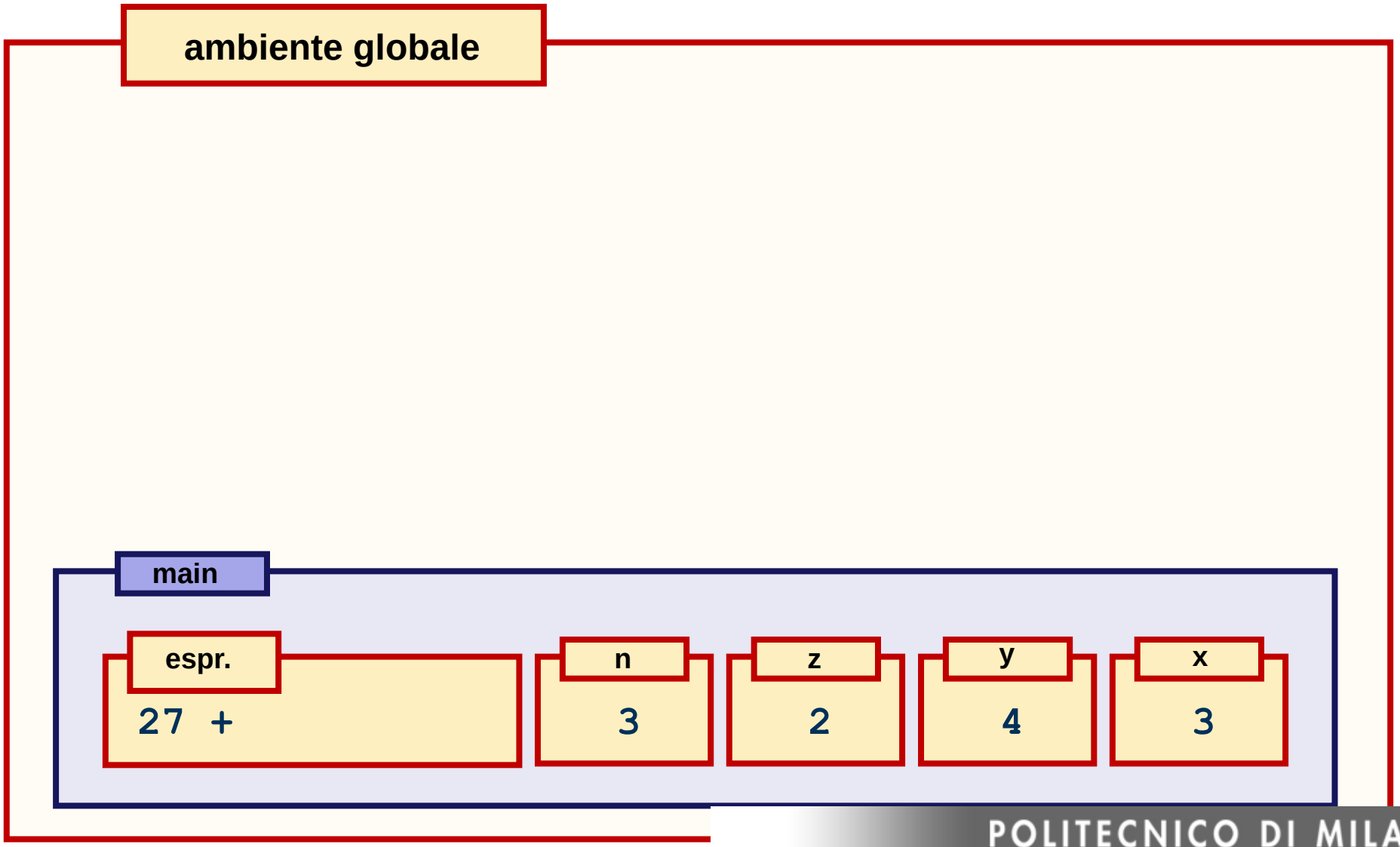

ambiente globale
elevaAPotenza_1
α
base _1
3
esponente_1
3
espr. _1
3 *
elevaAPotenza_2
δ
base _2
3
espr. _2
3 *
elevaAPotenza_3
δ
base _3
3
esponente_3
1
espr. _3
3
main
espr.
n
3
z
2
y
4
x
3


ambiente globale
elevaAPotenza_1
α
base _1
3
esponente_1
3
espr. _1
3 *
elevaAPotenza_2
δ
base _2
3
esponente_2
2
espr. _2
3 *
main
espr.
n
3
z
2
y
4
x
3


ambiente globale
elevaAPotenza_1
α
base _1
3
esponente_1
3
espr. _1
3 *
elevaAPotenza_2
δ
base _2
3
esponente_2
2
espr. _2
3 *
elevaAPotenza_3
δ
base _3
3
esponente_3
1
espr. _3
3
main
espr.
n
3
z
2
y
4
x
3


ambiente globale
elevaAPotenza_1
α
base _1
3
esponente_1
3
espr. _1
3 *
elevaAPotenza_2
δ
base _2
3
esponente_2
2
espr. _2
9
main
espr.
n
3
z
2
y
4
x
3


ambiente globale
elevaAPotenza_1
α
base _1
3
esponente_1
3
espr. _1
27
main
espr.
n
3
z
2
y
4
x
3


ambiente globale
main
espr.
27 +
n
3
z
2
y
4
x
3

Soluzione ricorsiva

```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

Soluzione iterativa

```
int elevaAPotenza(int base, int esponente)
{
  int prodMancanti; int potenza;
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
    potenza *= base;
  return potenza;
}
```

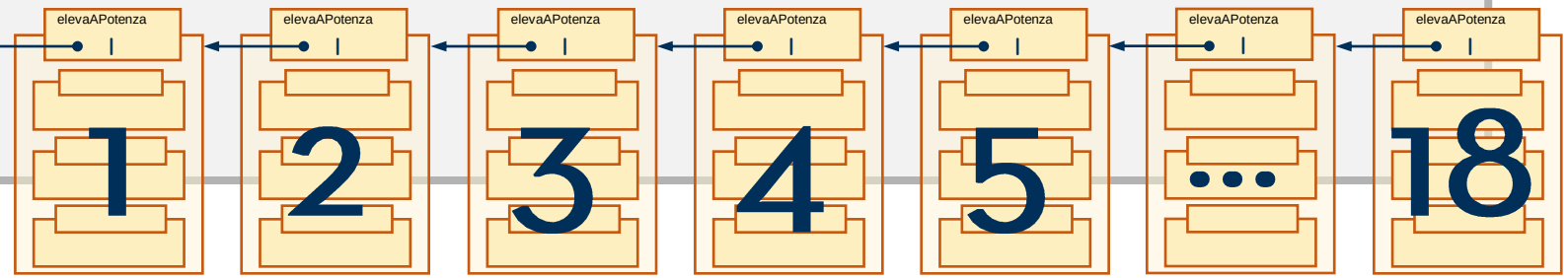
Soluzione ricorsiva

Soluzione iterativa

Aspetto secondario: efficienza

Soluzione ricorsiva

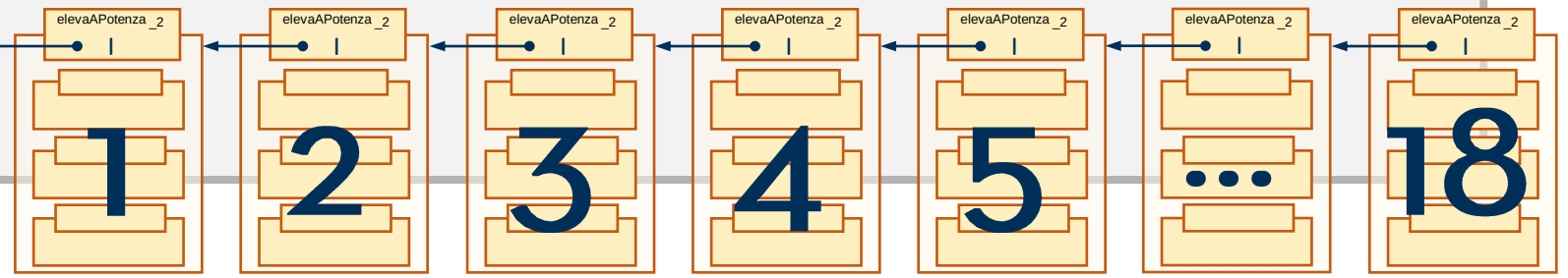
2^{18}



Soluzione iterativa

Soluzione ricorsiva

2^{18}



base_13 esponente_18 potenza_1 base_10 esponente_4 prodmancanti_8 prodmancanti_9 esponente_12 potenza_15 base_3 base_2 potenza_6 base_14 prodmancanti_17 esponente_5

Soluzione ricorsiva

Soluzione iterativa

Aspetto principale: semplicità

Soluzione ricorsiva ✓

Soluzione iterativa

Soluzione ricorsiva

se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$

```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

Soluzione ricorsiva ✓

se esponente = 1

$base^{esponente} = base$

se esponente > 1

$base^{esponente} = base * base^{esponente-1}$

Specifica

```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 1)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

Soluzione ricorsiva

$se\ esponente = 1$

$base^{esponente} = base$

$se\ esponente > 1$

$base^{esponente} = base * base^{esponente-1}$

Specifica

Meccanismo di soluzione

```
int elevaAPotenza(int base, int esponente)
{ if (esponente == 0)
    return base;
  if (esponente > 1)
    return base * elevaAPotenza(base, esponente-1);
}
```

se esponente = 1

$base^{esponente} = base$

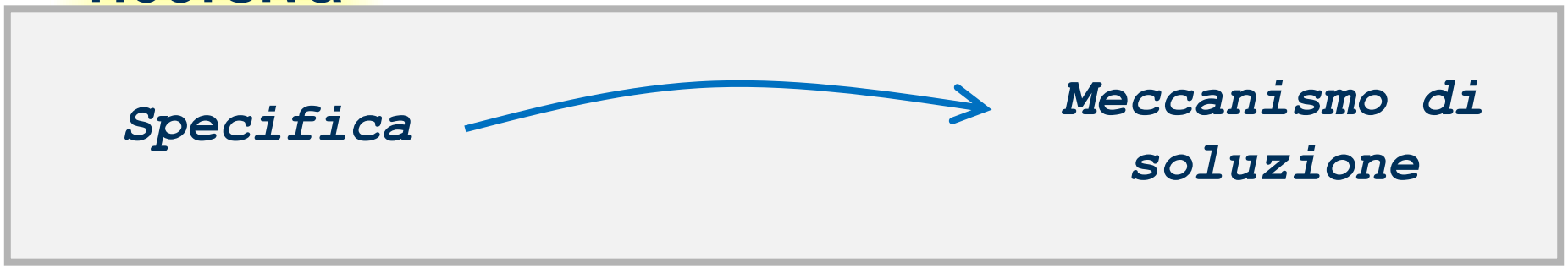
se esponente > 1

$base^{esponente} = base * base^{esponente-1}$

Soluzione iterativa

```
int elevaAPotenza(int base, int esponente)
{
  int prodMancanti; int potenza;
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
    potenza *= base;
  return potenza;
}
```

Soluzione ricorsiva



Soluzione iterativa