10 La gestion dynamique de la mémoire

Peter Schlagheck

Université de Liège

10 La gestion dynamique de la mémoire

10.1 Motivation

$\longrightarrow$ Créer une bibliothèque de fonctions et routines en lien avec des matrices quadratiques ...

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$\rightarrow$ trace de la matrice : $\mathrm{Tr}(A) = a_{11} + a_{22} + \ldots + a_{MM}$

$\rightarrow$ déterminant de la matrice : $\det(A) = \ldots$

$\rightarrow$ valeurs propres et vecteurs propres de la matrice

$\rightarrow$ inversion de la matrice : A^{-1}

$\rightarrow$ ...

10.1 Motivation

$\longrightarrow$ Créer une bibliothèque de fonctions et routines en lien avec des matrices quadratiques ...

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

en représentant la matrice par un tableau à deux dimensions ?

$$\texttt{a[i][j]} = a_{i+1,j+1}$$

10.1 Motivation

Le contenu de `matrix.cpp` :

```
const int M = 10;

double trace( double a[][ M ] )
{
  double tr = 0;
  for ( int i = 0; i < M; i++ )
    tr += a[ i ][ i ];
  return tr;
}

double determinant( double a[][ M ] )
{
  ...
}
```

10.1 Motivation

Le contenu de `matrix.cpp` :

```
#include <matrix.h>

double trace( double a[][ M ] )
{
  double tr = 0;
  for ( int i = 0; i < M; i++ )
    tr += a[ i ][ i ];
  return tr;
}

double determinant( double a[][ M ] )
{
  ...
}
```

10.1 Motivation

Le contenu de `matrix.h` :

```
const int M = 10;

double trace( double a[][ M ] );

double determinant( double a[][ M ] );

void inversion( double a[][ M ] );

void valeurs_propres( double a[][ M ], double vpr[] );

...
```

$\longrightarrow$ on est obligé de modifier `matrix.h` et de recompiler `matrix.cpp` si on veut changer la taille M de la matrice

$\longrightarrow$ utiliser des tableaux unidimensionnels de la taille $M \times M$

10.1 Motivation

Le contenu de `matrix.h` :

```
double trace( int M, double a[] );

double determinant( int M, double a[] );

void inversion( int M, double a[] );

void valeurs_propres( int M, double a[], double vpr[] );

...
```

10.1 Motivation

Le contenu de `matrix.cpp` :

```
#include <matrix.h>

double trace( int M, double a[] )
{
  double tr = 0;
  for ( int i = 0; i < M; i++ )
    tr += a[ i * M + i ]
  return tr;
}

double determinant( int M, double a[] )
{
  ...
}
```

10.1 Motivation

Le contenu de `matrix.cpp` :

```
#include <matrix.h>

double trace( int M, double a[] )
{
  double tr = 0;
  for ( int i = 0; i < M; i++ )
    tr += a[ i * M + i ]
  return tr;
}
```

$\longrightarrow$ l'élément a_{ij} de la matrice a est accessible via

```
a[ i * M + j ]
```

10.1 Motivation

Le contenu de `matrix.cpp` :

```
#include <matrix.h>

double trace( int M, double a[] )
{
  double tr = 0;
  for ( int i = 0; i < M; i++ )
    tr += a[ i * M + i ]
  return tr;
}
```

Problème : dans la partie principale du programme
M doit être défini en tant que constante

$\longrightarrow$ impossible de déterminer *M* d'une manière dynamique
$\rightarrow$ par un input de la part de l'utilisateur,
$\rightarrow$ par un calcul effectué dans le programme, ...

$\longrightarrow$ utilisation des tableaux dynamiques pour éviter ça

10.2 Les Pointeurs

Un pointeur est un type qui correspond à une adresse en mémoire

Déclaration:

```
int *ip;
```

$\longrightarrow$ `ip` peut contenir une adresse en mémoire où se trouve une variable entière du type `int`

10.2 Les Pointeurs

Un pointeur est un type qui correspond à une adresse en mémoire

Déclaration:

```
int* ip;
```

$\longrightarrow$ `ip` peut contenir une adresse en mémoire où se trouve une variable entière du type `int`

10.2 Les Pointeurs

Un pointeur est un type qui correspond à une adresse en mémoire

Déclaration:

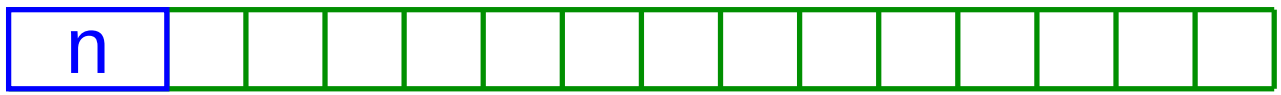
```
int *ip;
double *a;
```

$\longrightarrow$ `a` peut contenir une adresse en mémoire où se trouve une variable flottante du type `double`

$\longrightarrow$ comment (et pourquoi) utiliser ?

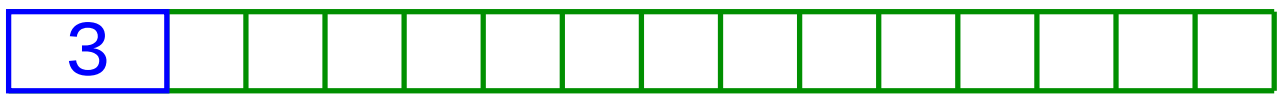
10.2 Les Pointeurs

```
int n = 3;
int *ip;
ip = &n;
int k = *ip;
```



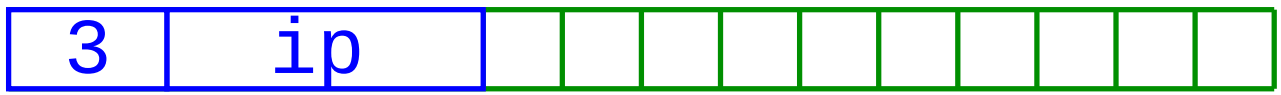
10.2 Les Pointeurs

```
int n = 3;
int *ip;
ip = &n;
int k = *ip;
```



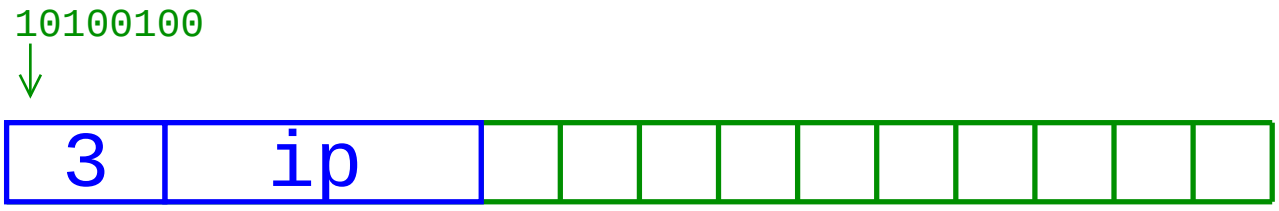
10.2 Les Pointeurs

```
int n = 3;
int *ip;
ip = &n;
int k = *ip;
```



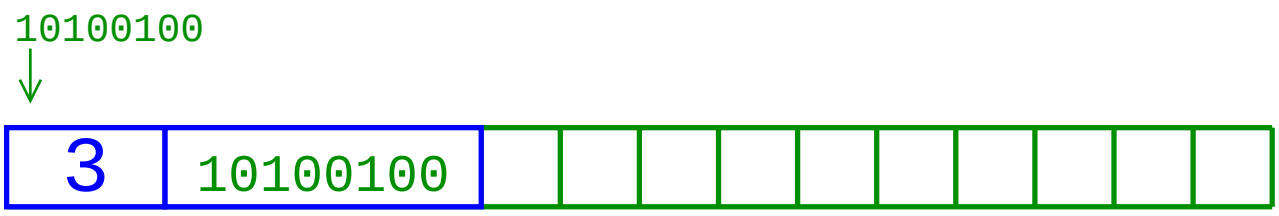
10.2 Les Pointeurs

```
int n = 3;
int *ip;
ip = &n;
int k = *ip;
```



10.2 Les Pointeurs

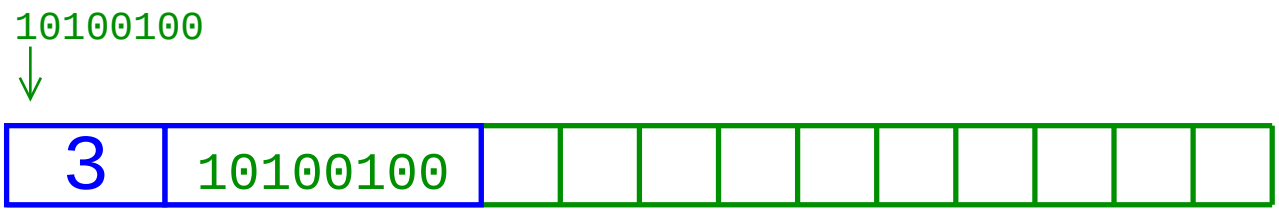
```
int n = 3;
int *ip;
ip = &n;
int k = *ip;
```



$\longrightarrow$ `ip` contient l'adresse de la variable `n`

10.2 Les Pointeurs

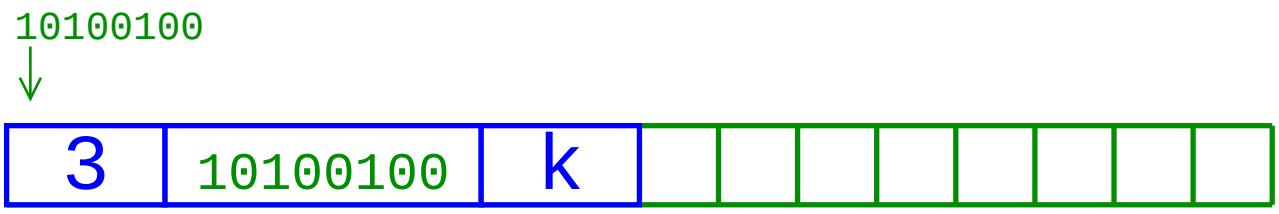
```
int n = 3;
int *ip = &n;
int k = *ip;
```



$\longrightarrow$ `ip` contient l'adresse de la variable `n`

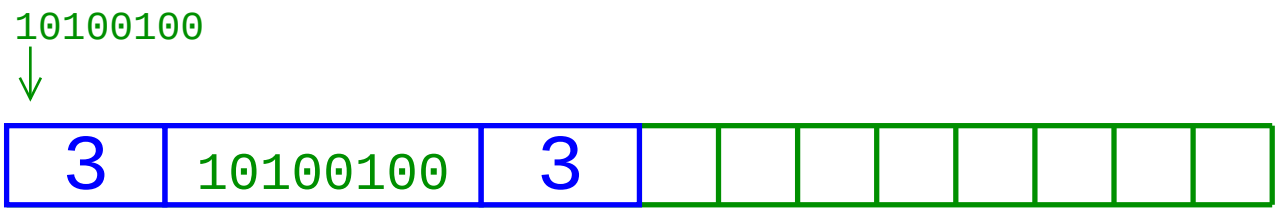
10.2 Les Pointeurs

```
int n = 3;
int *ip = &n;
int k = *ip;
```



10.2 Les Pointeurs

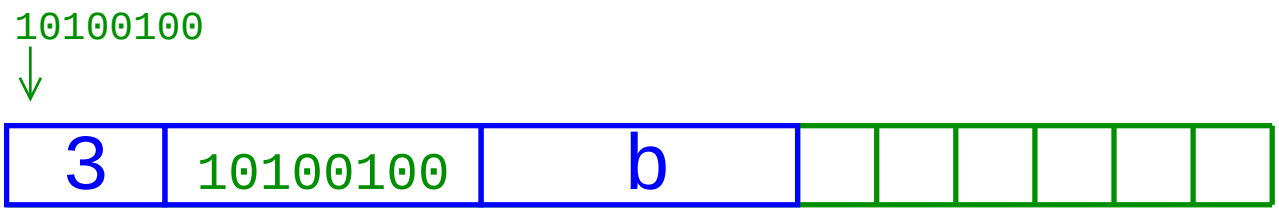
```
int n = 3;
int *ip = &n;
int k = *ip;
```



$\longrightarrow$ `k` contient le contenu de l'adresse `ip`

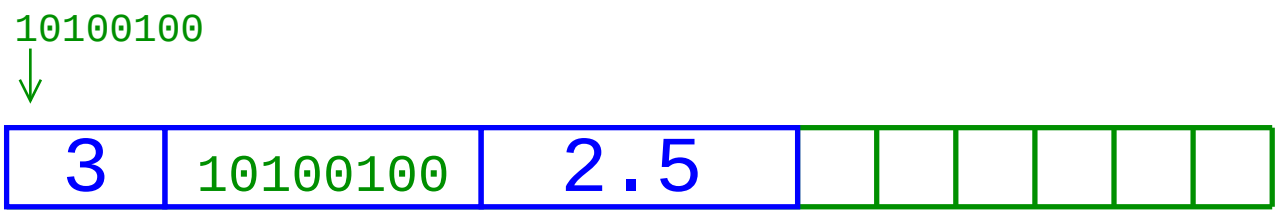
10.2 Les Pointeurs

```
int n = 3;
int *ip = &n;
double b = 2.5;
double *a = &b;
```



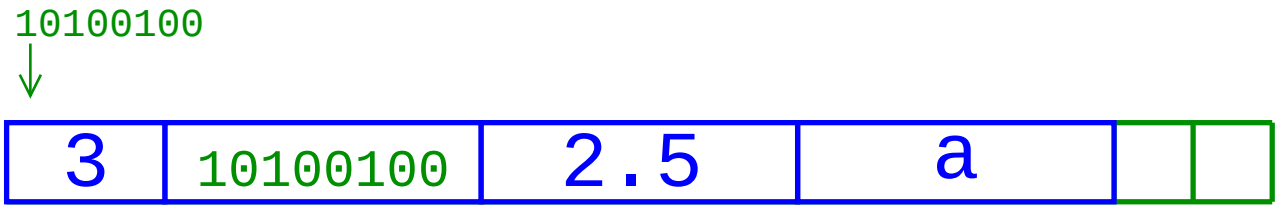
10.2 Les Pointeurs

```
int n = 3;
int *ip = &n;
double b = 2.5;
double *a = &b;
```



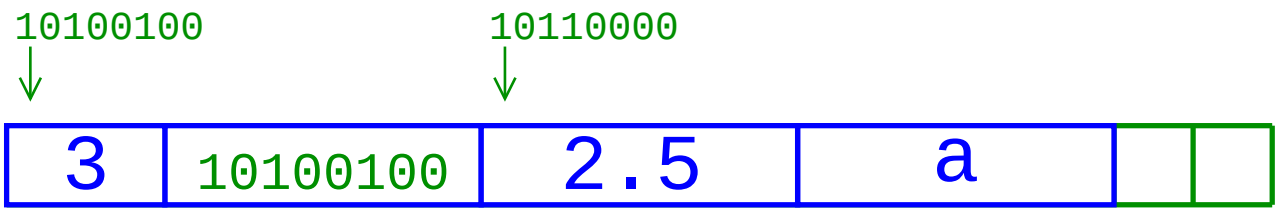
10.2 Les Pointeurs

```
int n = 3;
int *ip = &n;
double b = 2.5;
double *a = &b;
```



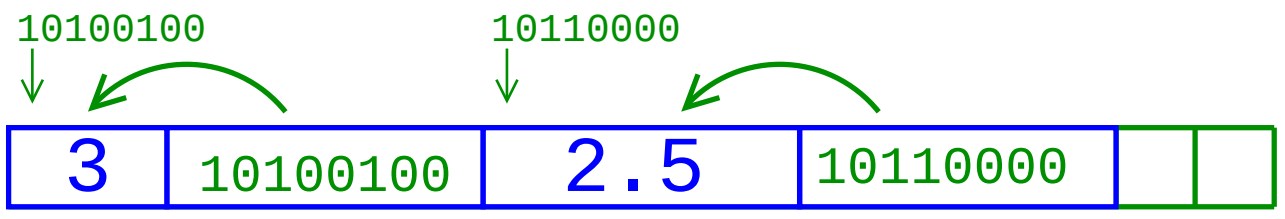
10.2 Les Pointeurs

```
int n = 3;
int *ip = &n;
double b = 2.5;
double *a = &b;
```



10.2 Les Pointeurs

```
int n = 3;
int *ip = &n;
double b = 2.5;
double *a = &b;
```



$\longrightarrow$ un pointeur occupe 8 octets,
indépendamment du type correspondant

(ou 4 octets, dans des vielles machines à 32 bits)

10.2 Les Pointeurs

⟨type⟩ ∗⟨nom⟩;

⟶ déclaration d'un pointeur nommé ⟨nom⟩ dont la valeur représente une adresse d'une variable du type ⟨type⟩

opérateur d'adresse: &⟨variable⟩
⟶ rend l'adresse en mémoire de la variable ⟨variable⟩

opérateur du contenu: ∗⟨adresse⟩
⟶ rend le contenu dans l'adresse ⟨adresse⟩

10.2 Les Pointeurs

⟨type⟩ *⟨nom⟩;

⟶ déclaration d'un pointeur nommé ⟨nom⟩ dont la valeur représente une adresse d'une variable du type ⟨type⟩

opérateur d'adresse: &⟨variable⟩
⟶ rend l'adresse en mémoire de la variable ⟨variable⟩

opérateur du contenu: *⟨adresse⟩
⟶ rend le contenu dans l'adresse ⟨adresse⟩

```
int n = 35;
cout << n << " " << &n << " " << *( &n ) << endl;
```

Exécution:

```
35 0xbfaee0bc 35
```

(notation hexadécimale de l'adresse)

10.2 Les Pointeurs

⟨type⟩ ∗⟨nom⟩;

⟶ déclaration d'un pointeur nommé ⟨nom⟩ dont la valeur représente une adresse d'une variable du type ⟨type⟩

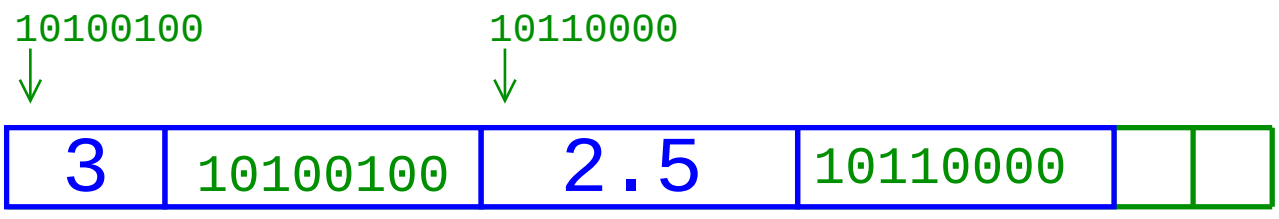
opérateur d'adresse: &⟨variable⟩
⟶ rend l'adresse en mémoire de la variable ⟨variable⟩

opérateur du contenu: ∗⟨adresse⟩
⟶ rend le contenu dans l'adresse ⟨adresse⟩

⟶ ∗⟨nom⟩ peut être utilisé non seulement comme expression explicite, mais aussi comme variable

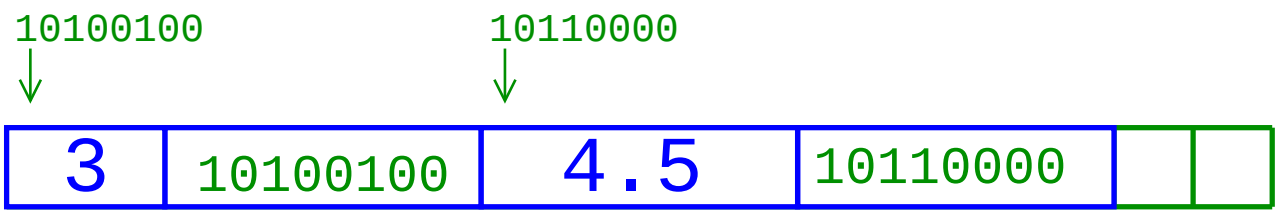
10.2 Les Pointeurs

```
int n = 3;
int *ip = &n;
double b = 2.5;
double *a = &b;
*a += 2;
cout << b << endl;
```



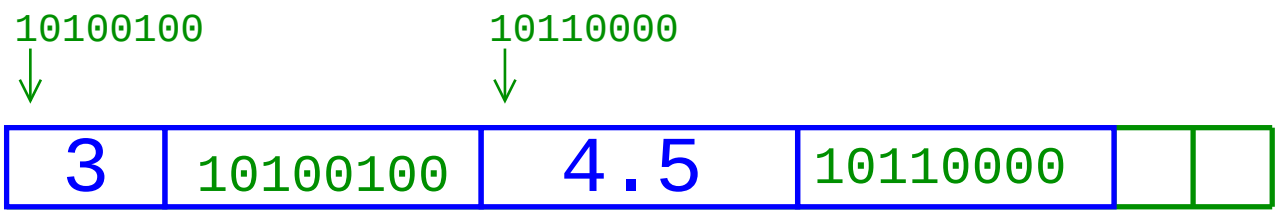
10.2 Les Pointeurs

```
int n = 3;
int *ip = &n;
double b = 2.5;
double *a = &b;
*a += 2;
cout << b << endl;
```



10.2 Les Pointeurs

```
int n = 3;
int *ip = &n;
double b = 2.5;
double *a = &b;
*a += 2;
cout << b << endl;
```



Exécution:

```
4.5
```

$\longrightarrow$ potentiel de manipulation énorme !!!

10.2 Les Pointeurs

```
void manip( double *p )
{
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << "  " << b << "  "
       << c << endl;
}
```

Exécution:

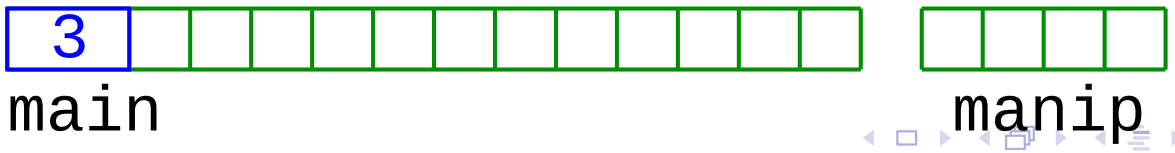
```
3  8  5
```

10.2 Les Pointeurs

```
void manip( double *p )
{
  p += 1;
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << " " << b << " "
       << c << endl;
}
```

Exécution:

```
3 7 6
```

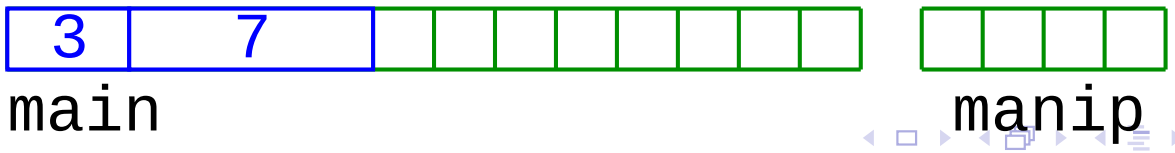


10.2 Les Pointeurs

```
void manip( double *p )
{
  p += 1;
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << " " << b << " "
       << c << endl;
}
```

Exécution:

```
3 7 6
```

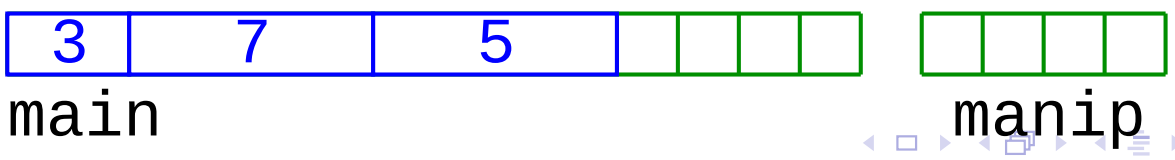


10.2 Les Pointeurs

```
void manip( double *p )
{
  p += 1;
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << " " << b << " "
       << c << endl;
}
```

Exécution:

```
3 7 6
```

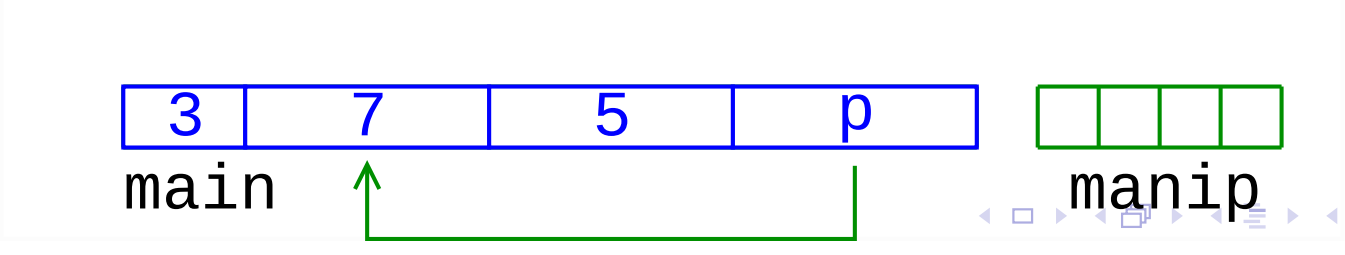


10.2 Les Pointeurs

```
void manip( double *p )
{
  p += 1;
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << " " << b << " "
       << c << endl;
}
```

Exécution:

```
3 7 6
```

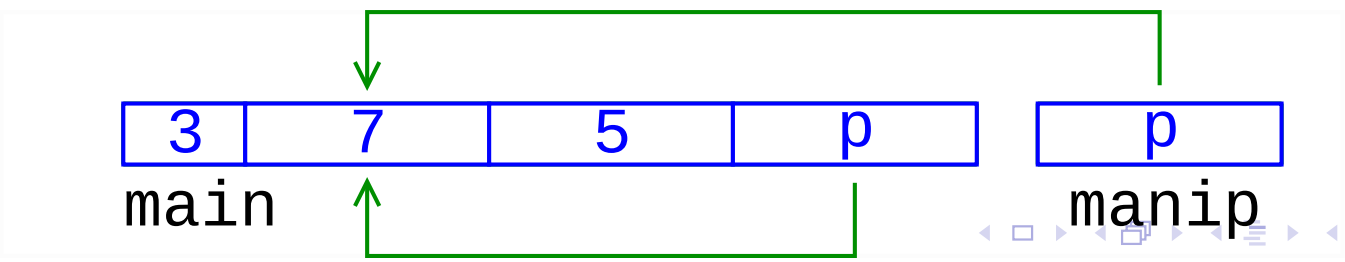


10.2 Les Pointeurs

```
void manip( double *p )
{
  p += 1;
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << " " << b << " "
       << c << endl;
}
```

Exécution:

```
3 7 6
```

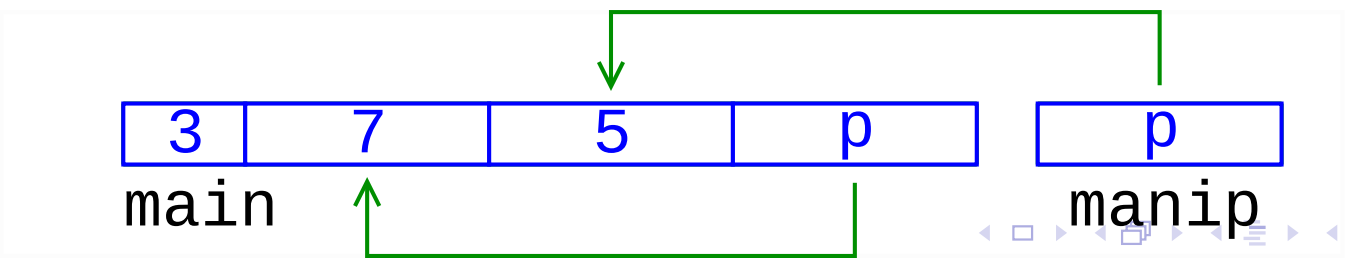


10.2 Les Pointeurs

```
void manip( double *p )
{
  p += 1;
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << " " << b << " "
       << c << endl;
}
```

Exécution:

```
3 7 6
```

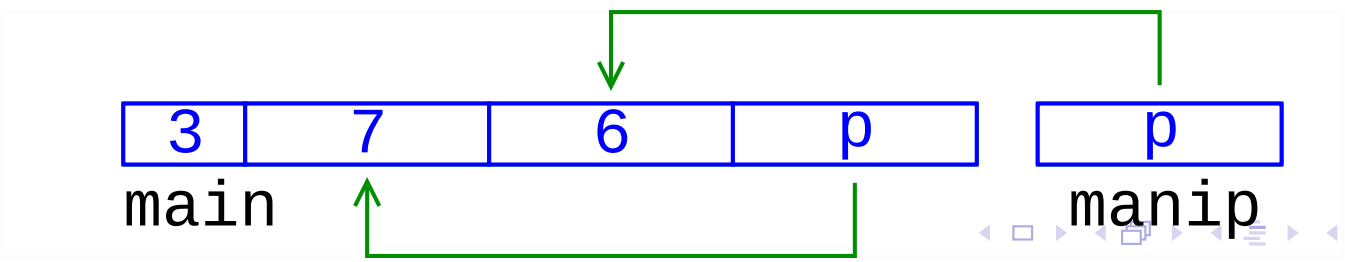


10.2 Les Pointeurs

```
void manip( double *p )
{
  p += 1;
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << " " << b << " "
       << c << endl;
}
```

Exécution:

```
3 7 6
```

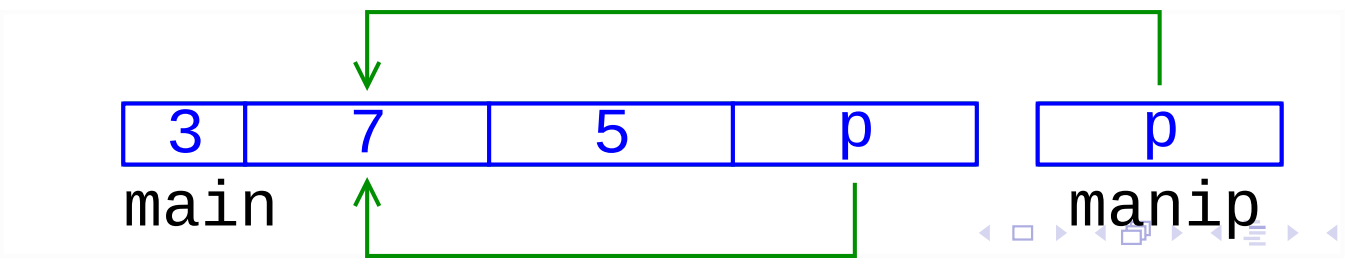


10.2 Les Pointeurs

```
void manip( double *p )
{
  p -= 1;
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << " " << b << " "
       << c << endl;
}
```

Exécution:

```
1072693248 7 5
```

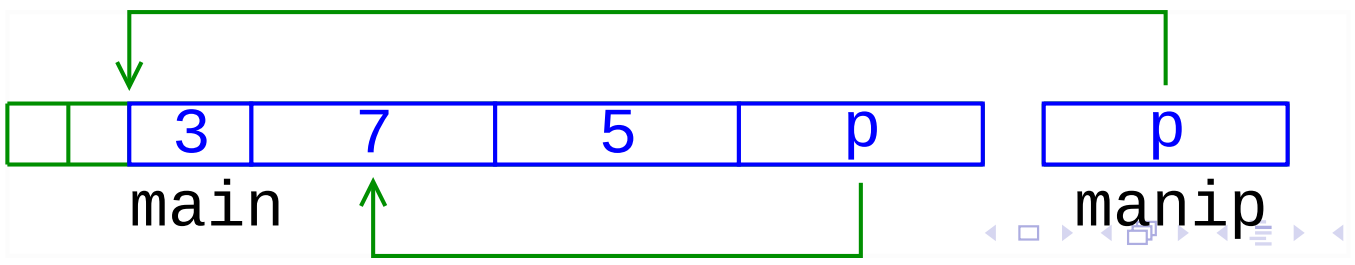


10.2 Les Pointeurs

```
void manip( double *p )
{
  p -= 1;
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << " " << b << " "
       << c << endl;
}
```

Exécution:

```
1072693248 7 5
```

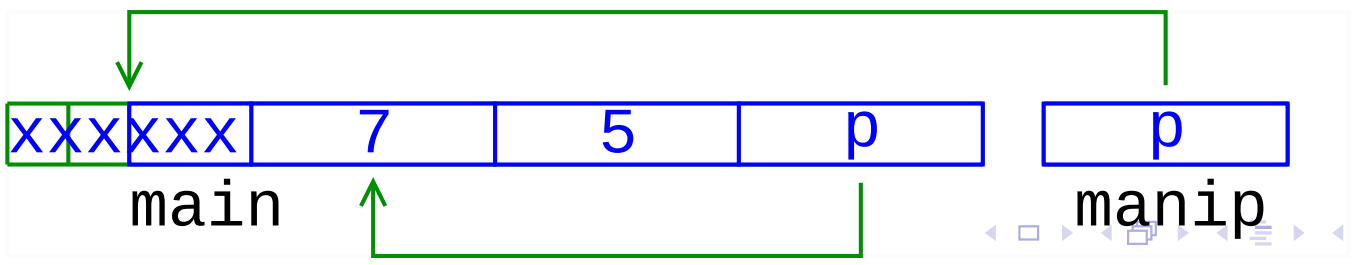


10.2 Les Pointeurs

```
void manip( double *p )
{
  p -= 1;
  *p += 1;
}

int main()
{
  int a = 3;
  double b = 7;
  double c = 5;
  double *p = &b;
  manip( p );
  cout << a << " " << b << " "
       << c << endl;
}
```

Exécution:

```
1072693248 7 5
```



10.2 Les Pointeurs

Opérations élémentaires pour des pointeurs:

⟨type⟩ ∗⟨nom⟩;
⟨nom⟩ = ⟨adresse⟩;

⟶ copie de l'adresse ⟨adresse⟩ dans le pointeur ⟨nom⟩

⟨adresse⟩ doit contenir une variable du type ⟨type⟩

10.2 Les Pointeurs

Opérations élémentaires pour des pointeurs:

⟨type⟩ ∗⟨nom⟩;
⟨nom⟩ = ⟨adresse⟩;
⟨nom⟩ += ⟨entier⟩;

⟶ augmentation de la valeur (l'adresse) dans ⟨nom⟩
par ⟨entier⟩×⟨taille⟩ octets
où ⟨taille⟩ est la taille d'une variable du type ⟨type⟩

on peut également écrire ⟨nom⟩ = ⟨nom⟩ + ⟨entier⟩;

⟶ on se déplace par ⟨entier⟩ variables en mémoire
(en supposant qu'elles soient toutes du type ⟨type⟩ . . .)

10.2 Les Pointeurs

Opérations élémentaires pour des pointeurs:

⟨type⟩ ∗⟨nom⟩;
⟨nom⟩ = ⟨adresse⟩;
⟨nom⟩ −= ⟨entier⟩;

⟶ diminuation de la valeur (l'adresse) dans ⟨nom⟩
par ⟨entier⟩ × ⟨taille⟩ octets
où ⟨taille⟩ est la taille d'une variable du type ⟨type⟩

on peut également écrire ⟨nom⟩ = ⟨nom⟩ − ⟨entier⟩;

10.2 Les Pointeurs

Opérations élémentaires pour des pointeurs:

⟨type⟩ *⟨nom⟩;
⟨nom⟩ = ⟨adresse⟩;
⟨nom⟩ += ⟨entier⟩;
⟨nom⟩++;

⟶ incrément de l'adresse dans ⟨nom⟩

on peut également écrire ⟨nom⟩ += 1;

10.2 Les Pointeurs

Opérations élémentaires pour des pointeurs:

⟨type⟩ *⟨nom⟩;
⟨nom⟩ = ⟨adresse⟩;
⟨nom⟩ += ⟨entier⟩;
⟨nom⟩--;

⟶ décrément de l'adresse dans ⟨nom⟩

on peut également écrire ⟨nom⟩ -= 1;

10.2 Les Pointeurs

Opérations élémentaires pour des pointeurs:

⟨type⟩ *⟨nom⟩;
⟨nom⟩ = ⟨adresse⟩;
⟨nom⟩ += ⟨entier⟩;
⟨nom⟩++;

⟨nom⟩+⟨entier⟩ rend l'adresse de la variable (du type ⟨type⟩)
⟨entier⟩ places à côté

*(⟨nom⟩+⟨entier⟩) rend le contenu de cette variable

⟨nom⟩[⟨entier⟩] rend aussi le contenu de cette variable

Les pointeurs et les tableaux

Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
cout << a << " " << *a << " " << a[ 0 ] << endl;
```

Exécution:

```
0xbfe71c34 8 8
```

Les pointeurs et les tableaux

Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a;
cout << b[ 1 ] << endl;
```

Exécution:

```
3
```

Les pointeurs et les tableaux

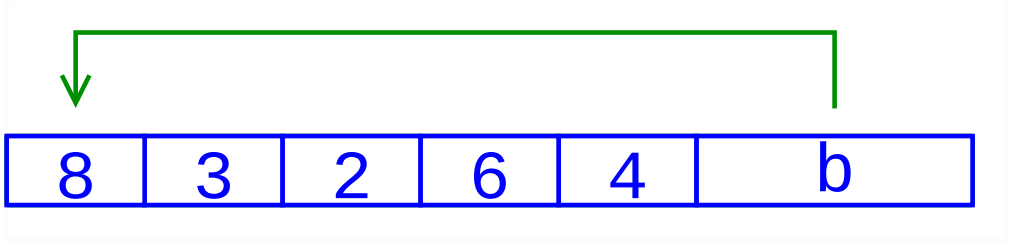
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a;
cout << b[ 1 ] << endl;
```

Exécution:

```
3
```



Les pointeurs et les tableaux

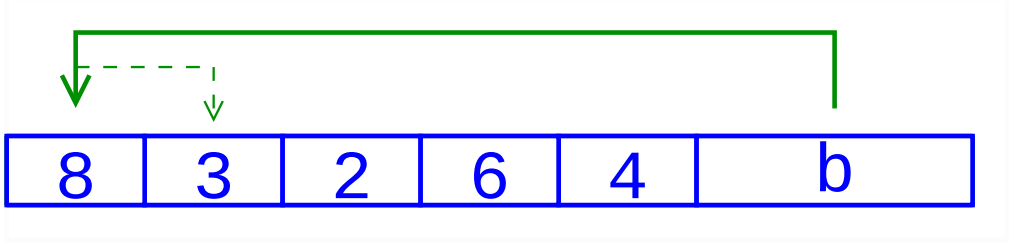
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a;
cout << b[ 1 ] << endl;
```

Exécution:

```
3
```



Les pointeurs et les tableaux

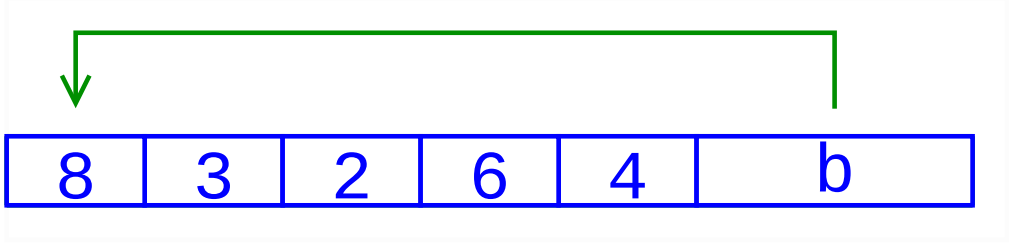
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a;
b++;
cout << b[ 1 ] << endl;
```

Exécution:

```
2
```



Les pointeurs et les tableaux

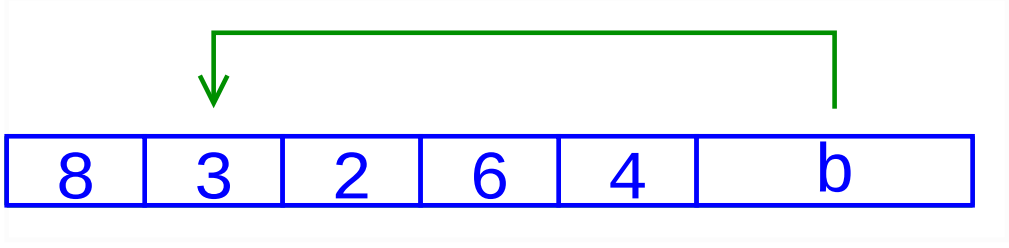
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a;
b++;
cout << b[ 1 ] << endl;
```

Exécution:

```
2
```



Les pointeurs et les tableaux

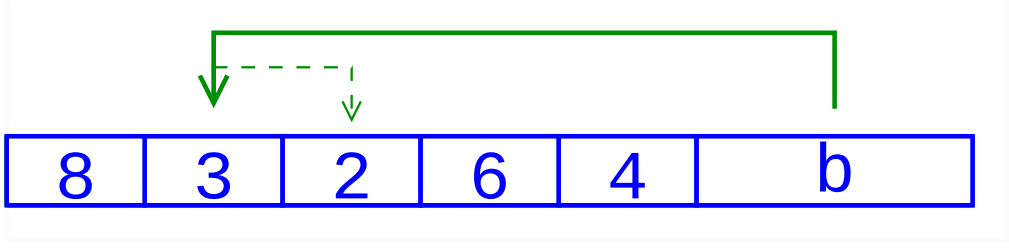
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a;
b++;
cout << b[ 1 ] << endl;
```

Exécution:

```
2
```



Les pointeurs et les tableaux

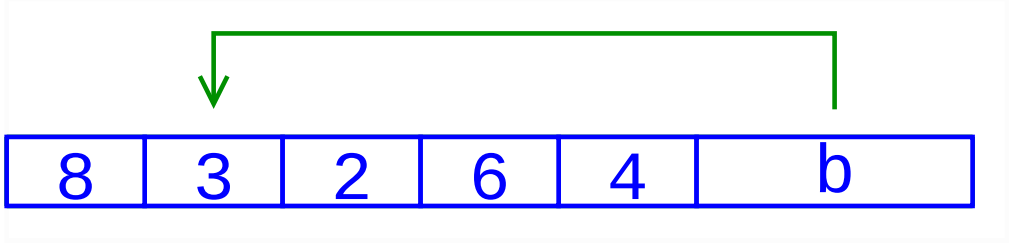
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a + 1;
b++;
cout << b[ 1 ] << endl;
```

Exécution:

```
6
```



Les pointeurs et les tableaux

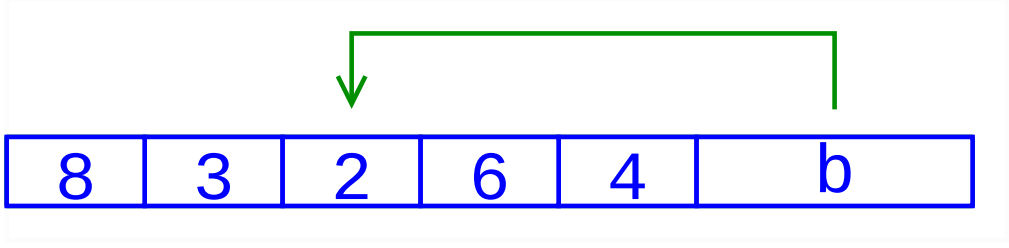
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a + 1;
b++;
cout << b[ 1 ] << endl;
```

Exécution:

```
6
```



Les pointeurs et les tableaux

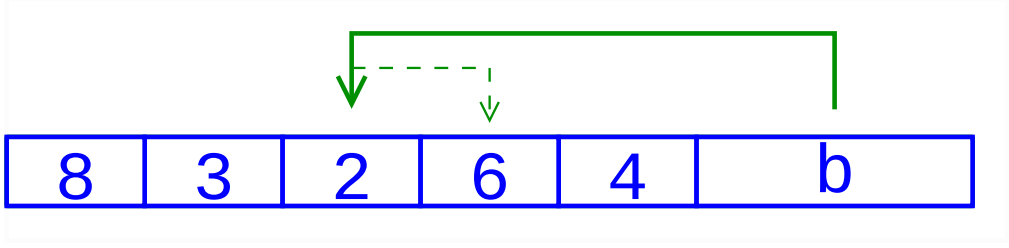
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a + 1;
b++;
cout << b[ 1 ] << endl;
```

Exécution:

```
6
```



Les pointeurs et les tableaux

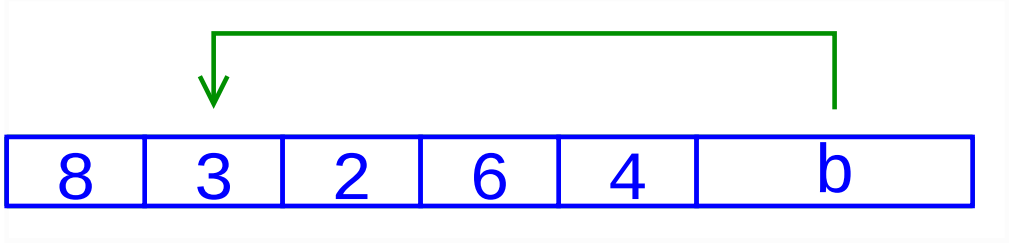
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a + 1;
*b += 2;
cout << a[ 1 ] << endl;
```

Exécution:

```
5
```



Les pointeurs et les tableaux

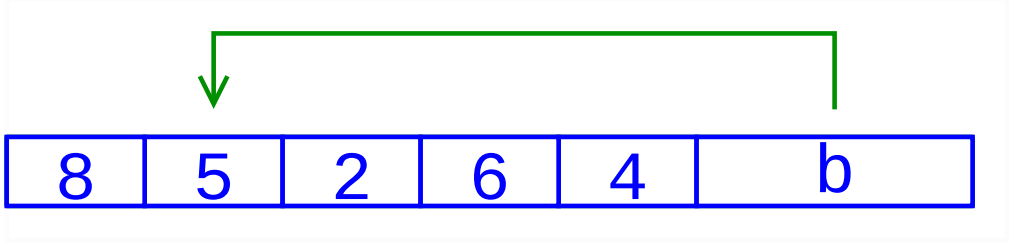
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

Cette adresse ne peut pas être changée. Elle reste constante.

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a + 1;
*b += 2;
cout << a[ 1 ] << endl;
```

Exécution:

```
5
```



Les pointeurs et les tableaux

Normalement, les tableaux sont transférés à des fonctions par des pointeurs

```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}
```

Les pointeurs et les tableaux

Normalement, les tableaux sont transférés à des fonctions par des pointeurs

```
void copie( int N, double *a, double *b )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}
```

Les pointeurs et les tableaux

Normalement, les tableaux sont transférés à des fonctions par des pointeurs

```
void copie( char *a, char *b )
{
  int i;
  for ( i = 0; a[ i ]; i++ )
    b[ i ] = a[ i ];
  b[ i ] = 0;
}
```

Les pointeurs et les tableaux

Normalement, les tableaux sont transférés à des fonctions par des pointeurs

```
void copie( char *a, char *b )
{
  while ( *a )
  {
    *b = *a;
    b++;
    a++;
  }
  *b = 0;
}
```

Les pointeurs et les tableaux

Normalement, les tableaux sont transférés à des fonctions par des pointeurs

```
void copie( char *a, char *b )
{
  while ( *b++ = *a++ );
}
```

$\longrightarrow$ correct et efficace . . . style 'expert'

Les pointeurs constants

`const` ⟨type⟩ `*`⟨nom⟩`;`

$\longrightarrow$ définition d'un pointeur ⟨nom⟩
qui ne peut pas manipuler le contenu de son adresse

Les pointeurs constants

`const` ⟨type⟩ `*`⟨nom⟩`;`

⟶ définition d'un pointeur ⟨nom⟩
qui ne peut pas manipuler le contenu de son adresse

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
const int *b = a + 1;
*b += 2;
cout << a[ 1 ] << endl;
```

⟶ `erreur: assignment of read-only location '* b'`

Les pointeurs constants

`const` ⟨type⟩ `*`⟨nom⟩`;`

$\longrightarrow$ définition d'un pointeur ⟨nom⟩
qui ne peut pas manipuler le contenu de son adresse

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
const int *b = a + 1;
b++;
cout << b[ 1 ] << endl;
```

Exécution:

```
6
```

Les pointeurs constants

`const` ⟨type⟩ `*`⟨nom⟩`;`

⟶ définition d'un pointeur ⟨nom⟩
qui ne peut pas manipuler le contenu de son adresse

```
int a[ 5 ] = { 8, 3, 2, 6, 4 };
const int *b = a + 1;
b++;
*b += 2;
cout << b[ 1 ] << endl;
```

⟶ `erreur: assignment of read-only location '* b'`

Les pointeurs constants

`const` ⟨type⟩ `*`⟨nom⟩`;`

$\longrightarrow$ définition d'un pointeur ⟨nom⟩
qui ne peut pas manipuler le contenu de son adresse

```
const int a[ 5 ] = { 8, 3, 2, 6, 4 };
const int *b = a + 1;
b++;
cout << b[ 1 ] << endl;
```

Exécution:

```
6
```

Les pointeurs constants

`const` ⟨type⟩ `*`⟨nom⟩`;`

⟶ définition d'un pointeur ⟨nom⟩
qui ne peut pas manipuler le contenu de son adresse

```
const int a[ 5 ] = { 8, 3, 2, 6, 4 };
int *b = a + 1;
b++;
cout << b[ 1 ] << endl;
```

⟶ erreur:

```
invalid conversion from 'const int*' to 'int*'
```

Les pointeurs constants

`const` ⟨type⟩ `*`⟨nom⟩`;`

$\longrightarrow$ définition d'un pointeur ⟨nom⟩
qui ne peut pas manipuler le contenu de son adresse

Des pointeurs "ordinaires" et des pointeurs "constants" représentent des types différents .

Une conversion implicite est possible
$\rightarrow$ d'un pointeur ordinaire à un pointeur constant

```
int *a;
const int *b = a;
```

Les pointeurs constants

`const` ⟨type⟩ `*`⟨nom⟩`;`

$\longrightarrow$ définition d'un pointeur ⟨nom⟩
qui ne peut pas manipuler le contenu de son adresse

Des pointeurs "ordinaires" et des pointeurs "constants" représentent des types différents .

Une conversion implicite est possible
$\rightarrow$ d'un pointeur ordinaire à un pointeur constant
$\rightarrow$ . . . mais pas d'un pointeur constant à un pointeur ordinaire.

```
const int *a;
int *b = a;
```

$\longrightarrow$ erreur:

```
invalid conversion from 'const int*' to 'int*'
```

Les pointeurs constants

`const` ⟨type⟩ `*`⟨nom⟩`;`

$\longrightarrow$ définition d'un pointeur ⟨nom⟩
qui ne peut pas manipuler le contenu de son adresse

`const` ⟨type⟩ ⟨tab⟩`[` ⟨taille⟩ `] = {` ⟨el_1⟩`, ..., ` ⟨el_N⟩ `};`

$\longrightarrow$ définition d'un tableau ⟨tab⟩
dont les éléments ne peuvent pas être modifiés

Les pointeurs constants

`const` ⟨type⟩ `*`⟨nom⟩`;`

⟶ définition d'un pointeur ⟨nom⟩
qui ne peut pas manipuler le contenu de son adresse

`const` ⟨type⟩ ⟨tab⟩`[] = {` ⟨el_1⟩`, ..., ` ⟨el_N⟩ `};`

⟶ définition d'un tableau ⟨tab⟩
dont les éléments ne peuvent pas être modifiés

⟨tab⟩ est parfaitement compatible au pointeur constant ⟨nom⟩

Les pointeurs constants

Normalement, des fonctions qui utilisent des tableaux sans les manipuler sont définies tel que les tableaux apparaissent dans la liste d'arguments comme des pointeurs constants.

```
void copie( int N, const double *a, double *b )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}
```

`new` et `delete`

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

$\longrightarrow$ instructions `new` et `delete`

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

⟨type⟩ *⟨nom⟩;

⟨nom⟩ = new ⟨type⟩;

⟶ réservation des blocs sur l'espace "libre"
(c-à-d ne pas occupé par les variables du programme)
pour une nouvelle variable du type ⟨type⟩

et copie de son adresse dans ⟨nom⟩

Cette nouvelle variable peut donc être accédée par *⟨nom⟩.

`new` et `delete`

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

⟨type⟩ `*`⟨nom⟩`;`

⟨nom⟩ `= new` ⟨type⟩`;`

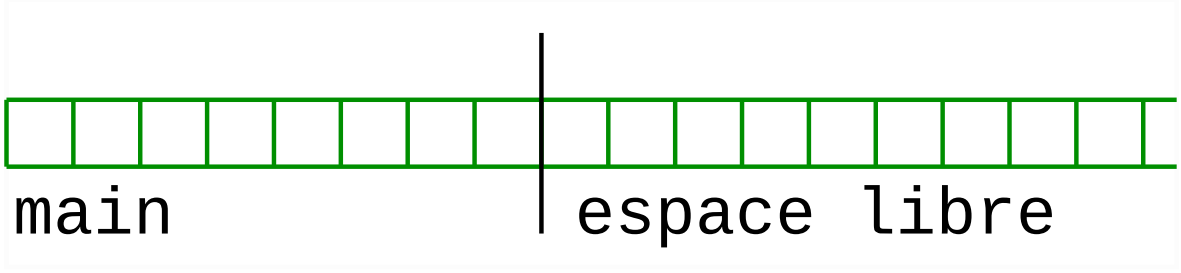
`delete` ⟨nom⟩`;`

$\longrightarrow$ libération de cet espace bloqué dans la mémoire

afin qu'il puisse être réutilisé par d'autres instructions `new`

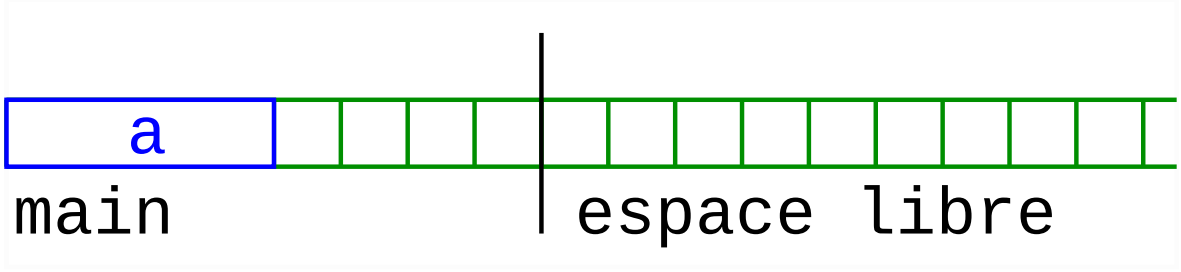
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
}
```



Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

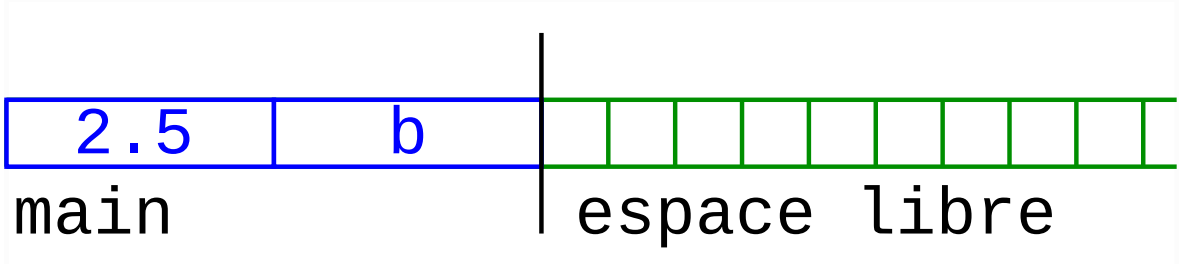
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
}
```



new et delete

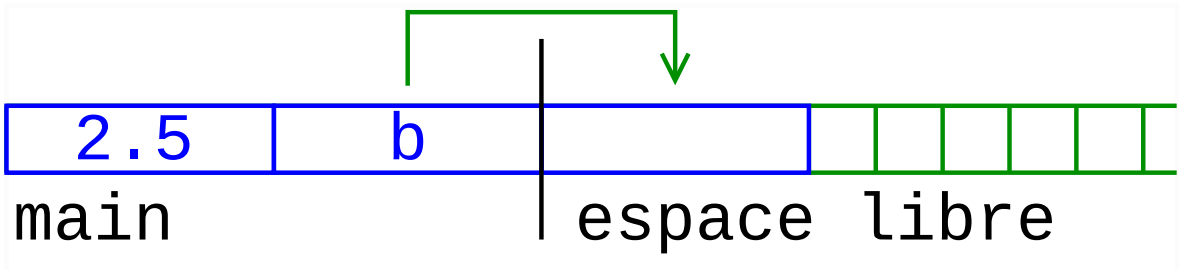
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
}
```



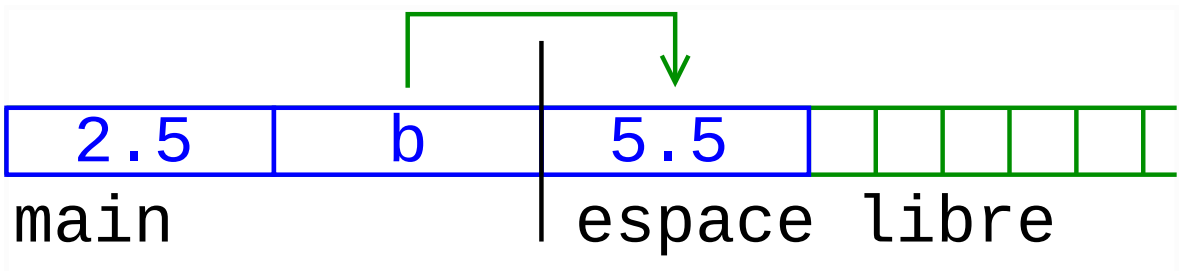
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
}
```



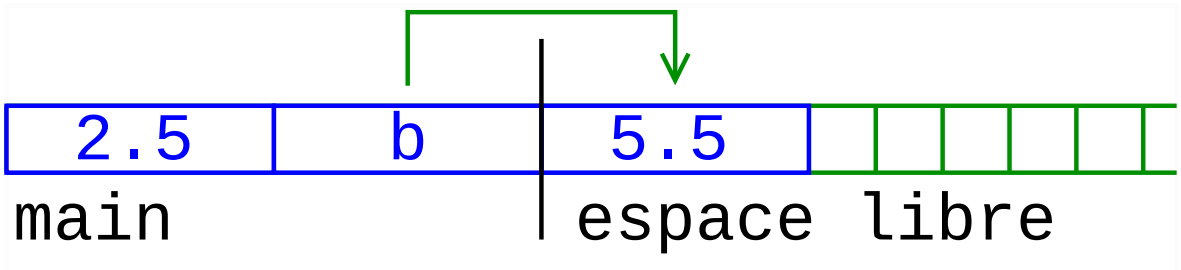
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
}
```



Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
}
```



Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

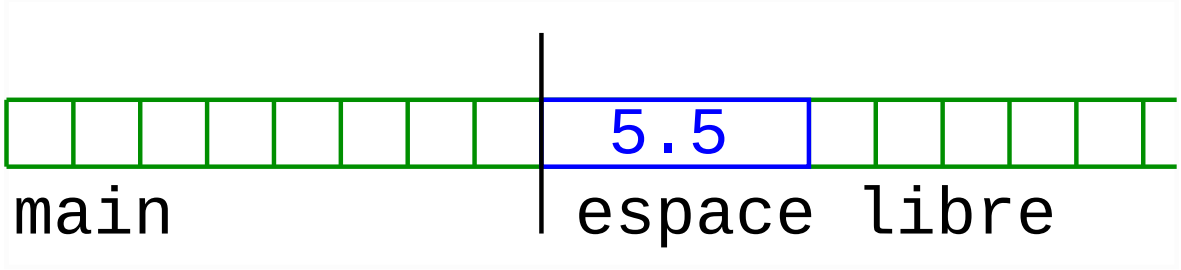
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
}
```

Exécution:

```
5.5
```

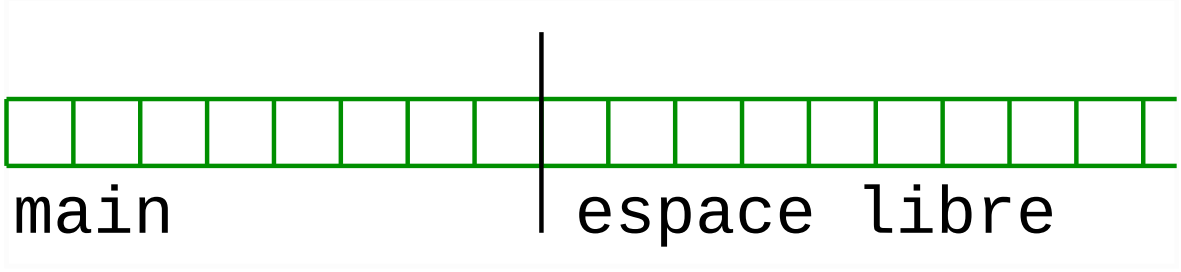
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
}
```



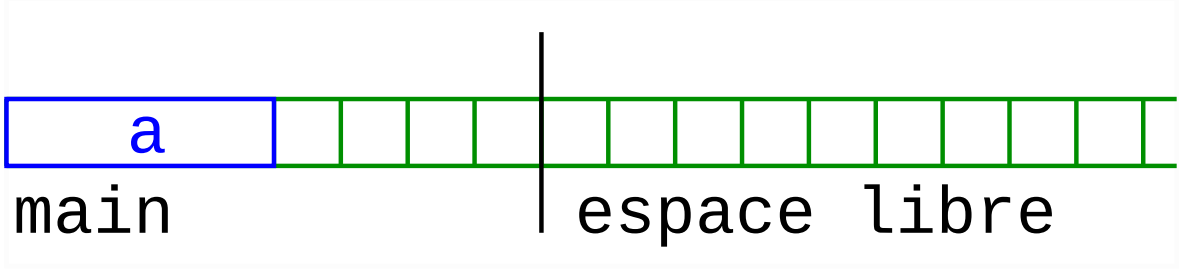
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

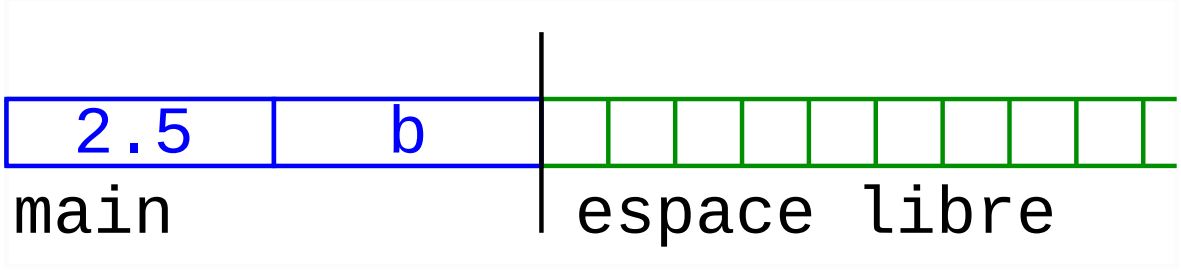
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



new et delete

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

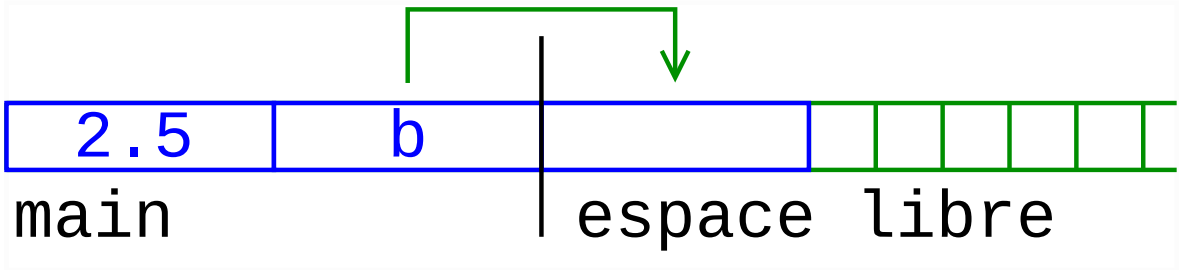
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



new et delete

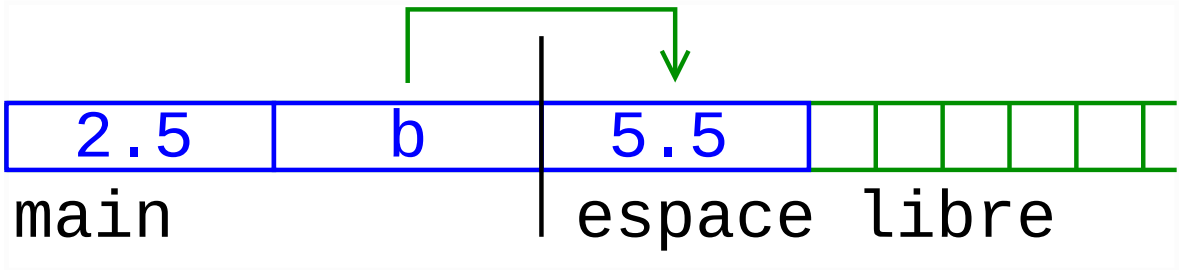
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

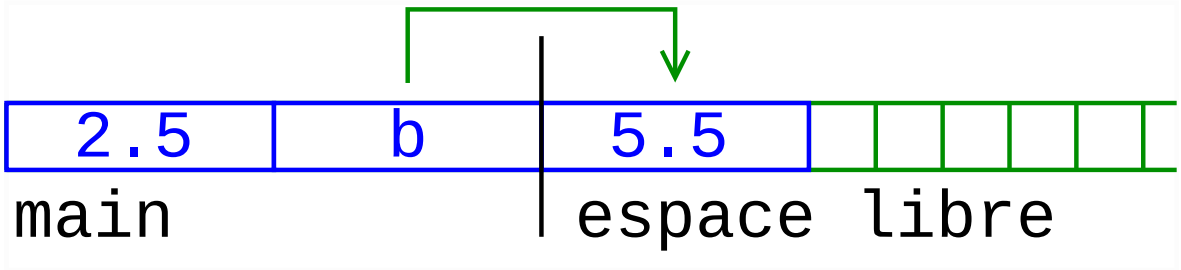
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



new et delete

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

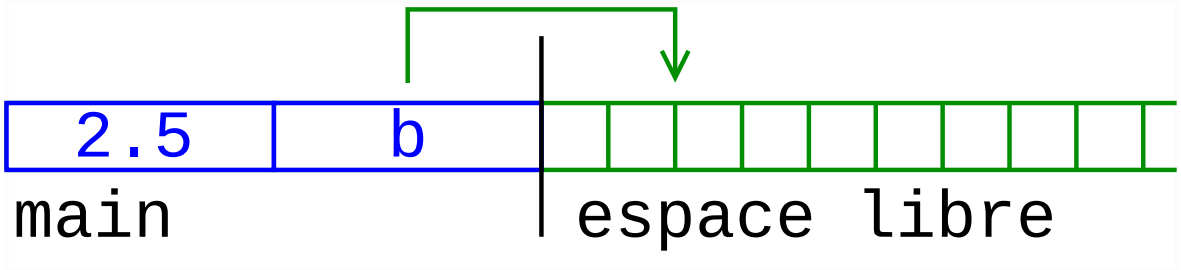
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



new et delete

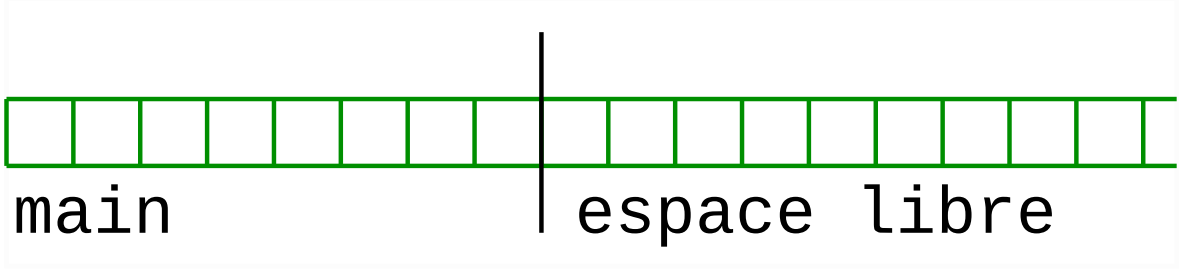
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



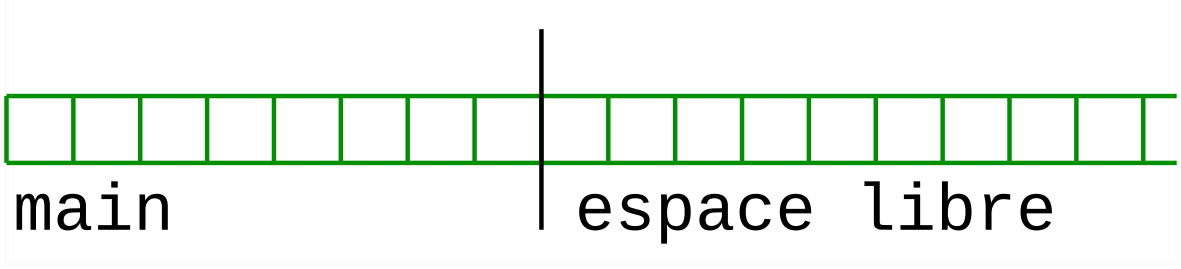
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



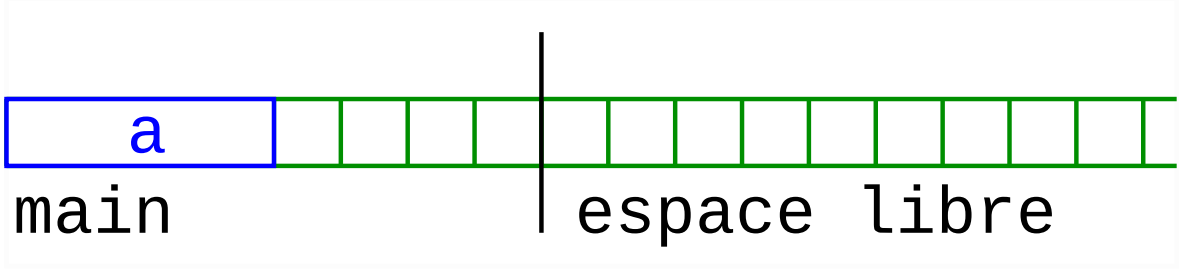
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

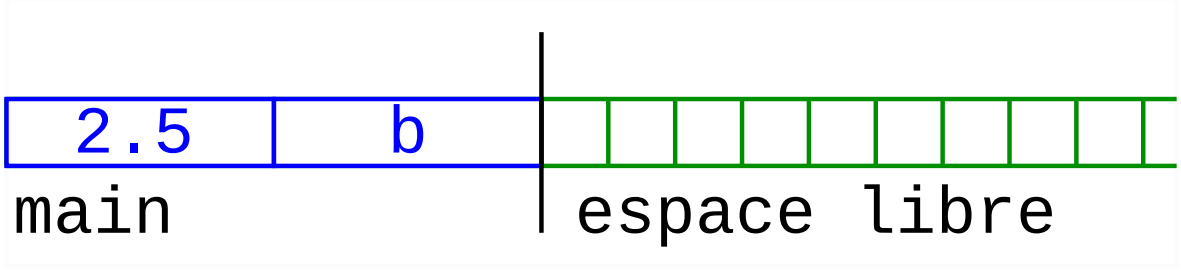
```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



new et delete

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

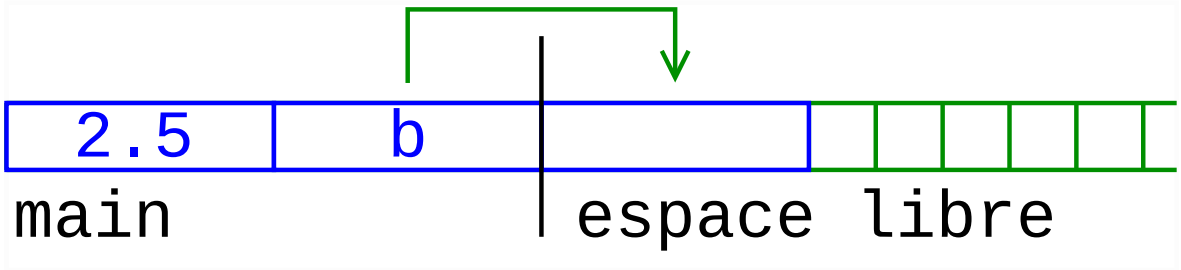
```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



new et delete

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

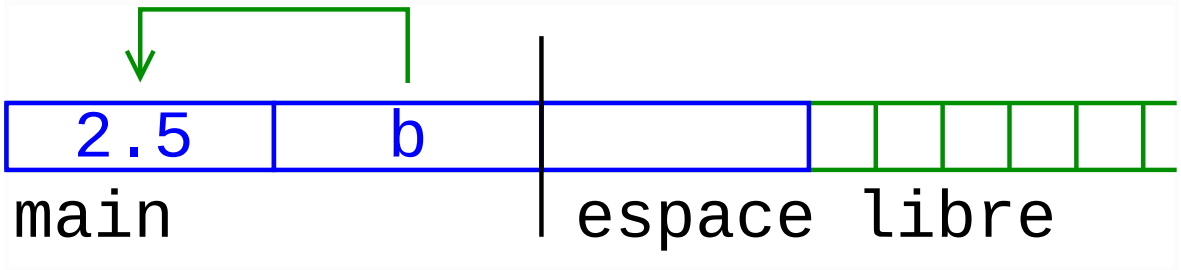
```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



new et delete

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

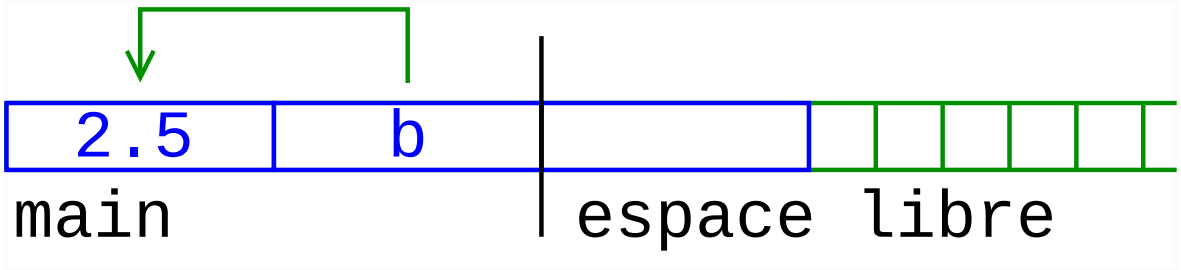
```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



new et delete

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

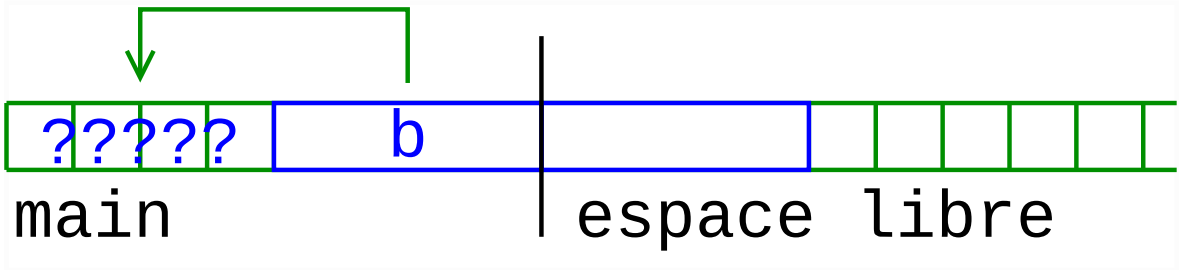
```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



new et delete

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



new et delete

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```

⟶ `erreur: double free or corruption ...`

new et delete

pschlagheck@pqs01: ~/cours/prog/progs - Shell - Konsole

Session Edit View Bookmarks Settings Help

```
pschlagheck@pqs01:~/cours/prog/progs$ ./prog1
2.5
*** glibc detected *** ./prog1: double free or corruption (out): 0xbfdd3ba0 ***
======= Backtrace: =========
/lib/i686/cmov/libc.so.6[0xb7d95624]
/lib/i686/cmov/libc.so.6(cfree+0x96)[0xb7d97826]
/usr/lib/libstdc++.so.6(_ZdlPv+0x21)[0xb7f6e2e1]
./prog1(__gxx_personality_v0+0x19b)[0x804879f]
/lib/i686/cmov/libc.so.6(__libc_start_main+0xe5)[0xb7d3d455]
./prog1(__gxx_personality_v0+0x3d)[0x8048641]
======= Memory map: ========
08048000-08049000 r-xp 00000000 08:02 15040917   /home/pschlagheck/cours/prog/progs/prog1
08049000-0804a000 rw-p 00000000 08:02 15040917   /home/pschlagheck/cours/prog/progs/prog1
0892a000-0894b000 rw-p 0892a000 00:00 0          [heap]
b7c00000-b7c21000 rw-p b7c00000 00:00 0
b7c21000-b7d00000 ---p b7c21000 00:00 0
b7d26000-b7d27000 rw-p b7d26000 00:00 0
b7d27000-b7e7c000 r-xp 00000000 08:02 17155267   /lib/i686/cmov/libc-2.7.so
b7e7c000-b7e7d000 r--p 00155000 08:02 17155267   /lib/i686/cmov/libc-2.7.so
b7e7d000-b7e7f000 rw-p 00156000 08:02 17155267   /lib/i686/cmov/libc-2.7.so
b7e7f000-b7e82000 rw-p b7e7f000 00:00 0
b7e82000-b7e8e000 r-xp 00000000 08:02 17145859   /lib/libgcc_s.so.1
b7e8e000-b7e8f000 rw-p 0000b000 08:02 17145859   /lib/libgcc_s.so.1
b7e8f000-b7e90000 rw-p b7e8f000 00:00 0
b7e90000-b7eb4000 r-xp 00000000 08:02 17155271   /lib/i686/cmov/libm-2.7.so
b7eb4000-b7eb6000 rw-p 00023000 08:02 17155271   /lib/i686/cmov/libm-2.7.so
b7eb6000-b7f99000 r-xp 00000000 08:02 6848842    /usr/lib/libstdc++.so.6.0.10
b7f99000-b7f9c000 r--p 000e2000 08:02 6848842    /usr/lib/libstdc++.so.6.0.10
b7f9c000-b7f9e000 rw-p 000e5000 08:02 6848842    /usr/lib/libstdc++.so.6.0.10
b7f9e000-b7fa4000 rw-p b7f9e000 00:00 0
b7fb7000-b7fba000 rw-p b7fb7000 00:00 0
b7fba000-b7fbb000 r-xp b7fba000 00:00 0          [vdso]
b7fbb000-b7fd5000 r-xp 00000000 08:02 17145858   /lib/ld-2.7.so
b7fd5000-b7fd7000 rw-p 0001a000 08:02 17145858   /lib/ld-2.7.so
bfdc1000-bfdd6000 rw-p bffeb000 00:00 0          [stack]
Aborted
pschlagheck@pqs01:~/cours/prog/progs$
```

Shell

Sent for Peter.Schlagheck@ LEO Deutsch-Französisches pschlagheck@pqs01: ~/

Xfig - flot_newmain8.fig prog8.pdf - Adobe Reader emacs@localhost [3]

20:15 23/04/10

`new` et `delete`

Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```

$\longrightarrow$ nécessaire de faire du `delete` ?

$\longrightarrow$ à quoi bon tout ça ??

10.3 Les tableaux dynamiques

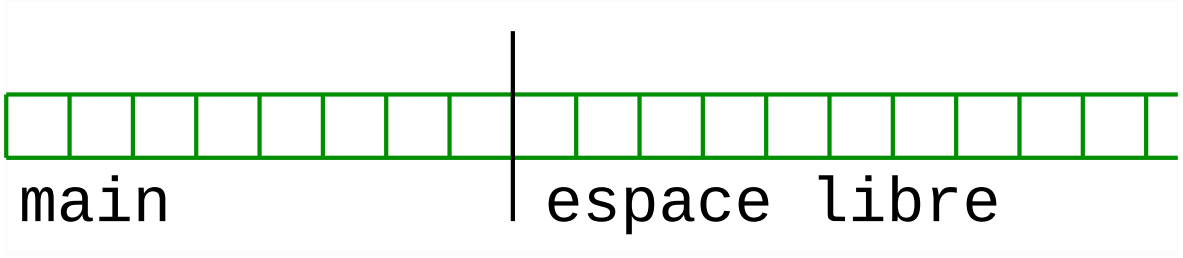
`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

```
int main()
{
  int N = 5;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```

10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

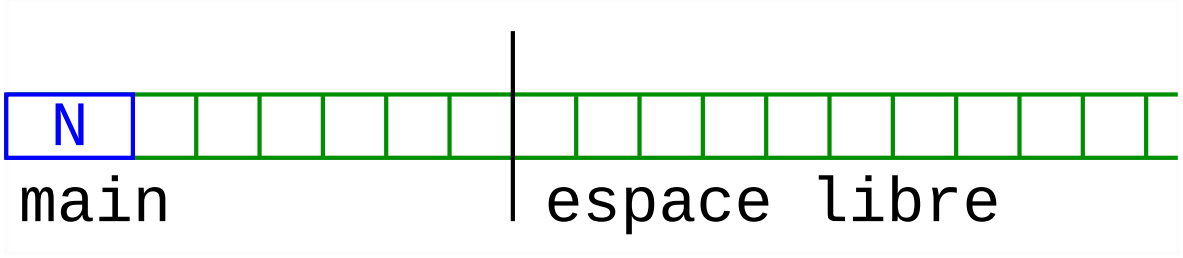
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

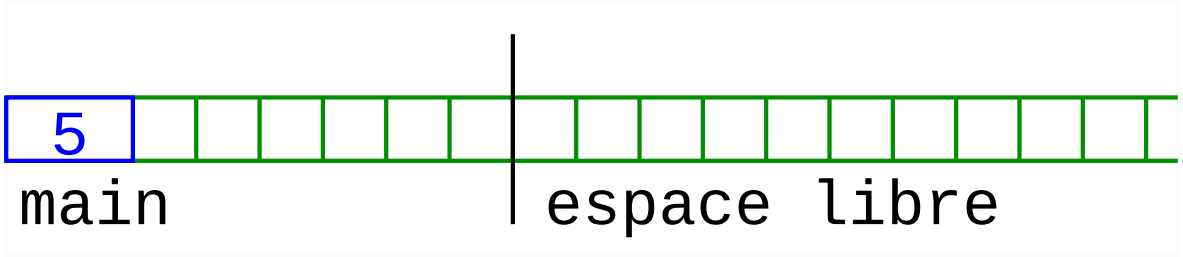
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

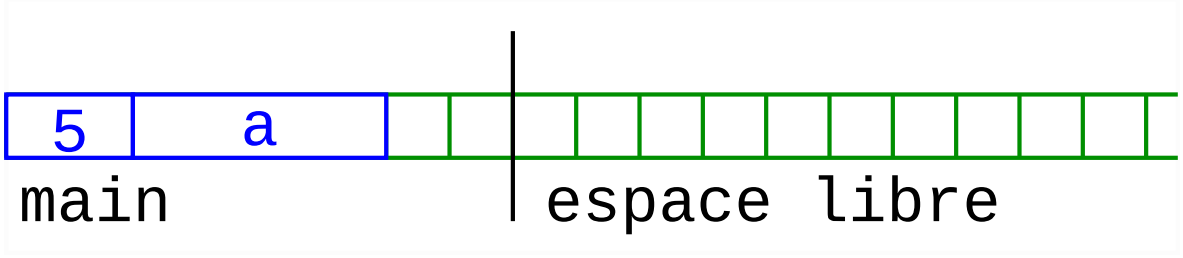
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

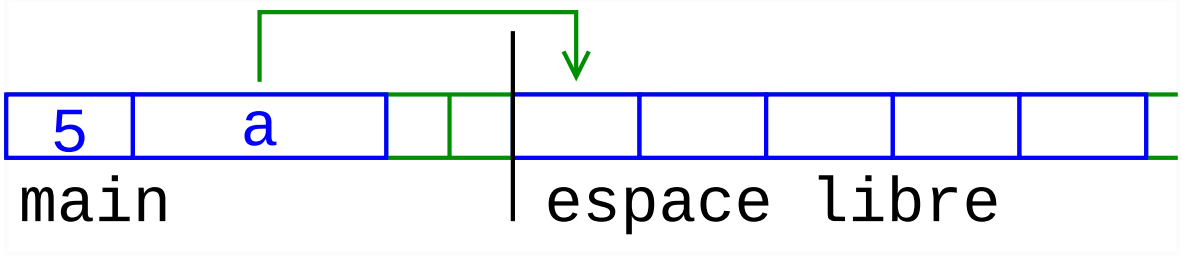
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

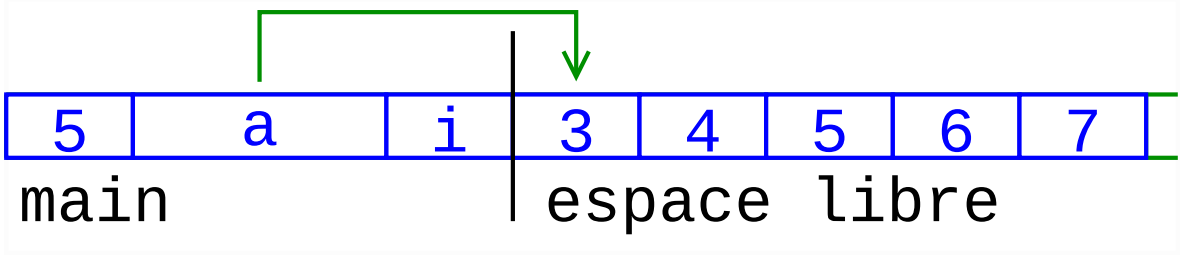
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

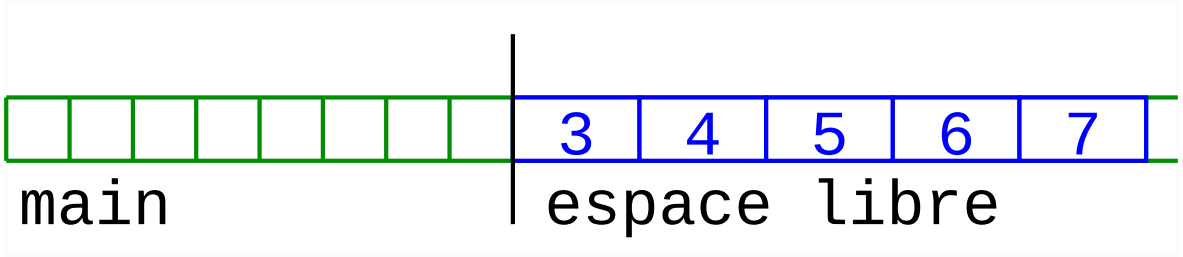
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

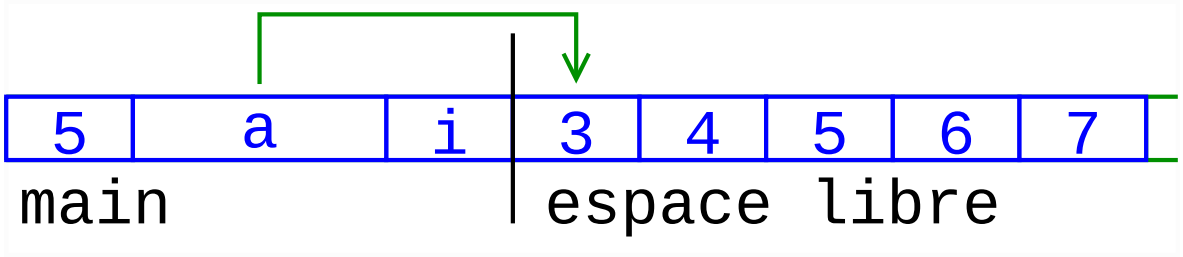
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

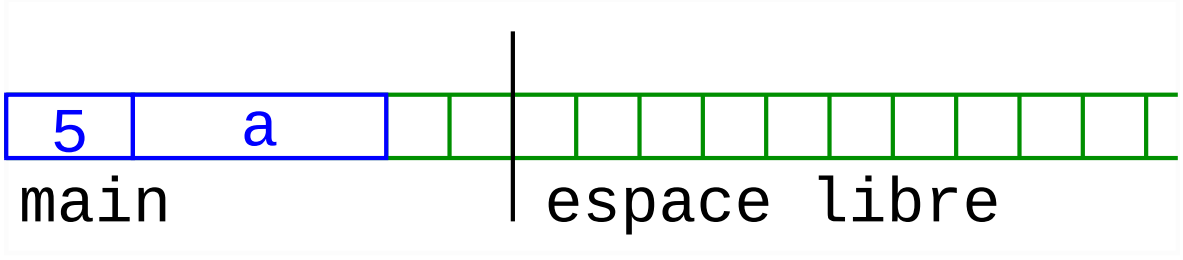
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
  delete[] a;
}
```



10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

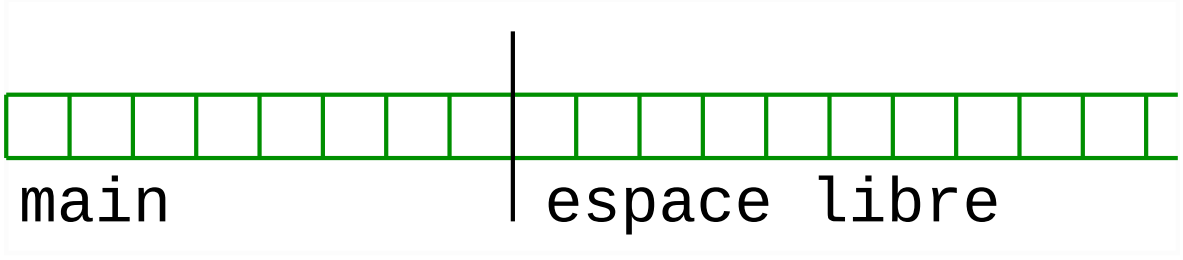
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
  delete[] a;
}
```



10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
  delete[] a;
}
```



10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

⟨type⟩ `*`⟨nom⟩`;`

⟨nom⟩ `=` `new` ⟨type⟩`[` ⟨taille⟩ `];`

$\longrightarrow$ réservation des blocs sur l’espace libre pour
un nouveau tableau de ⟨taille⟩ variables du type ⟨type⟩

et copie de son adresse dans ⟨nom⟩

Les éléments de ce tableau peuvent donc être accédés par ⟨nom⟩`[` ⟨entier⟩ `].`

10.3 Les tableaux dynamiques

`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

⟨type⟩ `*`⟨nom⟩`;`

⟨nom⟩ `= new` ⟨type⟩`[` ⟨taille⟩ `];`

`delete[]` ⟨nom⟩`;`

$\longrightarrow$ libération de cet espace bloqué dans la mémoire

afin qu’il puisse être réutilisé par d’autres instructions `new`

10.3 Les tableaux dynamiques

Exemple (mécanique quantique):

Calcul de la valeur propre la plus basse d'une grande matrice symétrique dont les éléments se construisent en fonction d'un ou plusieurs paramètres

```
double valprop( int N, double par )
{
  double *matr = new double[ N * N ];
  ...
  // remplir la matrice;
  ...
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}
```

10.3 Les tableaux dynamiques

```
#include <matrix.h>

double valprop( int N, double par )
{
  double *matr = new double[ N * N ];
  ...
  // remplir la matrice;
  ...
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}
```

Le contenu de `matrix.h` :

```
void diagonalise( int N, double *matr, double *vpr );
```

$\longrightarrow$ calculer les valeurs propres de la matrice `matr`
et les rendre ordonnées par le vecteur `vpr`

10.3 Les tableaux dynamiques

```
#include <matrix.h>

double valprop( int N, double par )
{
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}
```

Le contenu de `matrix.h` :

```
void diagonalise( int N, double *matr, double *vpr );
void constr_matrix( int N, double *matr, double par );
```

$\longrightarrow$ calculer les éléments de la matrice
en fonction du paramètre `par`

10.3 Les tableaux dynamiques

```
#include <matrix.h>

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}
```

Le contenu de `matrix.h` :

```
void diagonalise( int N, double *matr, double *vpr );
void constr_matrix( int N, double *matr, double par );
int taille_optimale( double par );
```

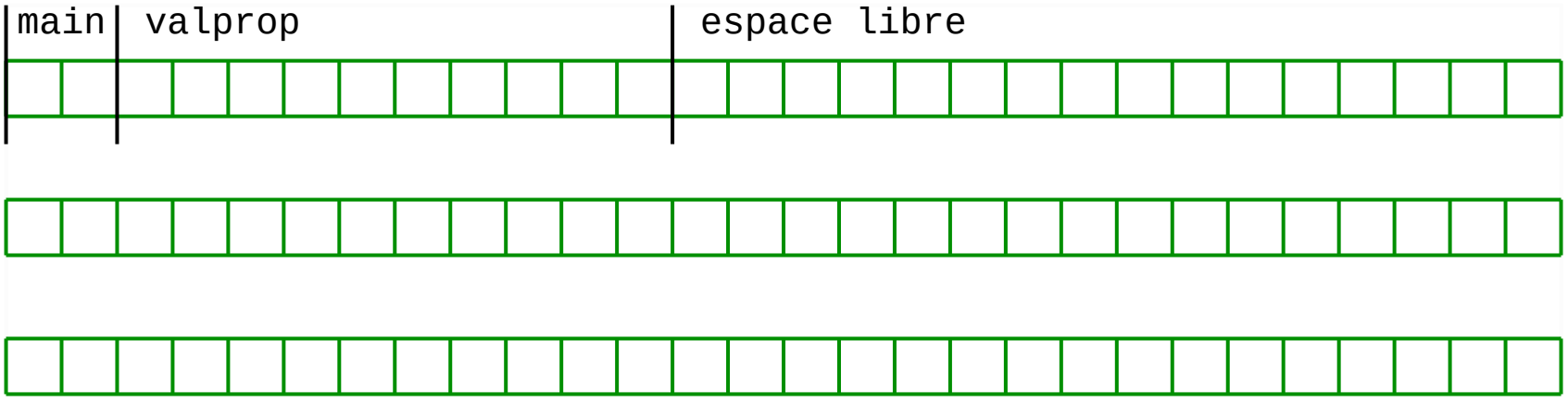
$\longrightarrow$ déterminer la taille optimale de la matrice

10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```

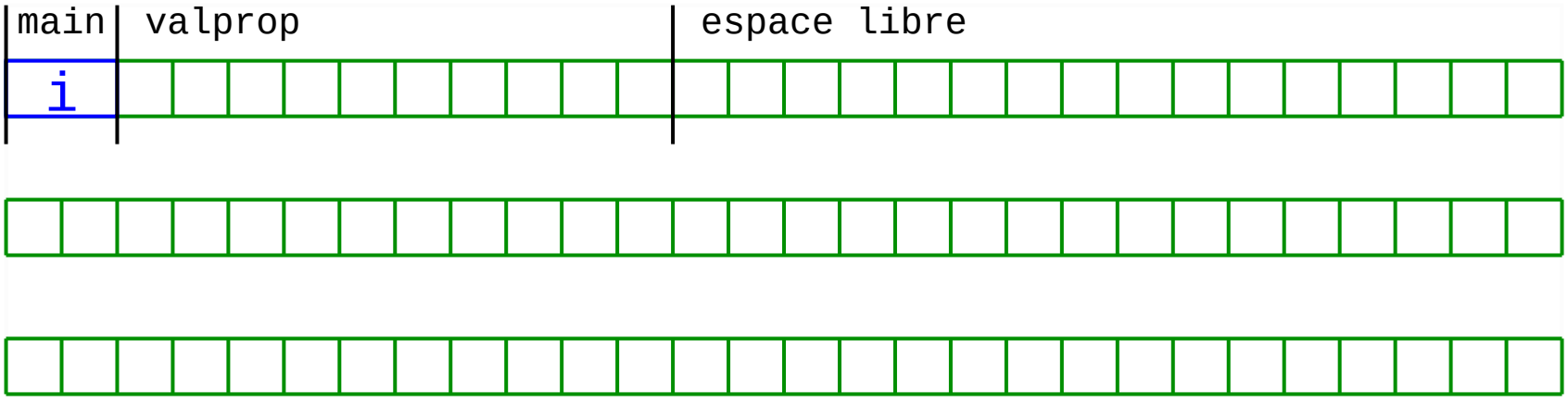
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



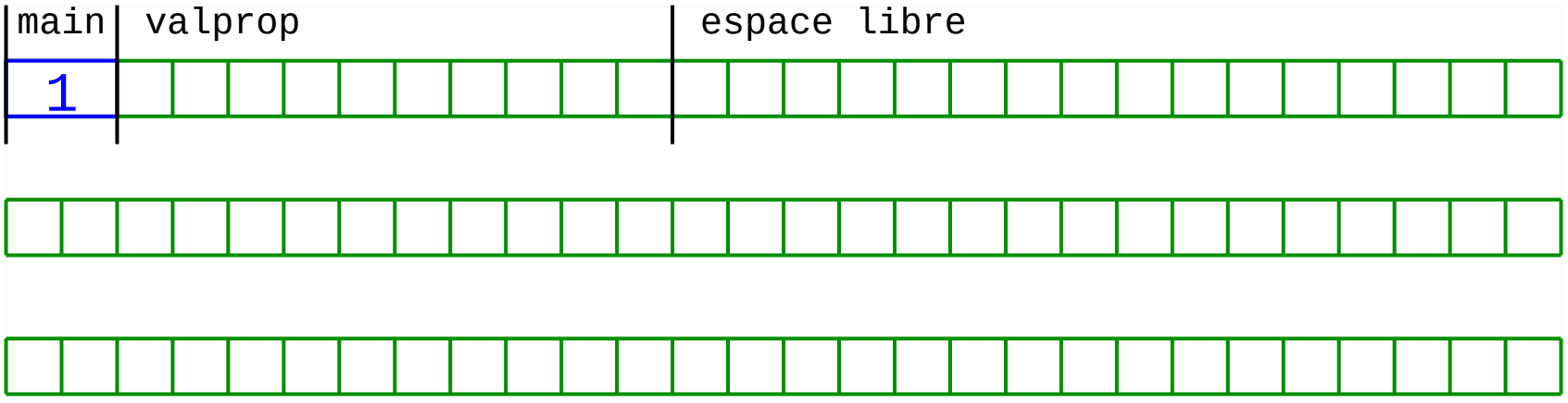
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



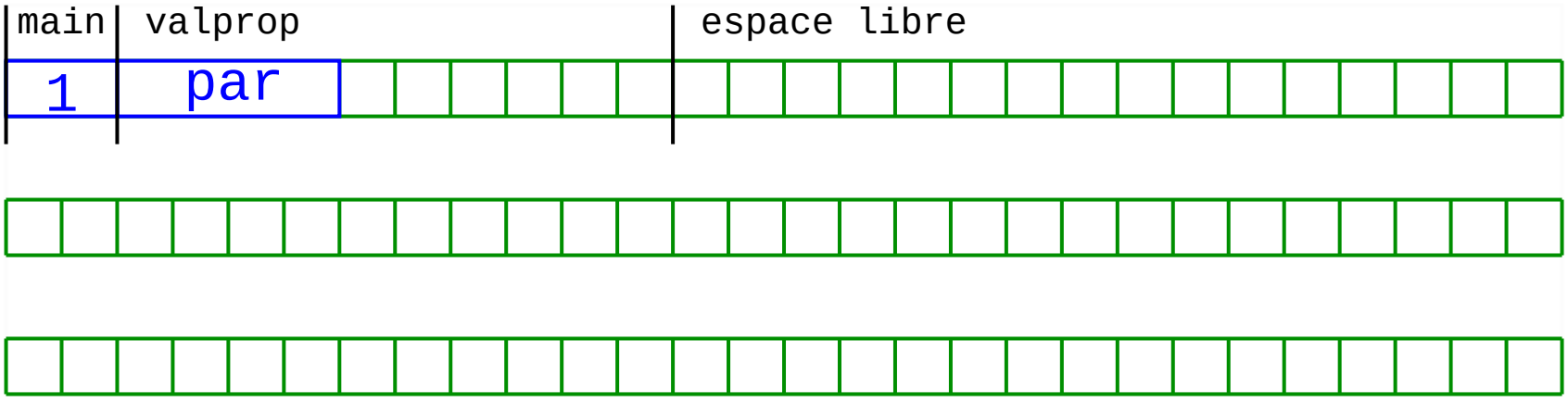
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



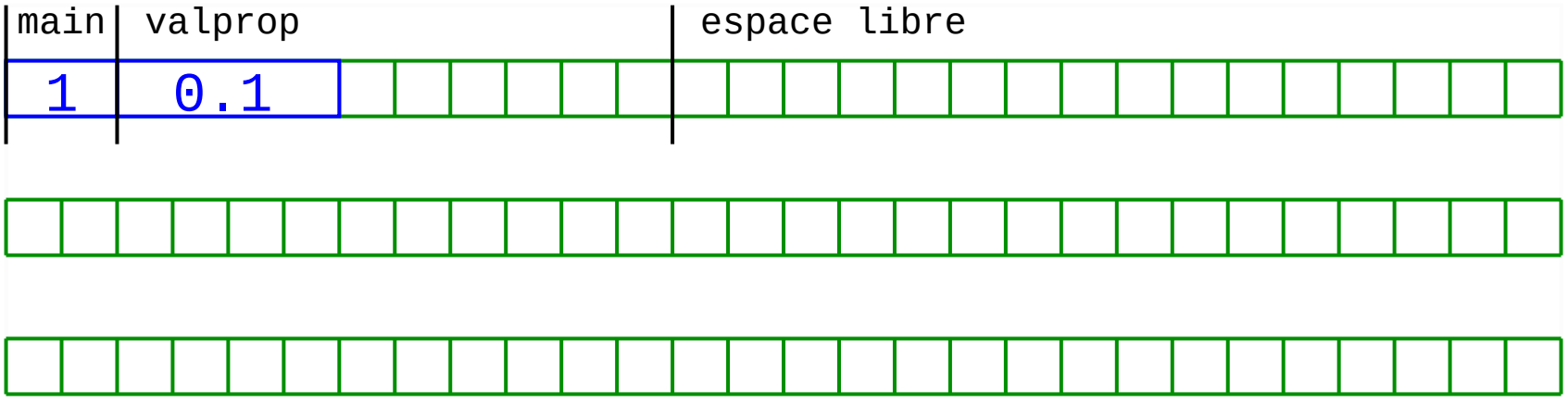
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



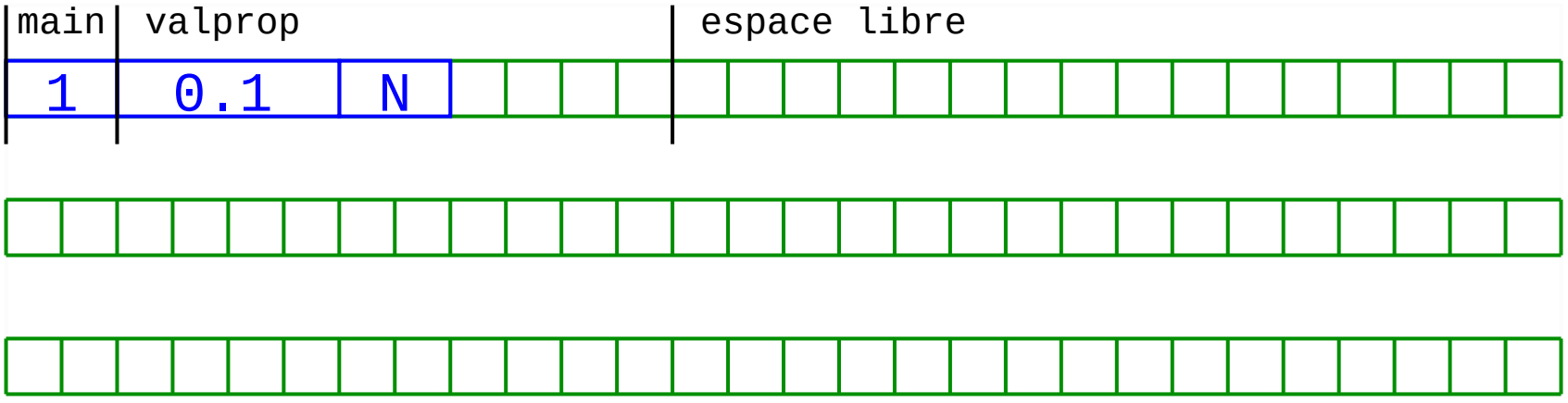
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



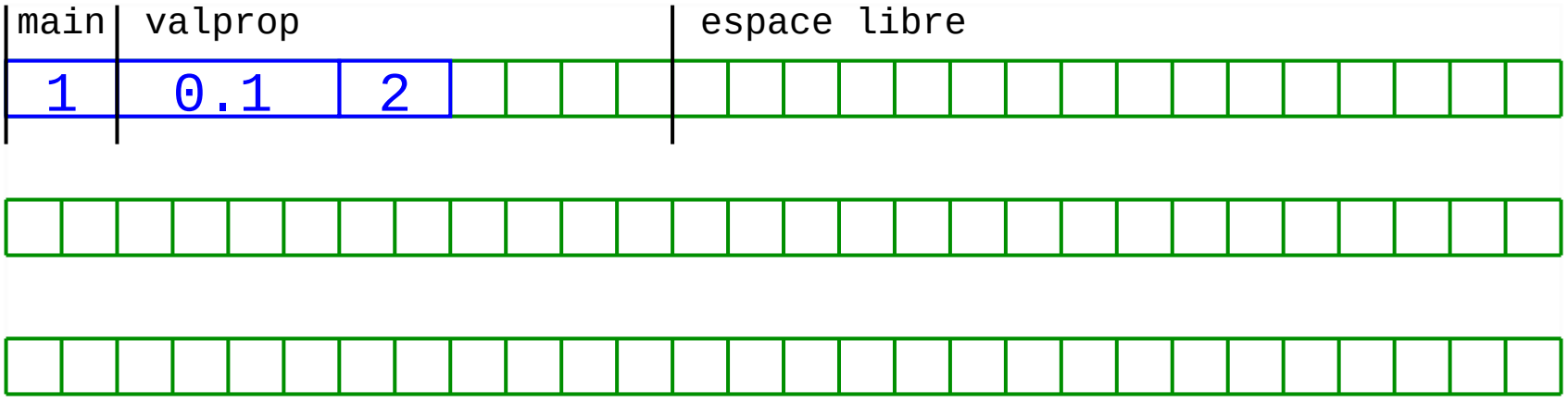
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



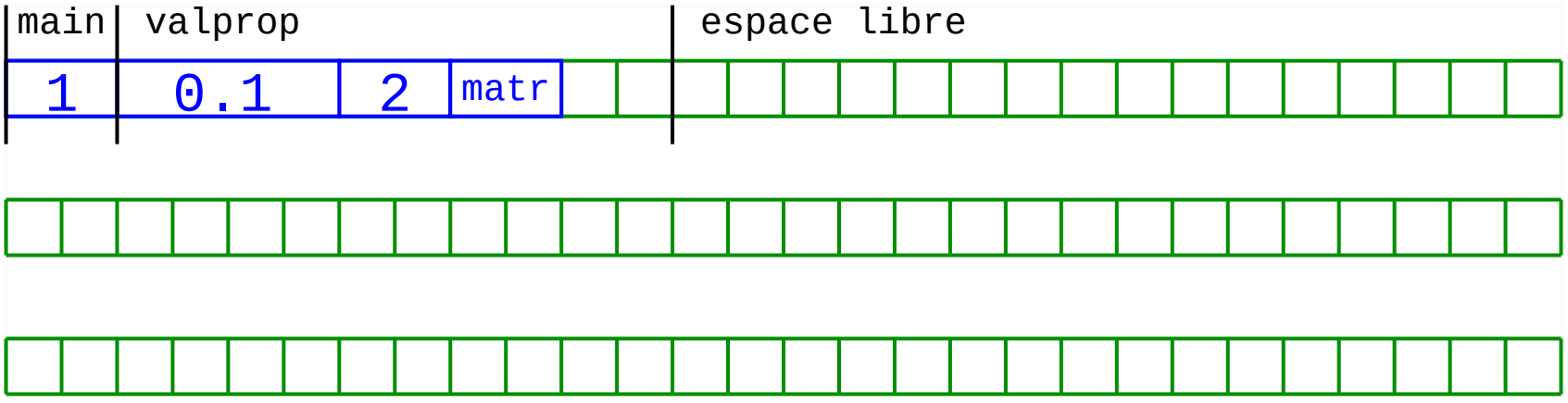
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



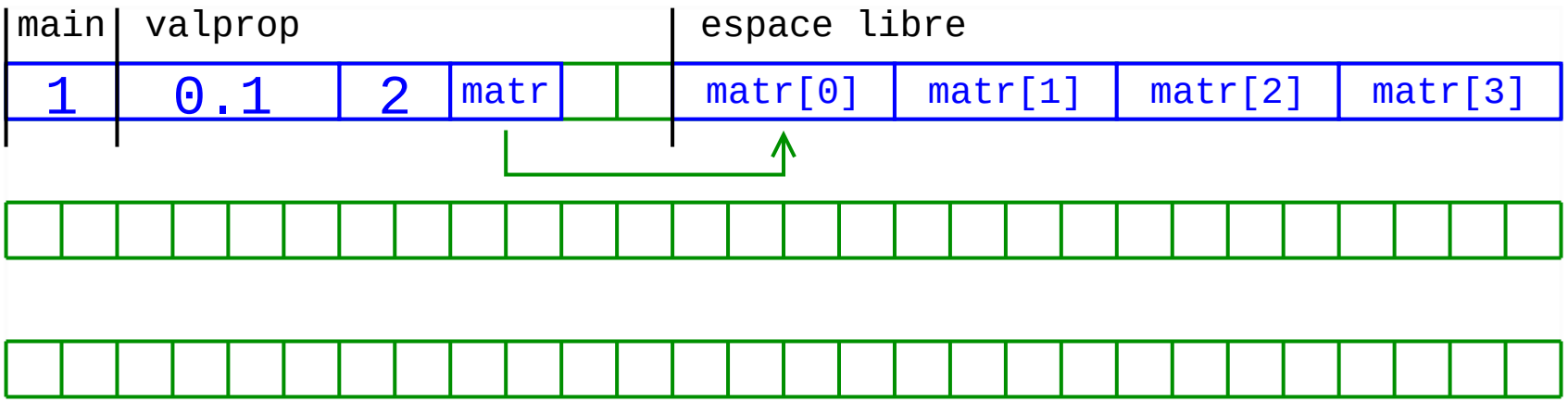
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



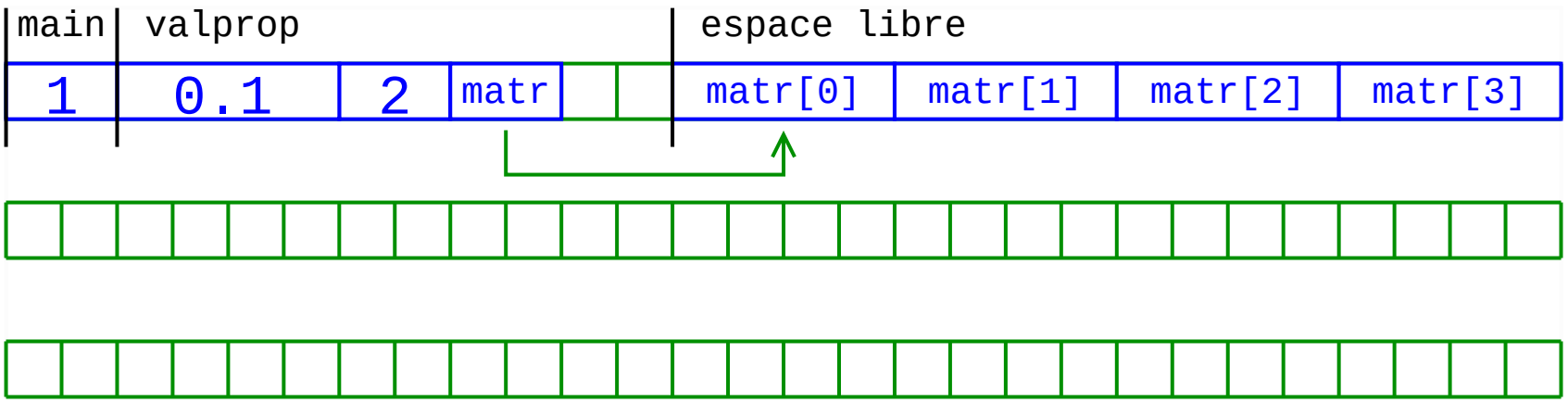
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



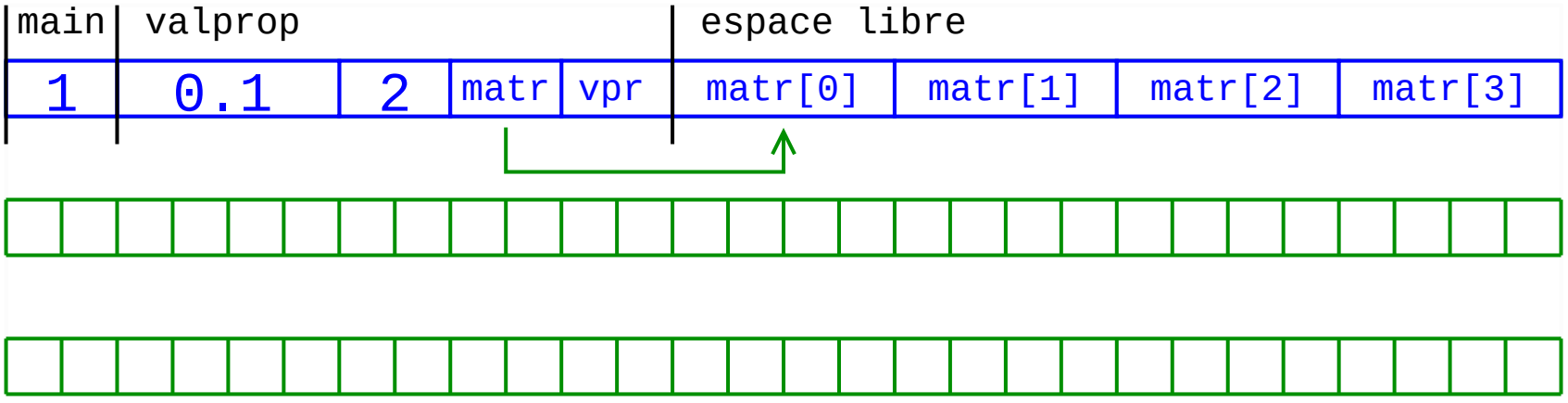
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



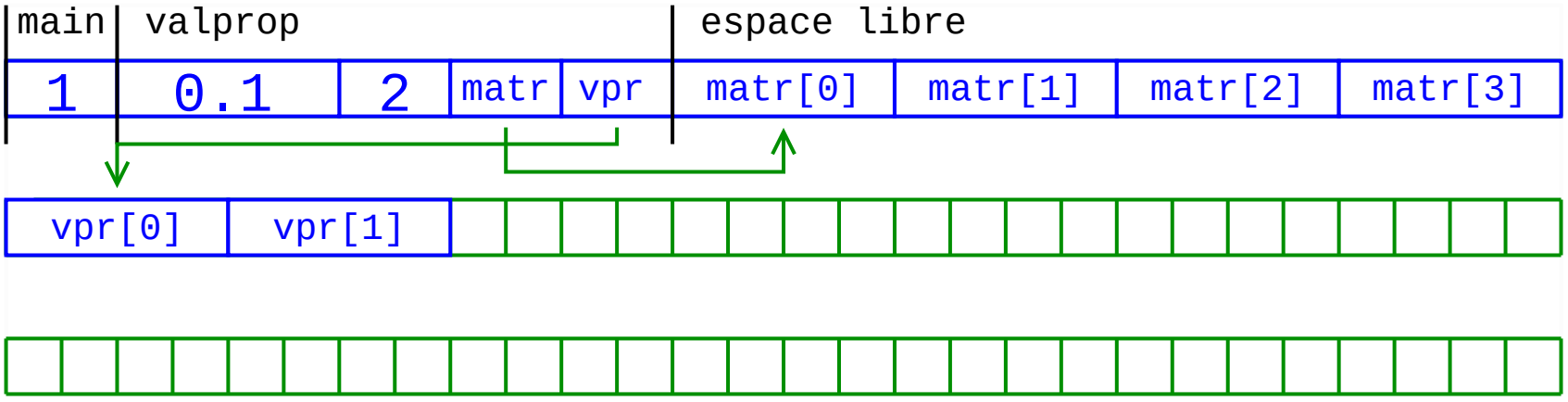
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



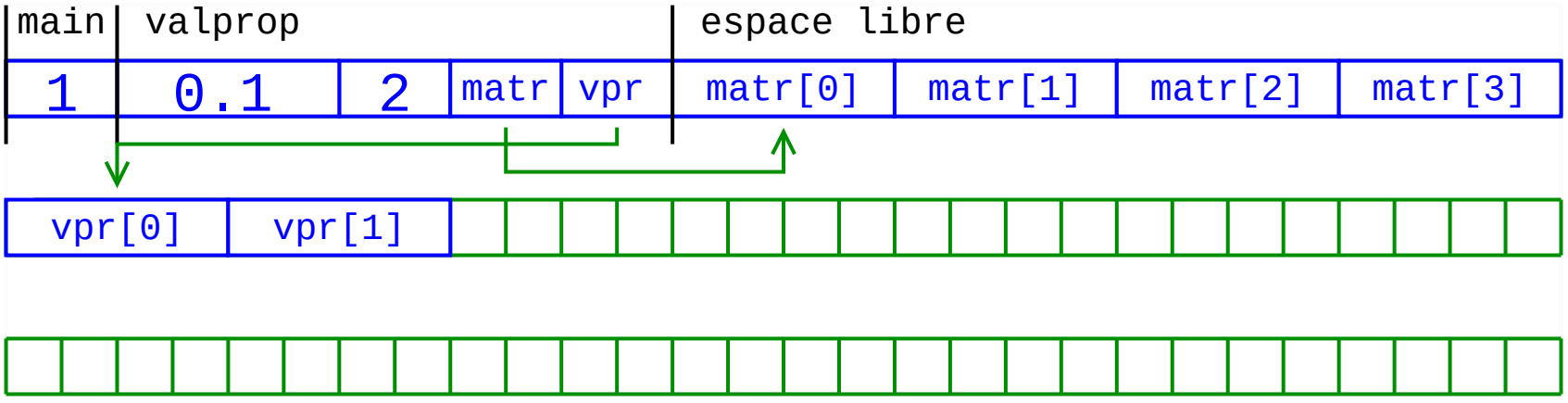
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



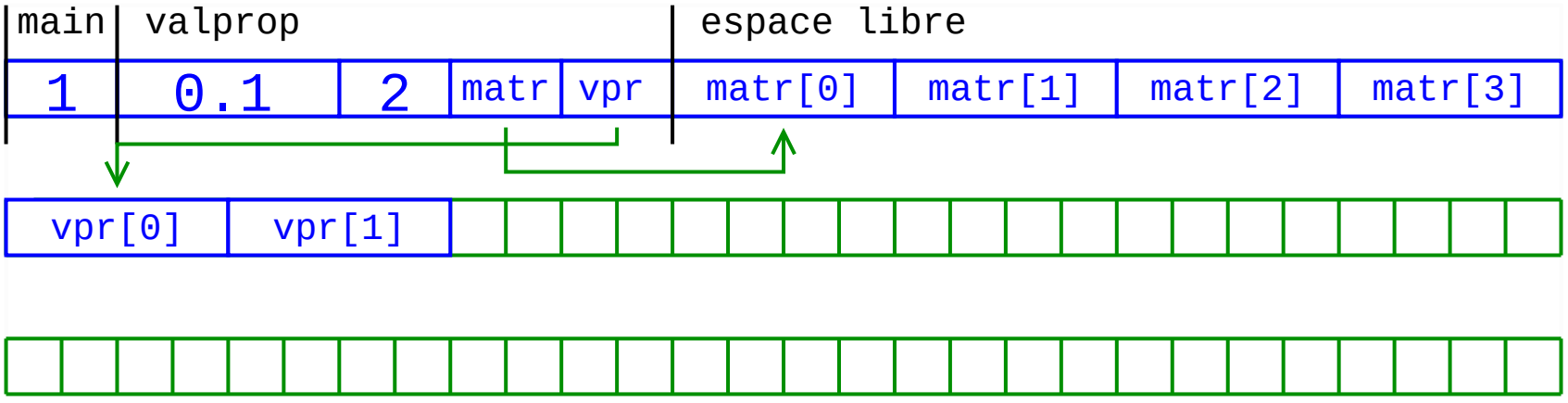
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



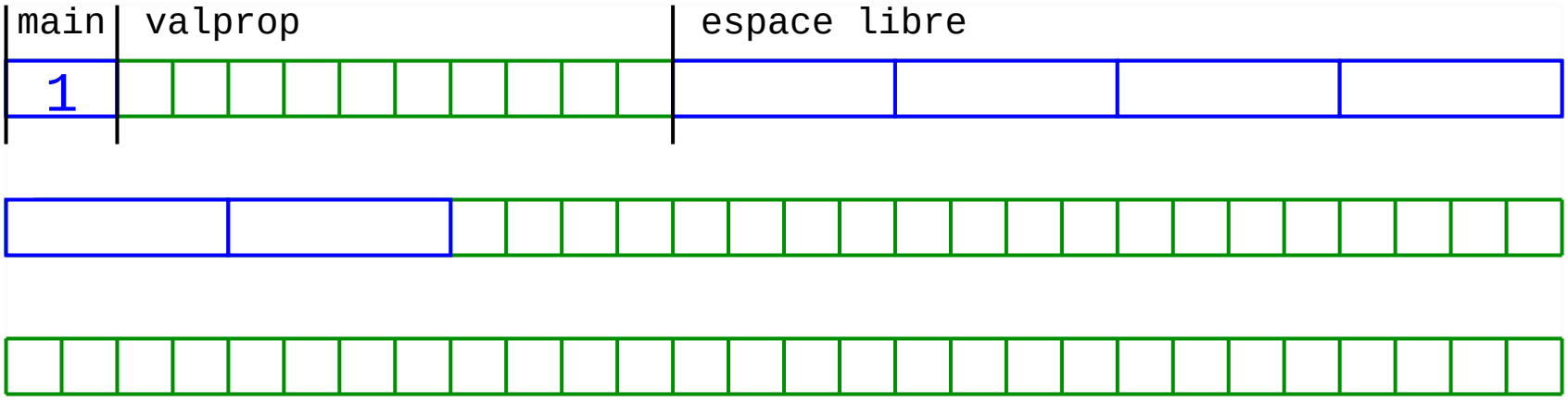
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



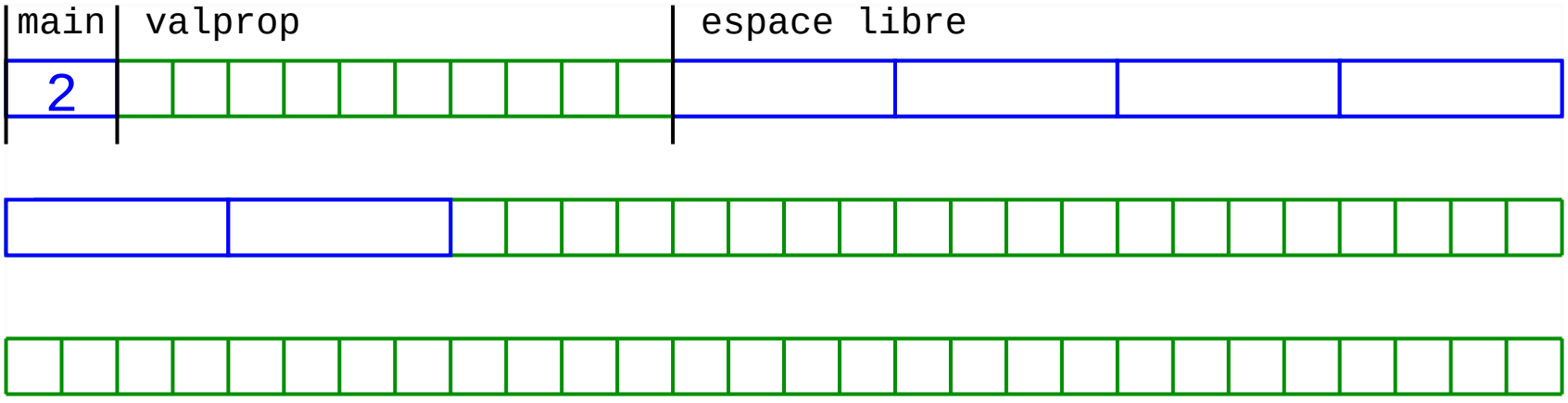
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



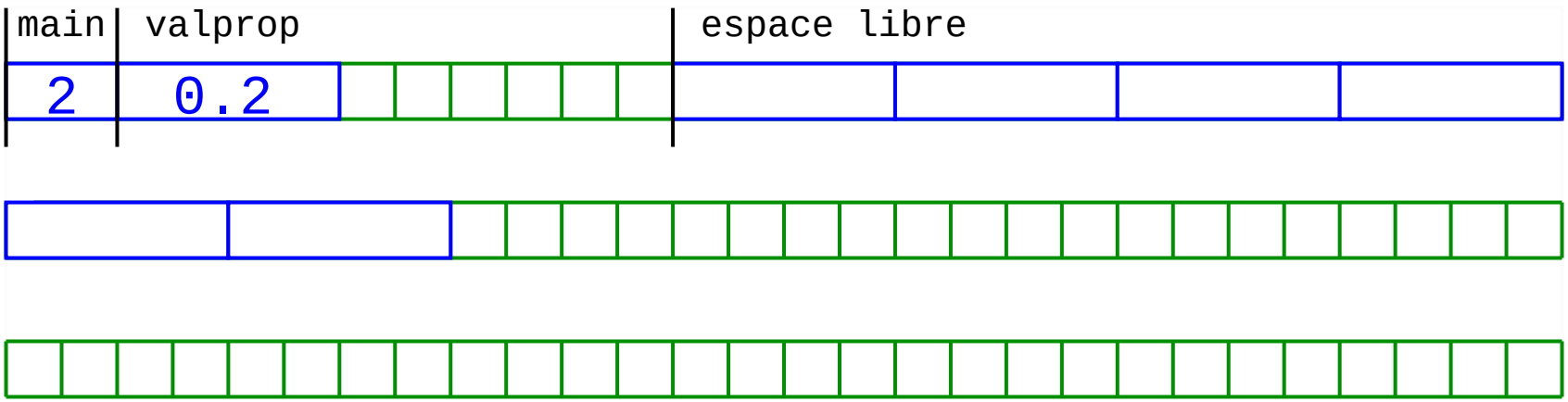
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



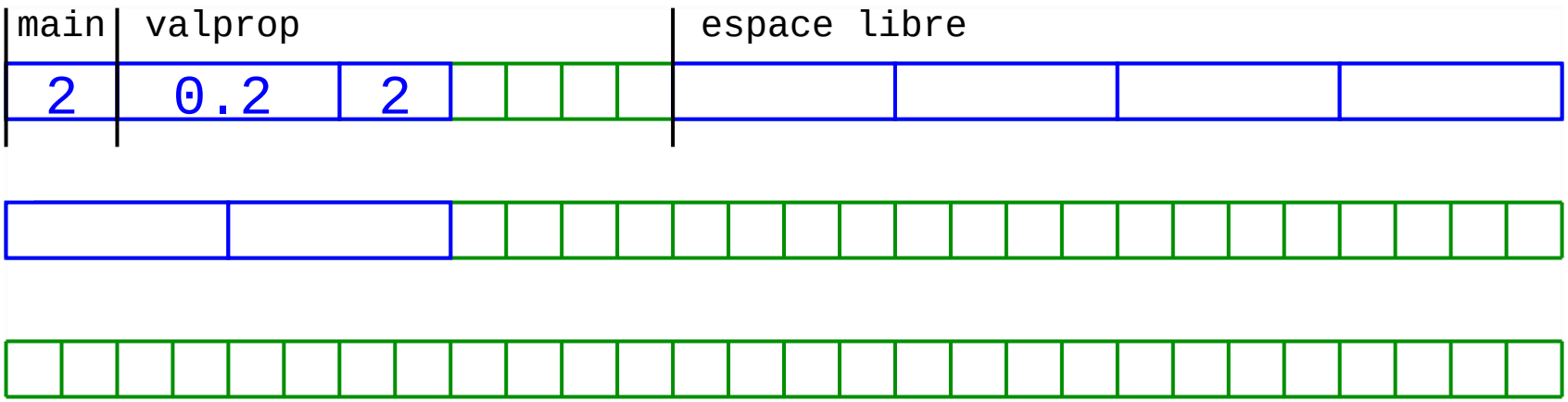
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



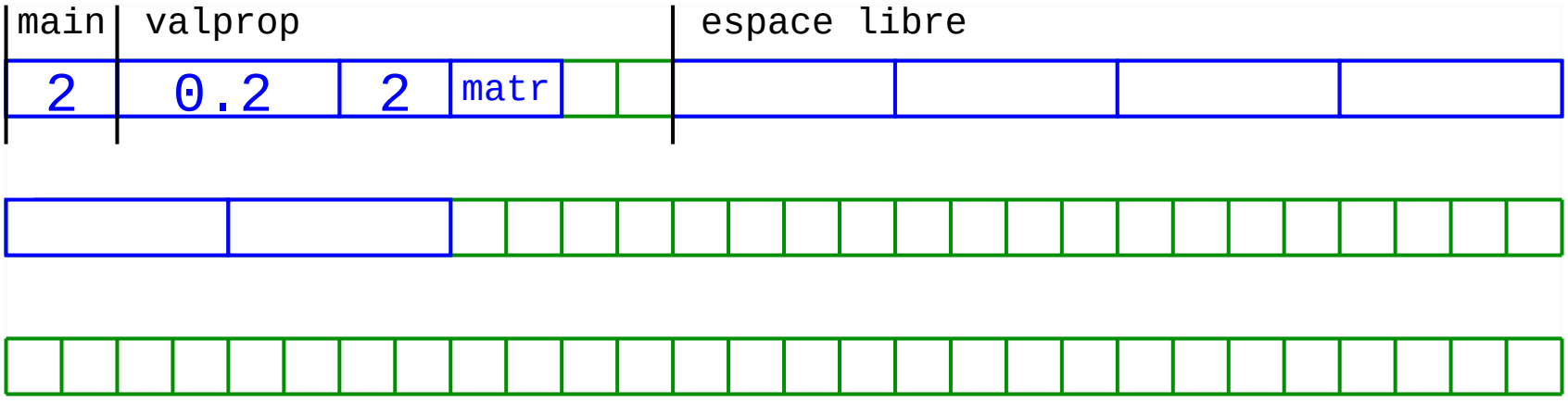
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



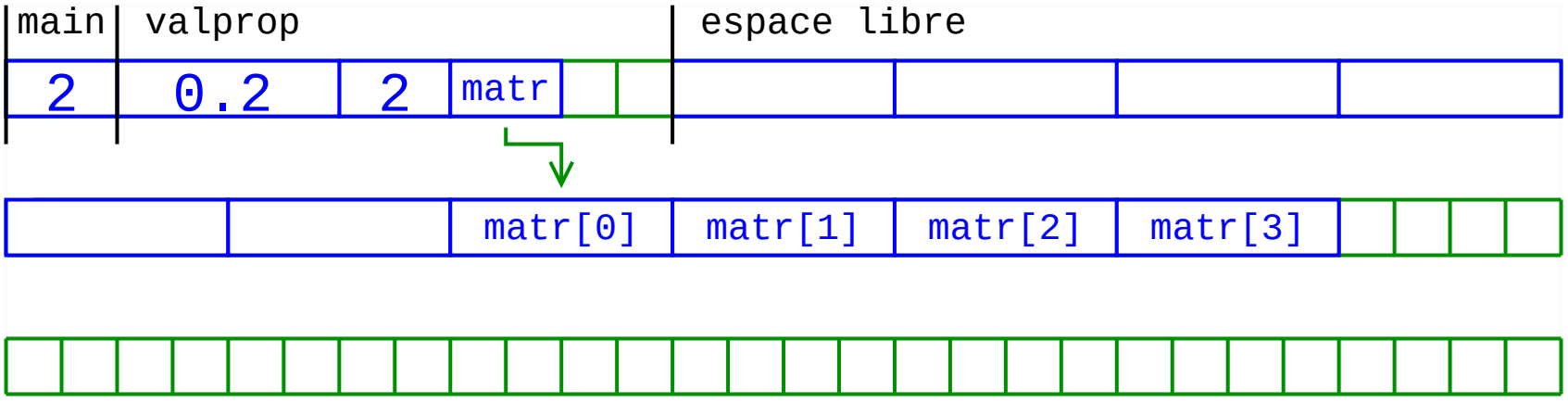
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



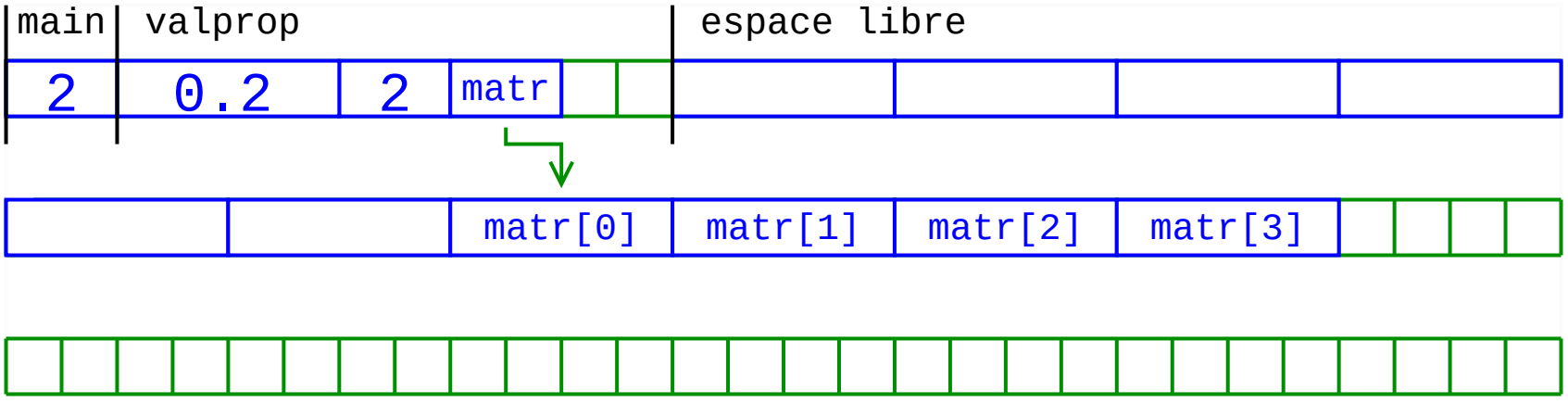
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



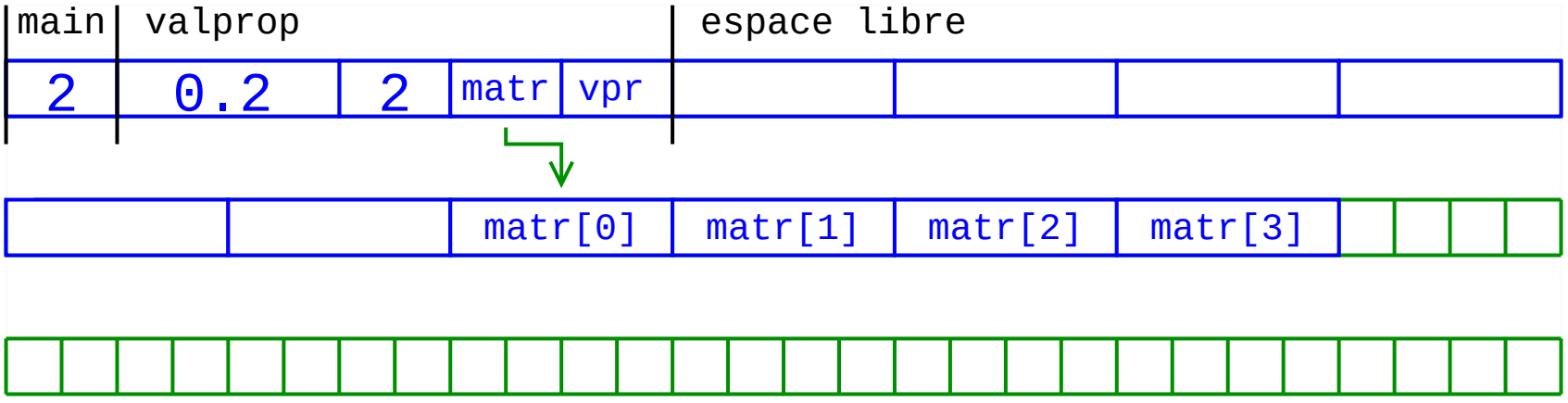
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



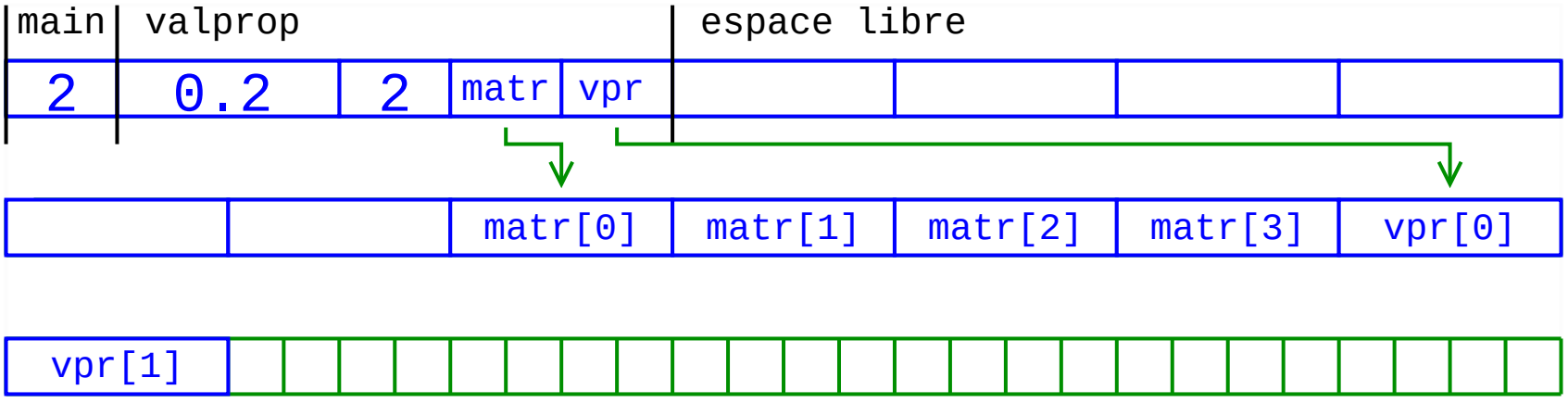
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



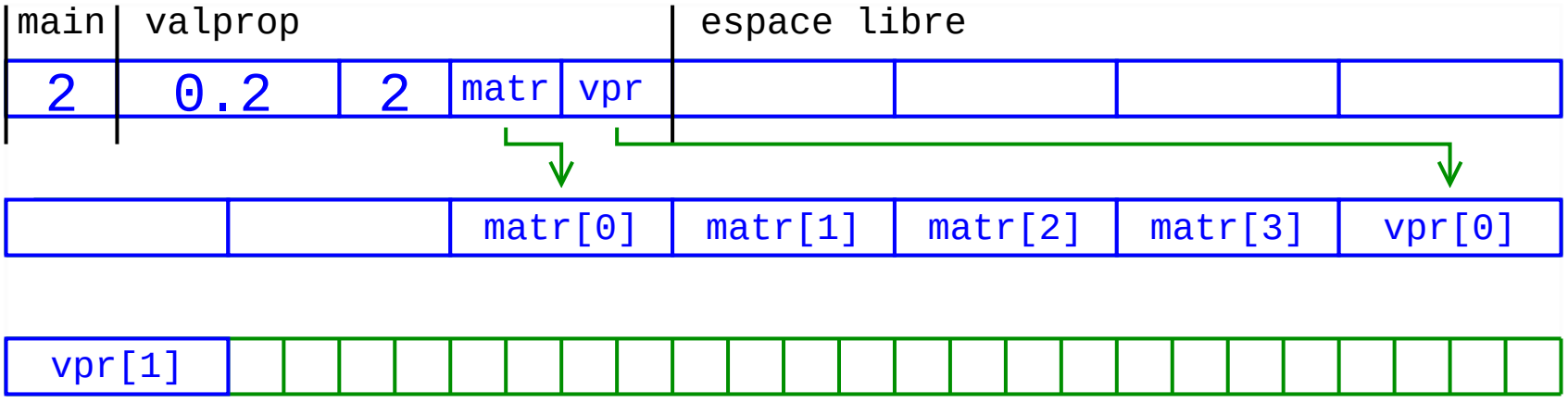
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



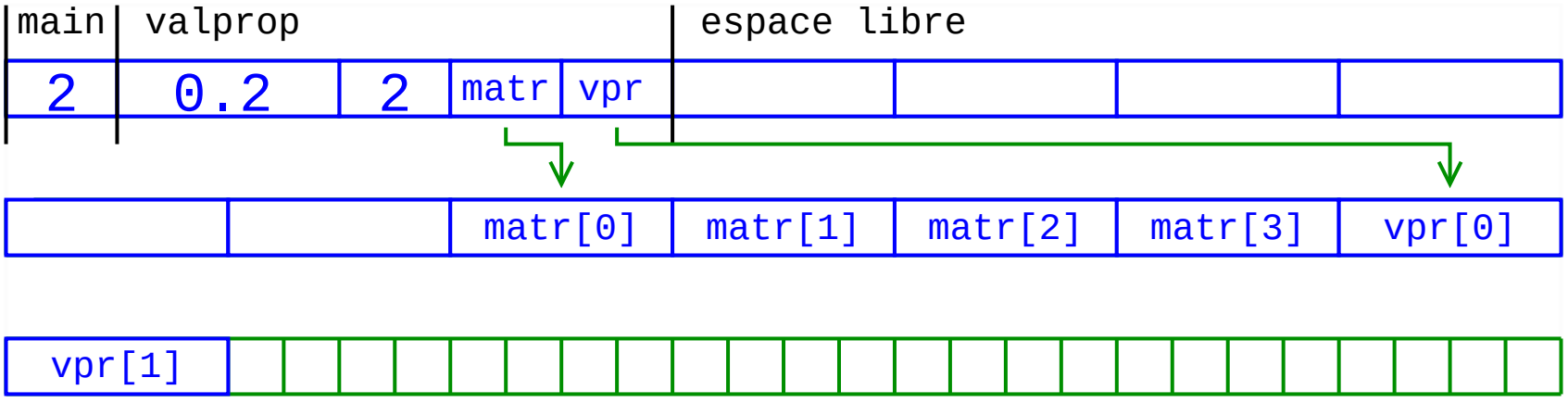
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



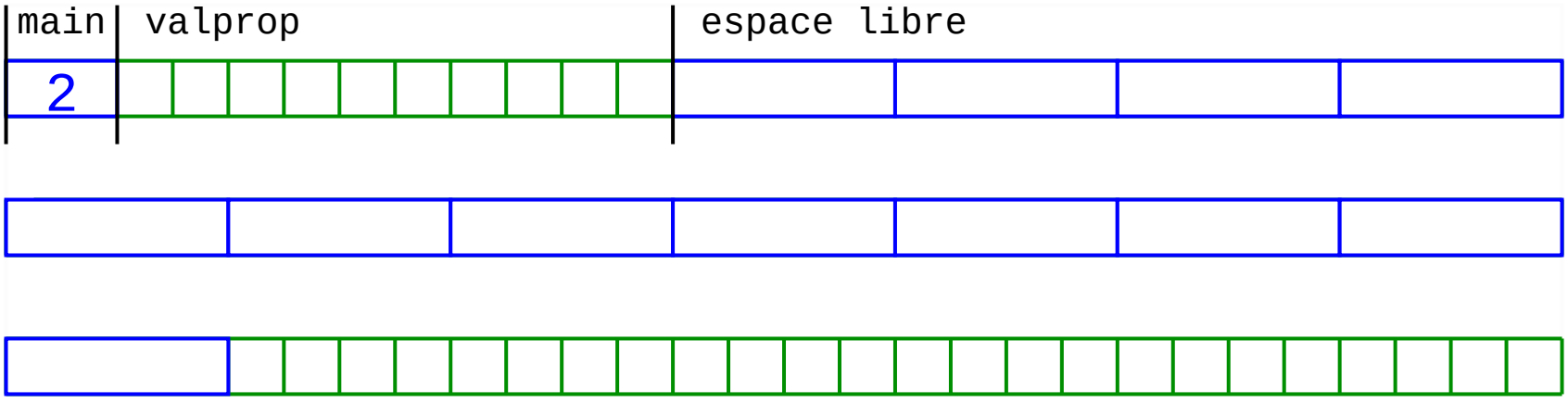
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



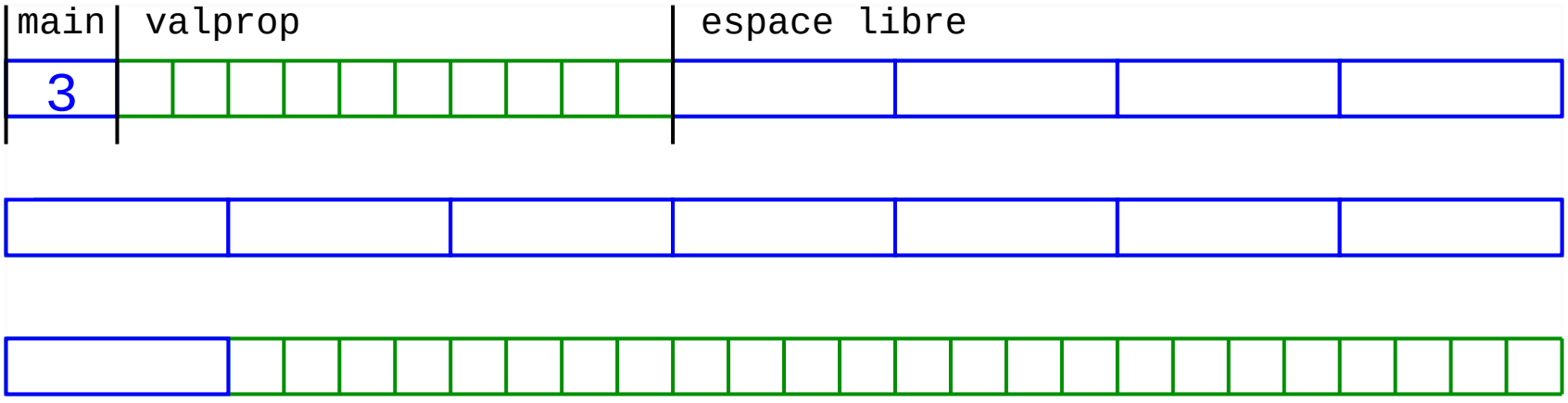
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



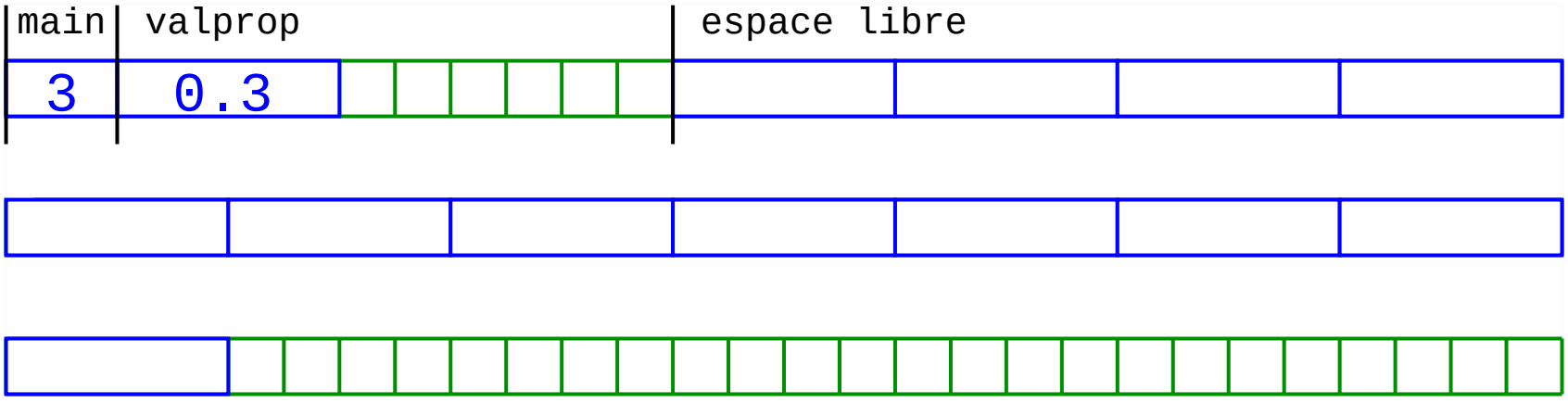
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



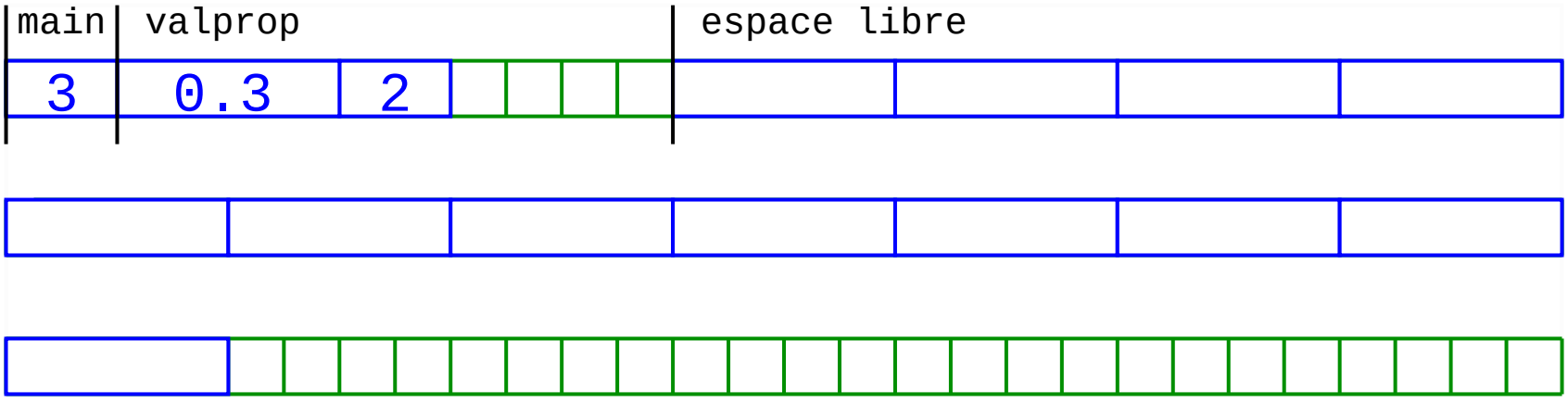
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



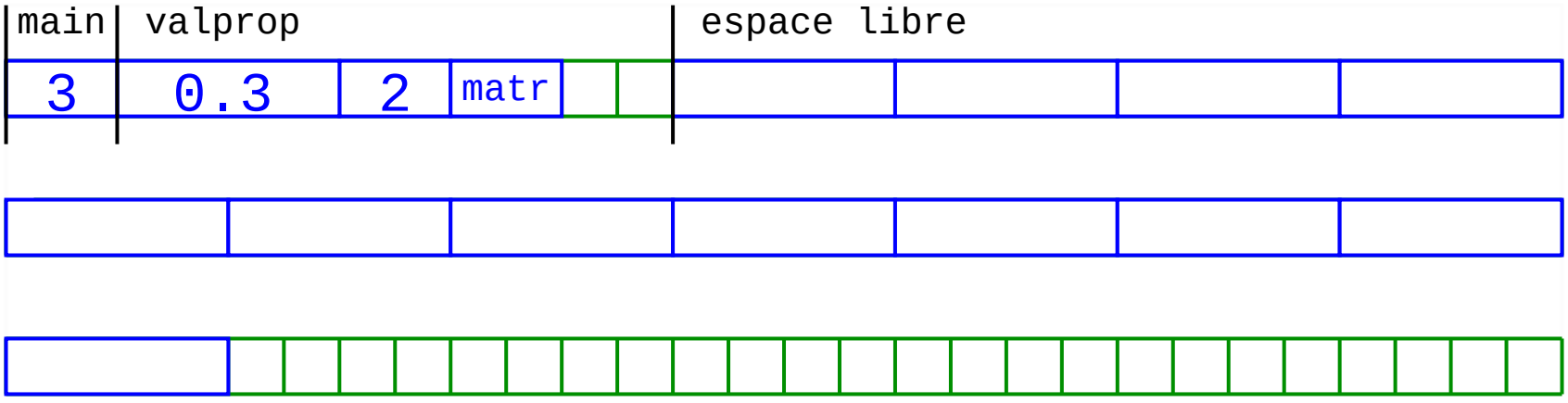
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



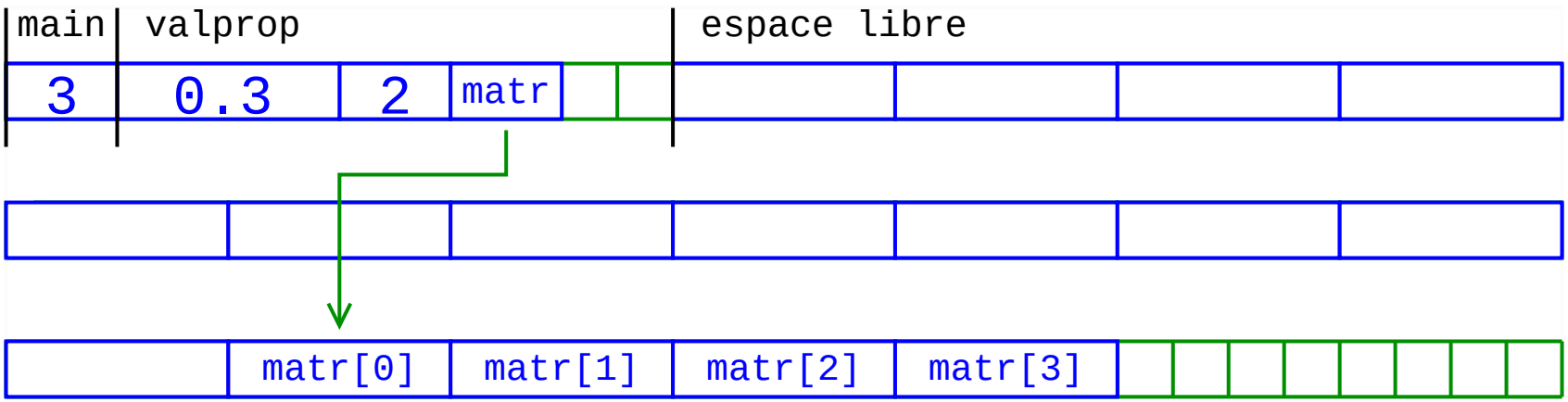
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



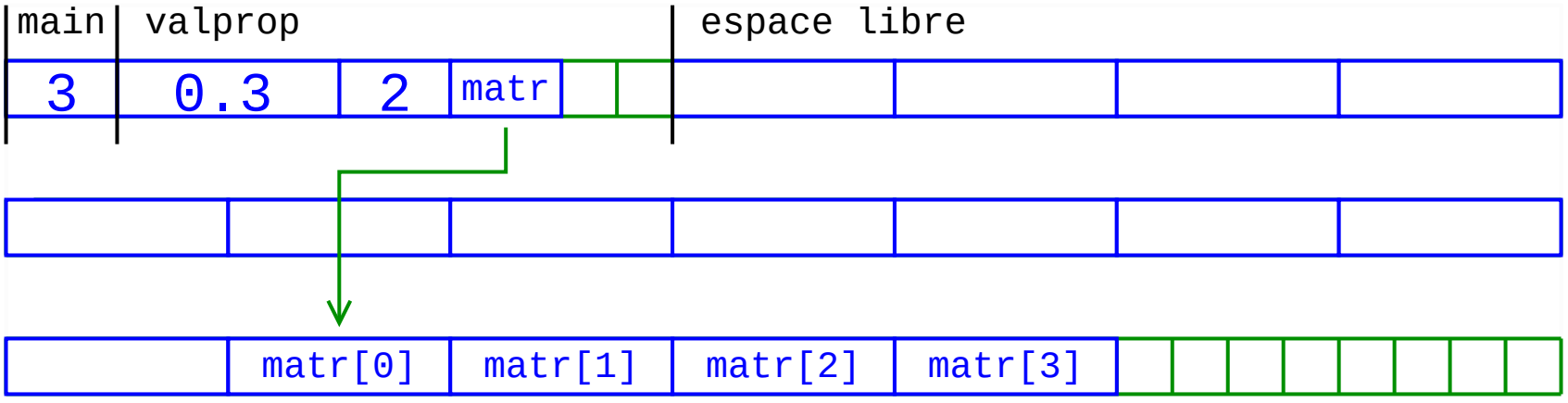
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



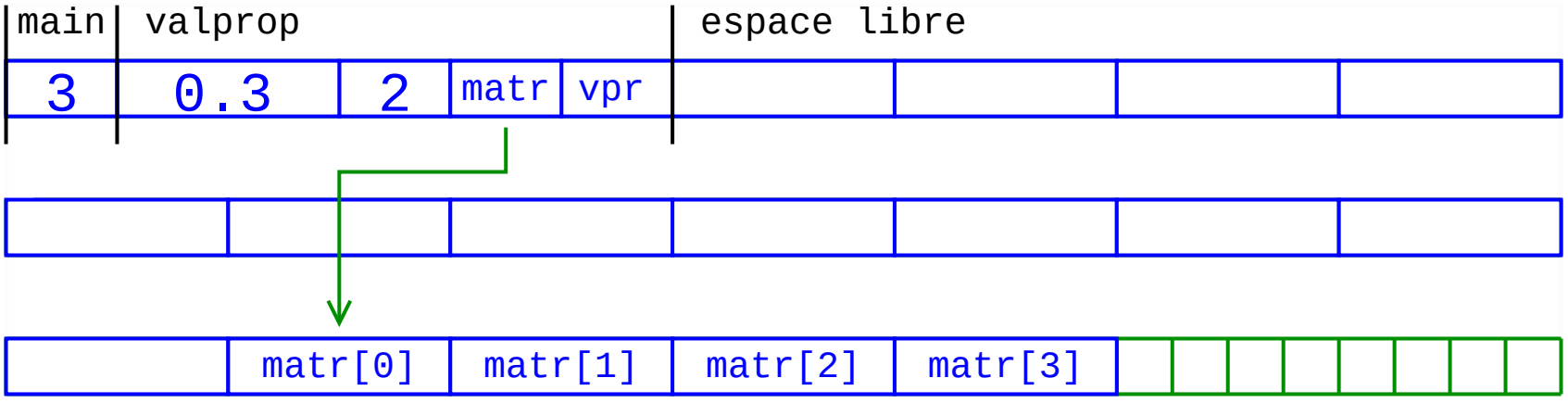
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



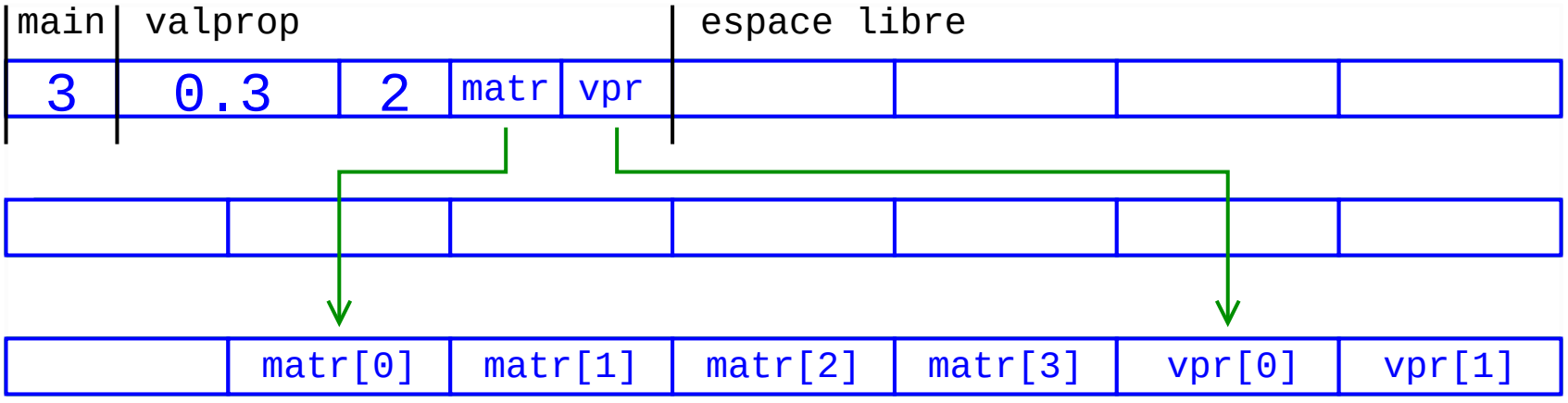
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



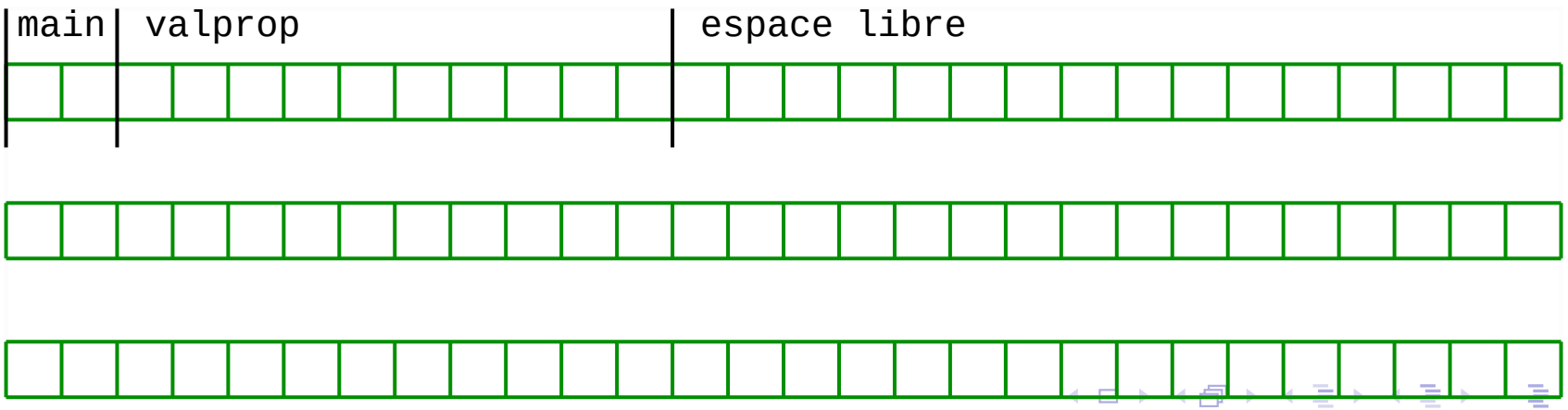
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```

$\longrightarrow$ on risque de bloquer la mémoire de l'ordinateur si on oublie de mettre `delete`

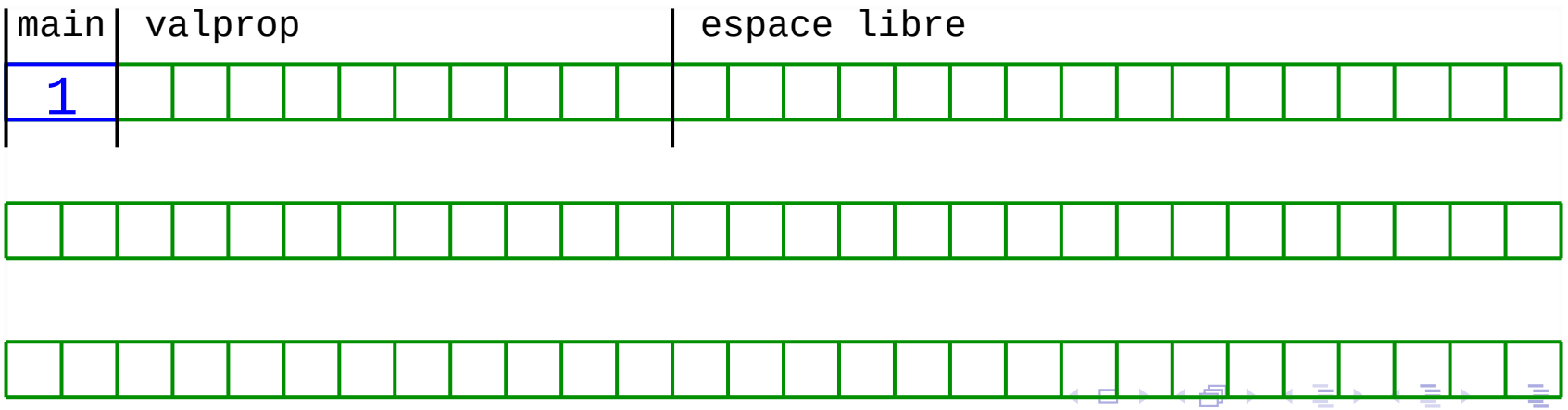
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



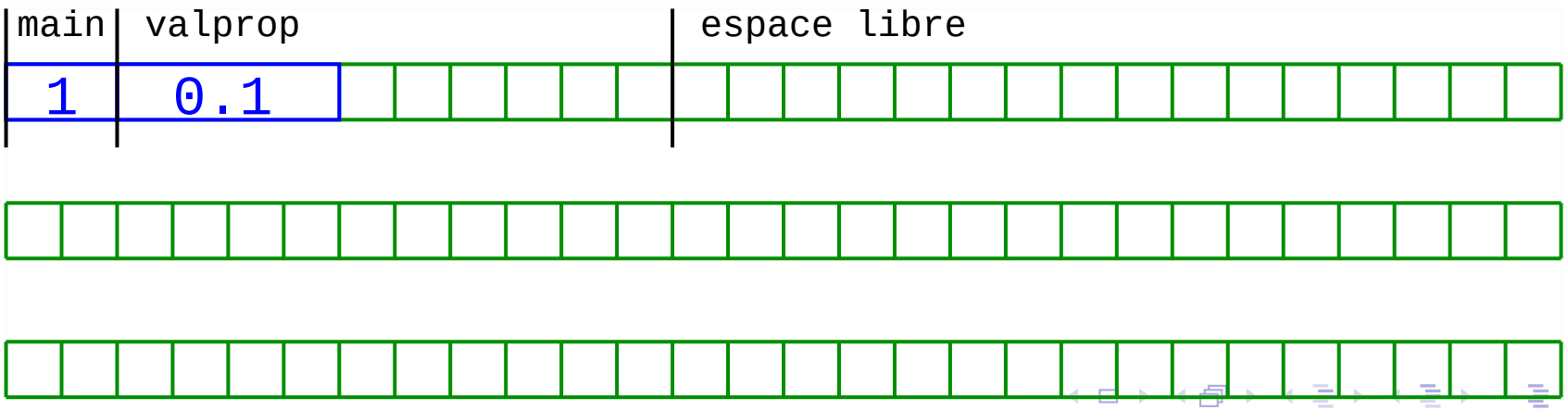
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



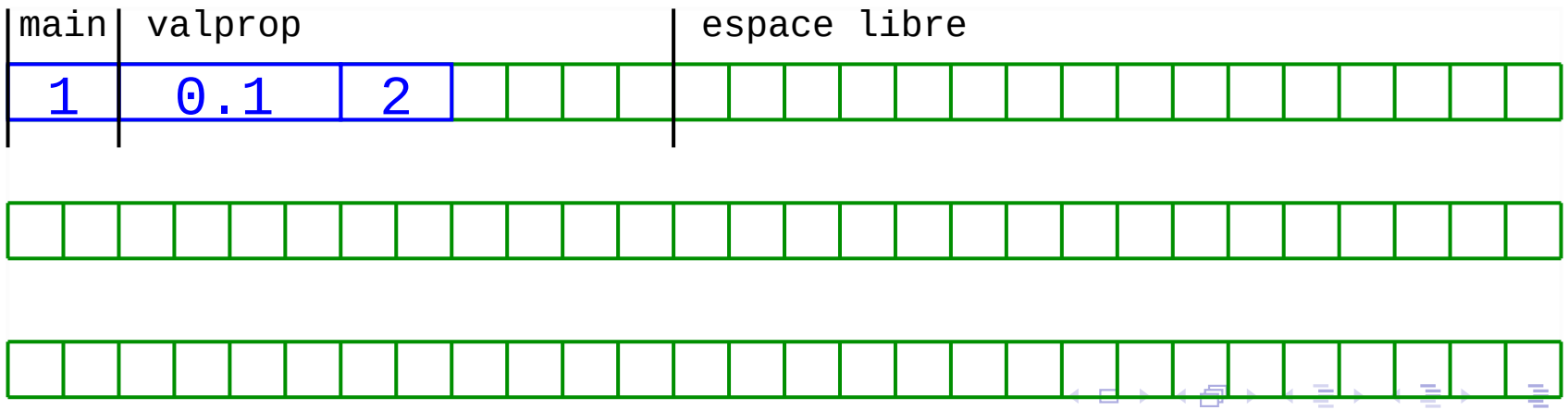
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



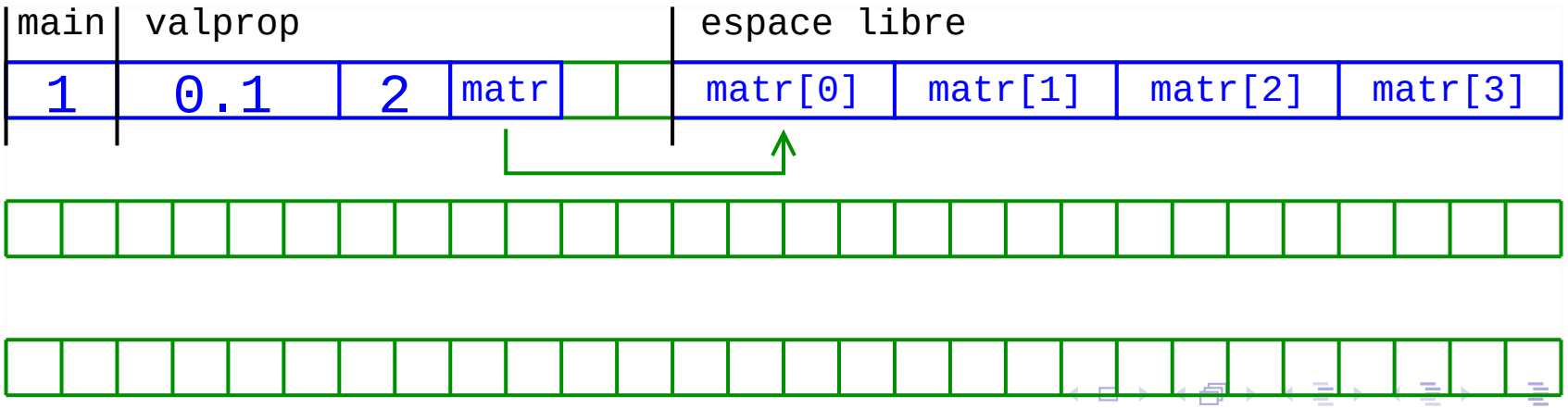
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



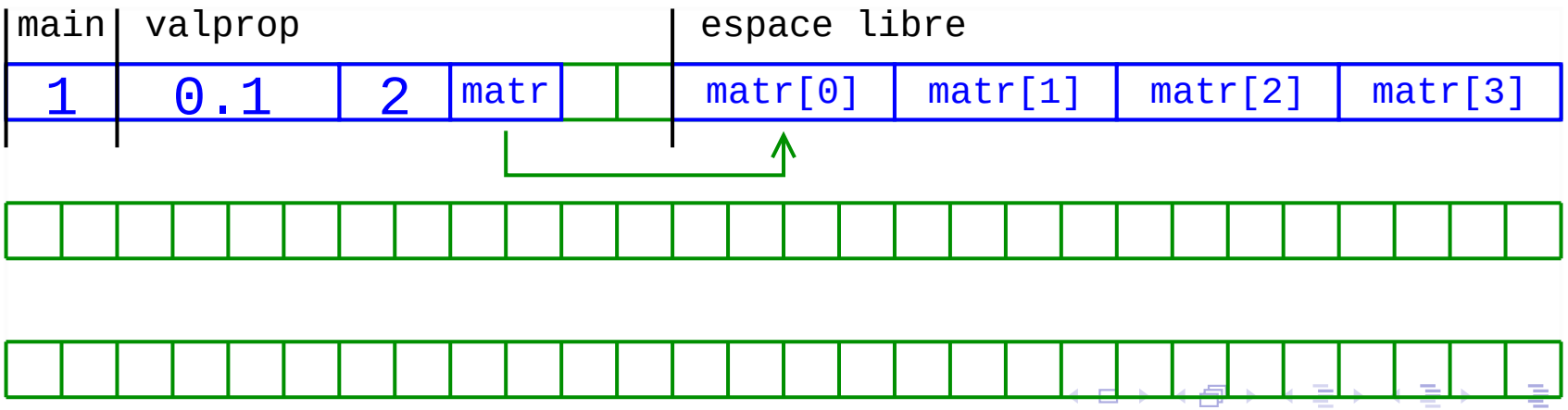
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



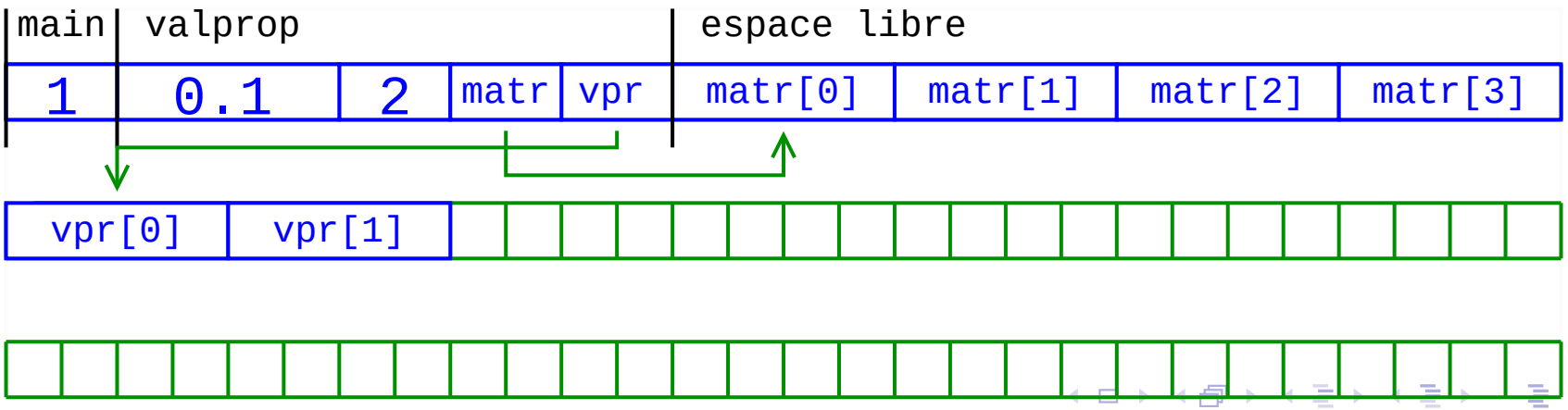
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



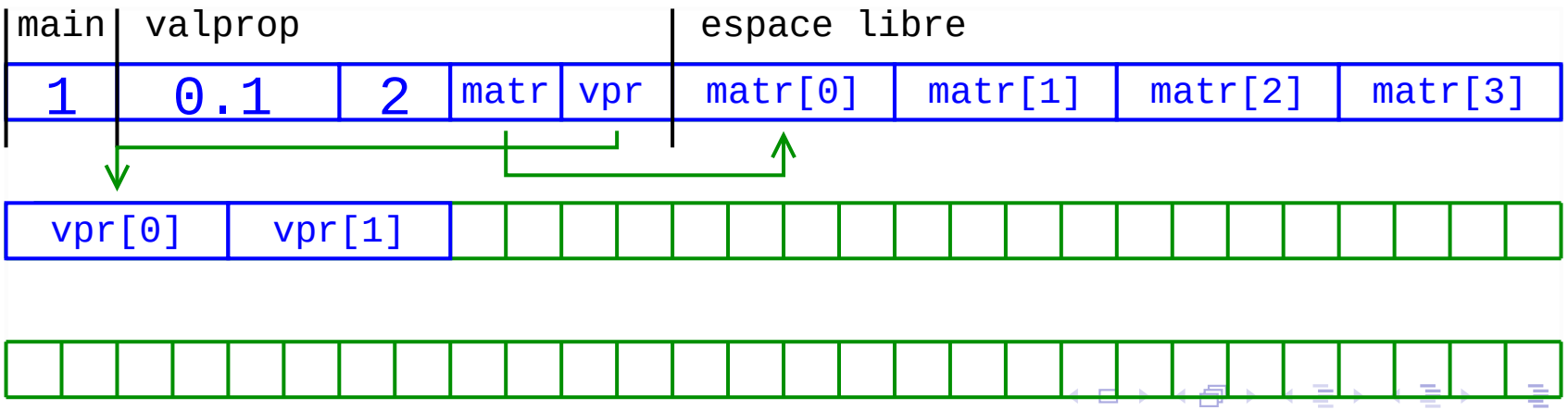
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



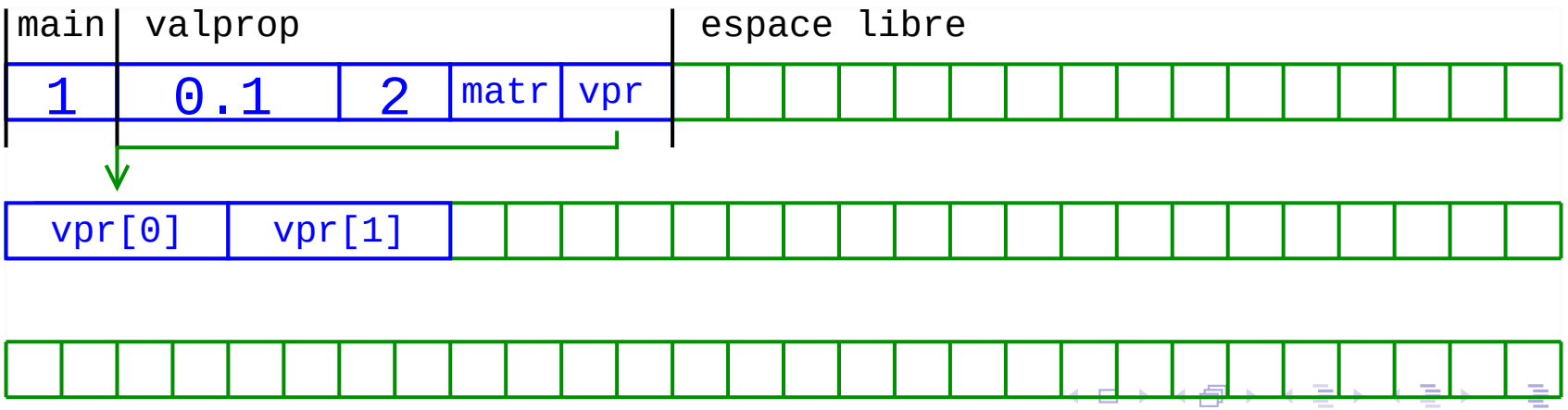
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



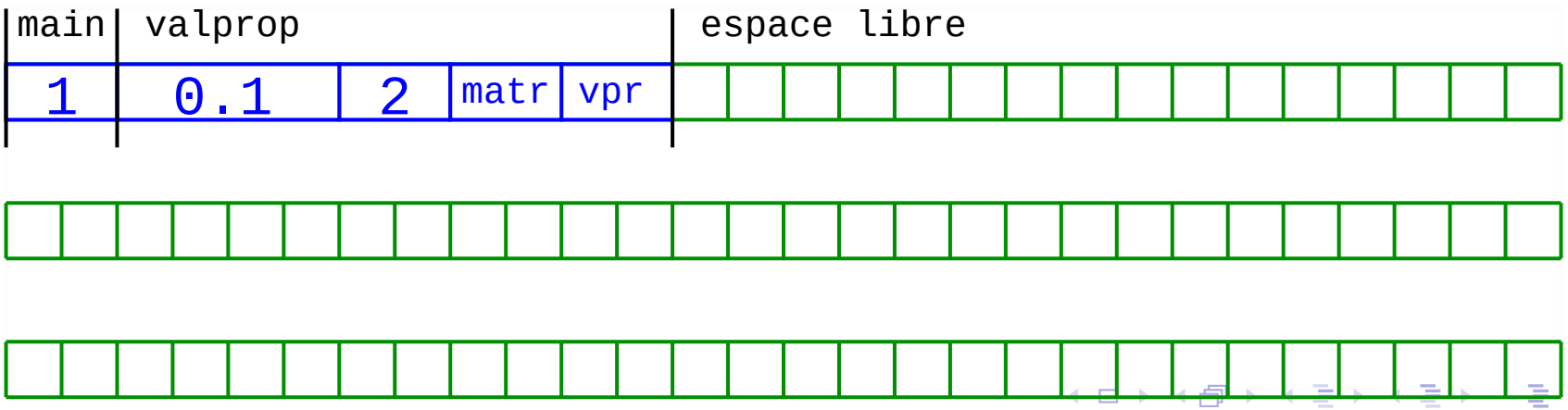
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



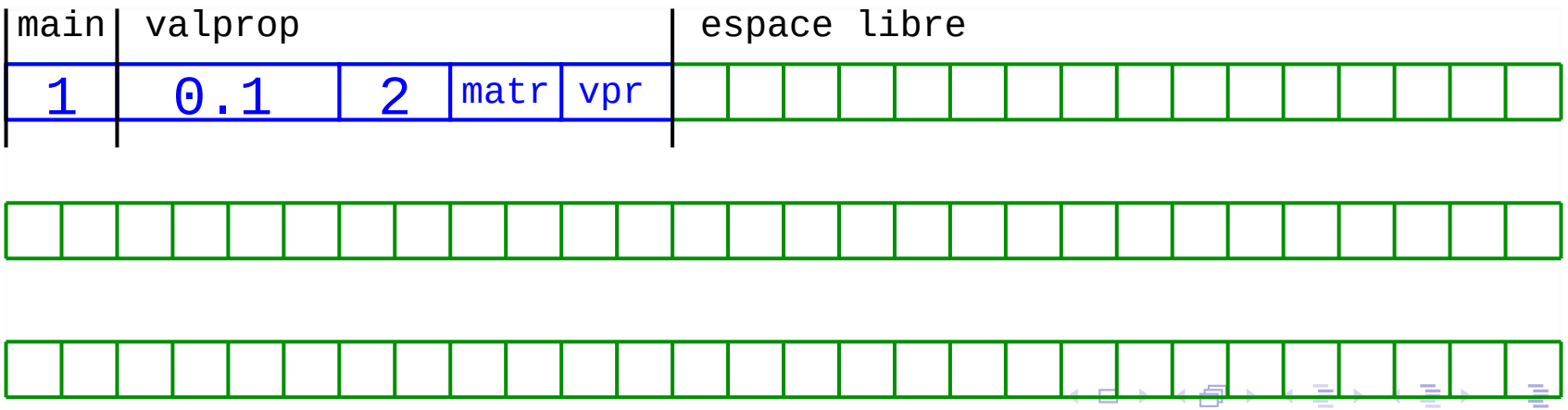
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



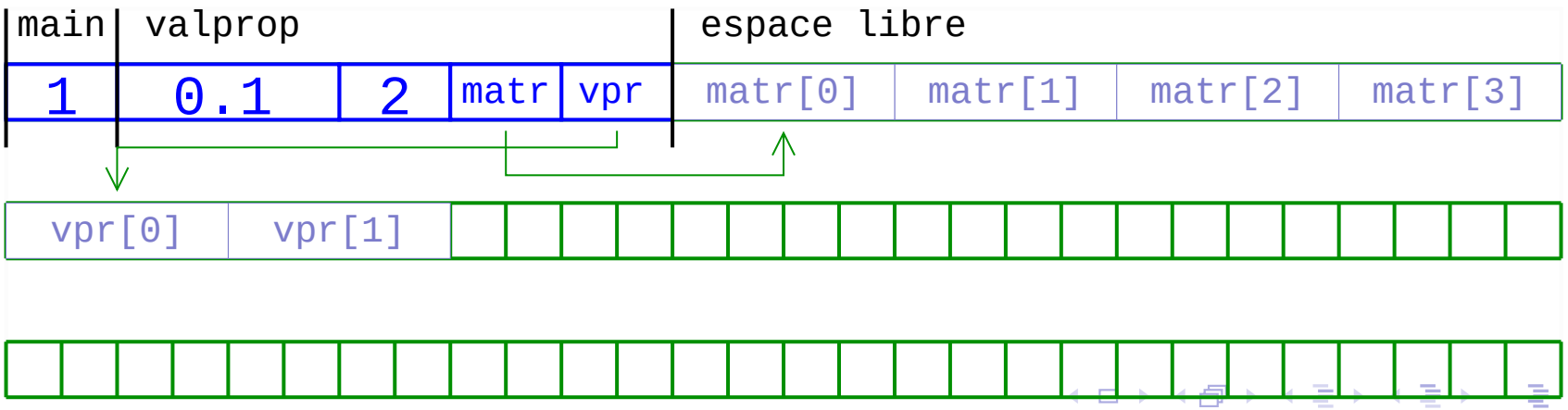
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



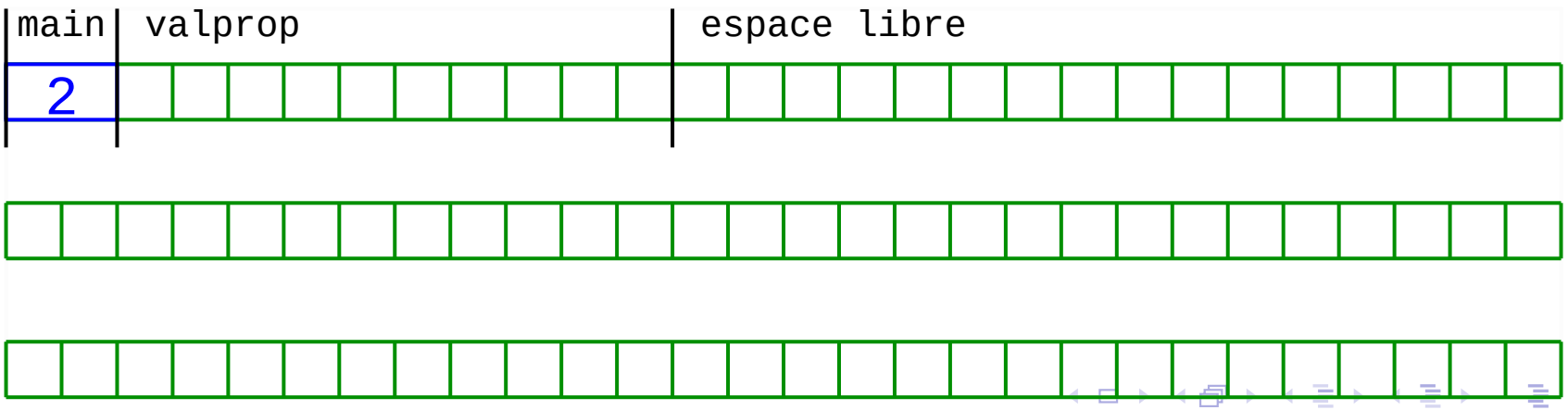
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



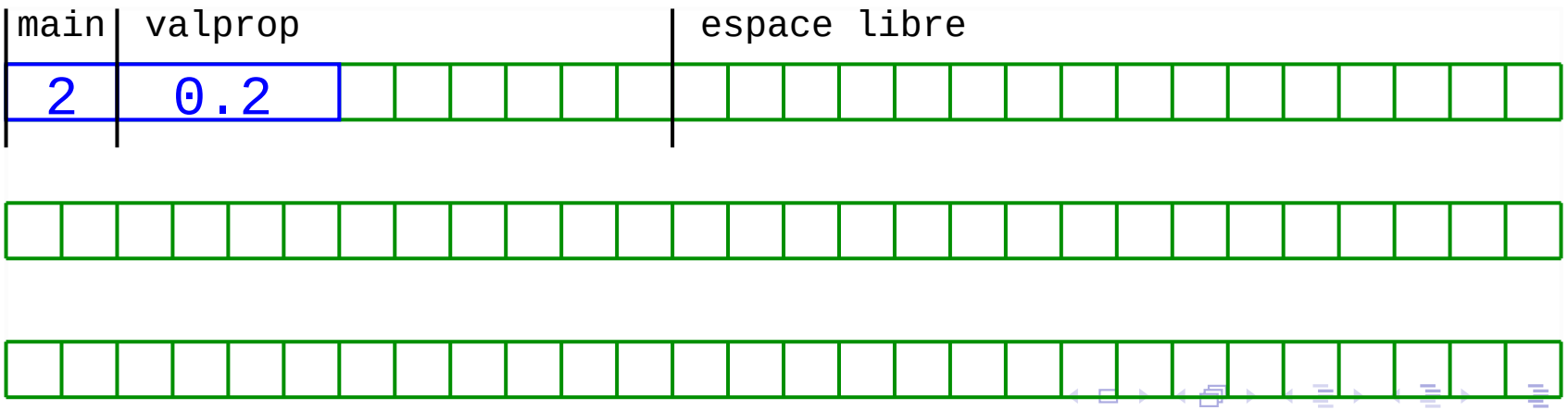
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



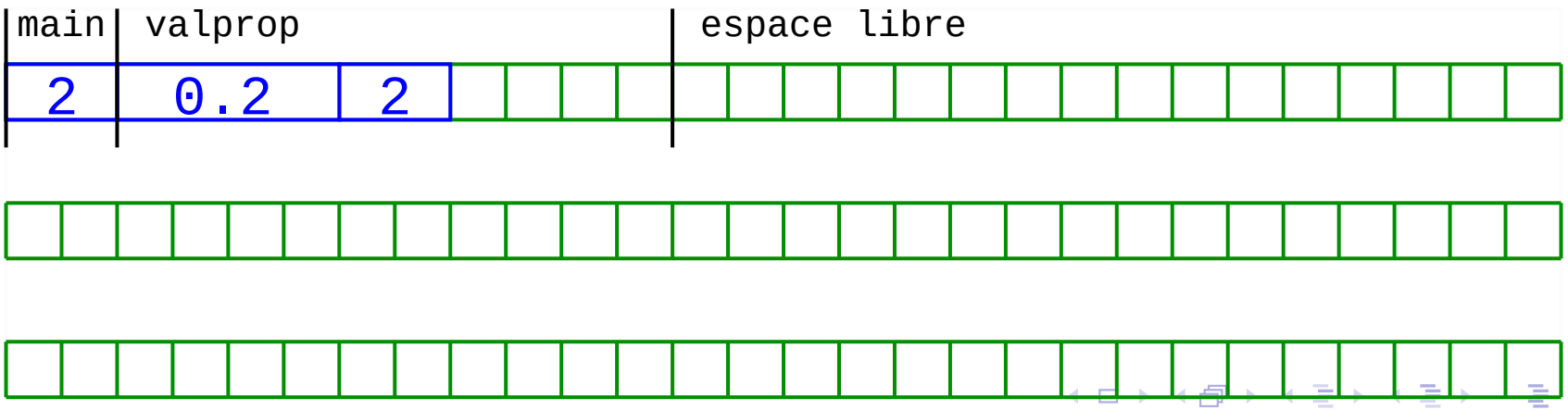
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



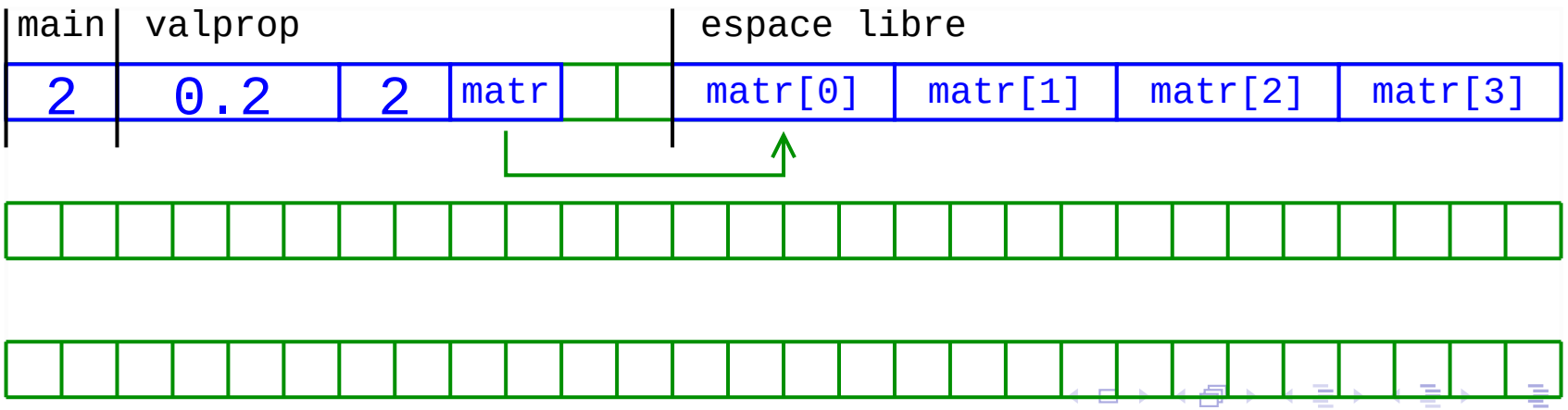
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



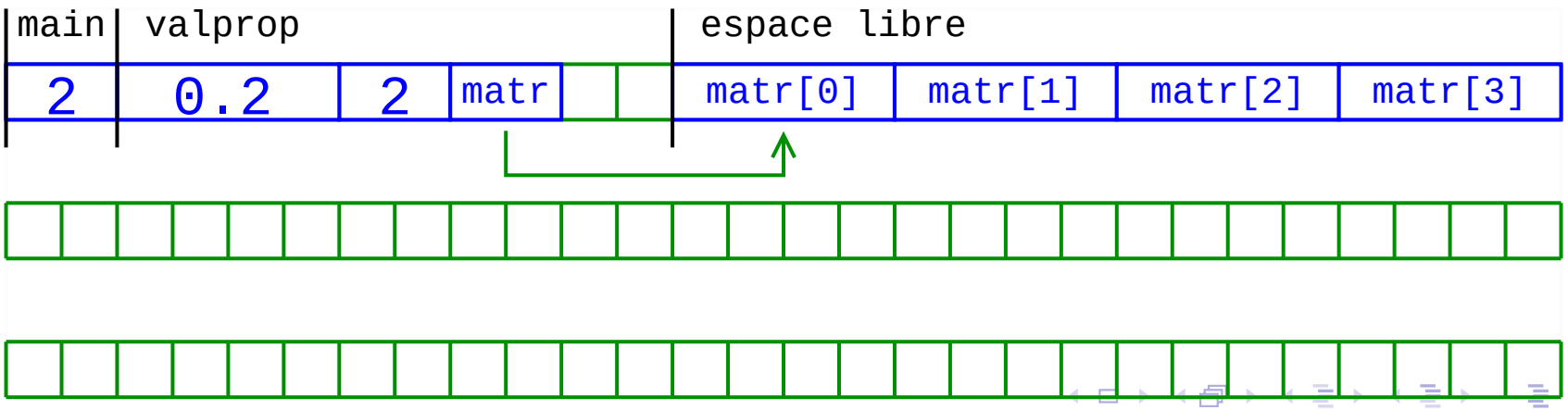
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



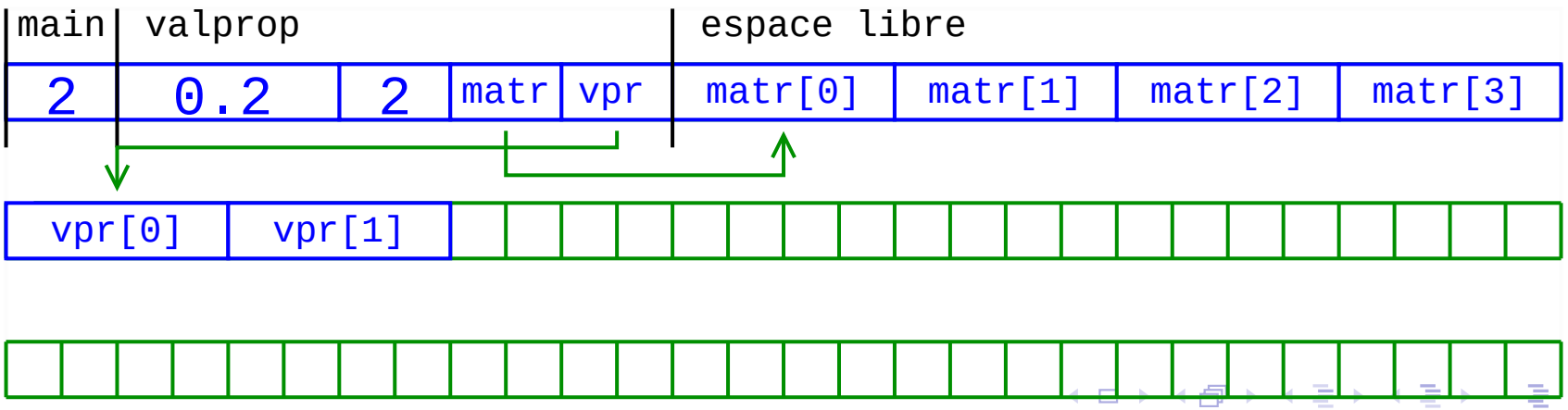
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



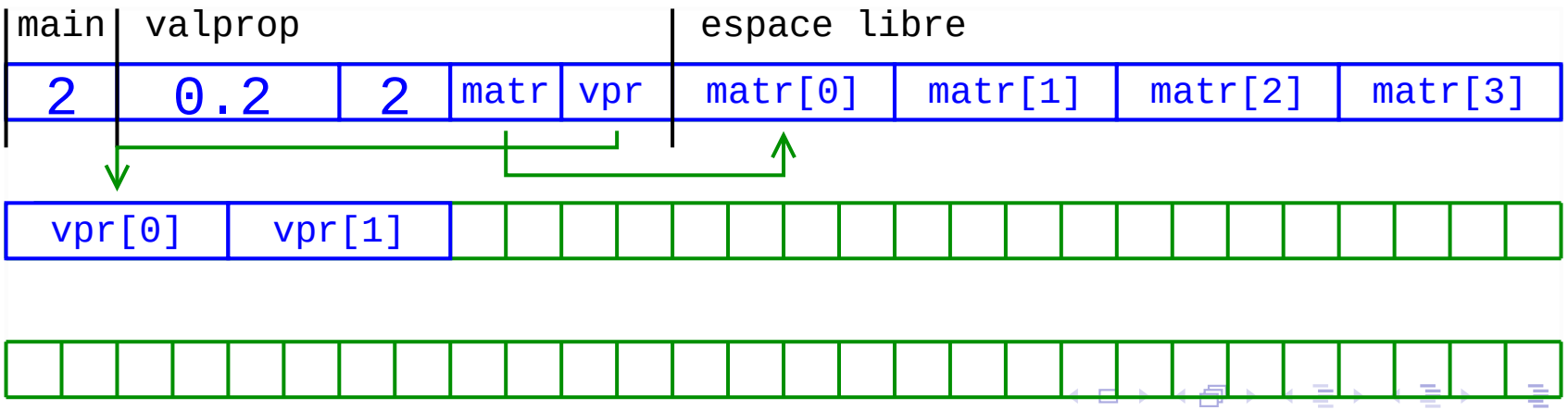
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



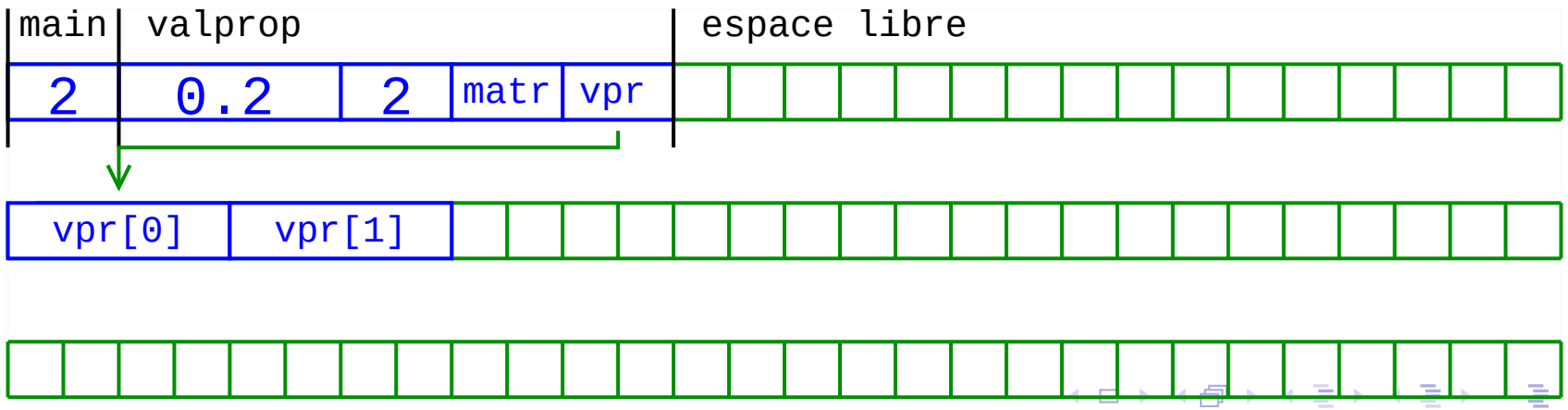
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



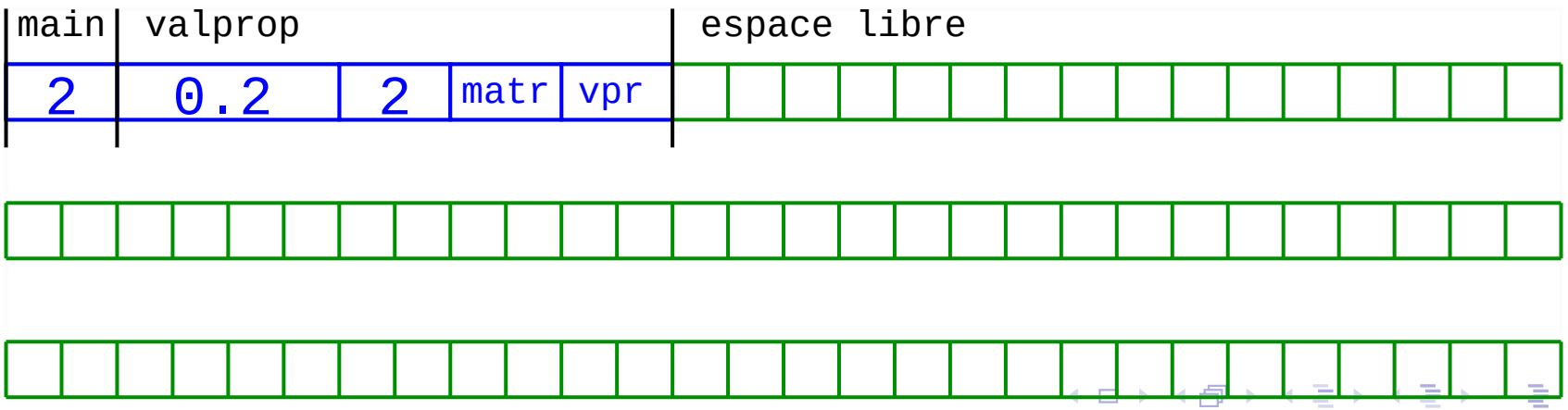
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



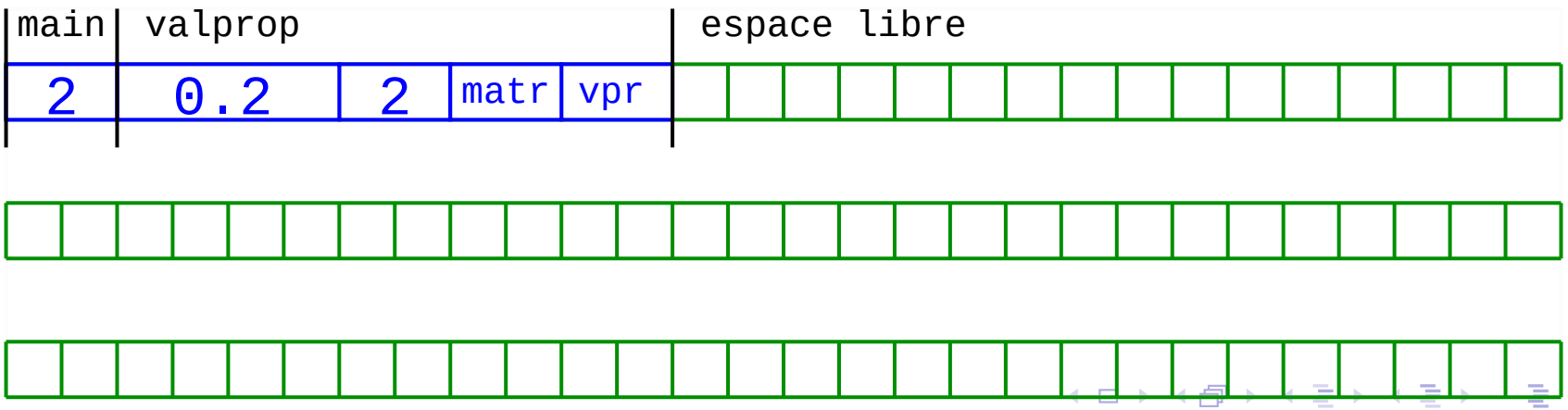
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



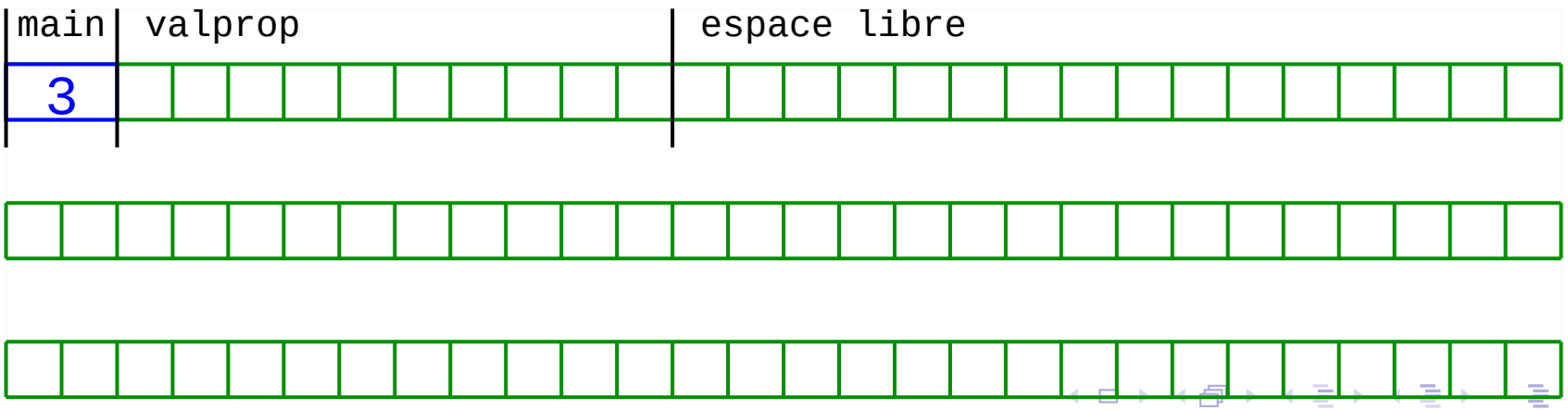
10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  delete[] matr;
  delete[] vpr;
  return ( vpr[ 0 ] );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```



10.3 Les tableaux dynamiques

```
#include <matrix.h>
#include <iostream>
using namespace std;

double valprop( double par )
{
  int N = taille_optimale( par );
  double *matr = new double[ N * N ];
  constr_matrix( N, matr, par );
  double *vpr = new double[ N ];
  diagonalise( N, matr, vpr );
  double valpr = vpr[ 0 ];
  delete[] matr;
  delete[] vpr;
  return ( valpr );
}

int main()
{
  for ( int i = 1; i <= 100; i++ )
    cout << valprop( i * 0.1 ) << endl;
}
```

Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```

`double **a;`
définit un pointeur sur un pointeur sur une variable `double`

Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

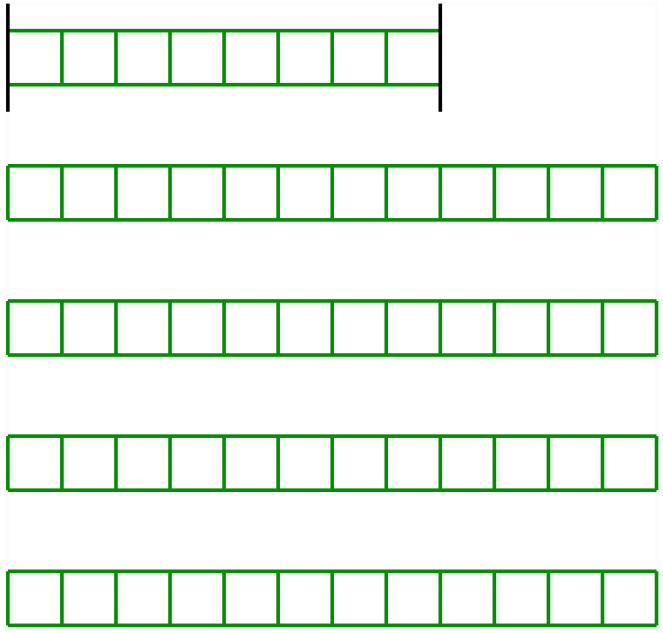
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```

`double **a = new double*[ N ];`
définit un tableau de pointeurs sur des variables `double` dans l'espace libre

Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

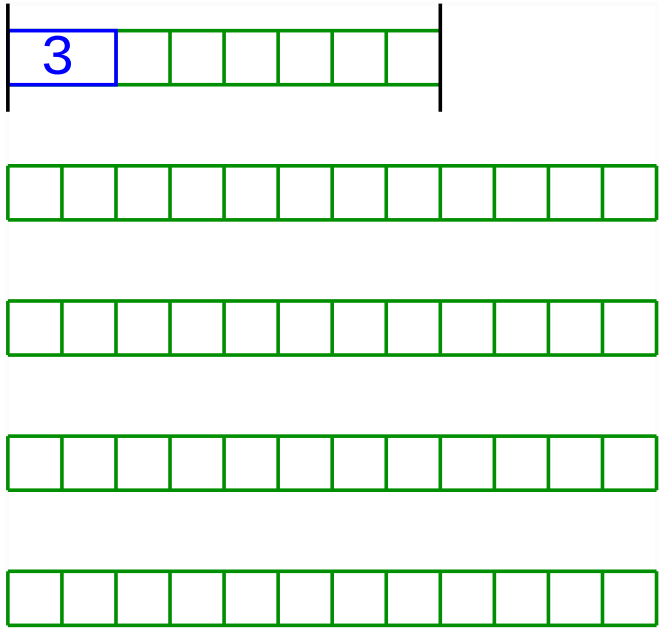
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

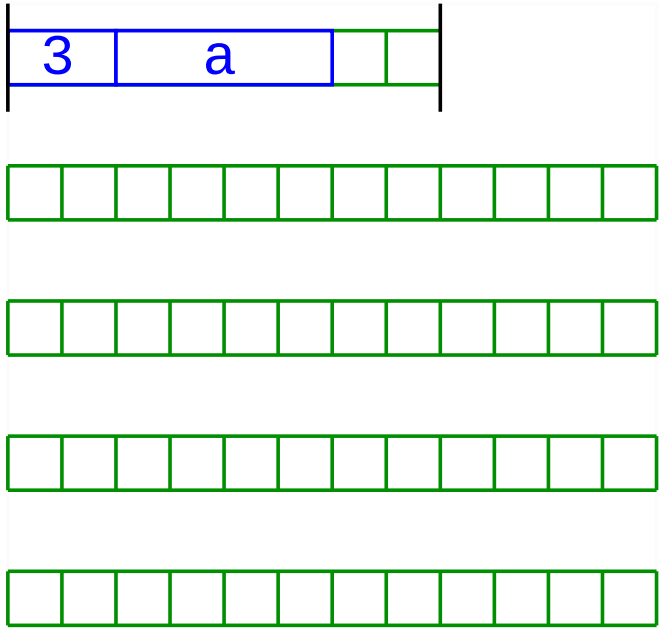
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

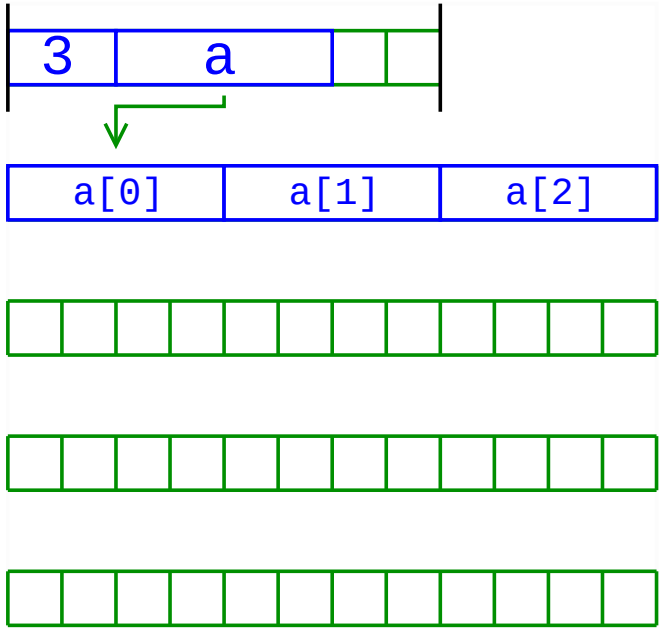
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

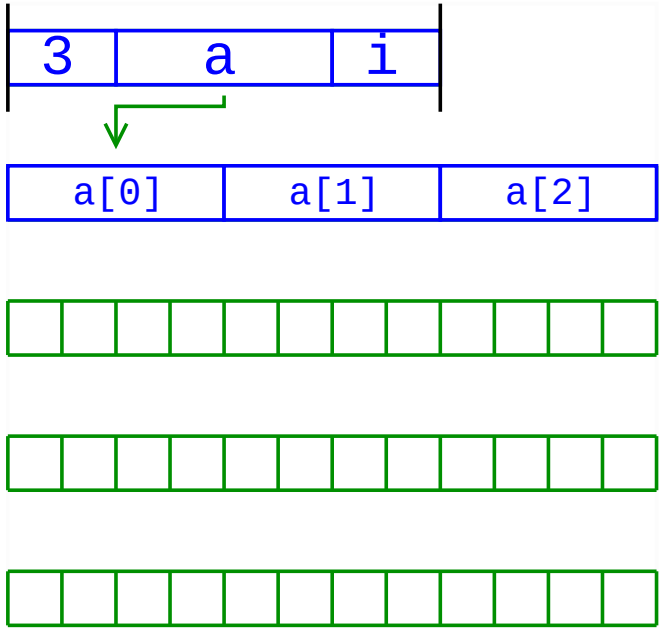
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

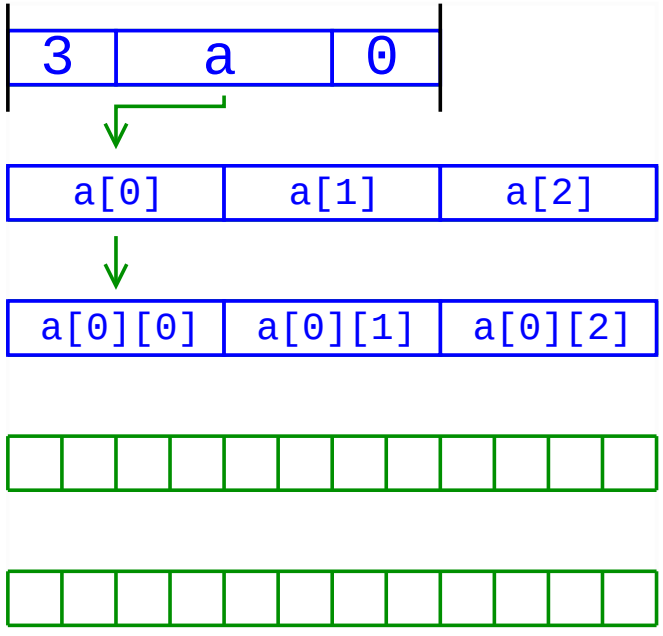
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

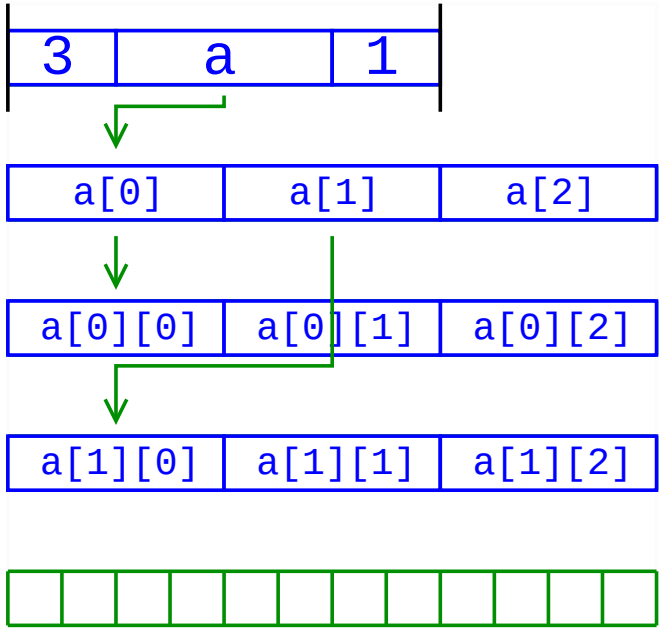
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

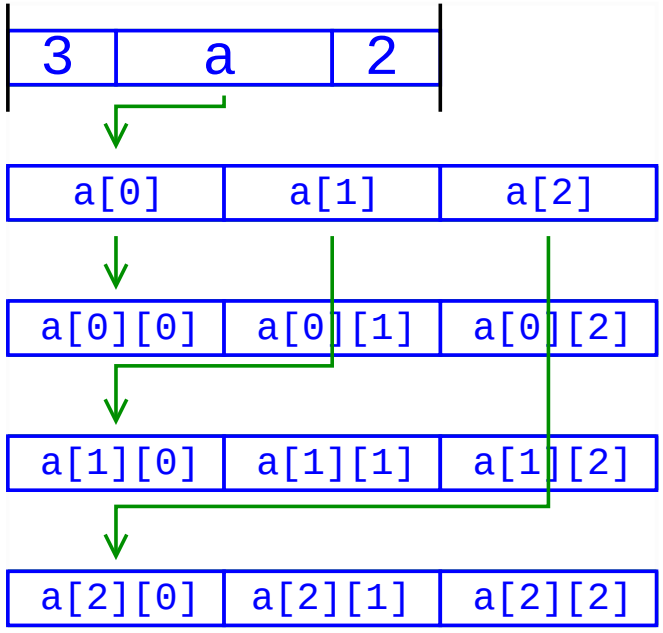
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

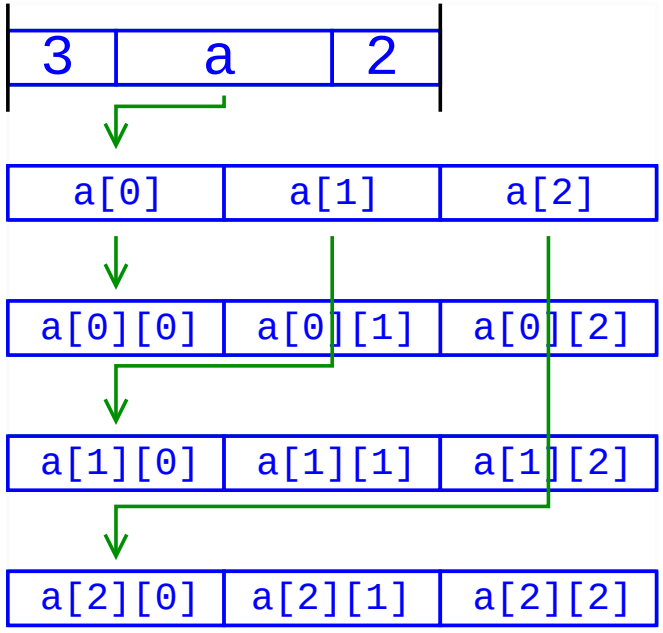
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

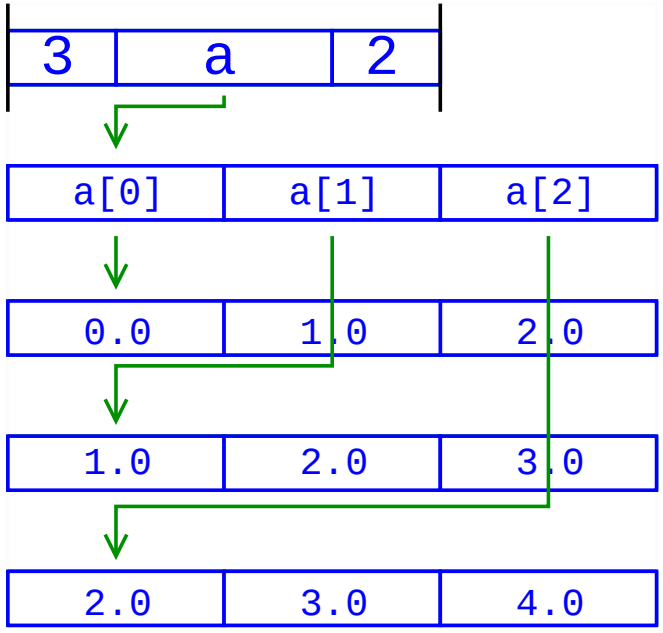
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j < N; j++ )
    a[ i ][ j ] = i + j;
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

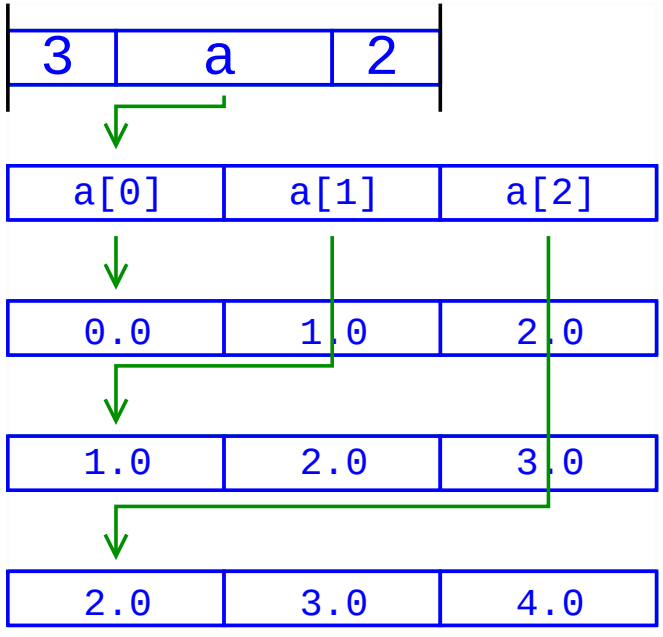
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j < N; j++ )
    a[ i ][ j ] = i + j;
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

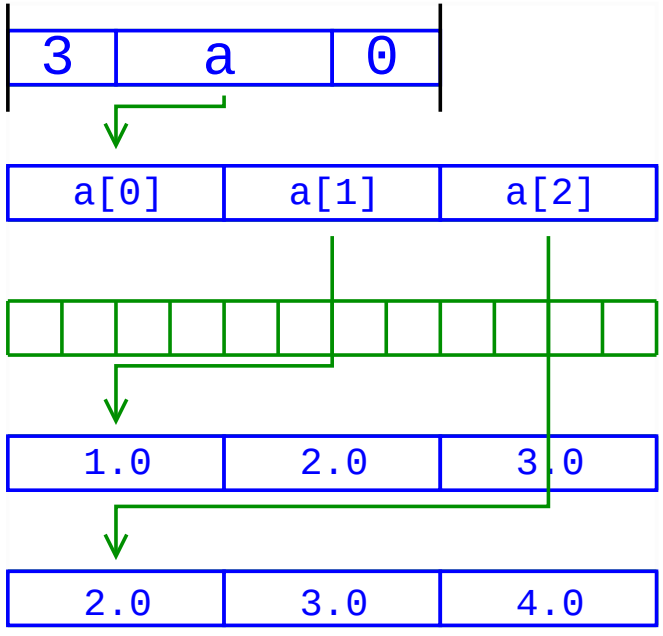
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j < N; j++ )
    a[ i ][ j ] = i + j;

for ( int i = 0; i < N; i++ )
  delete[] a[ i ];
delete[] a;
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

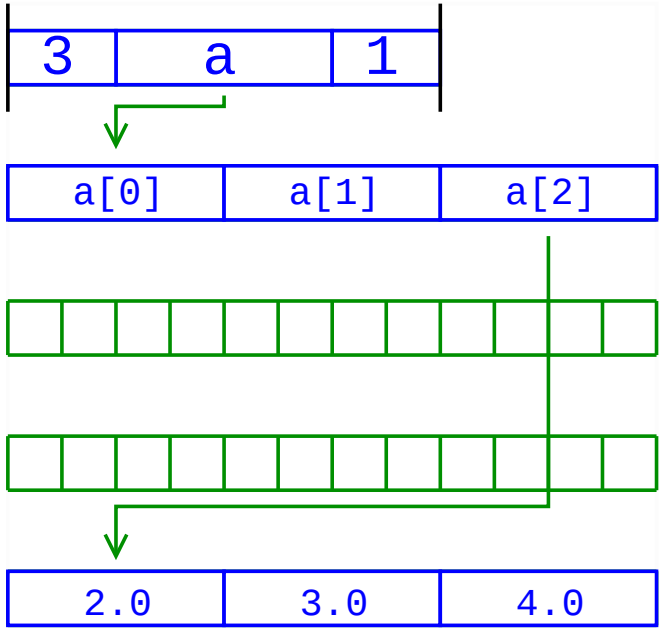
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j < N; j++ )
    a[ i ][ j ] = i + j;

for ( int i = 0; i < N; i++ )
  delete[] a[ i ];
delete[] a;
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

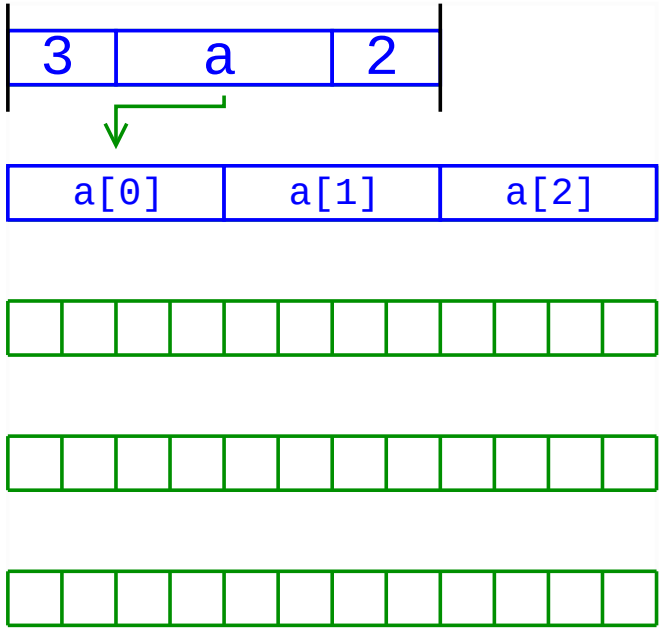
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j < N; j++ )
    a[ i ][ j ] = i + j;

for ( int i = 0; i < N; i++ )
  delete[] a[ i ];
delete[] a;
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

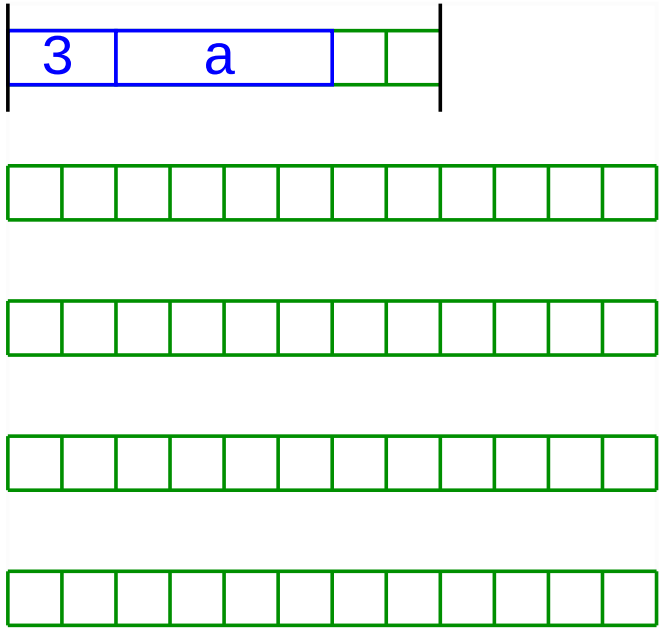
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j < N; j++ )
    a[ i ][ j ] = i + j;

for ( int i = 0; i < N; i++ )
  delete[] a[ i ];
delete[] a;
```



Les tableaux dynamiques multidimensionnels

Un tableau dynamique à deux dimensions peut être créé comme tableau de pointeurs dont chacun représente un tableau dynamique ordinaire.

```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j < N; j++ )
    a[ i ][ j ] = i + j;

for ( int i = 0; i < N; i++ )
  delete[] a[ i ];
delete[] a;
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

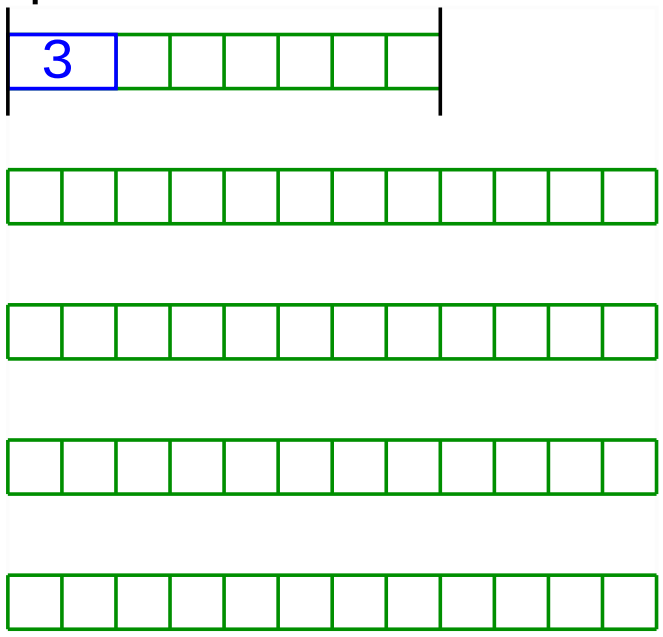
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$a_{ij} = a_{ji}$$

Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

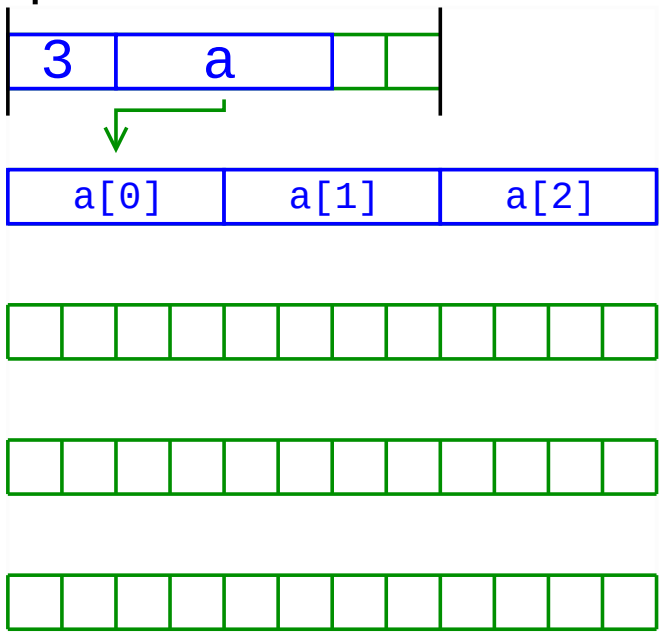
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

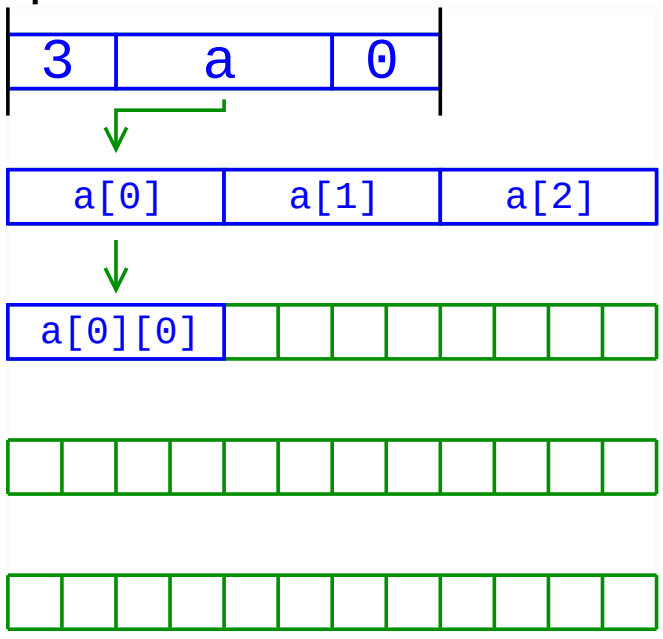
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

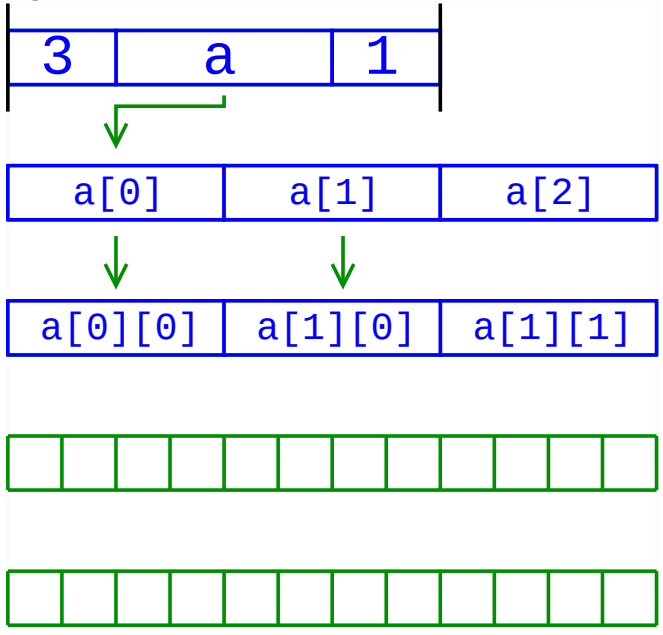
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

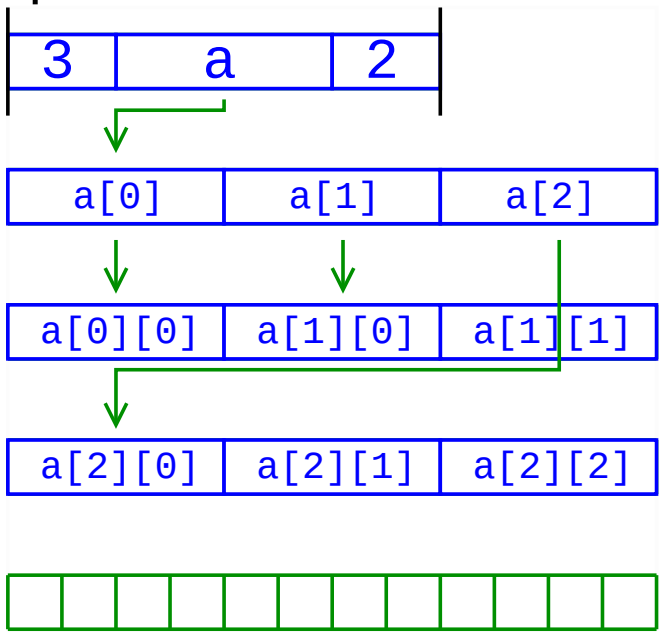
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

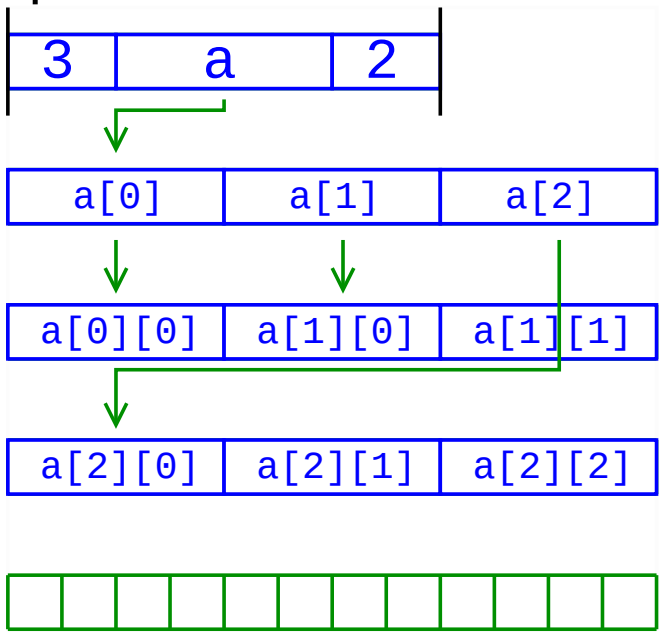
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

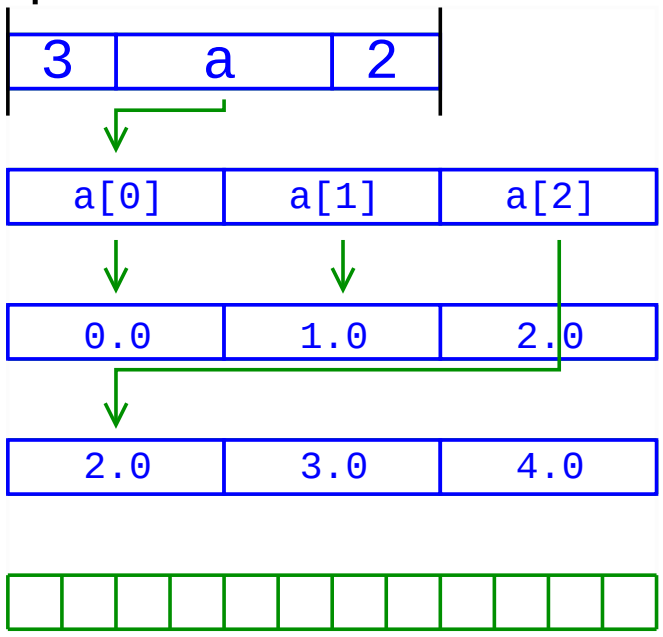
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j <= i; j++ )
    a[ i ][ j ] = i + j;
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

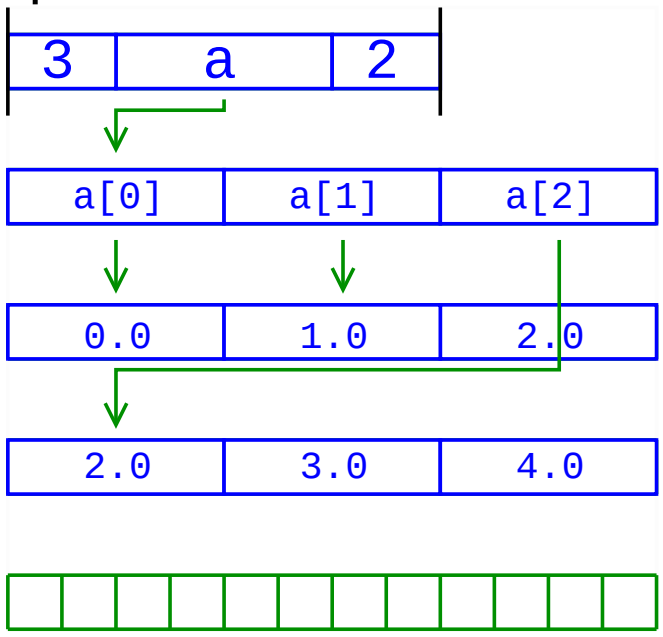
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j <= i; j++ )
    a[ i ][ j ] = i + j;
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

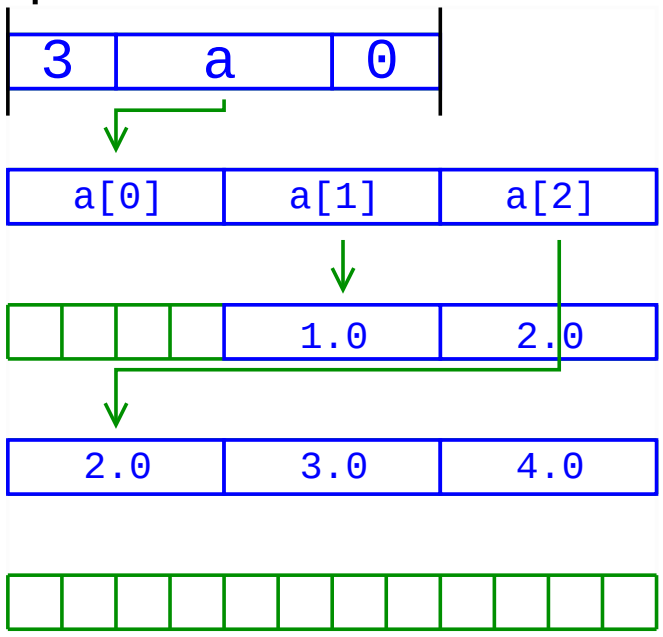
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j <= i; j++ )
    a[ i ][ j ] = i + j;

for ( int i = 0; i < N; i++ )
  delete[] a[ i ];
delete[] a;
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

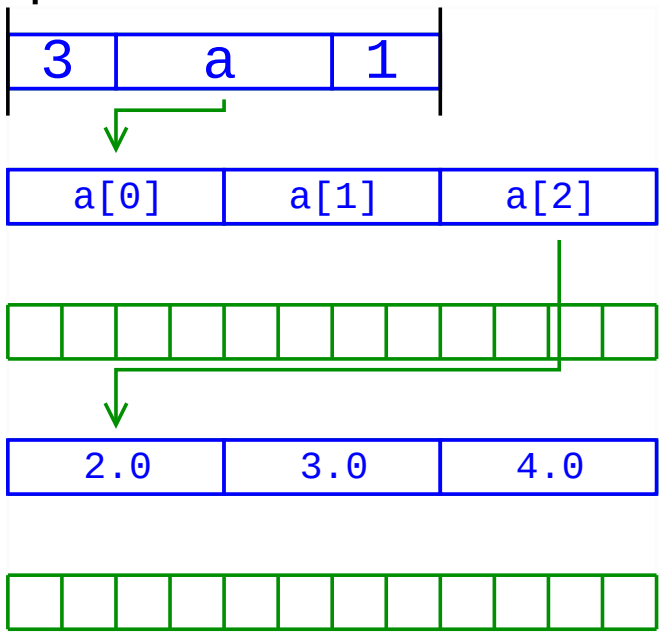
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j <= i; j++ )
    a[ i ][ j ] = i + j;

for ( int i = 0; i < N; i++ )
  delete[] a[ i ];
delete[] a;
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

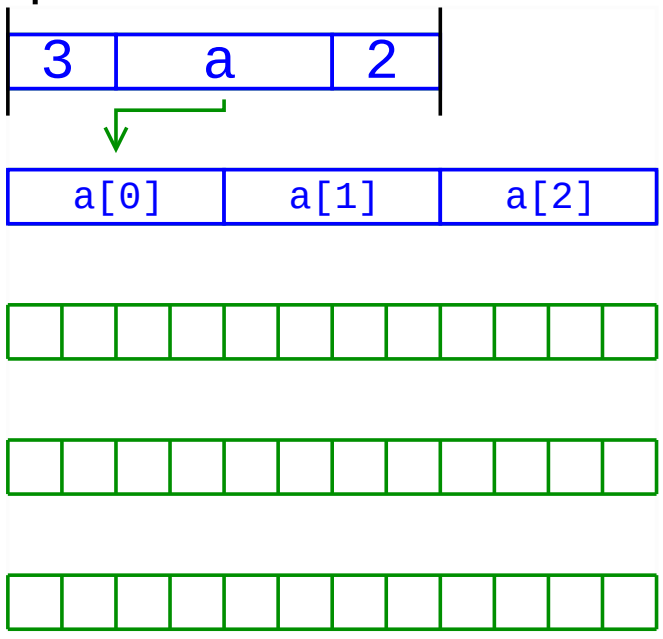
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j <= i; j++ )
    a[ i ][ j ] = i + j;

for ( int i = 0; i < N; i++ )
  delete[] a[ i ];
delete[] a;
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

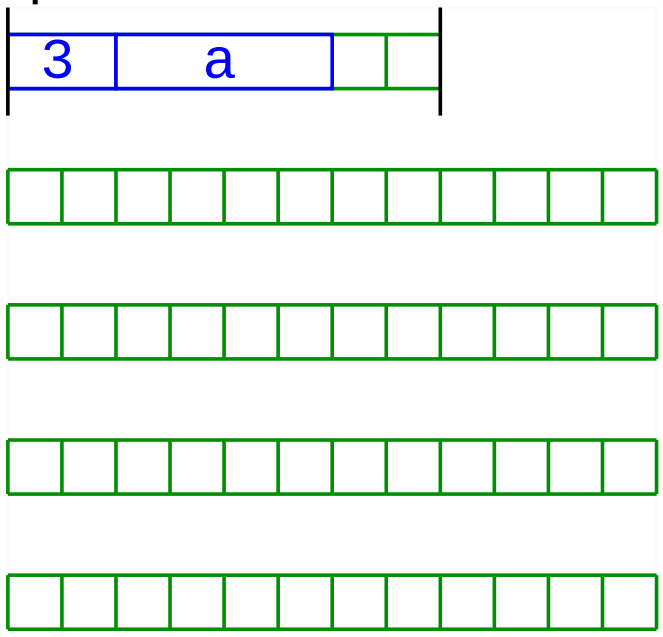
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j <= i; j++ )
    a[ i ][ j ] = i + j;

for ( int i = 0; i < N; i++ )
  delete[] a[ i ];
delete[] a;
```



Les tableaux dynamiques multidimensionnels

Comme ça, on peut même définir des tableaux inhomogènes, par exemple pour des matrices symétriques :

```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

for ( int i = 0; i < N; i++ )
  for ( int j = 0; j <= i; j++ )
    a[ i ][ j ] = i + j;

for ( int i = 0; i < N; i++ )
  delete[] a[ i ];
delete[] a;
```



Les tableaux dynamiques multidimensionnels

Des tableaux dynamiques multidimensionnels peuvent aussi être utilisés comme arguments des fonctions

Exemple: les arguments de la fonction `main`

Les tableaux dynamiques multidimensionnels

Des tableaux dynamiques multidimensionnels peuvent aussi être utilisés comme arguments des fonctions

Exemple: les arguments de la fonction `main`

Le programme `prog.cpp` :

```
#include <iostream>
using namespace std;

int main( int Narg, char **arg )
{
  for ( int i = 0; i < Narg; i++ )
    cout << arg[ i ] << endl;
}
```

```
c++ prog.cpp -o prog
./prog bonjour
prog
bonjour
```

Les tableaux dynamiques multidimensionnels

Des tableaux dynamiques multidimensionnels peuvent aussi être utilisés comme arguments des fonctions

Exemple: les arguments de la fonction `main`

Le programme `prog.cpp` :

```
#include <iostream>
using namespace std;

int main( int Narg, char **arg )
{
  for ( int i = 0; i < Narg; i++ )
    cout << arg[ i ] << endl;
}
```

```
c++ prog.cpp -o prog
./prog bonjour a tous
prog
bonjour
a
tous
```

Les tableaux dynamiques multidimensionnels

Des tableaux dynamiques multidimensionnels peuvent aussi être utilisés comme arguments des fonctions

Exemple: les arguments de la fonction `main`

Le programme `prog.cpp` :

```
#include <iostream>
using namespace std;

int main( int Narg, char **arg )
{
  for ( int i = 0; i < Narg; i++ )
    cout << arg[ i ] << endl;
}
```

$\longrightarrow$ `Narg` contient le nombre de strings de la ligne de commande (incluant le nom du programme lui-même)

$\longrightarrow$ `arg` contient les strings dans un tableau de caractères

Les tableaux dynamiques multidimensionnels

Des tableaux dynamiques multidimensionnels peuvent aussi être utilisés comme arguments des fonctions

Exemple: les arguments de la fonction `main`

Le programme `prog.cpp` :

```
#include <iostream>
using namespace std;

int main( int Narg, char **arg )
{
  for ( int i = 0; i < Narg; i++ )
    cout << arg[ i ] << endl;
}
```

$\longrightarrow$ on peut utiliser les fonctions `atoi` et `atof` (disponible avec `<cstdlib>`) pour convertir des strings `arg[i]` en entiers et flottants

Les tableaux dynamiques multidimensionnels

Des tableaux dynamiques multidimensionnels peuvent aussi être utilisés comme arguments des fonctions

Exemple: les arguments de la fonction `main`

Le programme `mult.cpp` :

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main( int Narg, char **arg )
{
  int n = atoi( arg[ 1 ] );
  double a = atof( arg[ 2 ] );
  cout << n * a << endl;
}
```

Les tableaux dynamiques multidimensionnels

Des tableaux dynamiques multidimensionnels peuvent aussi être utilisés comme arguments des fonctions

Exemple: les arguments de la fonction `main`

Le programme `mult.cpp` :

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main( int Narg, char **arg )
{
  if ( Narg < 3 )
    return 0;
  int n = atoi( arg[ 1 ] );
  double a = atof( arg[ 2 ] );
  cout << n * a << endl;
}
```

```
c++ mult.cpp -o mult
./mult 2 3.5
7
```