POLITECNICO DI MILANO

INFORMATICA

L'uso di funzioni all'interno di espressioni

Funzioni all'interno di espressioni

```
elevaAPotenza( a, b )
beta  = alfa +  (a elevato a b);
```

Funzioni all'interno di espressioni

```
...
 elevaAPotenza( a, b, k);
 beta  = alfa +  k;
...
```

C++

```
...
elevaAPotenza( a, b, k);

beta  = alfa + k;
...
```

C++

ritornare un valore
associandolo al nome
della funzione

prefiggendo il tipo

```
void elevaAPotenza(int base, int esponente, int & potenza);
```

```
int elevaAPotenza(int base, int esponente)
{...}

beta  = alfa +  elevaAPotenza(a, b);

...
```

```
void elevaAPotenza(int base, int esponente, int & potenza);
```

```
int elevaAPotenza(int base, int esponente)
{...}
...
beta  = alfa +  elevaAPotenza(a, b);
...
```

Ritorno del valore di potenza

```
int elevaAPotenza(int base, int esponente)

{ //versione per esponente positivo
    int prodMancanti;  // variabile locale alla funzione
    int potenza;                  // variabile necessaria per il calcolo
                        // del valore di ritorno

    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
      potenza *= base;
    return potenza;
}
```

return – come parola chiave senza argomento termina la funzione

Ritorno del valore di potenza

```
int elevaAPotenza(int base, int esponente)

{ //versione per esponente positivo
    int prodMancanti;  // variabile locale alla funzione
    int potenza;                  // variabile necessaria per il calcolo
                        // del valore di ritorno

    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
      potenza *= base;
    return potenza;
}
```

return [*espressione*]– come parola chiave con argomento termina la funzione . . .

Ritorno del valore di potenza

```
int elevaAPotenza(int base, int esponente)

{ //versione per esponente positivo
    int prodMancanti;  // variabile locale alla funzione
    int potenza;                    // variabile necessaria per il calcolo
                          // del valore di ritorno

    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
      potenza *= base;
    return potenza;
}
```

```
void elevaAPotenza(int base, int esponente, int & potenza)
```

Ritorno del valore di potenza

```
int elevaAPotenza(int base, int esponente)

{ //versione per esponente positivo
    int prodMancanti;  // variabile locale alla funzione
    int potenza;                  // variabile necessaria per il calcolo
                        // del valore di ritorno

    potenza = 1;
    for (prodMancanti = esponente; prodMancanti > 0; prodMancanti--)
      potenza *= base;
    return potenza;
}
```

return [*espressione*]– come parola chiave con argomento termina la funzione e ritorna il valore di [*espressione*] alla funzione chiamante

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * La funzione fa uso di parametri solo per ricevere dati, mentre
 * l'unico risultato viene fornito associandolo al nome della funzione.
 */
#include <iostream.h>

void elevaAPotenza(int base, int esponente, int & potenza)
{…}

void main()
{…
   elevaAPotenza(x, n, xAllaN);
 …
}

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

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

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

  if (xAllaN + yAllaN == zAllaN)
  …
}
```

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * La funzione fa uso di parametri solo per ricevere dati, mentre
 * l'unico risultato viene fornito associandolo al nome della funzione.
 */
#include <iostream.h>

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 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
      xAllaN, yAllaN, zAllaN;           //contengono x,y,z elevati a n

  "presenta le funzionalità del programma"
  "leggi i dati e verifica che rispondano alle specifiche"

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

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

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

  if (xAllaN + yAllaN == zAllaN)
  …
}
```

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * La funzione fa uso di parametri solo per ricevere dati, mentre
 * l'unico risultato viene fornito associandolo al nome della funzione.
 */
#include <iostream.h>

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 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
      xAllaN, yAllaN, zAllaN;           //contengono x,y,z elevati a n

  "presenta le funzionalità del programma"
  "leggi i dati e verifica che rispondano alle specifiche"

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

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

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

  if (xAllaN + yAllaN == zAllaN)
  ...
}
```

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
 * La funzione fa uso di parametri solo per ricevere dati, mentre
 * l'unico risultato viene fornito associandolo al nome della funzione.
 */
#include <iostream.h>

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

Più efficace e sintetico

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
                                    ri solo per ricevere dati, mentre
                                    to associandolo al nome della funzione.

#

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

Chiamata in un espressione

```
/* Tentativo senza speranza di dimostrare la falsità dell'ultimo
 * teorema di Fermat
                                ri solo per ricevere dati, mentre
                                to associandolo al nome della funzione.
#
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 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
  "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 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))
```

ambiente globale

```
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaA
```

```
{ 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))
```

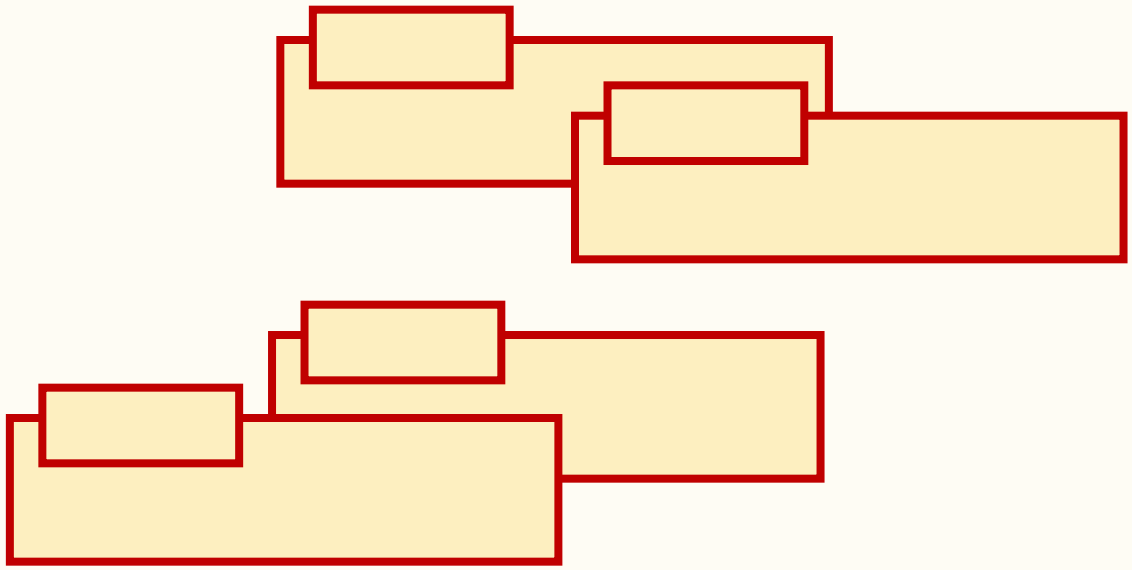
ambiente globale

00101110100101101100011010110010111010011001010**1110**

```
{ int x, y, z, n,          //dati su cui operare
  "presenta le funzionalità del programma"
  "leggi i dati e verifica che rispondano alle spe
  if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
```

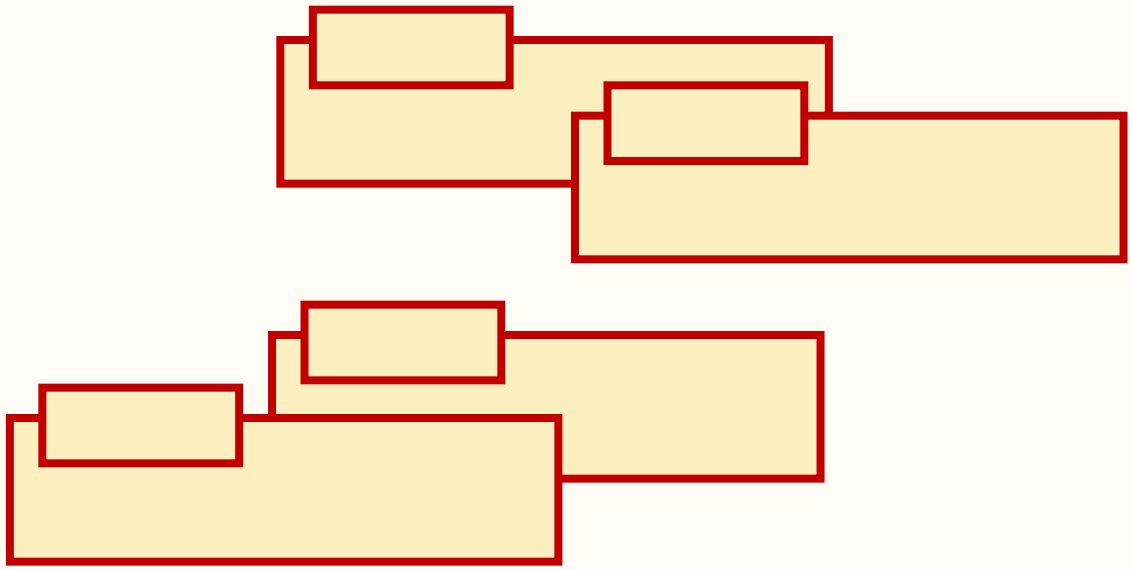
Espressione

ambiente globale



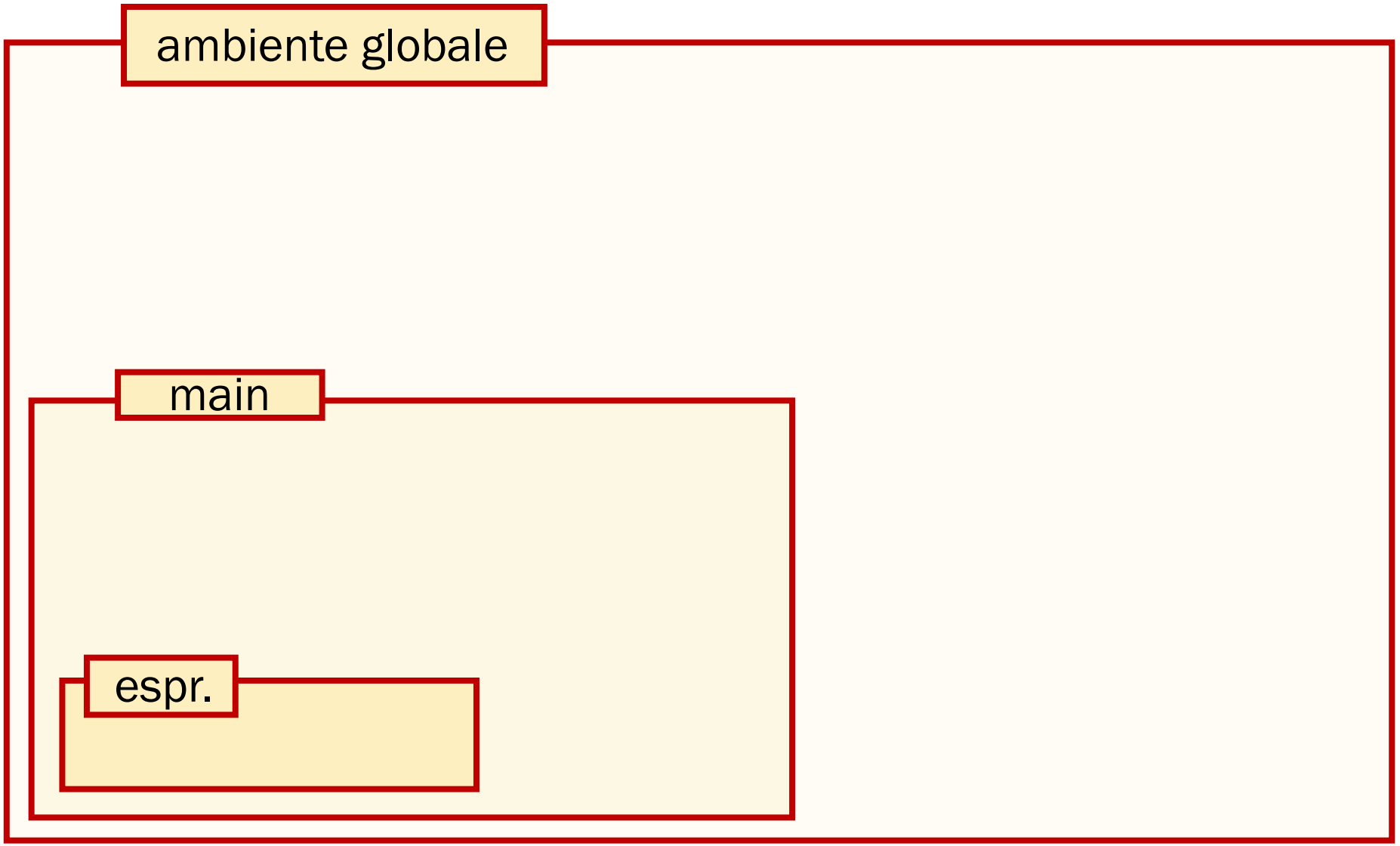
```
{ int x, y, z, n,          //dati su cui operare
  "presenta le funzionalità del programma"
  "leggi i dati e verifica che rispondano alle specifiche"
  if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
```

ambiente globale



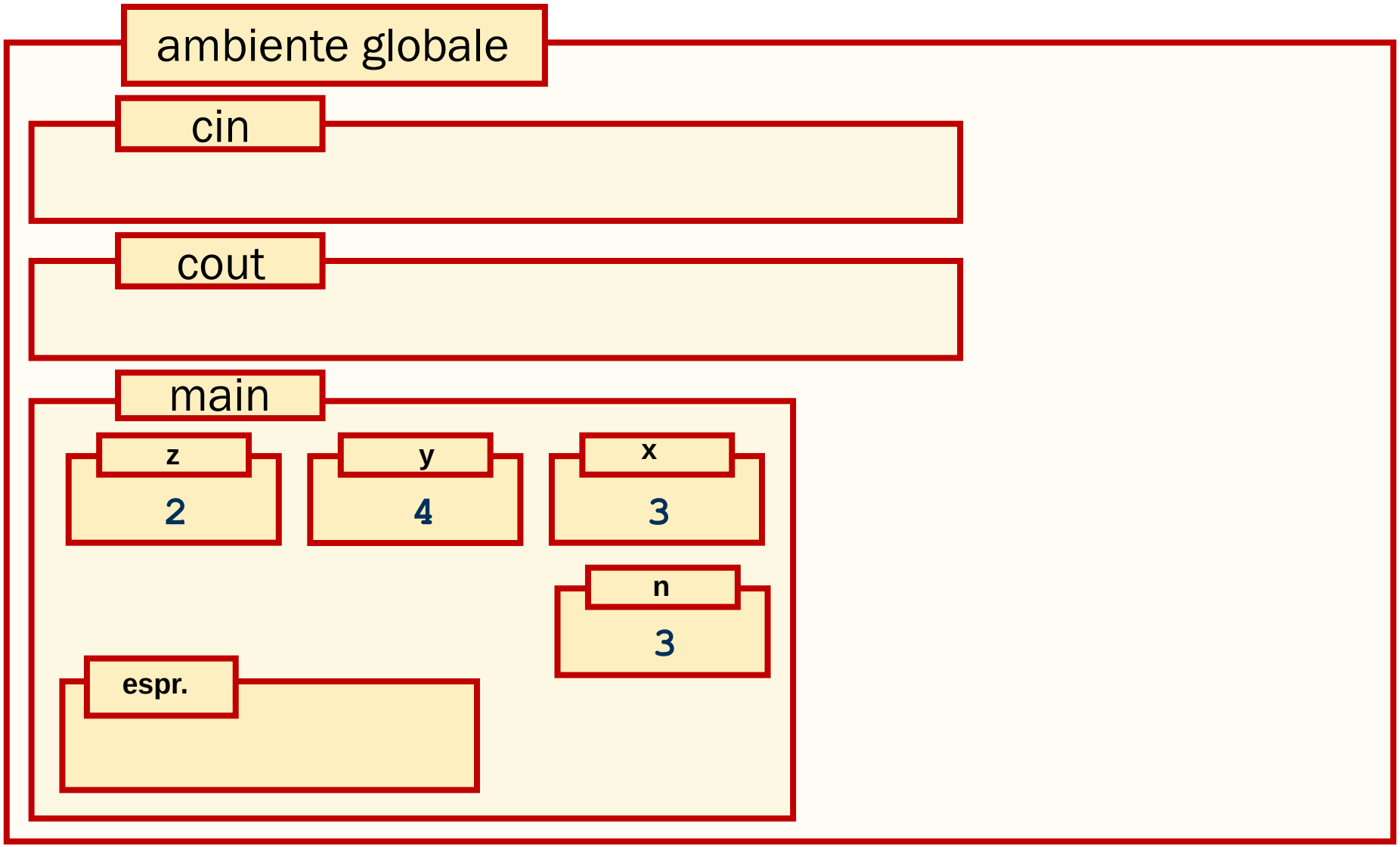
```
{ int x, y, z, n,          //dati su cui operare
  "presenta le funzionalità del programma"
  "leggi i dati e verifica che rispondano alle spe
  if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
```

espr

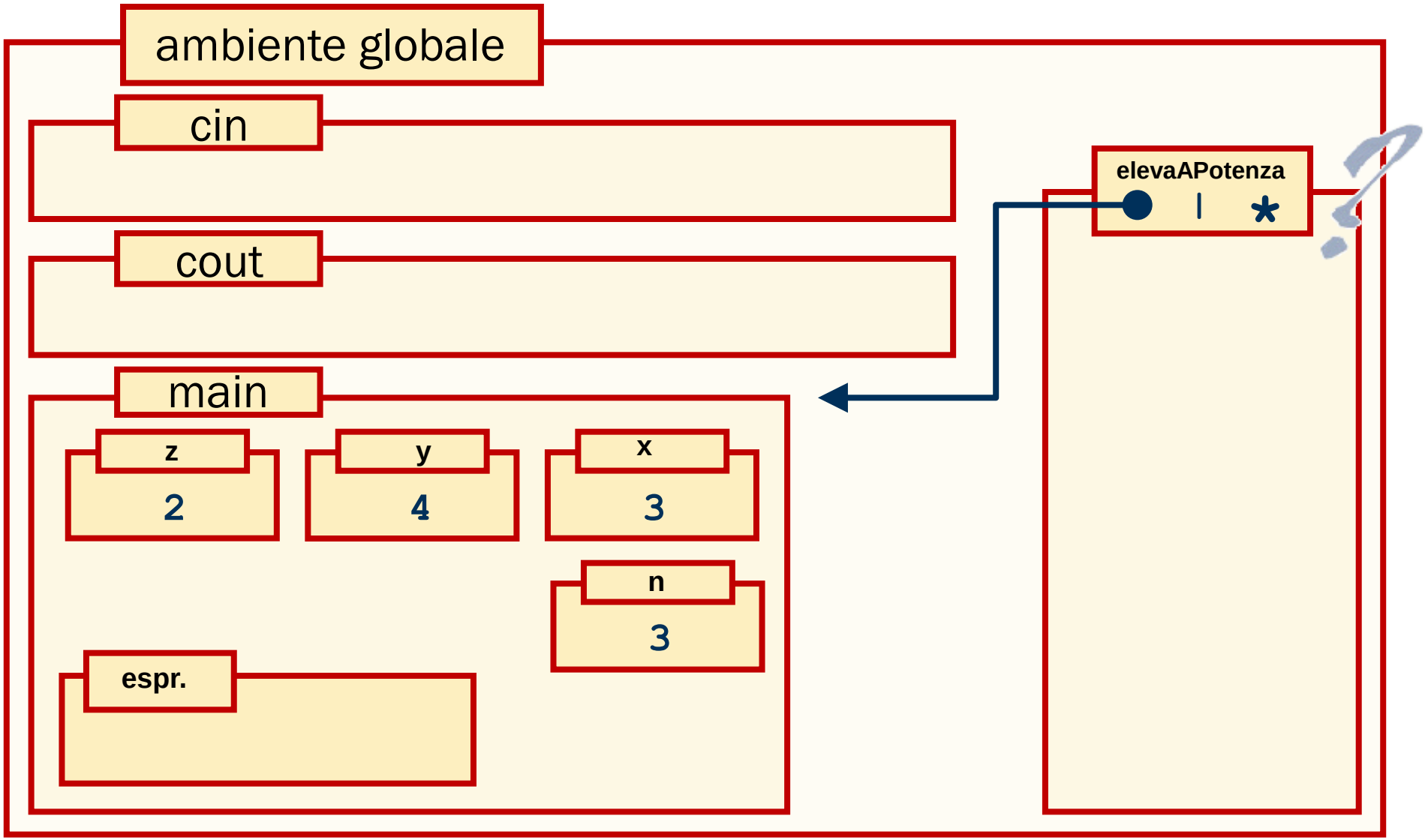


```
}
void main()
{ int x, y, x, n,                    // dati su cui operare
  " Presenta le funzionalità del programma"
 "leggi i dati e verifica che rispondano alle specifiche"
   if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
```

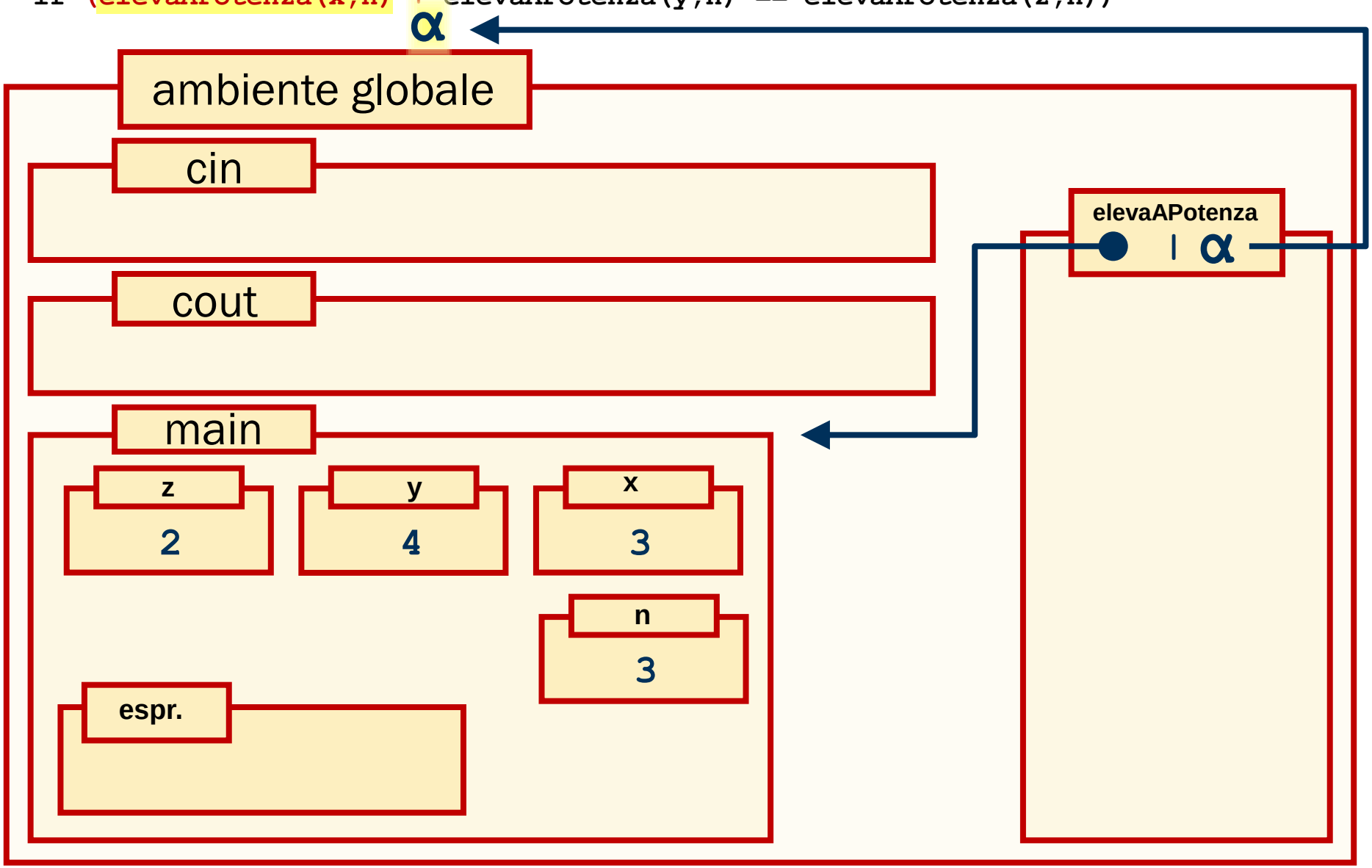
```
"presenta le funzionalità del programma"
"leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
```



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

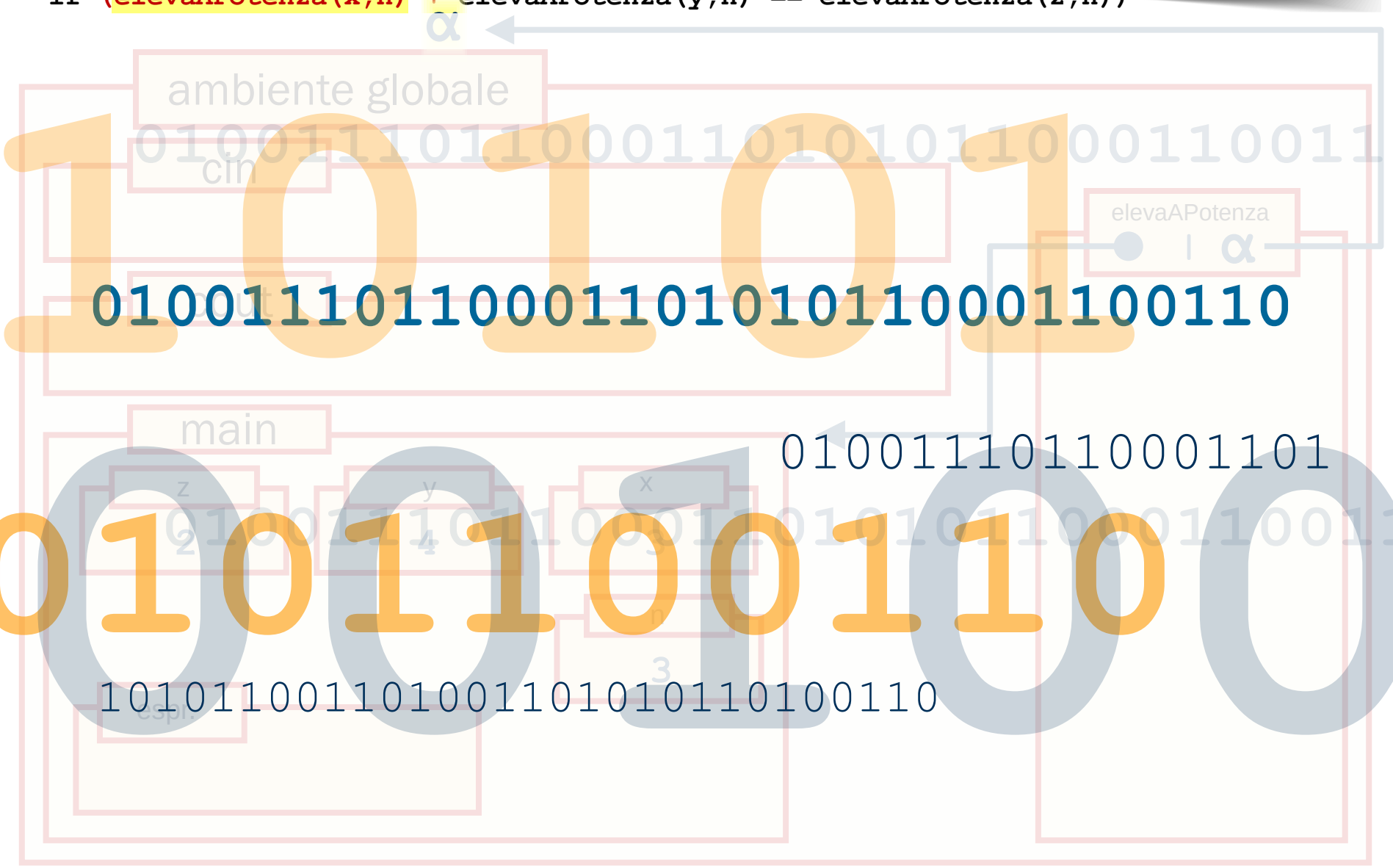


```
"presenta le funzionalità del programma"
"leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
```

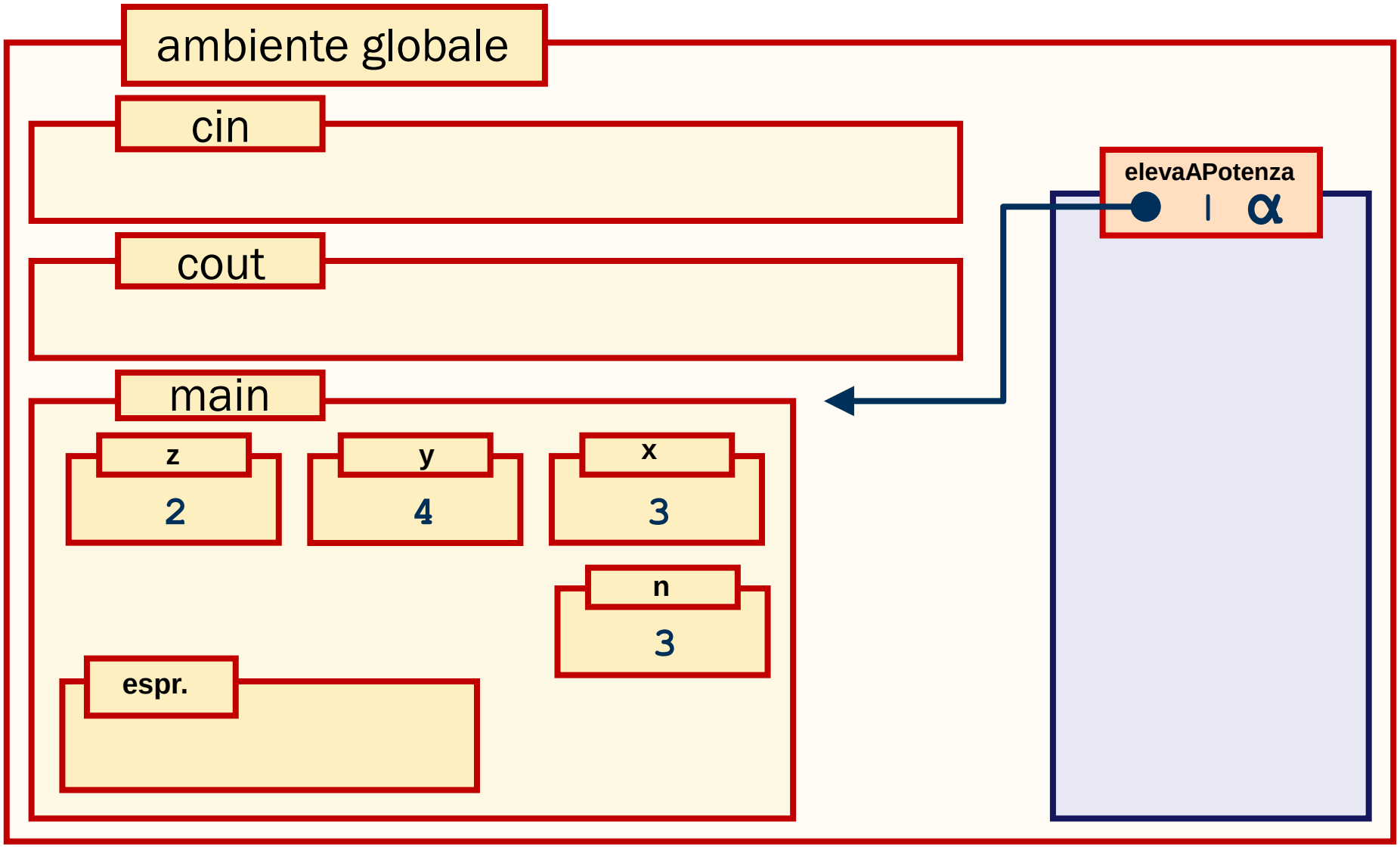


```
"presenta le funzionalità del programma"
"leggi i dati e verifica che rispondano alle specifiche
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
```

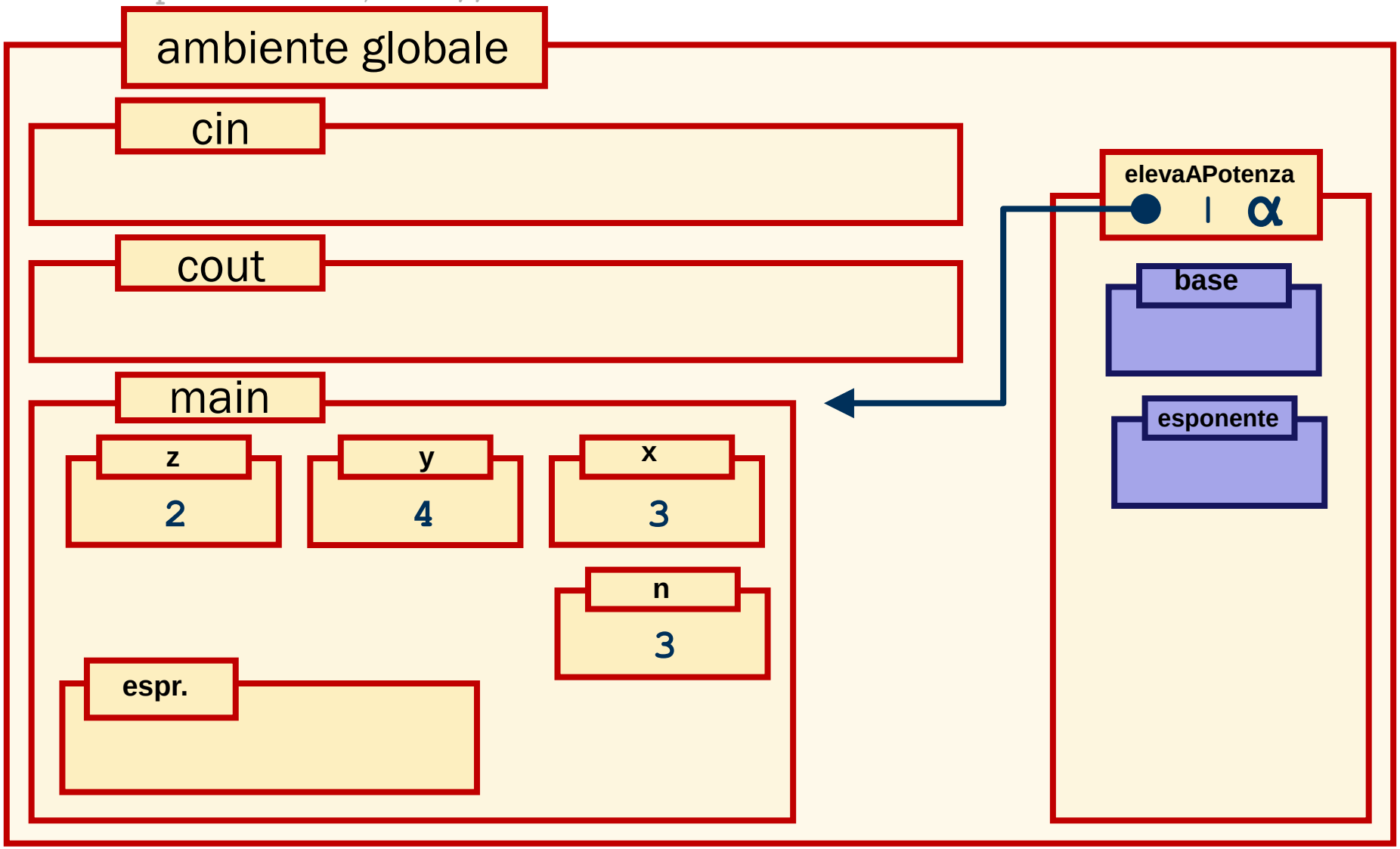
Linguaggio macchina



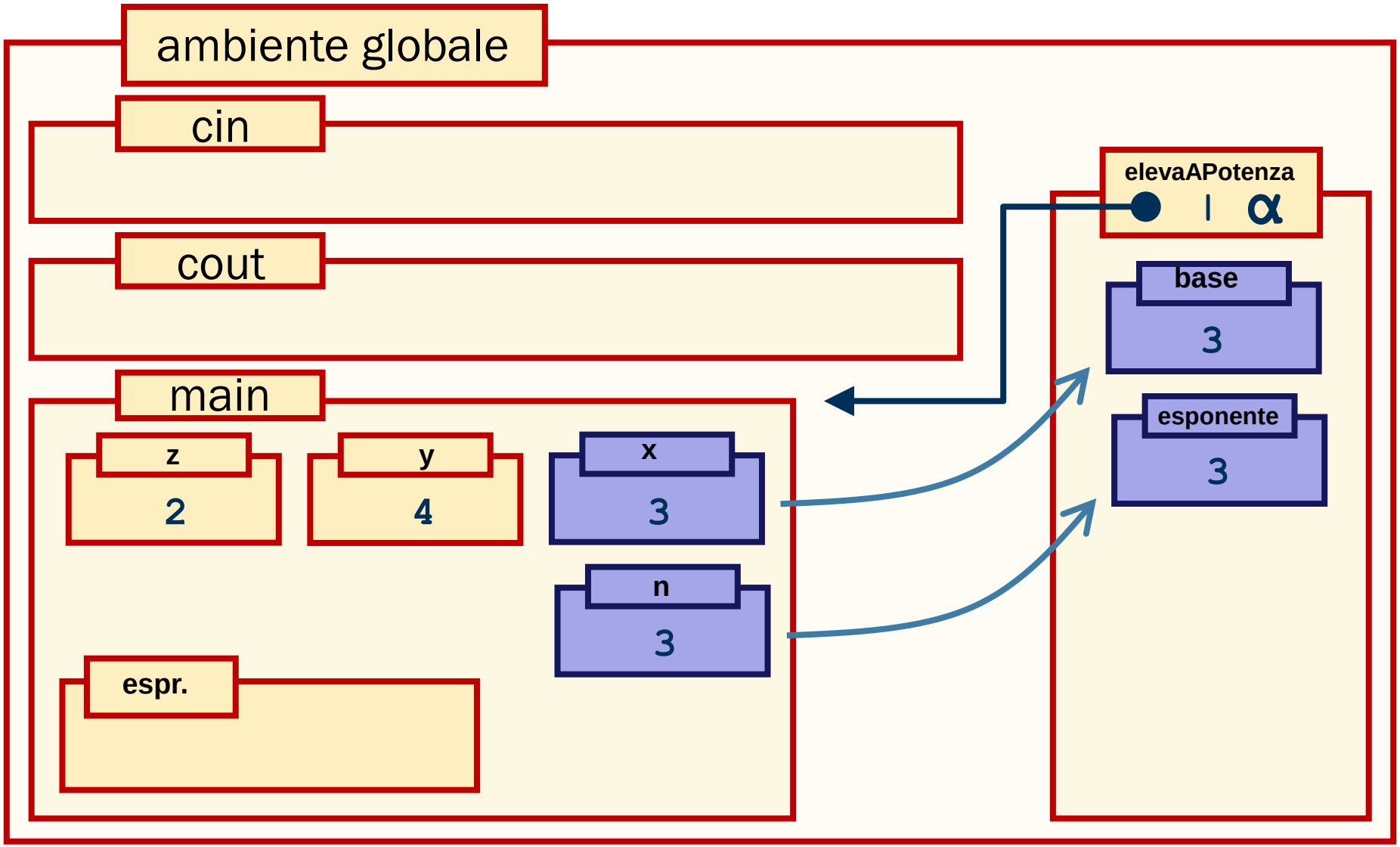
```
"presenta le funzionalità del programma"
"leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
```



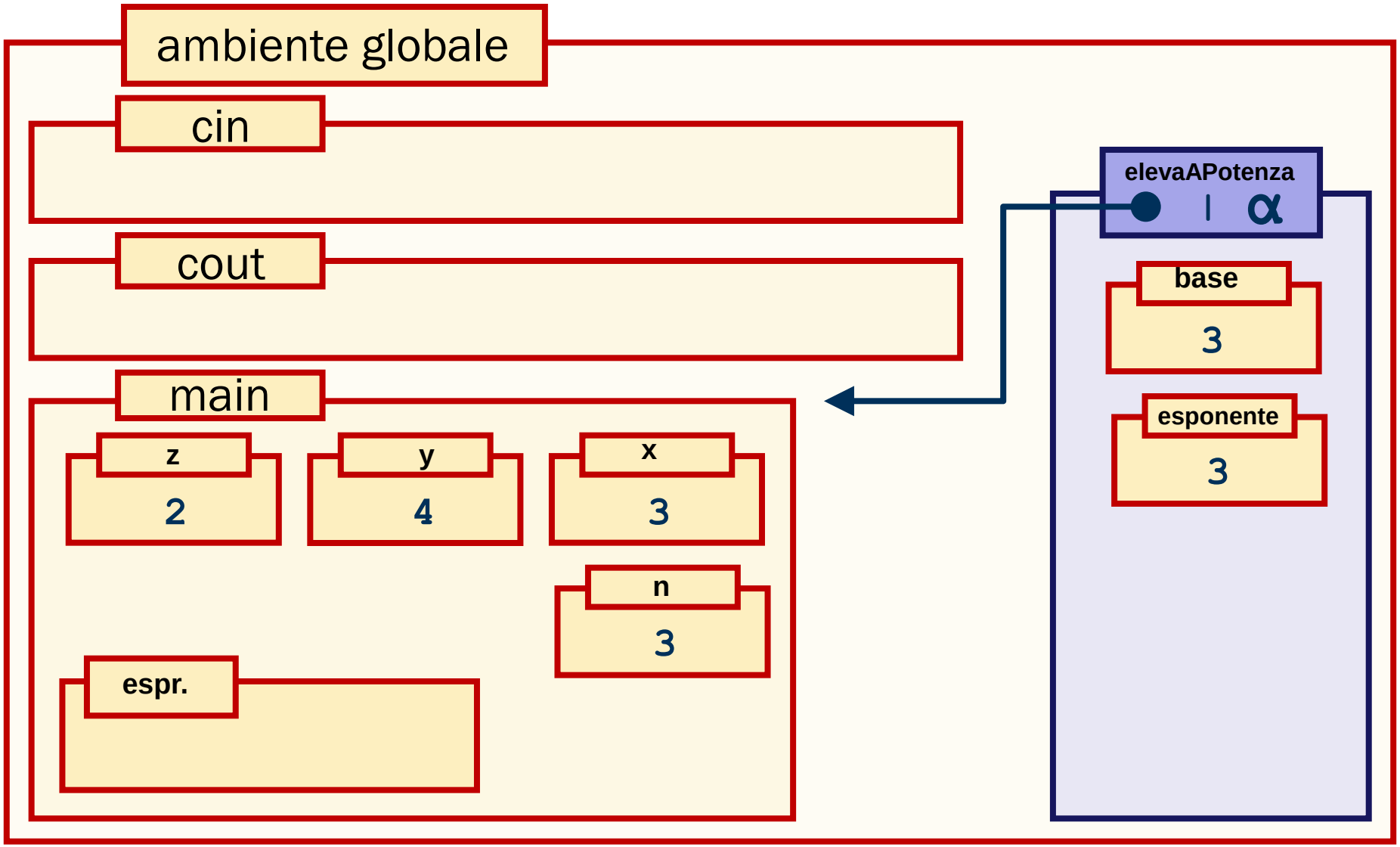
```
#include <iostream.h>

int elevaAPotenza(int base, int esponente) // funzione con valore di ritorno
{ //versione con esponente positivo
    int prodMancanti;     // variabile locale alla funzione elevaAPotenza
```



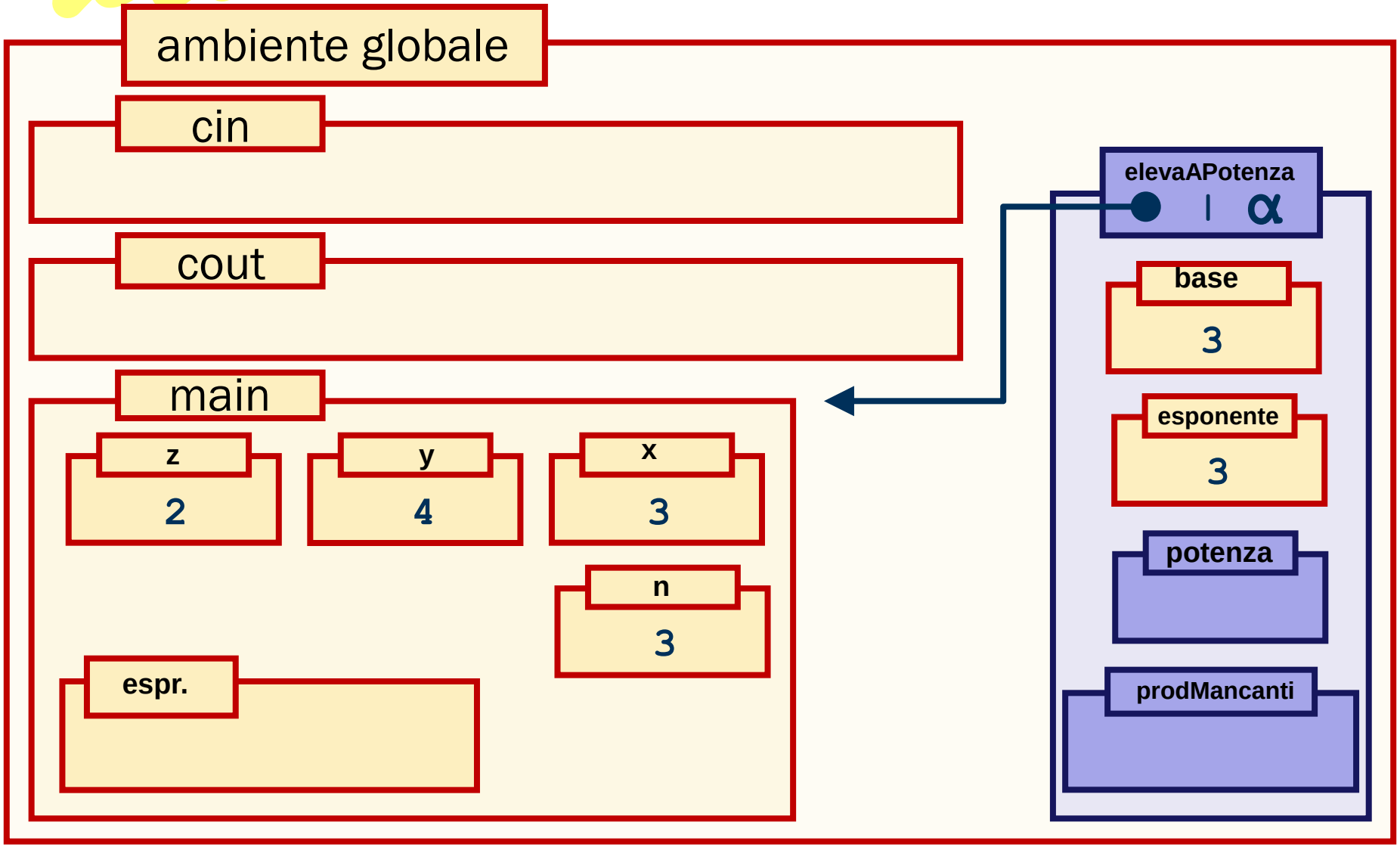
```
"presenta le funzionalità del programma"
"leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
```



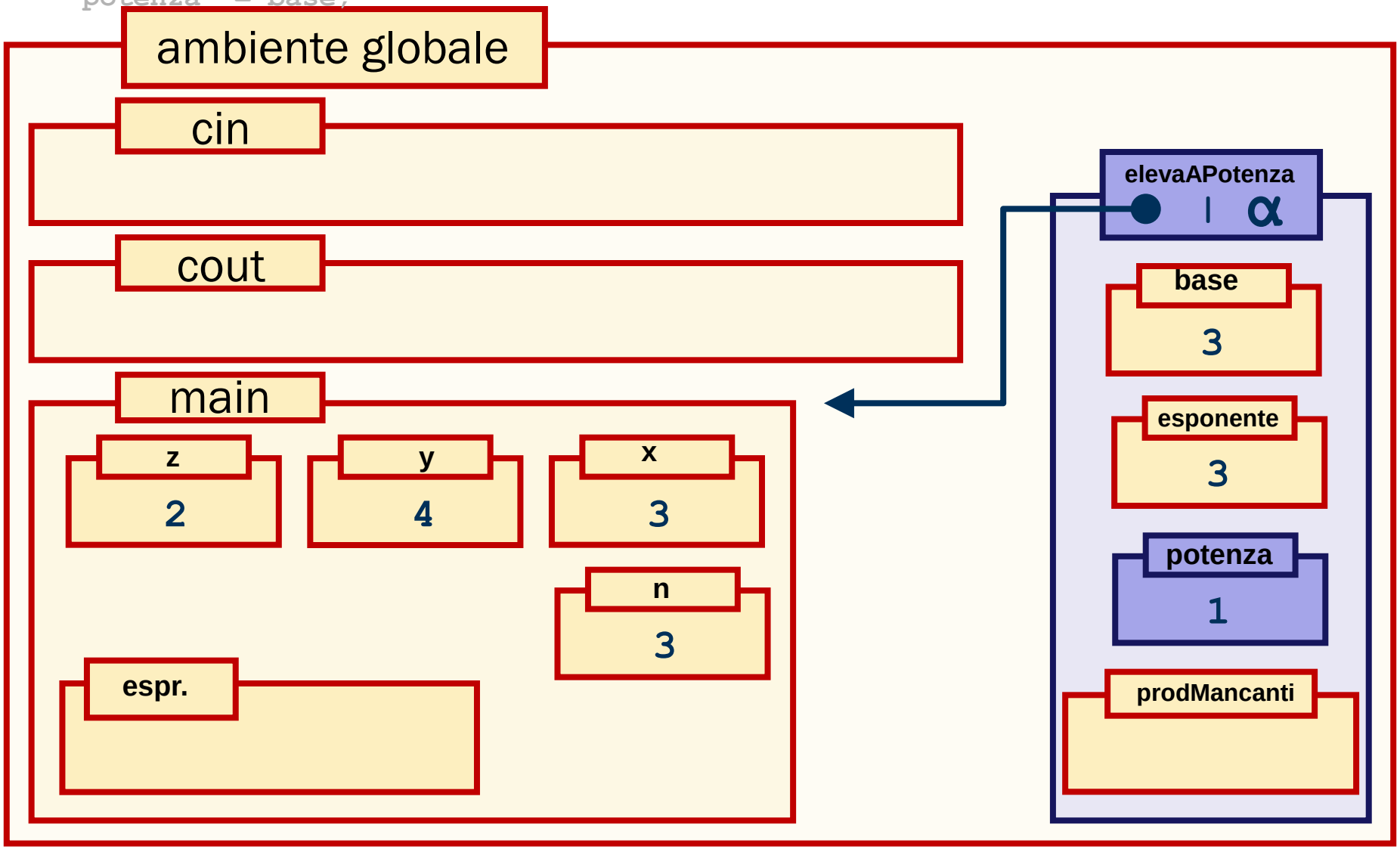
```
int elevaAPotenza(int base, int esponente) // funzione con valore di ritorno
{ //versione con esponente positivo
    int prodMancanti;     // variabile locale alla funzione elevaAPotenza
    int potenza;          // variabile necessaria per il calcolo
```



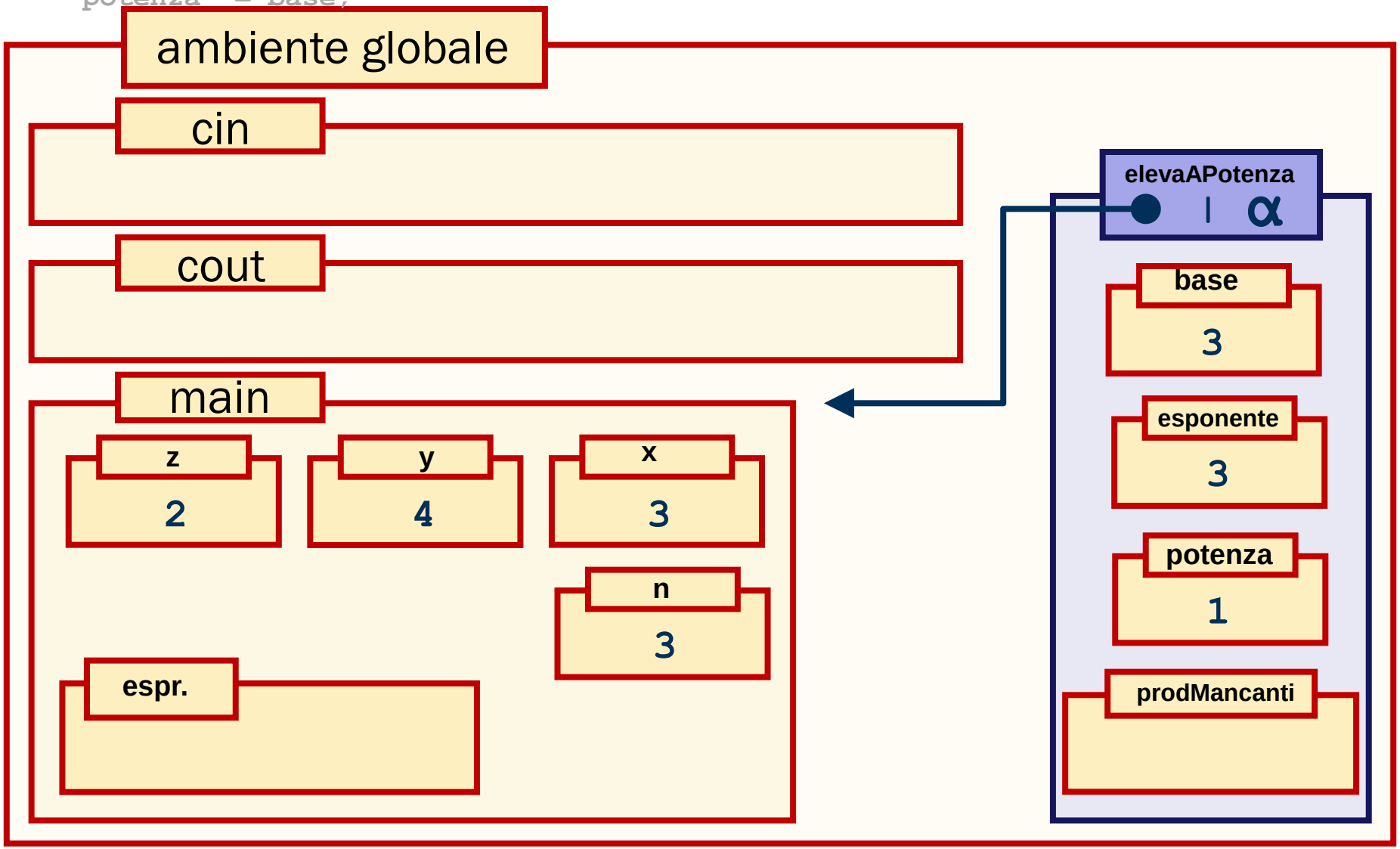
```
int elevaAPotenza(int base, int esponente) // funzione con valore di ritorno
{ //versione con esponente positivo
   int prodMancanti;              // variabile locale alla funzione elevaAPotenza
   int potenza;           // variabile necessaria per il calcolo
```



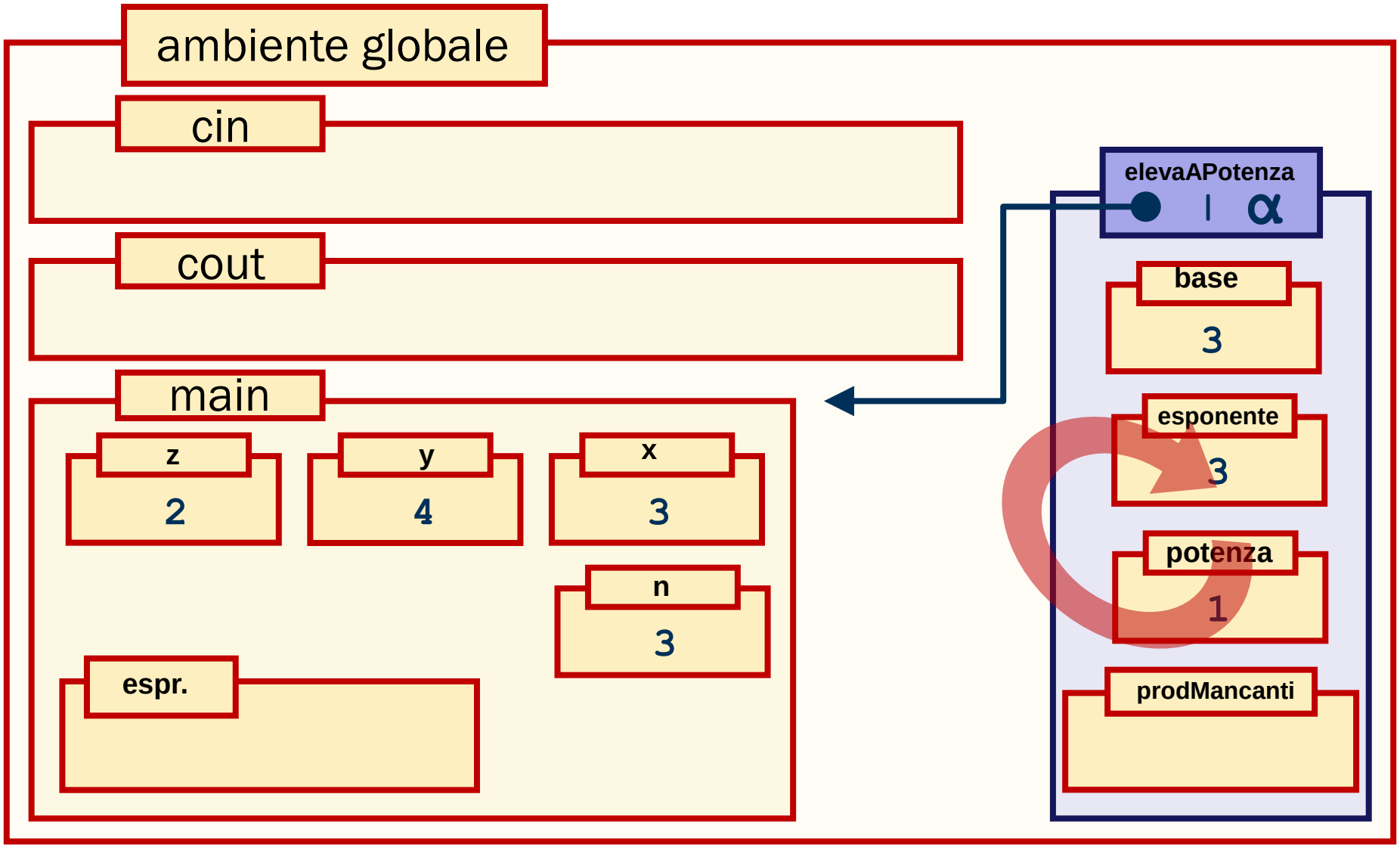
```
int potenza;              // variabile necessaria per il calcolo
                          // del valore di ritorno
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
  potenza *= base;
```



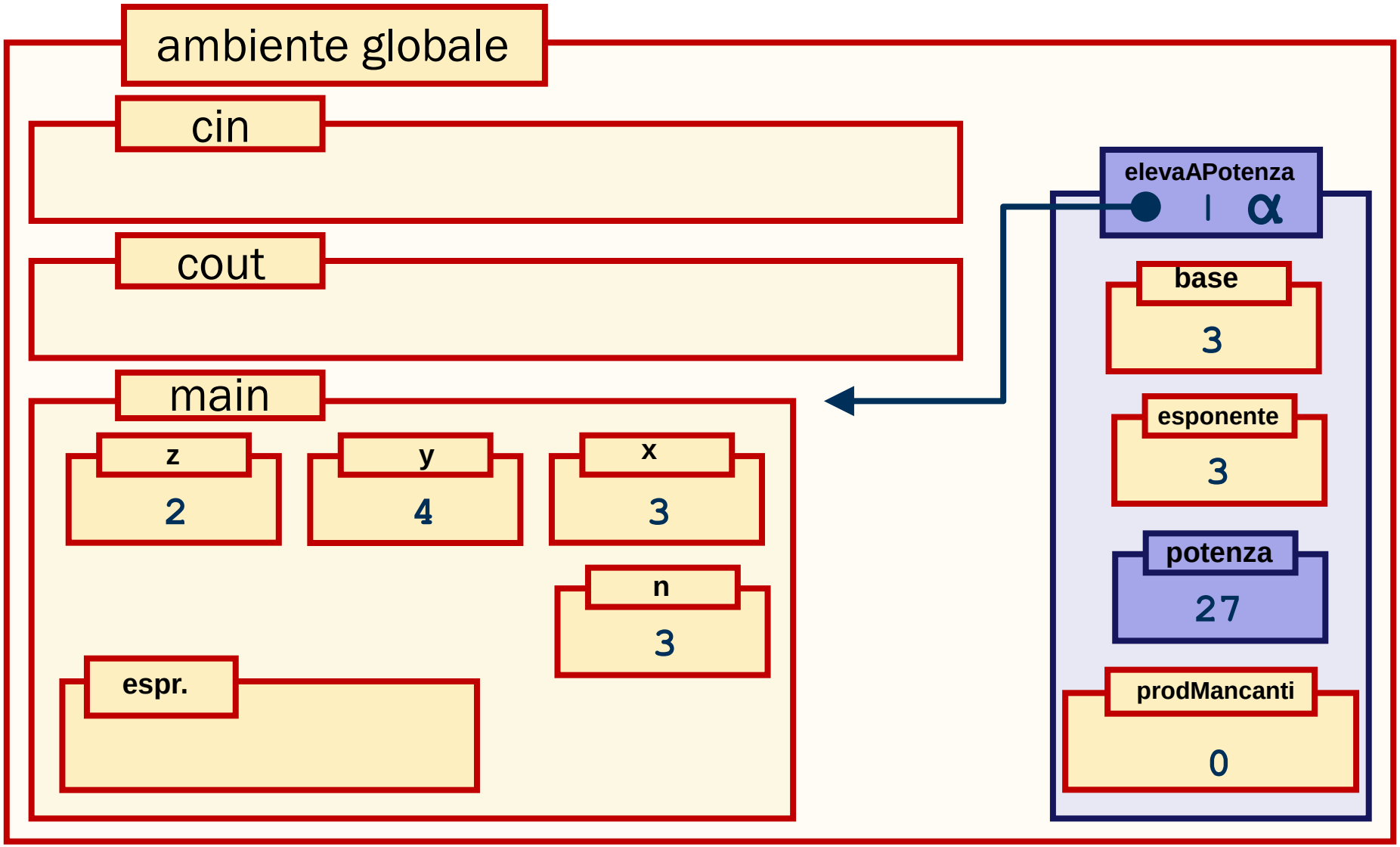
```
int potenza;              // variabile necessaria per il calcolo
                          // del valore di ritorno
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
  potenza *= base;
```



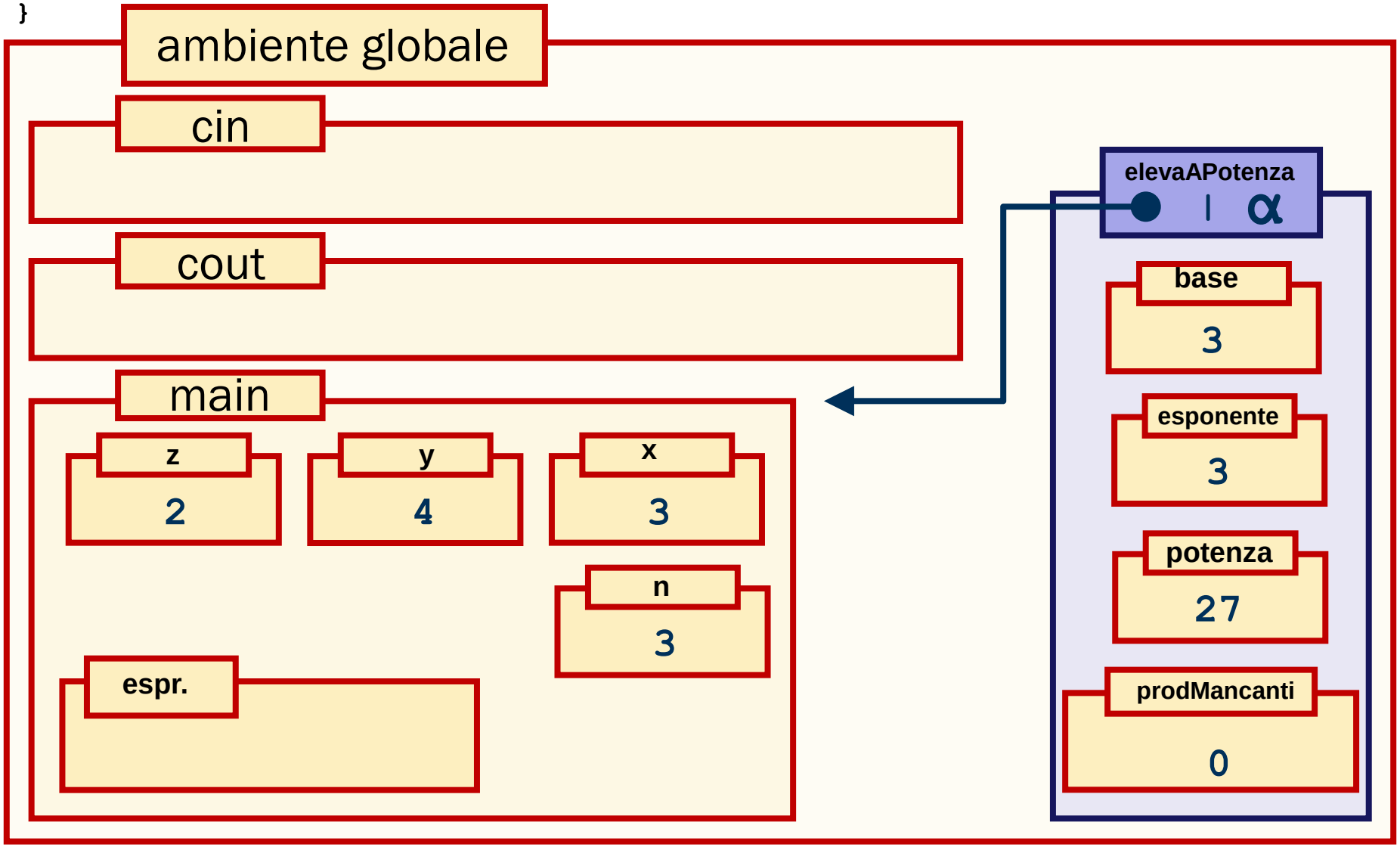
```
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
  potenza *= base;
return potenza;
```



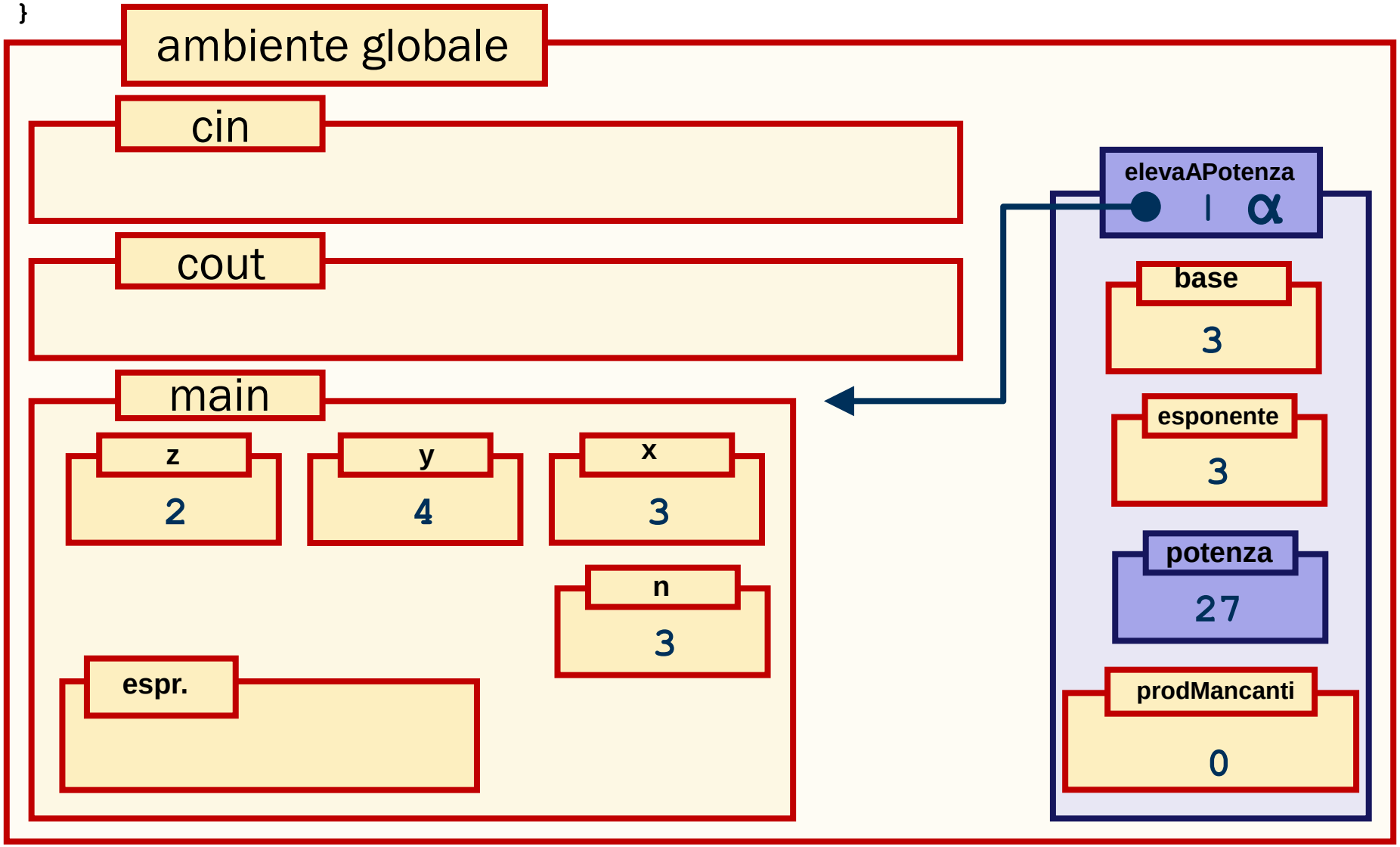
```
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
  potenza *= base;
return potenza;
```



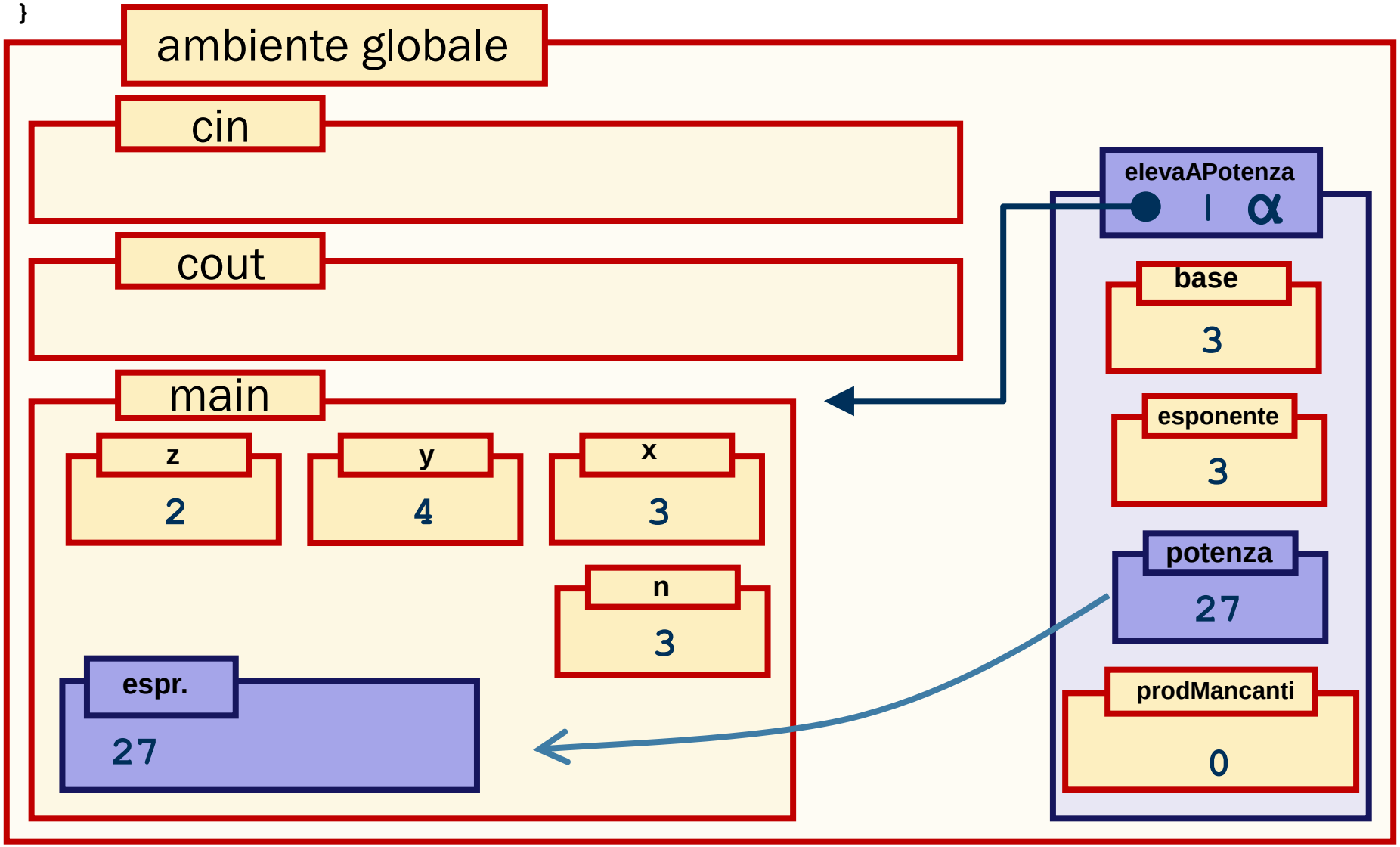
```
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
    potenza *= base;
  return potenza;
}
```



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

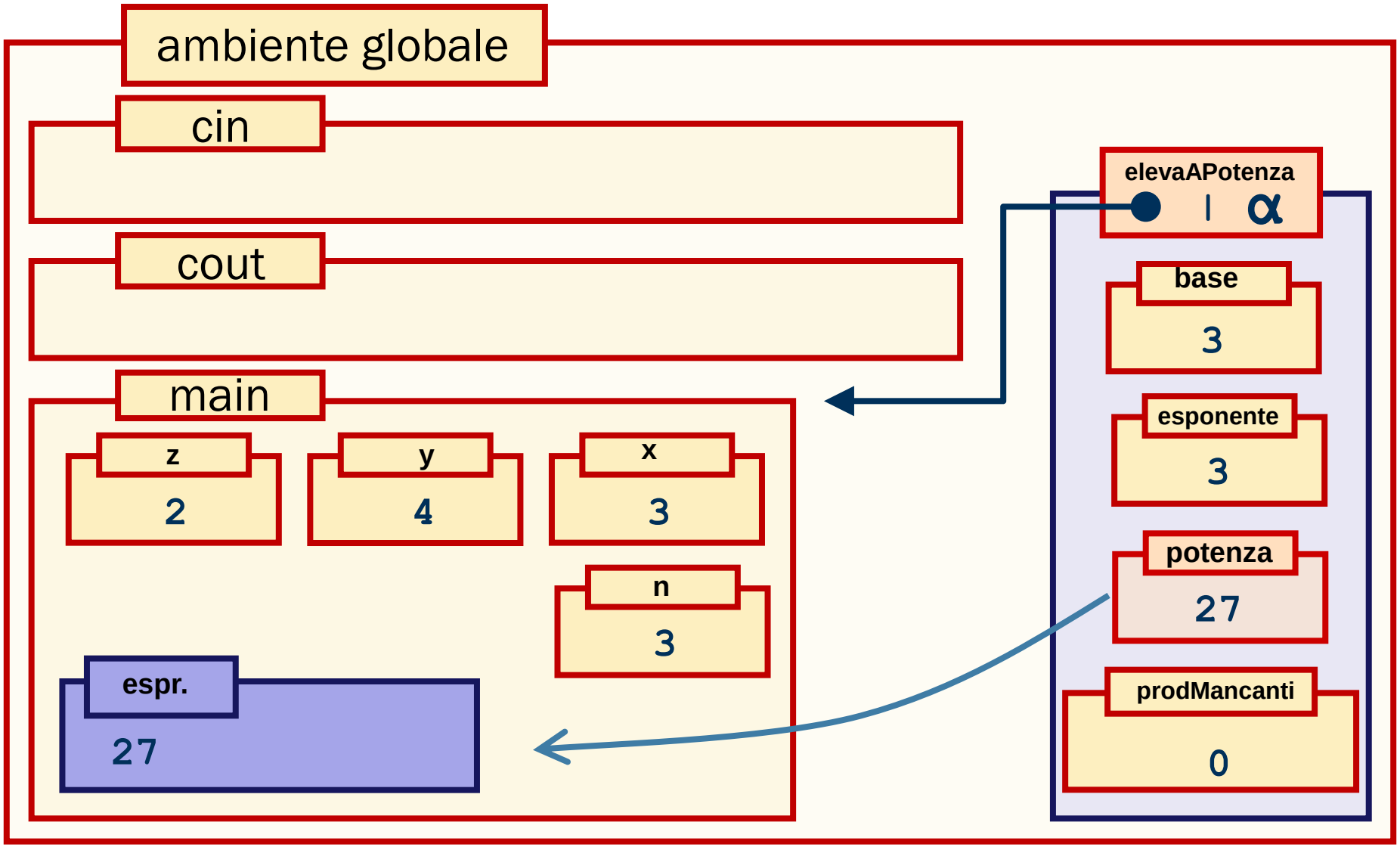


```
  potenza = 1;
  for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
    potenza *= base;
  return potenza;
}
```

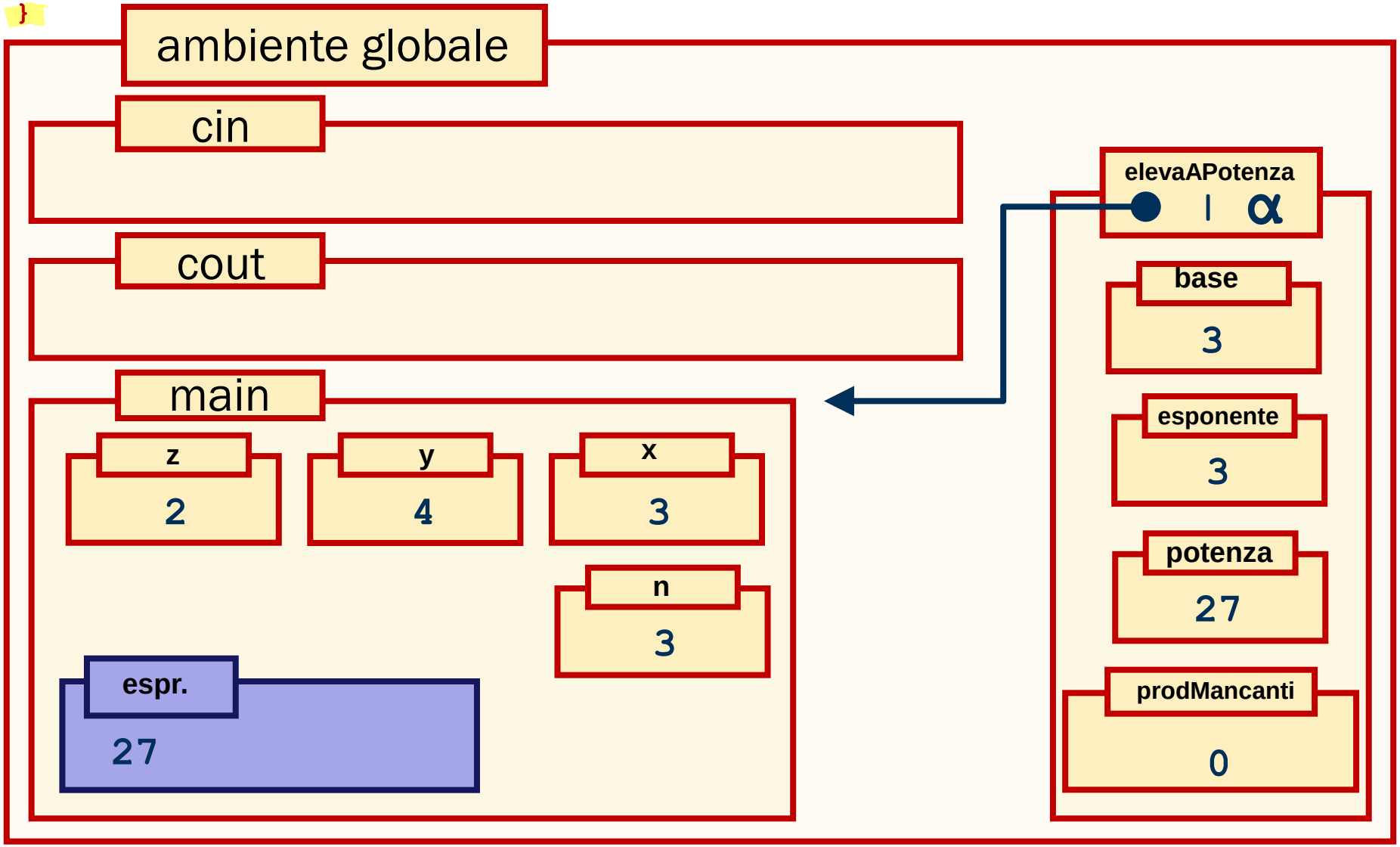


```
"leggi i dati e verifica che rispondano alle specifiche"
 if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
```

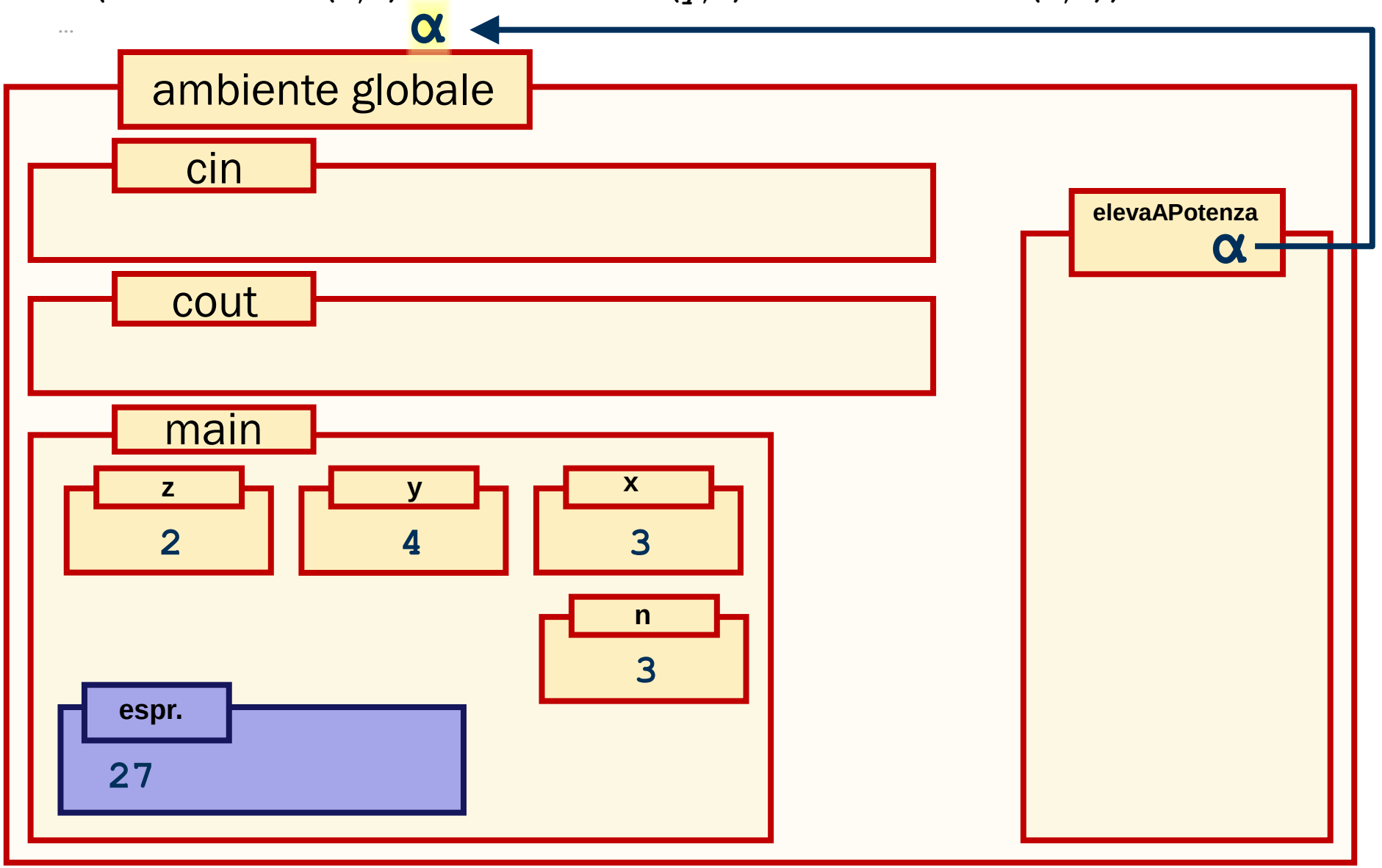
27



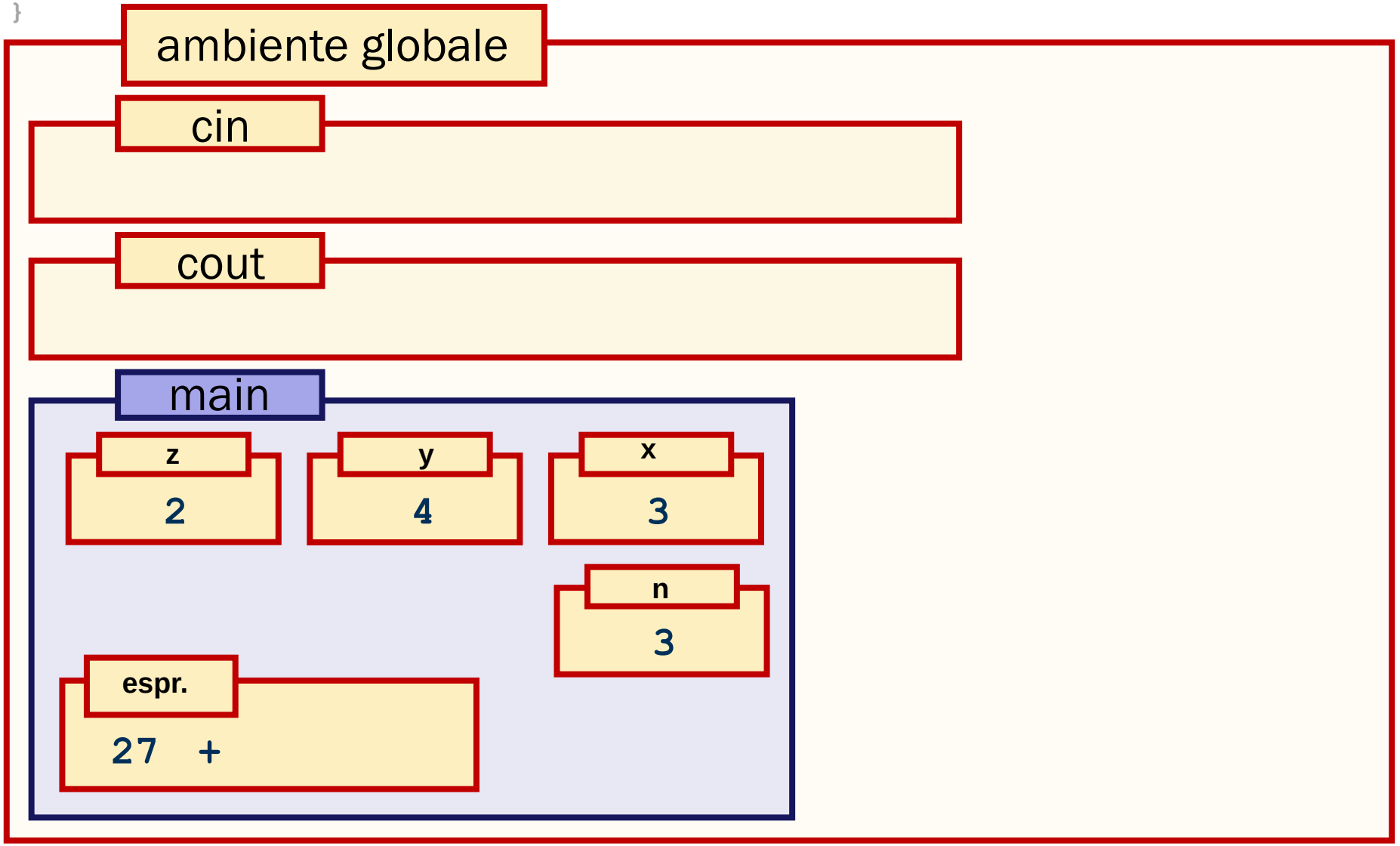
```
potenza = 1;
for (prodMancanti = esponente; prodMancanti > 0; proMancanti--)
  potenza *= base;
return potenza;
}
```



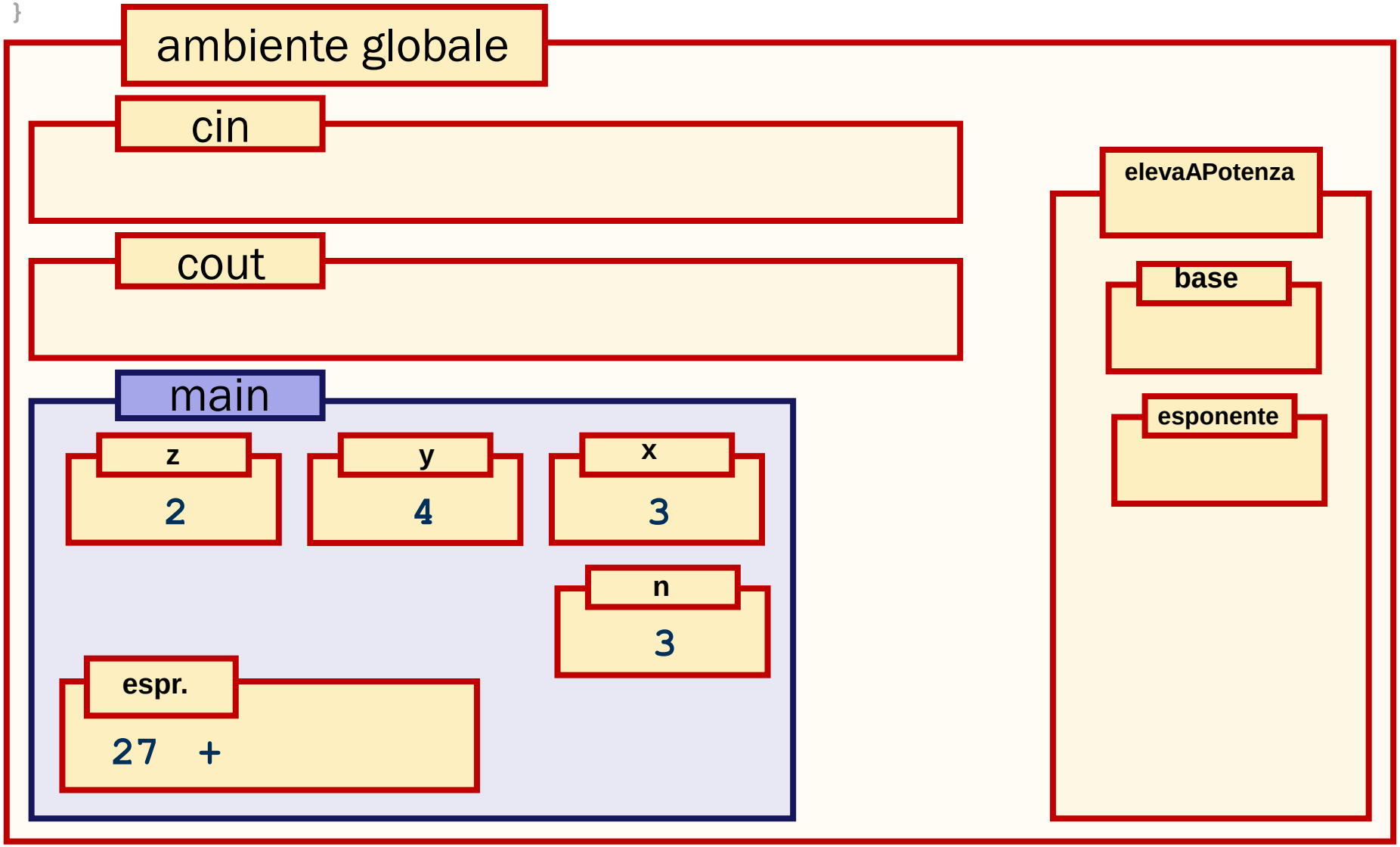
```
"presenta le funzionalità del programma"
"leggi i dati e verifica che rispondano alle specifiche"
if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
 ...
```



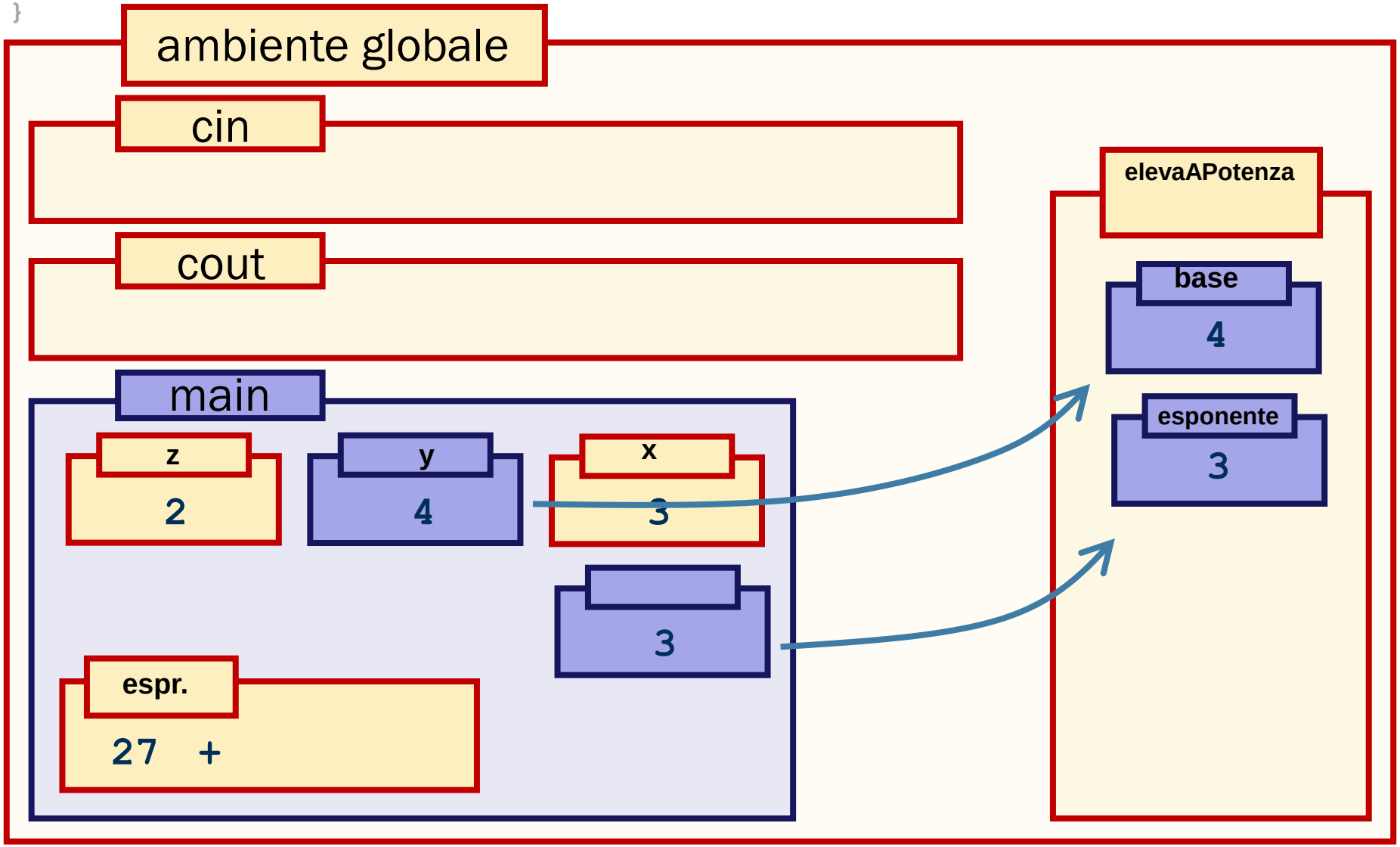
```
    "presenta le funzionalità del programma"
    "leggi i dati e verifica che rispondano alle specifiche"
    if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
      …
}
```



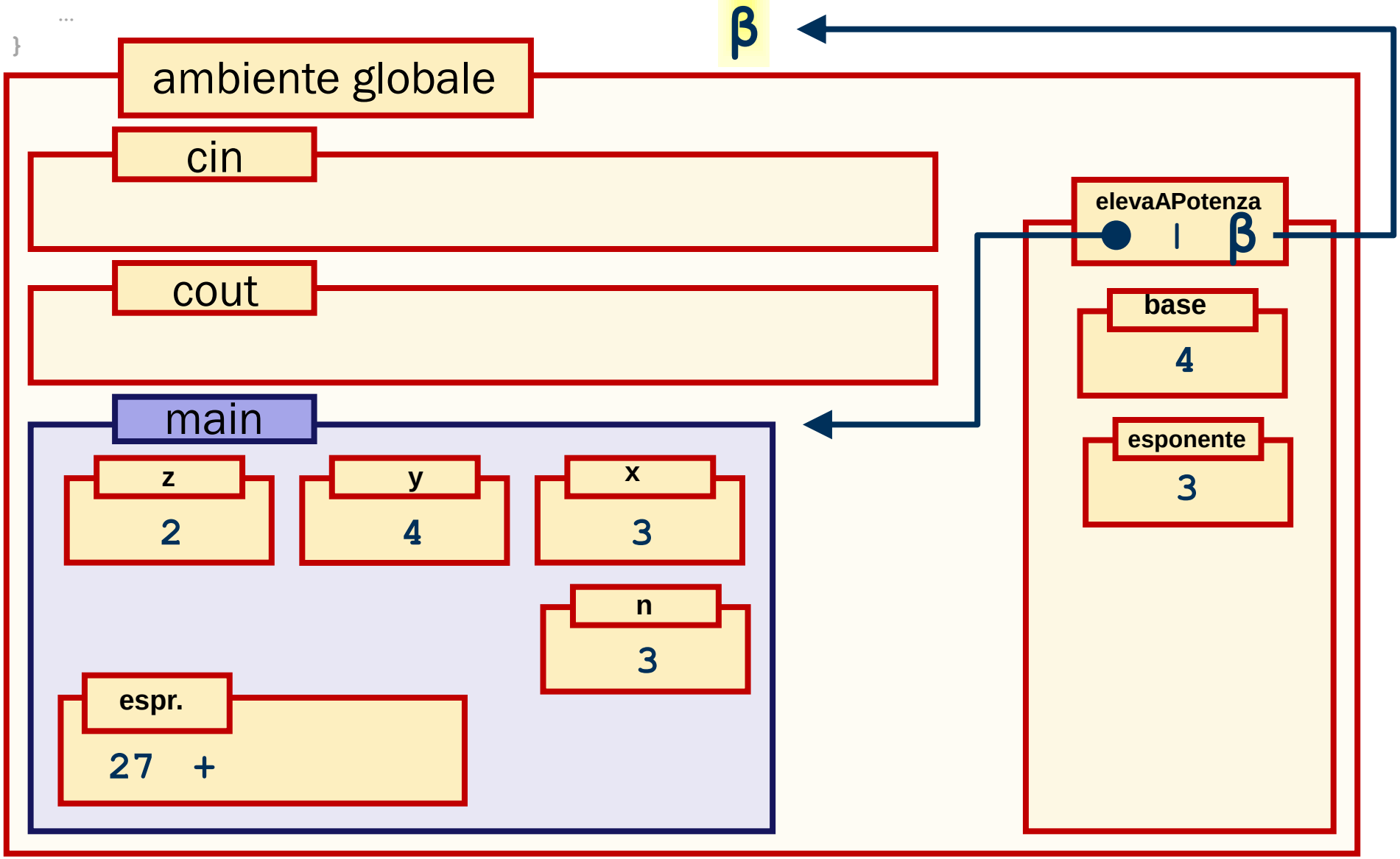
```
    "presenta le funzionalità del programma"
    "leggi i dati e verifica che rispondano alle specifiche"
    if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
      …
}
```



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

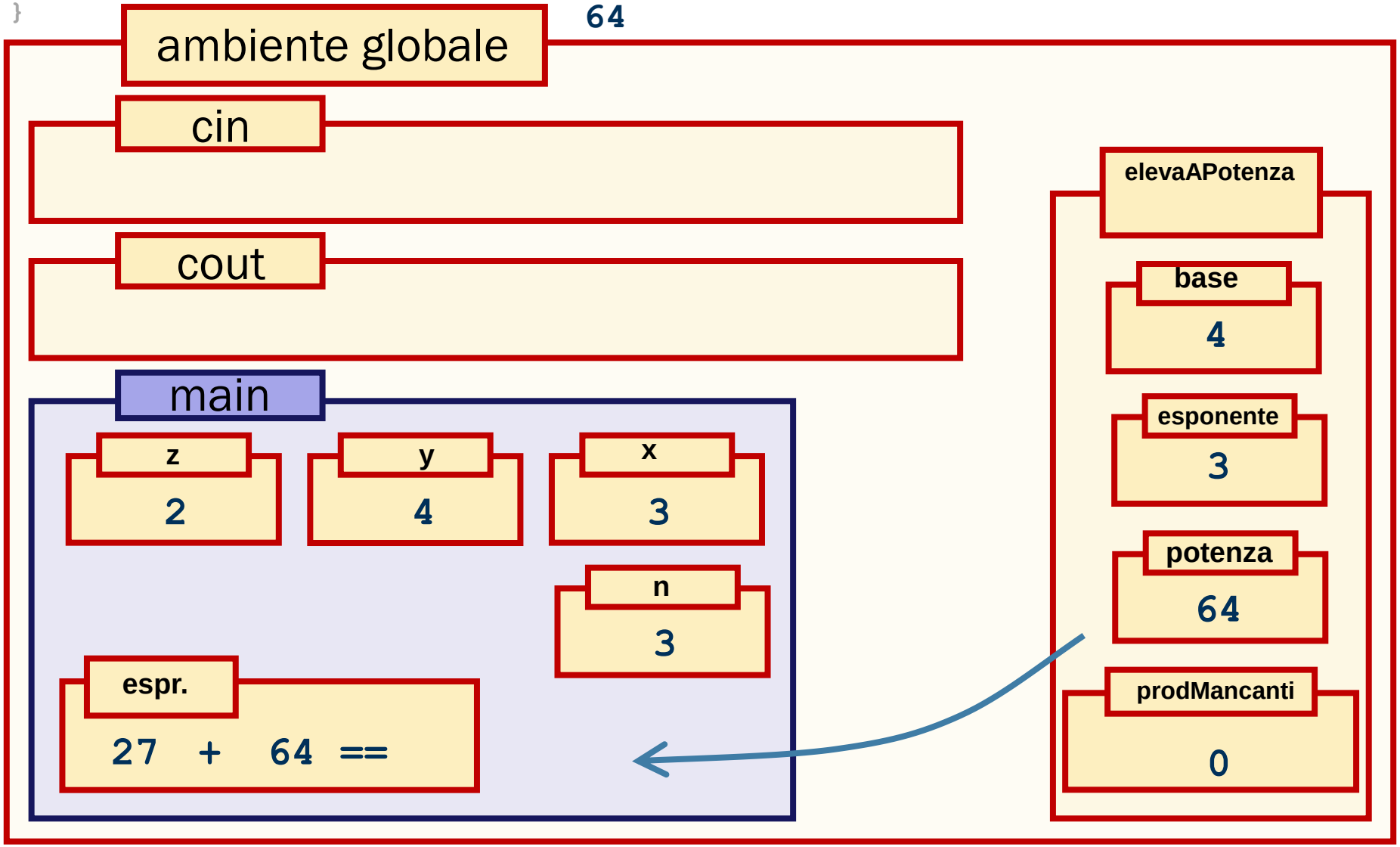


```
    "presenta le funzionalità del programma"
    "leggi i dati e verifica che rispondano alle specifiche"
    if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
      …
}
```

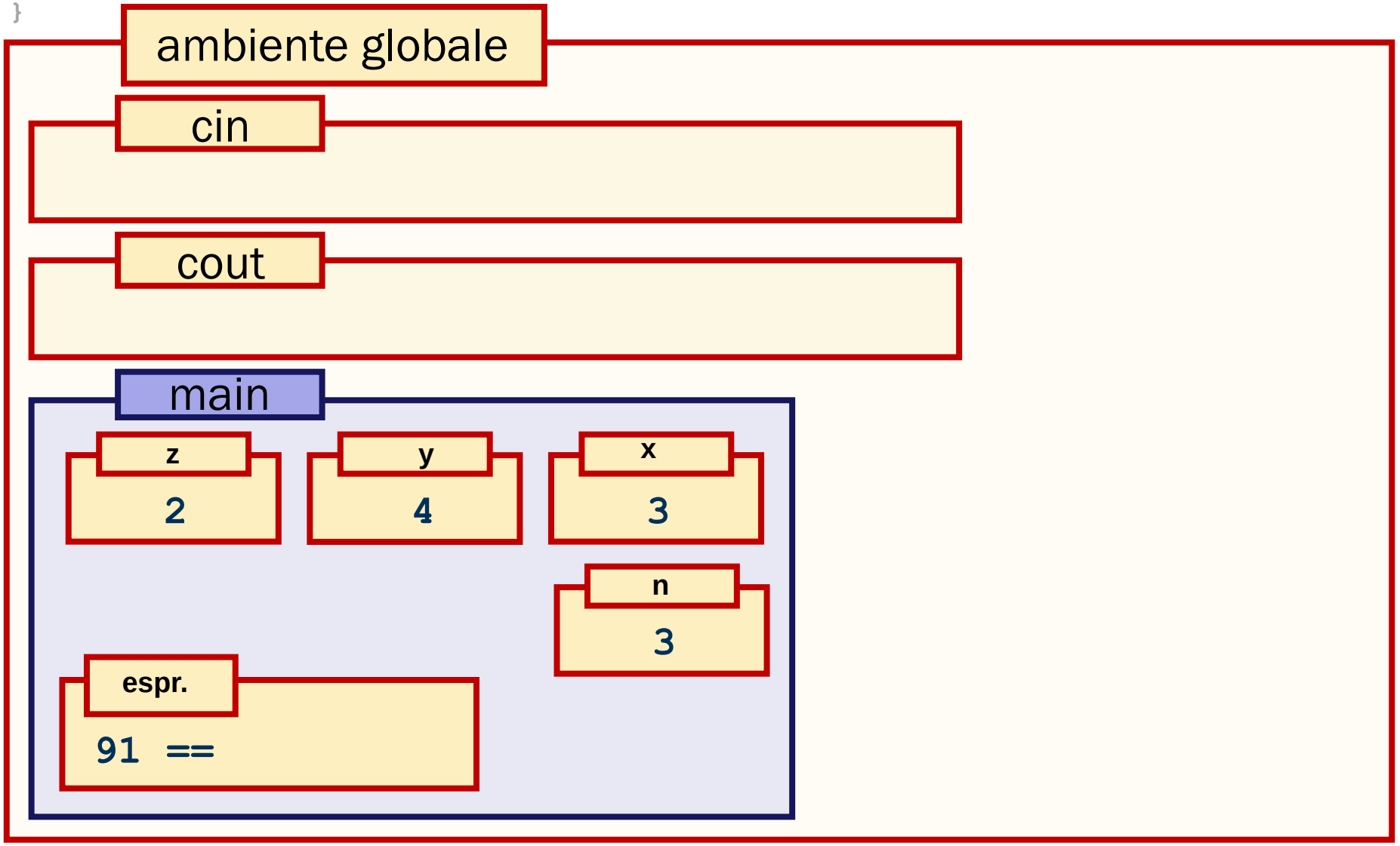


```
    "presenta le funzionalità del programma"
    "leggi i dati e verifica che rispondano alle specifiche"
    if (elevaAPotenza(x,n) + elevaAPotenza(y,n) == elevaAPotenza(z,n))
      ...
}
```

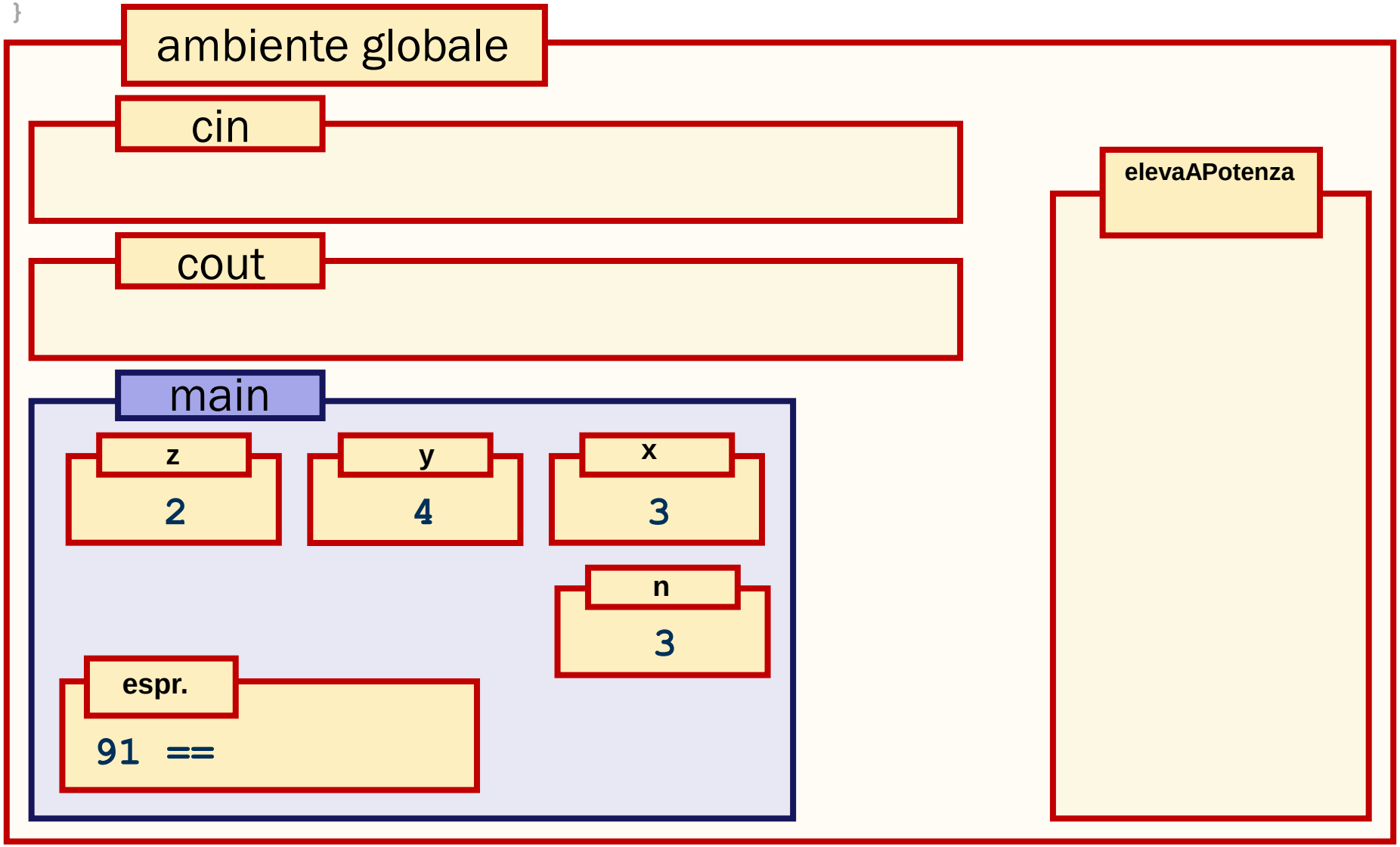
64



```
    "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))
      …
}
```