8 Les tableaux

Peter Schlagheck

Université de Liège

8 Les tableaux

8.1 Les tableaux conventionnels

8.2 Les strings

8.3 Les pointeurs

8.4 Les tableaux dynamiques

8.1 Les tableaux conventionnels

Souvent on veut appliquer une séquence d'opérations non seulement pour une seule variable, mais pour un ensemble de plusieurs variables équivalentes.

Exemple:

traitement des données de mésure contenues dans le fichier `donnees.dat`

$\longrightarrow$ lire le fichier `donnees.dat`

$\longrightarrow$ déterminer la donnée minimale

$\longrightarrow$ écrire les données modifiées (soustraction avec la valeur minimale) dans un autre fichier

`donnees.dat`:

```
53.98
48.58
55.28
62.85
```

8.1 Les tableaux conventionnels

Souvent on veut appliquer une séquence d'opérations non seulement pour une seule variable, mais pour un ensemble de plusieurs variables équivalentes.

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

int main()
{
  ifstream inp( "donnees.dat" );
  double a1, a2, a3, a4;
  inp >> a1 >> a2 >> a3 >> a4;
  double amin = a1;
  if ( a2 < amin )
    amin = a2;
  if ( a3 < amin )
    amin = a3;
  if ( a4 < amin )
    amin = a4;
  ofstream out( "donnees2.dat" );
  out << a1 - amin << endl << a2 - amin << endl
      << a3 - amin << endl << a4 - amin << endl;
}
```

$\longrightarrow$ difficile de généraliser ce programme pour des données plus nombreuses ...

donnees.dat:

```
53.98
48.58
55.28
62.85
```

donnees2.dat:

```
5.4
0
6.7
14.27
```

8.1 Les tableaux conventionnels

Souvent on veut appliquer une séquence d'opérations non seulement pour une seule variable, mais pour un ensemble de plusieurs variables équivalentes.

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

int main()
{
  ifstream inp( "donnees.dat" );
  double a1, a2, a3, a4;
  inp >> a1 >> a2 >> a3 >> a4;
  double amin = a1;
  if ( a2 < amin )
    amin = a2;
  if ( a3 < amin )
    amin = a3;
  if ( a4 < amin )
    amin = a4;
  ofstream out( "donnees2.dat" );
  out << a1 - amin << endl << a2 - amin << endl
      << a3 - amin << endl << a4 - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
63.00
56.17
52.77
54.65
60.65
54.60
```

$\longrightarrow$ difficile de généraliser ce programme pour des données plus nombreuses . . .

8.1 Les tableaux conventionnels

Souvent on veut appliquer une séquence d'opérations non seulement pour une seule variable, mais pour un ensemble de plusieurs variables équivalentes.

$\longrightarrow$ utilisation des tableaux :

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

int main()
{
  ifstream inp( "donnees.dat" );
  double a[ 10 ];
  for ( int i = 0; i < 10; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < 10; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < 10; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
63.00
56.17
52.77
54.65
60.65
54.60
```

8.1 Les tableaux conventionnels

Souvent on veut appliquer une séquence d'opérations non seulement pour une seule variable, mais pour un ensemble de plusieurs variables équivalentes.

$\longrightarrow$ utilisation des tableaux :

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

int main()
{
  ifstream inp( "donnees.dat" );
  double a[ 10 ];
  for ( int i = 0; i < 10; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < 10; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < 10; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees2.dat:

```
5.4
0
6.7
14.27
14.42
7.59
4.19
6.07
12.07
6.02
```

8.1 Les tableaux conventionnels

Souvent on veut appliquer une séquence d'opérations non seulement pour une seule variable, mais pour un ensemble de plusieurs variables équivalentes.

$\longrightarrow$ utilisation des tableaux :

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

int main()
{
  const int N = 10;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees2.dat:

```
5.4
0
6.7
14.27
14.42
7.59
4.19
6.07
12.07
6.02
```

8.1 Les tableaux conventionnels

Souvent on veut appliquer une séquence d'opérations non seulement pour une seule variable, mais pour un ensemble de plusieurs variables équivalentes.

$\longrightarrow$ utilisation des tableaux :

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

int main()
{
  const int N = 20;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees2.dat:

```
5.4
0
6.7
14.27
14.42
7.59
4.19
6.07
12.07
6.02
```

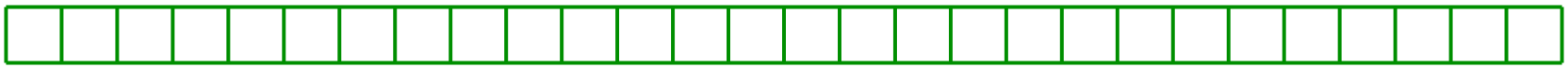
8.1 Les tableaux conventionnels

Le flot du programme dans la mémoire:

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```



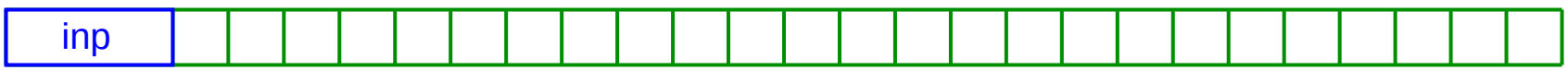
8.1 Les tableaux conventionnels

Le flot du programme dans la mémoire:

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```



8.1 Les tableaux conventionnels

Le flot du programme dans la mémoire:

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```

8.1 Les tableaux conventionnels

Le flot du programme dans la mémoire:

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```

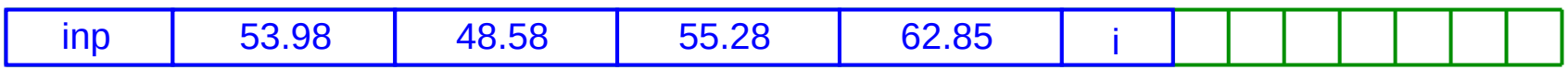
8.1 Les tableaux conventionnels

Le flot du programme dans la mémoire:

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```



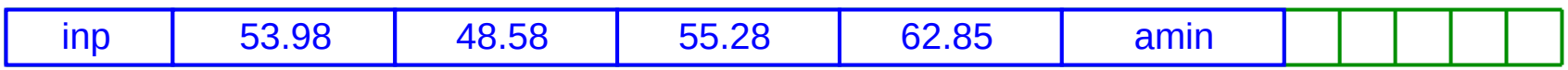
8.1 Les tableaux conventionnels

Le flot du programme dans la mémoire:

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```



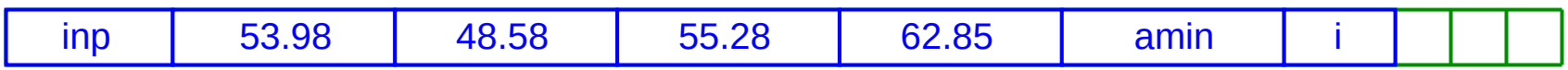
8.1 Les tableaux conventionnels

Le flot du programme dans la mémoire:

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```



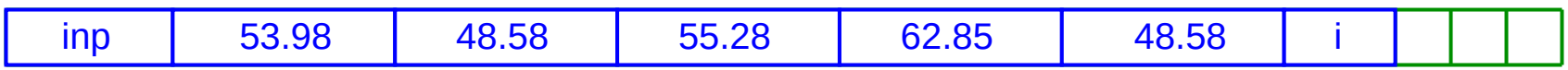
8.1 Les tableaux conventionnels

Le flot du programme dans la mémoire:

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```



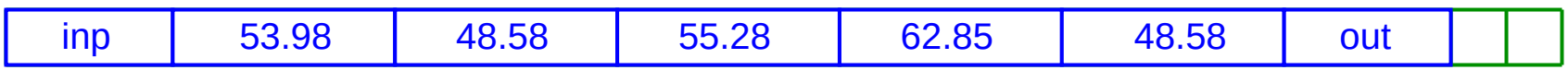
8.1 Les tableaux conventionnels

Le flot du programme dans la mémoire:

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```



8.1 Les tableaux conventionnels

Le flot du programme dans la mémoire:

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  double amin = a[ 0 ];
  for ( int i = 1; i < N; i++ )
    if ( a[ i ] < amin )
      amin = a[ i ];
  ofstream out( "donnees2.dat" );
  for ( int i = 0; i < N; i++ )
    out << a[ i ] - amin << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```

inp	53.98	48.58	55.28	62.85	48.58	out	i

8.1 Les tableaux conventionnels

⟨type⟩ ⟨nom⟩ [⟨taille⟩];

⟶ définition d'un tableau nommé ⟨nom⟩
contenant ⟨taille⟩ variables du type ⟨type⟩

⟶ ⟨taille⟩ doit être une expression entière constante

Accès aux éléments du tableau:

⟨nom⟩ [⟨expression entière⟩]

⟶ ça rend l'élément numéro ⟨expression entière⟩+1
du tableau ⟨nom⟩

8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  cout << a[ 0 ] << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```

Exécution:

```
53.98
```

8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  cout << a[ 3 ] << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```

Exécution:

```
62.85
```

8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  for ( int i = 0; i < N; i++ )
    cout << a[ i + 1 ] << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```

Exécution:

```
48.58
55.28
62.85
1.00768e-313
```

8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  for ( int i = 0; i < N; i++ )
    cout << a[ i - 1 ] << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```

Exécution:

```
-4.90985e-39
53.98
48.58
55.28
```

8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  cout << a[ 4 ] << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```

Exécution:

```
1.00768e-313
```

8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  cout << a[ -1 ] << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```

Exécution:

```
-4.90985e-39
```

8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 4;
  ifstream inp( "donnees.dat" );
  double a[ N ];
  for ( int i = 0; i < N; i++ )
    inp >> a[ i ];
  cout << a[ -2 ] << endl;
}
```

donnees.dat:

```
53.98
48.58
55.28
62.85
```

Exécution:

```
-3.0138e-39
```

$\longrightarrow$ valeur aléatoire (sans message d'erreur)
si l'indice dépasse la taille du tableau

$\longrightarrow$ vous risquez de manipuler d'autres variables
si vous ne faites pas attention avec les indices !

8.1 Les tableaux conventionnels

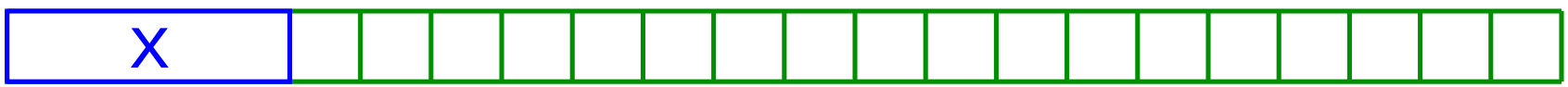
```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```

Exécution:

```
2.3 1
```

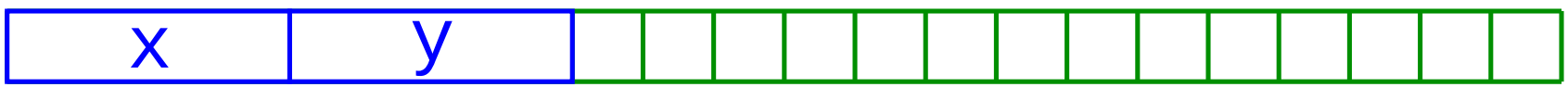
8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```



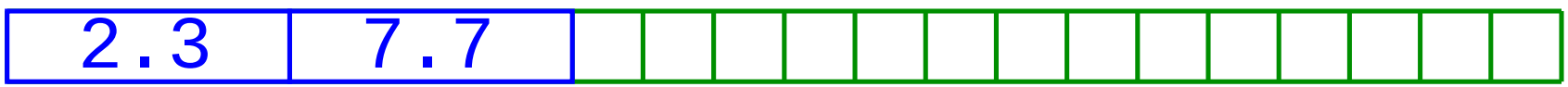
8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```



8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```



8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```

8.1 Les tableaux conventionnels

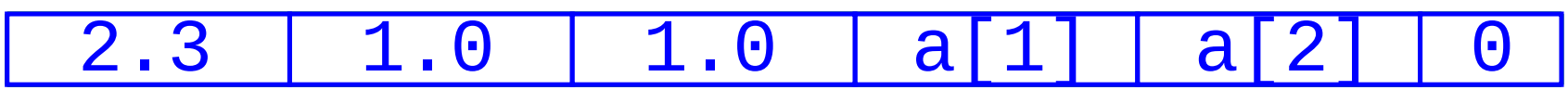
```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```

8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```

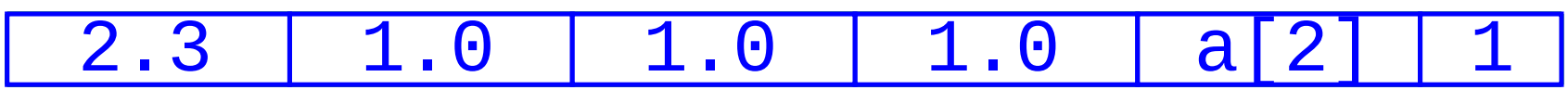
8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```



8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```



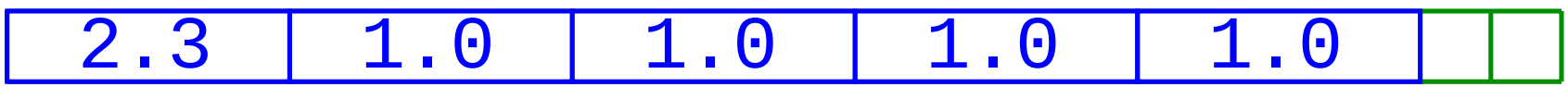
8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```

2.3	1.0	1.0	1.0	1.0	2

8.1 Les tableaux conventionnels

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

int main()
{
  const int N = 3;
  double x = 2.3;
  double y = 7.7;
  double a[ N ];
  for ( int i = -1; i < N; i++ )
    a[ i ] = 1.0;
  cout << x << " " << y << endl;
}
```



Initialisation des tableaux

⟨type⟩ ⟨nom⟩`[` ⟨taille⟩ `]` = { ⟨valeur_0⟩`,` ⟨valeur_1⟩`,` `...` };

⟶ ⟨nom⟩`[0]` est initialisé avec la valeur ⟨valeur_0⟩
⟨nom⟩`[1]` est initialisé avec la valeur ⟨valeur_1⟩
...

Initialisation des tableaux

⟨type⟩ ⟨nom⟩`[` ⟨taille⟩ `]` = { ⟨valeur_0⟩, ⟨valeur_1⟩, ... };

⟶ ⟨valeur_0⟩, ⟨valeur_1⟩ ... doivent être des valeurs du type (ou convertibles en) ⟨type⟩

⟶ le nombre des valeurs spécifiées dans l'initialisation ne doit pas dépasser la taille du tableau (mais il peut bien être inférieur à elle)

```
const int n = 3;
double a[ n ] = { 1.2, 3, -0.1 };
for ( int i = 0; i < n; i++ )
  cout << a[ i ] << endl;
```

Exécution:

```
1.2
3
-0.1
```

Initialisation des tableaux

⟨type⟩ ⟨nom⟩`[` ⟨taille⟩ `]` = `{` ⟨valeur_0⟩`,` ⟨valeur_1⟩`,` `...` `};`

⟶ ⟨valeur_0⟩, ⟨valeur_1⟩ ...doivent être des valeurs du type (ou convertibles en) ⟨type⟩

⟶ le nombre des valeurs spécifiées dans l'initialisation ne doit pas dépasser la taille du tableau (mais il peut bien être inférieur à elle)

```
const int n = 3;
double a[ n ] = { 1.2, 3 };
for ( int i = 0; i < n; i++ )
  cout << a[ i ] << endl;
```

Exécution:

```
1.2
3
0
```

Initialisation des tableaux

⟨type⟩ ⟨nom⟩`[` ⟨taille⟩ `]` = { ⟨valeur_0⟩, ⟨valeur_1⟩, ... };

⟶ ⟨valeur_0⟩, ⟨valeur_1⟩ ... doivent être des valeurs du type (ou convertibles en) ⟨type⟩

⟶ le nombre des valeurs spécifiées dans l'initialisation ne doit pas dépasser la taille du tableau (mais il peut bien être inférieur à elle)

```
const int n = 3;
double a[ n ] = { 1.2, 3, -0.1, 4.1 };
for ( int i = 0; i < n; i++ )
  cout << a[ i ] << endl;
```

⟶ `erreur: too many initializers for 'double [3]'`

Initialisation des tableaux

⟨type⟩ ⟨nom⟩`[` ⟨taille⟩ `]` = { ⟨valeur_0⟩`,` ⟨valeur_1⟩`,` `...` };

⟶ ⟨valeur_0⟩, ⟨valeur_1⟩ ...doivent être des valeurs du type (ou convertibles en) ⟨type⟩

⟶ le nombre des valeurs spécifiées dans l'initialisation ne doit pas dépasser la taille du tableau (mais il peut bien être inférieur à elle)

```
const int n = 3;
double a[ n ] = { 1.2, 3, -0.1 };
double b[ n ] = a;
for ( int i = 0; i < n; i++ )
  cout << b[ i ] << endl;
```

⟶ `erreur: array must be initialized with a brace-enclosed initializer`

Initialisation des tableaux

⟨type⟩ ⟨nom⟩`[` ⟨taille⟩ `]` = { ⟨valeur_0⟩`,` ⟨valeur_1⟩`,` `...` };

⟶ ⟨valeur_0⟩, ⟨valeur_1⟩ ...doivent être des valeurs du type (ou convertibles en) ⟨type⟩

⟶ le nombre des valeurs spécifiées dans l'initialisation ne doit pas dépasser la taille du tableau (mais il peut bien être inférieur à elle)

```
const int n = 3;
double a[ n ] = { 1.2, 3, -0.1 };
double b[ n ];
for ( int i = 0; i < n; i++ )
  b[ i ] = a[ i ];
for ( int i = 0; i < n; i++ )
  cout << b[ i ] << endl;
```

Exécution:

```
1.2
3
-0.1
```

Initialisation des tableaux

⟨type⟩ ⟨nom⟩`[]` = { ⟨valeur_1⟩`, ... ,` ⟨valeur_N⟩ }`;`

$\longrightarrow$ définition et initialisation d'un tableau avec *N* éléments:

⟨nom⟩`[ 0 ]` est initialisé avec la valeur ⟨valeur_1⟩

...

⟨nom⟩`[ N - 1 ]` est initialisé avec la valeur ⟨valeur_N⟩

Initialisation des tableaux

⟨type⟩ ⟨nom⟩`[]` = { ⟨valeur_1⟩, ... , ⟨valeur_N⟩ };

⟶ définition et initialisation d'un tableau avec *N* éléments:

⟨nom⟩`[ 0 ]` est initialisé avec la valeur ⟨valeur_1⟩

...

⟨nom⟩`[ N - 1 ]` est initialisé avec la valeur ⟨valeur_N⟩

```
const int n = 3;
double a[ n ] = { 1.2, 3, -0.1, 4.1 };
for ( int i = 0; i < n; i++ )
  cout << a[ i ] << endl;
```

⟶ `erreur: too many initializers for 'double [3]'`

Initialisation des tableaux

⟨type⟩ ⟨nom⟩`[]` = { ⟨valeur_1⟩, ... , ⟨valeur_N⟩ };

⟶ définition et initialisation d'un tableau avec *N* éléments:

⟨nom⟩`[ 0 ]` est initialisé avec la valeur ⟨valeur_1⟩

...

⟨nom⟩`[ N - 1 ]` est initialisé avec la valeur ⟨valeur_N⟩

```
const int n = 3;
double a[] = { 1.2, 3, -0.1, 4.1 };
for ( int i = 0; i < n; i++ )
  cout << a[ i ] << endl;
```

Exécution:

```
1.2
3
-0.1
```

Initialisation des tableaux

⟨type⟩ ⟨nom⟩`[]` = { ⟨valeur_1⟩, ... , ⟨valeur_N⟩ };

⟶ définition et initialisation d'un tableau avec *N* éléments:

⟨nom⟩`[ 0 ]` est initialisé avec la valeur ⟨valeur_1⟩

...

⟨nom⟩`[ N - 1 ]` est initialisé avec la valeur ⟨valeur_N⟩

```
const int n = 3;
double a[] = { 1.2, 3, -0.1, 4.1 };
for ( int i = 0; i < 4; i++ )
  cout << a[ i ] << endl;
```

Exécution:

```
1.2
3
-0.1
4.1
```

Les tableaux comme arguments des fonctions

Déclaration d'une fonction ⟨type_f⟩ qui prend un tableau des variables du type ⟨type_a⟩ comme argument:

⟨type_f⟩ ⟨nom_f⟩`(` `...` `,` ⟨type_a⟩ ⟨nom_a⟩`[]`, `...` `);`

Appel de cette fonction, après la définition du tableau ⟨type_a⟩ ⟨nom⟩`[` ⟨taille⟩ `];` :

⟨nom_f⟩`(` `...` `,` ⟨nom⟩, `...` `)`

$\longrightarrow$ on transmet l'adresse du premier élément du tableau à la fonction

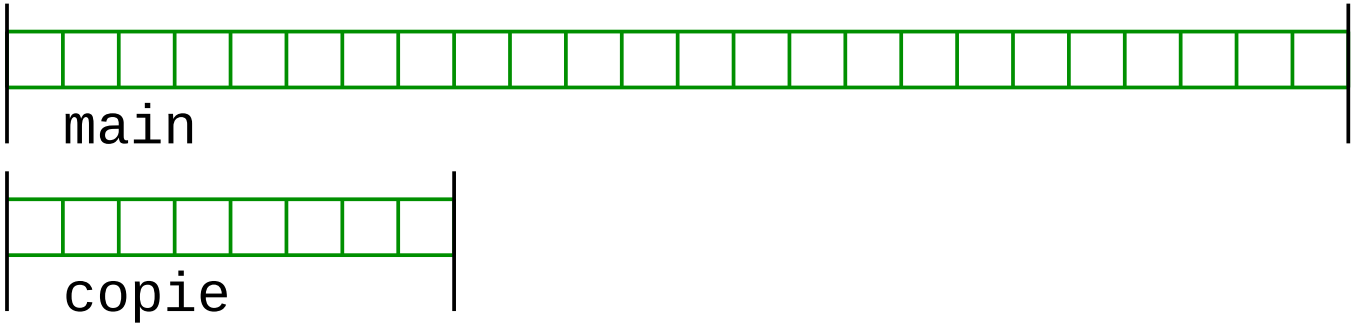
$\longrightarrow$ la fonction travaille donc forcément avec des variables originales du tableau

...mais elle ne connaît pas la taille du tableau

Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

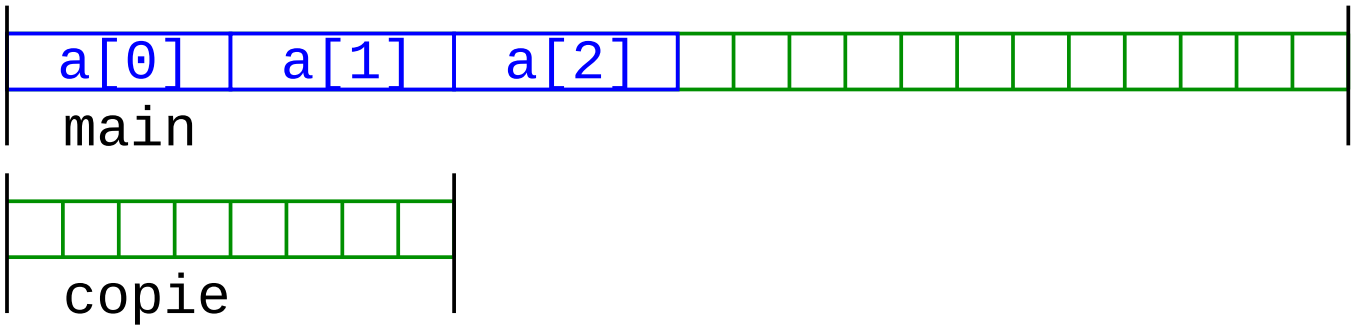
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

int main()
{
  double a[] = { 1.2, -0.1, 4.1 };
  double b[] = { 0, 0, 0 };
  copie( 3, a, b );
}
```



Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

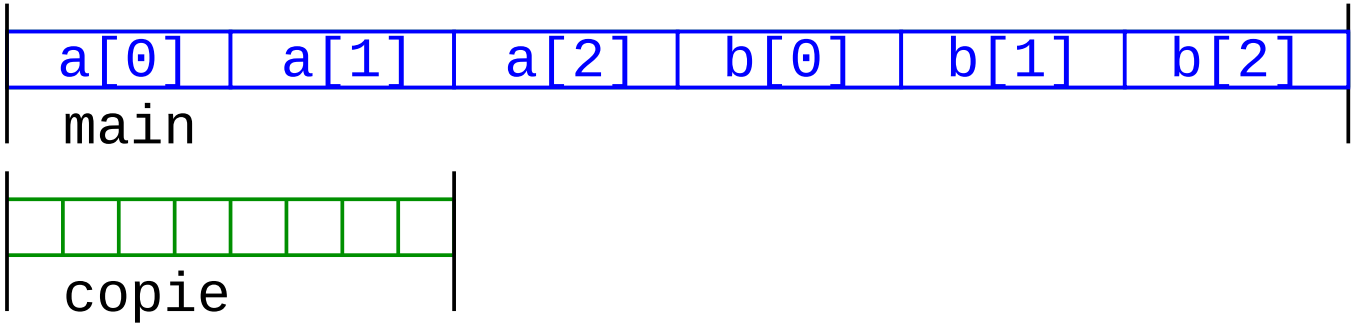
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

int main()
{
  double a[] = { 1.2, -0.1, 4.1 };
  double b[] = { 0, 0, 0 };
  copie( 3, a, b );
}
```



Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

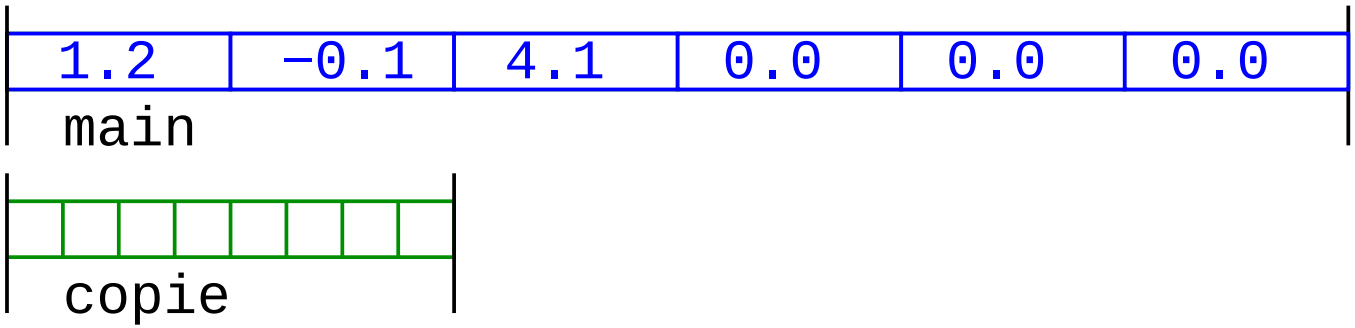
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

int main()
{
  double a[] = { 1.2, -0.1, 4.1 };
  double b[] = { 0, 0, 0 };
  copie( 3, a, b );
}
```



Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

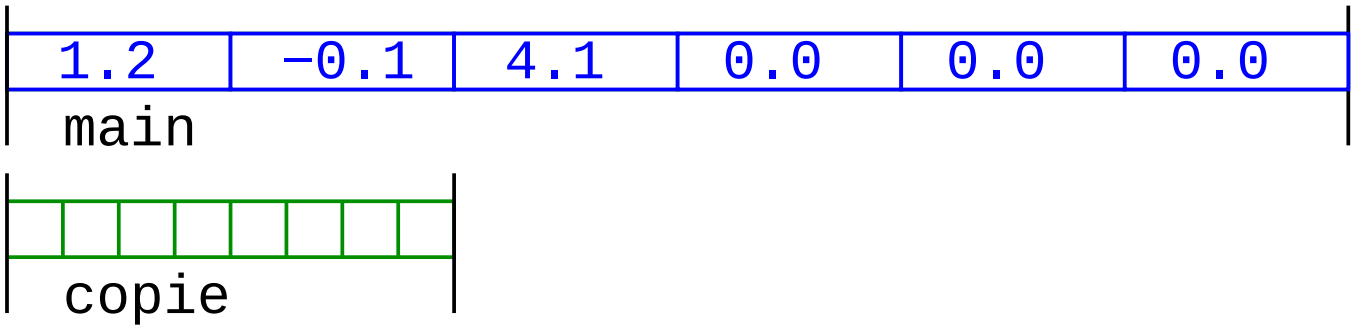
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

int main()
{
  double a[] = { 1.2, -0.1, 4.1 };
  double b[] = { 0, 0, 0 };
  copie( 3, a, b );
}
```



Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

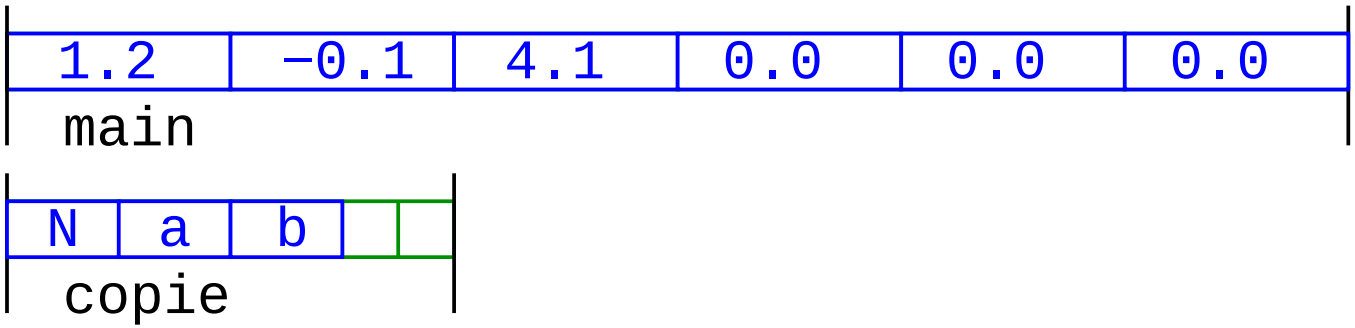
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

int main()
{
  double a[] = { 1.2, -0.1, 4.1 };
  double b[] = { 0, 0, 0 };
  copie( 3, a, b );
}
```



Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

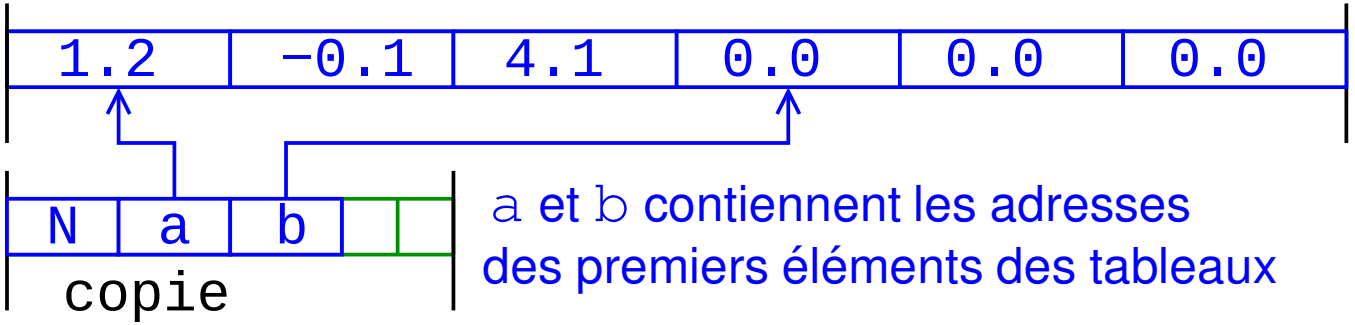
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

int main()
{
  double a[] = { 1.2, -0.1, 4.1 };
  double b[] = { 0, 0, 0 };
  copie( 3, a, b );
}
```



Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

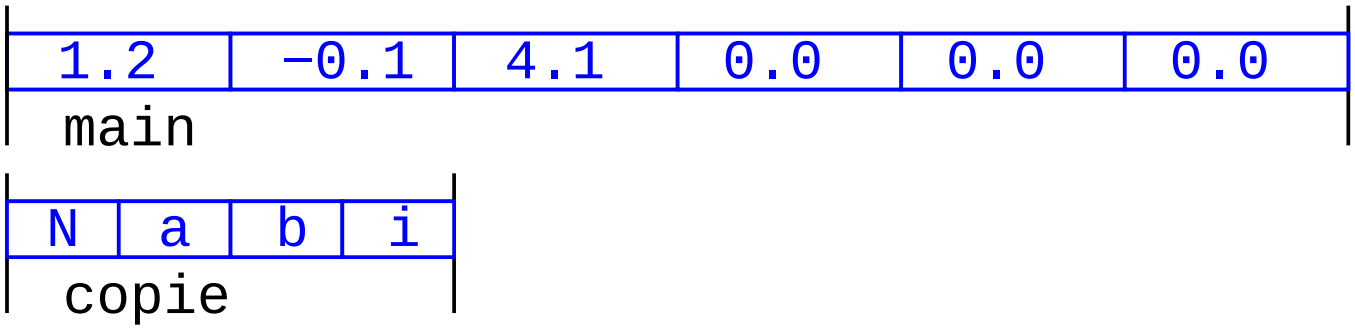
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

int main()
{
  double a[] = { 1.2, -0.1, 4.1 };
  double b[] = { 0, 0, 0 };
  copie( 3, a, b );
}
```



Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

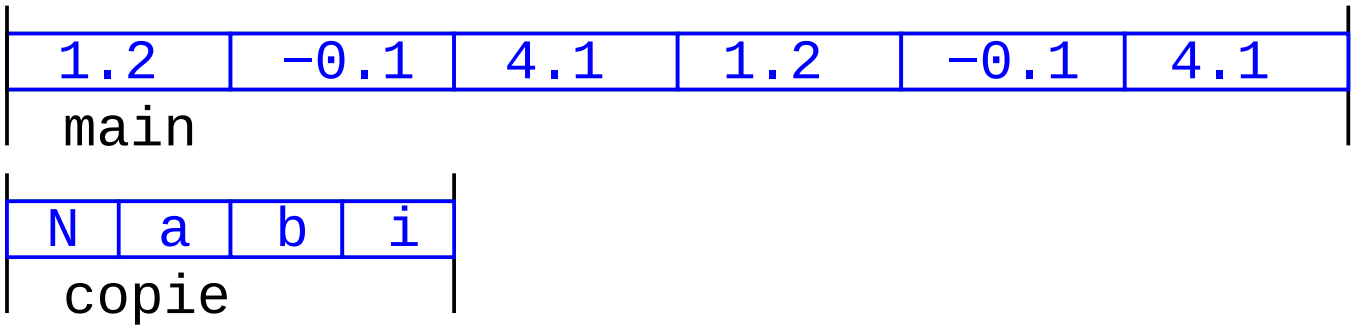
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

int main()
{
  double a[] = { 1.2, -0.1, 4.1 };
  double b[] = { 0, 0, 0 };
  copie( 3, a, b );
}
```



Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

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

int main()
{
  double a[] = { 1.2, -0.1, 4.1 };
  double b[] = { 0, 0, 0 };
  copie( 3, a, b );
}
```



Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

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

int main()
{
  double a[] = { 1.2, -0.1, 4.1 };
  double b[] = { 0, 0, 0 };
  copie( 3, a, b );
}
```

$\longrightarrow$ il n'est pas possible de définir des fonctions
qui rendent des tableaux comme "valeur de retour"

Les tableaux multidimensionnels

⟨type⟩ ⟨nom⟩ [⟨taille_1⟩] [⟨taille_2⟩];

⟶ définition d'un tableau à deux dimensions
= "un tableau des tableaux"

Définition avec initialisation:

```
⟨type⟩ ⟨nom⟩ [ ⟨taille_1⟩ ] [ ⟨taille_2⟩ ] =
   { { ⟨elem_11⟩, ⟨elem_12⟩, ... ⟨elem_1M⟩ },
     { ⟨elem_21⟩, ⟨elem_22⟩, ... ⟨elem_2M⟩ },
       ...
     { ⟨elem_N1⟩, ⟨elem_N2⟩, ... ⟨elem_NM⟩ } };
```

⟶ N doit correspondre à la valeur de ⟨taille_1⟩
M doit correspondre à la valeur de ⟨taille_2⟩

Les tableaux multidimensionnels

⟨type⟩ ⟨nom⟩`[` ⟨taille_1⟩ `][` ⟨taille_2⟩ `];`

⟶ définition d'un tableau à deux dimensions
= "un tableau des tableaux"

Définition avec initialisation:

⟨type⟩ ⟨nom⟩`[][] =`
`{ {` ⟨elem_11⟩`,` ⟨elem_12⟩`, ...` ⟨elem_1M⟩ `},`
`{` ⟨elem_21⟩`,` ⟨elem_22⟩`, ...` ⟨elem_2M⟩ `},`
`...`
`{` ⟨elem_N1⟩`,` ⟨elem_N2⟩`, ...` ⟨elem_NM⟩ `} };`

⟶ `erreur: declaration of '`⟨nom⟩`' as multidimensional array must have bounds for all dimensions except the first`

Les tableaux multidimensionnels

⟨type⟩ ⟨nom⟩ [⟨taille_1⟩] [⟨taille_2⟩];

⟶ définition d'un tableau à deux dimensions
= "un tableau des tableaux"

Définition avec initialisation:

```
⟨type⟩ ⟨nom⟩ [ ] [ ⟨taille_2⟩ ] =
   { { ⟨elem_11⟩, ⟨elem_12⟩, ... ⟨elem_1M⟩ },
     { ⟨elem_21⟩, ⟨elem_22⟩, ... ⟨elem_2M⟩ },
       ...
     { ⟨elem_N1⟩, ⟨elem_N2⟩, ... ⟨elem_NM⟩ } };
```

⟶ N définit la première dimension du tableau

Les tableaux multidimensionnels

⟨type⟩ ⟨nom⟩ [⟨taille_1⟩] [⟨taille_2⟩];

⟶ définition d'un tableau à deux dimensions
= "un tableau des tableaux"

Définition avec initialisation:

```
⟨type⟩ ⟨nom⟩ [ ] [ ⟨taille_2⟩ ] =
   { { ⟨elem_11⟩, ⟨elem_12⟩, ... ⟨elem_1M⟩ },
     { ⟨elem_21⟩, ⟨elem_22⟩, ... ⟨elem_2M⟩ },
       ...
     { ⟨elem_N1⟩, ⟨elem_N2⟩, ... ⟨elem_NM⟩ } };
```

Accès aux éléments: ⟨nom⟩ [⟨int_expr_1⟩] [⟨int_expr_2⟩]

⟶ ça rend l'élément no. $\langle \text{int\_expr\_1} \rangle \times \langle \text{taille\_2} \rangle + \langle \text{int\_expr\_2} \rangle$
dans la séquence des valeurs en mémoire

Les tableaux multidimensionnels

⟨type⟩ ⟨nom⟩ [⟨taille_1⟩] [⟨taille_2⟩] [⟨taille_3⟩] ;

$\longrightarrow$ définition d'un tableau à trois dimensions
= "un tableau des tableaux des tableaux"

Définition avec initialisation:

⟨type⟩ ⟨nom⟩ [ ] [⟨taille_2⟩] [⟨taille_3⟩] =

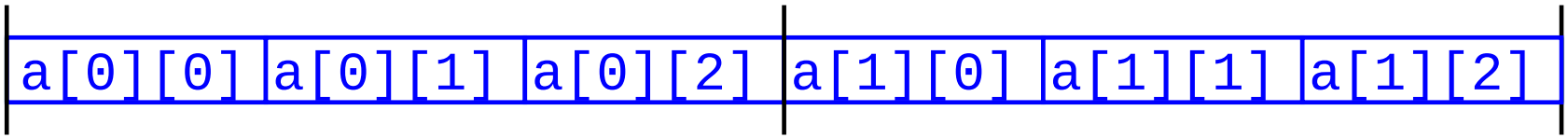
```
{ { { ... }, { ... }, ..., { ... } },
  { { ... }, { ... }, ..., { ... } },
      ...
  { { ... }, { ... }, ..., { ... } } };
```

Accès aux éléments:

⟨nom⟩ [⟨int_expr_1⟩] [⟨int_expr_2⟩] [⟨int_expr_3⟩]

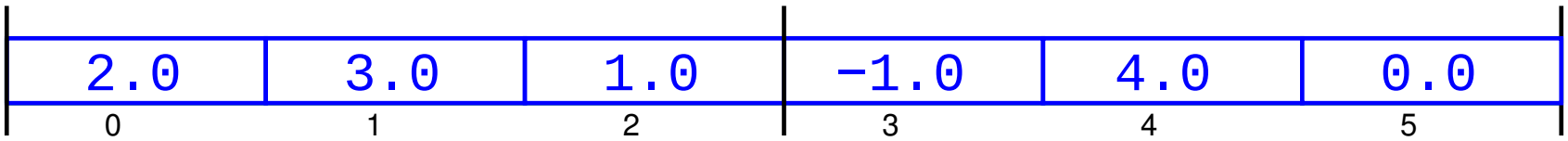
Les tableaux multidimensionnels

```
int main()
{
  double a[ 2 ][ 3 ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  cout << a[ 1 ][ 1 ] << endl;
}
```



Les tableaux multidimensionnels

```
int main()
{
  double a[ 2 ][ 3 ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  cout << a[ 1 ][ 1 ] << endl;
}
```



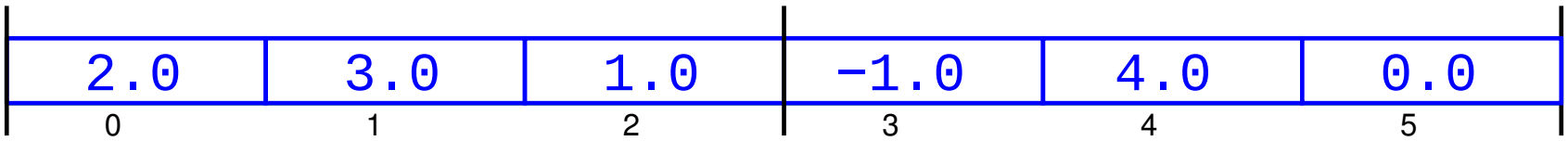
Exécution:

```
4
```

$\longrightarrow$ accès à l'élément no. $3 \times 1 + 1 = 4$ du tableau `a`

Les tableaux multidimensionnels

```
int main()
{
  double a[ 2 ][ 3 ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  cout << a[ 0 ][ 3 ] << endl;
}
```



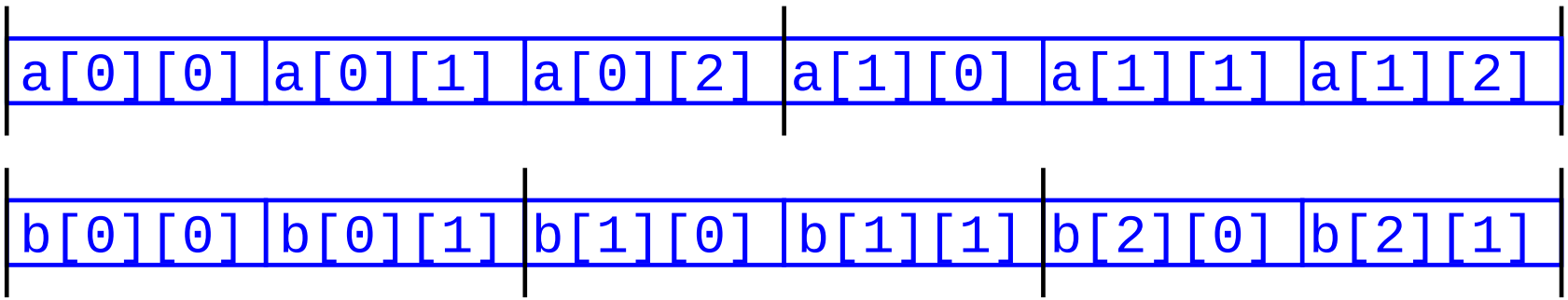
Exécution:

```
-1
```

$\longrightarrow$ accès à l'élément no. $3 \times 0 + 3 = 3$ du tableau `a`

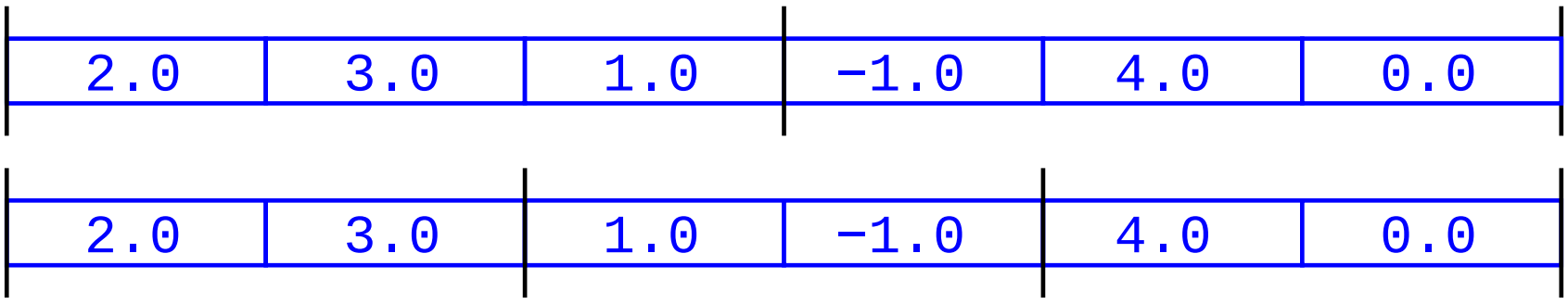
Les tableaux multidimensionnels

```
int main()
{
  double a[ 2 ][ 3 ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ 3 ][ 2 ] = { { 2, 3 }, { 1, -1 }, { 4, 0 } };
  cout << a[ 1 ][ 1 ] << endl;
  cout << b[ 1 ][ 1 ] << endl;
}
```



Les tableaux multidimensionnels

```
int main()
{
  double a[ 2 ][ 3 ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ 3 ][ 2 ] = { { 2, 3 }, { 1, -1 }, { 4, 0 } };
  cout << a[ 1 ][ 1 ] << endl;
  cout << b[ 1 ][ 1 ] << endl;
}
```

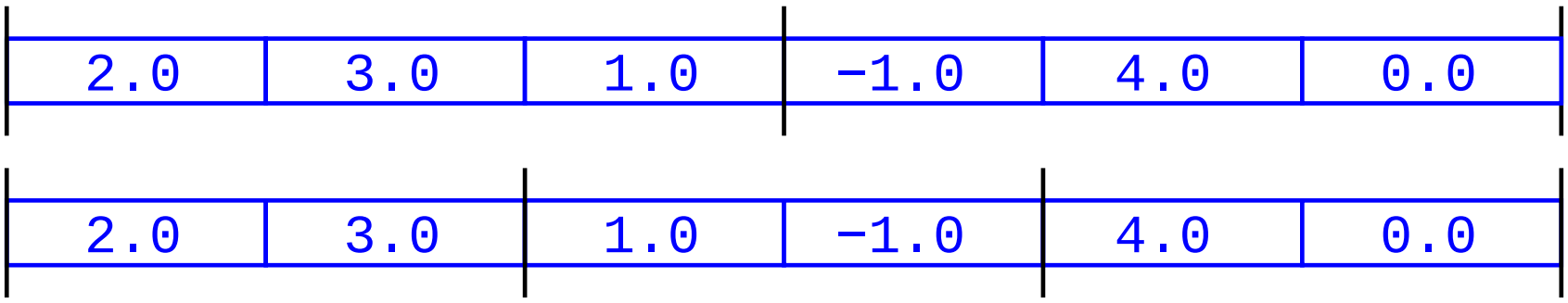


Exécution:

```
4
-1
```

Les tableaux multidimensionnels

```
int main()
{
  double a[ 2 ][ 3 ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ 3 ][ 2 ] = { { 2, 3 }, { 1, -1 }, { 4, 0 } };
  cout << a[ 0 ][ 3 ] << endl;
  cout << b[ 0 ][ 3 ] << endl;
}
```



Exécution:

```
-1
-1
```

Les tableaux multidimensionnels

. . . comme arguments des fonctions:

```
int main()
{
  double a[ 2 ][ 3 ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ 2 ][ 3 ];
  copie( a, b );
  cout << b[ 1 ][ 1 ] << endl;
}
```

Les tableaux multidimensionnels

```
void copie( double a[][], double b[][] )
{
  for ( int i = 0; i < 2; i++ )
    for ( int j = 0; j < 3; j++ )
      b[ i ][ j ] = a[ i ][ j ];
}

int main()
{
  double a[ 2 ][ 3 ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ 2 ][ 3 ];
  copie( a, b );
  cout << b[ 1 ][ 1 ] << endl;
}
```

$\longrightarrow$ erreur:

```
declaration of 'a' as multidimensional array must have
bounds for all dimensions except the first
declaration of 'b' as multidimensional array must have
bounds for all dimensions except the first
```

Les tableaux multidimensionnels

```
void copie( double a[][ 3 ], double b[][ 3 ] )
{
  for ( int i = 0; i < 2; i++ )
    for ( int j = 0; j < 3; j++ )
      b[ i ][ j ] = a[ i ][ j ];
}

int main()
{
  double a[ 2 ][ 3 ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ 2 ][ 3 ];
  copie( a, b );
  cout << b[ 1 ][ 1 ] << endl;
}
```

Exécution:

```
4
```

Les tableaux multidimensionnels

```
const int M = 3;

void copie( double a[][ M ], double b[][ M ] )
{
  for ( int i = 0; i < 2; i++ )
    for ( int j = 0; j < M; j++ )
      b[ i ][ j ] = a[ i ][ j ];
}

int main()
{
  double a[ 2 ][ M ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ 2 ][ M ];
  copie( a, b );
  cout << b[ 1 ][ 1 ] << endl;
}
```

$\longrightarrow$ définition de la taille M hors des blocs de fonctions

Les tableaux multidimensionnels

```
const int M = 3;

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

int main()
{
  const int N = 2;
  double a[ N ][ M ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ N ][ M ];
  copie( N, a, b );
  cout << b[ 1 ][ 1 ] << endl;
}
```

Les tableaux multidimensionnels

```
const int M = 4;

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

int main()
{
  const int N = 3;
  double a[ N ][ M ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ N ][ M ];
  copie( N, a, b );
  cout << b[ 1 ][ 1 ] << endl;
}
```

Exécution:

```
4
```

Les tableaux multidimensionnels

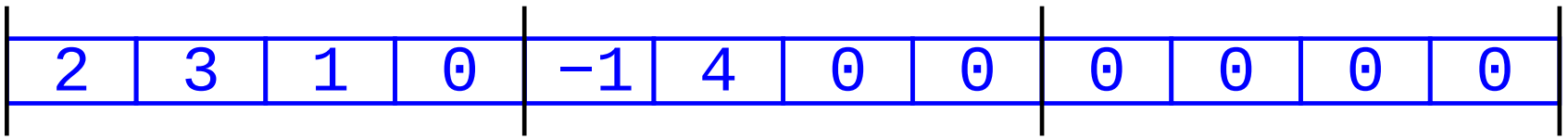
```
const int M = 4;

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

int main()
{
  const int N = 3;
  double a[ N ][ M ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ N ][ M ];
  copie( N, a, b );
  cout << b[ 0 ][ 3 ] << endl;
}
```

Exécution:

0

Les tableaux multidimensionnels

```
const int M = 4;

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

int main()
{
  const int N = 3;
  double a[ N ][ M ] = { { 2, 3, 1 }, { -1, 4, 0 } };
  double b[ N ][ M ];
  copie( N, a, b );
  cout << b[ 0 ][ 3 ] << endl;
}
```



Les tableaux multidimensionnels

Utilisation pour des routines de manipulation des matrices ?

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 ] )
{
  ...
}
```

Les tableaux multidimensionnels

Utilisation pour des routines de manipulation des matrices ?

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 ] )
{
  ...
}
```

Les tableaux multidimensionnels

Utilisation pour des routines de manipulation des matrices ?

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 $M \times M$

Les tableaux multidimensionnels

Utilisation pour des routines de manipulation des matrices ?

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[] );

...
```

Les tableaux multidimensionnels

Utilisation pour des routines de manipulation des matrices ?

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[] )
{
  ...
}
```

Les tableaux multidimensionnels

Utilisation pour des routines de manipulation des matrices ?

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 par

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

$\longrightarrow$ alternative: utilisation des tableaux dynamiques

8.2 Les strings (chaînes de caractères)

Un string est un tableau de caractères . . .

```
char nom[] = { 'P', 'e', 't', 'e', 'r' };
for ( int i = 0; i < 5; i++ )
  cout << nom[ i ];
cout << endl;
```

Exécution:

```
Peter
```

8.2 Les strings (chaînes de caractères)

Un string est un tableau de caractères . . .

. . . qui termine avec le caractère 'nul'

```
char nom[] = { 'P', 'e', 't', 'e', 'r', 0 };
for ( int i = 0; i < 5; i++ )
  cout << nom[ i ];
cout << endl;
```

8.2 Les strings (chaînes de caractères)

Un string est un tableau de caractères . . .

. . . qui termine avec le caractère 'nul'

```
char nom[] = "Peter";
for ( int i = 0; i < 5; i++ )
  cout << nom[ i ];
cout << endl;
```

8.2 Les strings (chaînes de caractères)

Un string est un tableau de caractères . . .

. . . qui termine avec le caractère 'nul'

```
char nom[] = "Peter";
cout << nom << endl;
```

Exécution:

```
Peter
```

$\longrightarrow$ initialisation "directe" avec des guillemets

$\longrightarrow$ input et output "directs" avec le nom du string

8.2 Les strings (chaînes de caractères)

Un string est un tableau de caractères . . .

. . . qui termine avec le caractère 'nul'

```
char nom[] = "Peter";
for ( int i = 0; i < 6; i++ )
  cout << int( nom[ i ] ) << " ";
cout << endl;
```

Exécution:

```
80 101 116 101 114 0
```

8.2 Les strings (chaînes de caractères)

```
char nom[ 10 ];
cout << "entrez votre nom: ";
cin >> nom;
cout << "voila votre nom: " << nom << endl;
```

Exécution:

```
entrez votre nom: Peter
voila votre nom: Peter
```

8.2 Les strings (chaînes de caractères)

```
char nom[ 10 ];
cout << "entrez votre nom: ";
cin >> nom;
cout << "voila votre nom: " << nom << endl;
for ( int i = 0; i < 10; i++ )
  cout << int( nom[ i ] ) << " ";
cout << endl;
```

Exécution:

```
entrez votre nom: Peter
voila votre nom: Peter
80 101 116 101 114 0 8 -88 91 -37
```

$\longrightarrow$ `cin` met les caractères lus depuis le clavier dans `nom` et ajoute le caractère 'nul'

$\longrightarrow$ `cout` affiche tous les caractères de `nom` jusqu'au caractère 'nul'

Manipulations des strings

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

Manipulations des strings

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

$\longrightarrow$ pas besoin de spécifier la taille du string

$\longrightarrow$ de telles fonctions de manipulation des strings sont disponibles en incluant le header `<cstring>` (`<string.h>` en C)

Manipulations des strings

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

int main()
{
  char prenom[] = "Peter";
  char nom[ 100 ];
  strcpy( nom, prenom );
  cout << nom << endl;
}
```

Exécution:

```
Peter
```

`strcpy( a, b )` copie le contenu du string `b` dans le string `a` (“$a = b$”)

Manipulations des strings

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

int main()
{
  char prenom[] = "Peter";
  char nom[ 100 ];
  strcpy( nom, prenom );
  char nom_famille[] = "Schlagheck";
  strcat( nom, nom_famille );
  cout << nom << endl;
}
```

Exécution:

```
PeterSchlagheck
```

`strcat( a, b )` ajoute le contenu du string `b` au string `a` (“$a = a + b$”)

Manipulations des strings

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

int main()
{
  char prenom[] = "Peter";
  char nom[ 100 ];
  strcpy( nom, prenom );
  char nom_famille[] = "Schlagheck";
  strcat( nom, " " );
  strcat( nom, nom_famille );
  cout << nom << endl;
}
```

Exécution:

```
Peter Schlagheck
```

Manipulations des strings

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

int main()
{
  char prenom[] = "Peter";
  char nom[ 100 ];
  strcpy( nom, prenom );
  char nom_famille[] = "Schlagheck";
  strcat( nom, " " );
  strcat( nom, nom_famille );
  cout << nom << " -- " << strlen( nom ) << endl;
}
```

Exécution:

```
Peter Schlagheck -- 16
```

`strlen( a )` rend la longueur du string `a`

Manipulations des strings

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

int main()
{
  char motpasse[] = "asdj25gf";
  char mot[ 100 ];
  cout << "saisissez le mot de passe: " << endl;
  cin >> mot;
  if ( strcmp( mot, motpasse ) )
    cout << "pas correct" << endl;
  else
    cout << "correct" << endl;
}
```

Exécution:

```
saisissez le mot de passe: asdj25gf
correct
```

Manipulations des strings

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

int main()
{
  char motpasse[] = "asdj25gf";
  char mot[ 100 ];
  cout << "saisissez le mot de passe: " << endl;
  cin >> mot;
  if ( strcmp( mot, motpasse ) )
    cout << "pas correct" << endl;
  else
    cout << "correct" << endl;
}
```

Exécution:

```
saisissez le mot de passe: asdj25g
pas correct
```

Manipulations des strings

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

int main()
{
  char motpasse[] = "asdj25gf";
  char mot[ 100 ];
  cout << "saisissez le mot de passe: " << endl;
  cin >> mot;
  if ( strcmp( mot, motpasse ) )
    cout << "pas correct" << endl;
  else
    cout << "correct" << endl;
}
```

Exécution:

```
saisissez le mot de passe: asdj25gfc
pas correct
```

Manipulations des strings

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

int main()
{
  char motpasse[] = "asdj25gf";
  char mot[ 100 ];
  cout << "saisissez le mot de passe: " << endl;
  cin >> mot;
  if ( strcmp( mot, motpasse ) )
    cout << "pas correct" << endl;
  else
    cout << "correct" << endl;
}
```

`strcmp( a, b )` rend 0 si les strings `a` et `b` sont égaux, 1 si $a > b$ (alphabétiquement) et -1 si $a < b$

Des tableaux des strings

```
char nom[ 12 ][ 50 ];
```

$\longrightarrow$ déclaration d'un tableau de 12 strings
avec une taille maximale de 50 caractères

Des tableaux des strings

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

int main()
{
  const int N = 13;
  char nom[ N ][ 50 ];
  char prenom[ N ][ 50 ];
  ifstream inp( "noms.dat" );
  for ( int i = 0; i < N; i++ )
    inp >> prenom[ i ] >> nom[ i ];
  cout << prenom[ 4 ]
       << nom[ 4 ] << endl;
}
```

Exécution:

```
ThierryBastin
```

Le fichier `noms.dat`:

```
Nicolas Vandewalle
Philippe Ghosez
Alain Seret
Geoffroy Lumay
Thierry Bastin
Laurent Dreesen
Maryse Hoebeke
John Martin
Peter Schlagheck
Matthieu Verstraete
Alejandro Silhanek
Herve Caps
David Strivay
```

Des tableaux des strings

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

int main()
{
  const int N = 13;
  char nom[ N ][ 50 ];
  char prenom[ N ][ 50 ];
  ifstream inp( "noms.dat" );
  for ( int i = 0; i < N; i++ )
    inp >> prenom[ i ] >> nom[ i ];
  cout << prenom[ 4 ] << " "
       << nom[ 4 ] << endl;
}
```

Exécution:

```
Thierry Bastin
```

Le fichier `noms.dat`:

```
Nicolas Vandewalle
Philippe Ghosez
Alain Seret
Geoffroy Lumay
Thierry Bastin
Laurent Dreesen
Maryse Hoebeke
John Martin
Peter Schlagheck
Matthieu Verstraete
Alejandro Silhanek
Herve Caps
David Strivay
```

Des tableaux des strings

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

int main()
{
  const int N = 13;
  char nom[ N ][ 50 ];
  char prenom[ N ][ 50 ];
  ifstream inp( "noms.dat" );
  for ( int i = 0; i < N; i++ )
    inp >> prenom[ i ] >> nom[ i ];
  cout << prenom[ 4 ] << " "
       << nom[ 4 ] << endl;
}
```

Le fichier `noms.dat`:

```
Nicolas Vandewalle
Philippe Ghosez
Alain Seret
Geoffroy Lumay
Thierry Bastin
Laurent Dreesen
Maryse Hoebeke
John Martin
Peter Schlagheck
Matthieu Verstraete
Alejandro Silhanek
Herve Caps
David Strivay
```

$\longrightarrow$ le caractère 'espace' (no. 32) sépare les différents strings dans le fichier `noms.dat`

Des tableaux des strings

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

int main()
{
  const int N = 13;
  char nom[ N ][ 50 ];
  char prenom[ N ][ 50 ];
  ifstream inp( "noms.dat" );
  for ( int i = 0; i < N; i++ )
    inp >> prenom[ i ] >> nom[ i ];
  cout << prenom[ 4 ] << " "
       << nom[ 4 ] << endl;
}
```

Le fichier `noms.dat`:

```
NicolasVandewalle
PhilippeGhosez
AlainSeret
GeoffroyLumay
ThierryBastin
LaurentDreesen
MaryseHoebeke
JohnMartin
PeterSchlagheck
MatthieuVerstraete
AlejandroSilhanek
HerveCaps
DavidStrivay
```

Des tableaux des strings

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

int main()
{
  const int N = 13;
  char nom[ N ][ 50 ];
  char prenom[ N ][ 50 ];
  ifstream inp( "noms.dat" );
  for ( int i = 0; i < N; i++ )
    inp >> prenom[ i ] >> nom[ i ];
  cout << prenom[ 4 ] << " "
       << nom[ 4 ] << endl;
}
```

Le fichier `noms.dat`:

```
NicolasVandewalle
PhilippeGhosez
AlainSeret
GeoffroyLumay
ThierryBastin
LaurentDreesen
MaryseHoebeke
JohnMartin
PeterSchlagheck
MatthieuVerstraete
AlejandroSilhanek
HerveCaps
DavidStrivay
```

Exécution:

```
PeterSchlagheck MatthieuVerstraete
```

Des tableaux des strings

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

int main()
{
  const int N = 13;
  char nom[ N ][ 50 ];
  char prenom[ N ][ 50 ];
  ifstream inp( "noms.dat" );
  for ( int i = 0; i < N; i++ )
    inp >> prenom[ i ] >> nom[ i ];
  cout << prenom[ 4 ] << " "
       << nom[ 4 ] << endl;
}
```

Le fichier `noms.dat`:

```
Nicolas Vandewalle
Philippe Ghosez
Alain Seret
Geoffroy Lumay
Thierry Bastin
Laurent Dreesen
Maryse Hoebeke
John Martin
Peter Schlagheck
Matthieu Verstraete
Alejandro Silhanek
Herve Caps
David Strivay
```

$\longrightarrow$ trouver le premier nom dans l'alphabète !

Des tableaux des strings

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

int main()
{
  const int N = 13;
  char nom[ N ][ 50 ];
  char prenom[ N ][ 50 ];
  ifstream inp( "noms.dat" );
  for ( int i = 0; i < N; i++ )
    inp >> prenom[ i ] >> nom[ i ];

  char nom0[ 50 ] = "|";
  for ( int i = 0; i < N; i++ )
    if ( strcmp( nom[ i ], nom0 ) < 0 )
      strcpy( nom0, nom[ i ] );
  cout << nom0 << endl;
}
```

Exécution:

```
Bastin
```

Le fichier `noms.dat`:

```
Nicolas Vandewalle
Philippe Ghosez
Alain Seret
Geoffroy Lumay
Thierry Bastin
Laurent Dreesen
Maryse Hoebeke
John Martin
Peter Schlagheck
Matthieu Verstraete
Alejandro Silhanek
Herve Caps
David Strivay
```

Des tableaux des strings

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

int main()
{
  const int N = 13;
  char nom[ N ][ 50 ];
  char prenom[ N ][ 50 ];
  ifstream inp( "noms.dat" );
  for ( int i = 0; i < N; i++ )
    inp >> prenom[ i ] >> nom[ i ];

  char nom0[ 50 ] = "|";
  char prenom0[ 50 ];
  for ( int i = 0; i < N; i++ )
    if ( strcmp( nom[ i ], nom0 ) < 0 )
    {
      strcpy( nom0, nom[ i ] );
      strcpy( prenom0, prenom[ i ] );
    }
  cout << prenom0 << " " << nom0 << endl;
}
```

Exécution:

```
Thierry Bastin
```

Le fichier `noms.dat`:

```
Nicolas Vandewalle
Philippe Ghosez
Alain Seret
Geoffroy Lumay
Thierry Bastin
Laurent Dreesen
Maryse Hoebeke
John Martin
Peter Schlagheck
Matthieu Verstraete
Alejandro Silhanek
Herve Caps
David Strivay
```

8.3 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`

8.3 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`

8.3 Les Pointeurs

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

Déclaration:

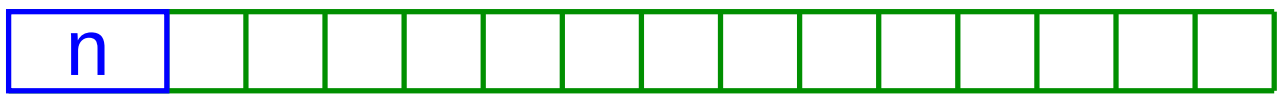
```
int *ip;
double *a;
```

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

$\longrightarrow$ comment (et pourquoi) utiliser ?

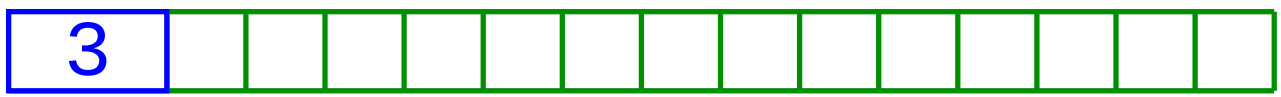
8.3 Les Pointeurs

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



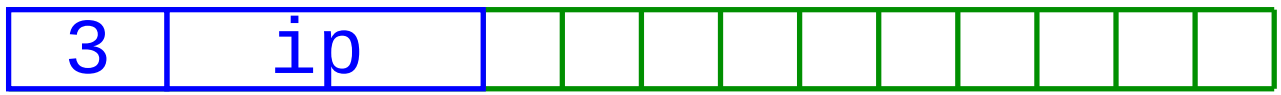
8.3 Les Pointeurs

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



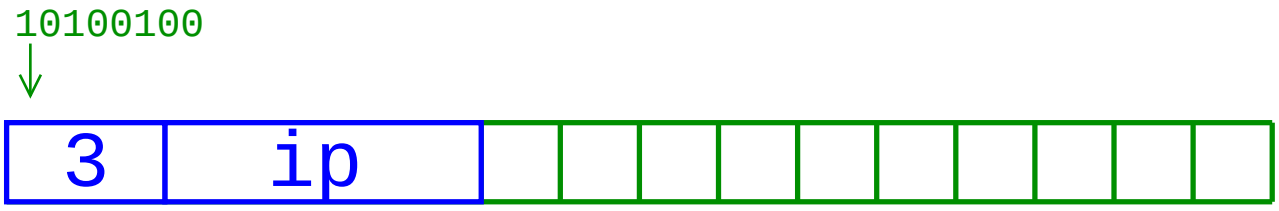
8.3 Les Pointeurs

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



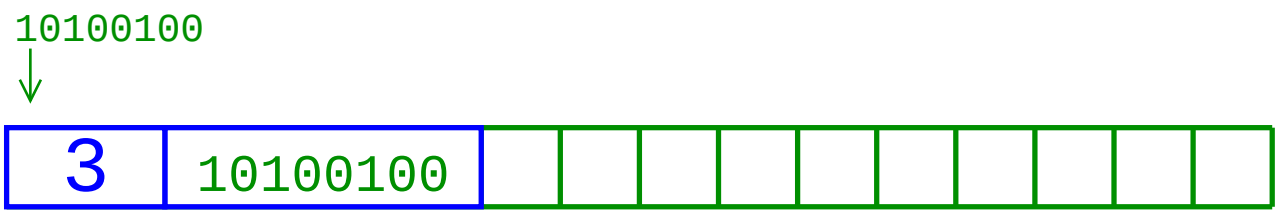
8.3 Les Pointeurs

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



8.3 Les Pointeurs

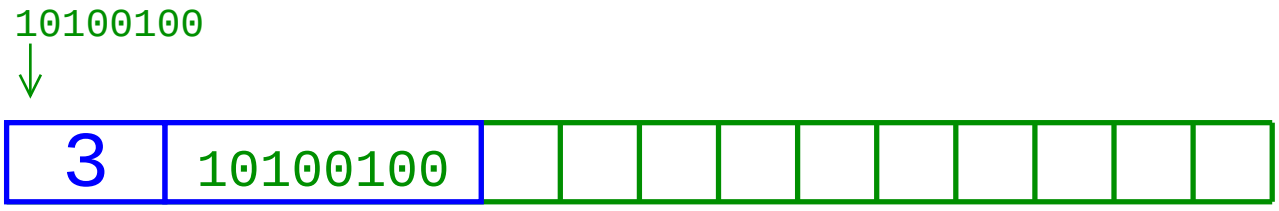
```
int n = 3;
int *ip;
ip = &n;
int k = *ip;
```



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

8.3 Les Pointeurs

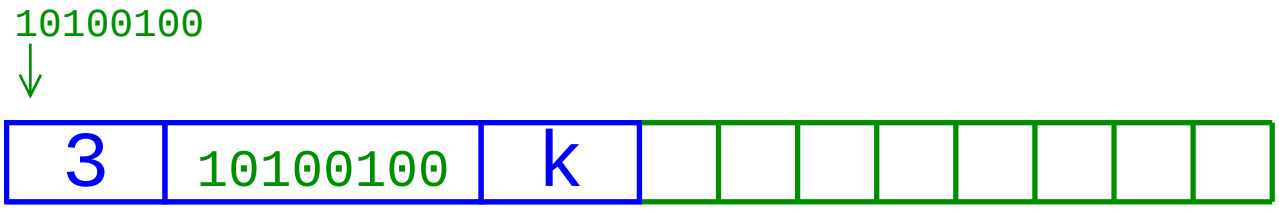
```
int n = 3;
int *ip = &n;
int k = *ip;
```



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

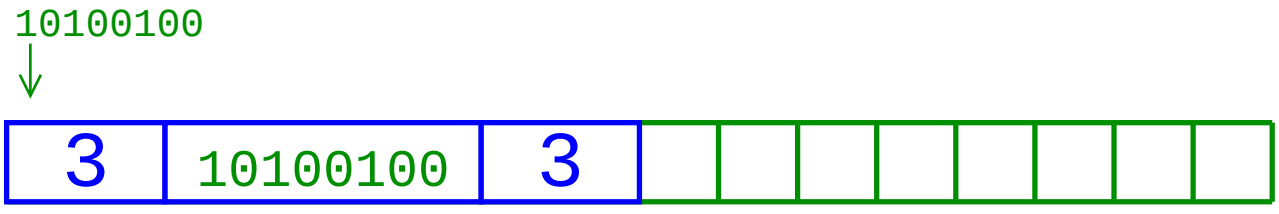
8.3 Les Pointeurs

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



8.3 Les Pointeurs

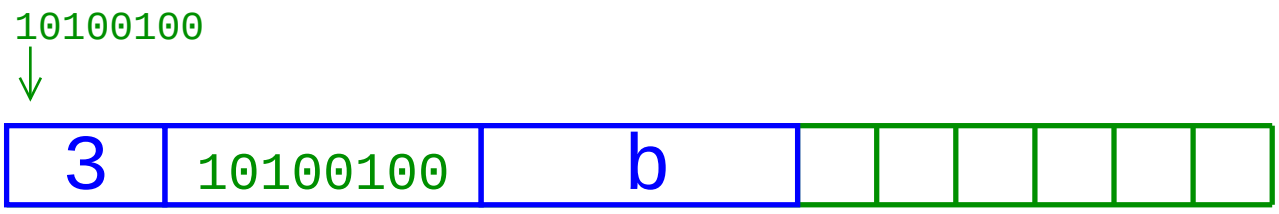
```
int n = 3;
int *ip = &n;
int k = *ip;
```



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

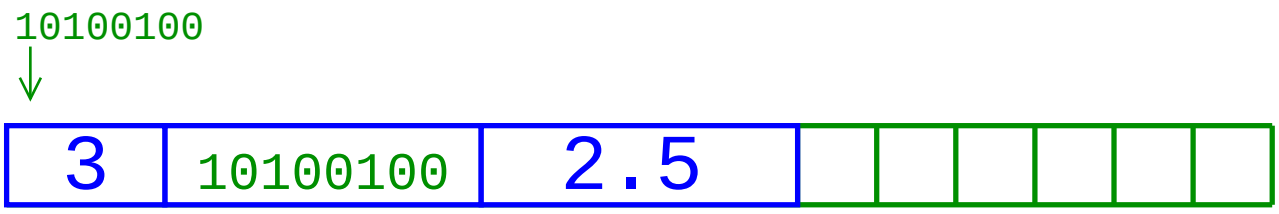
8.3 Les Pointeurs

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



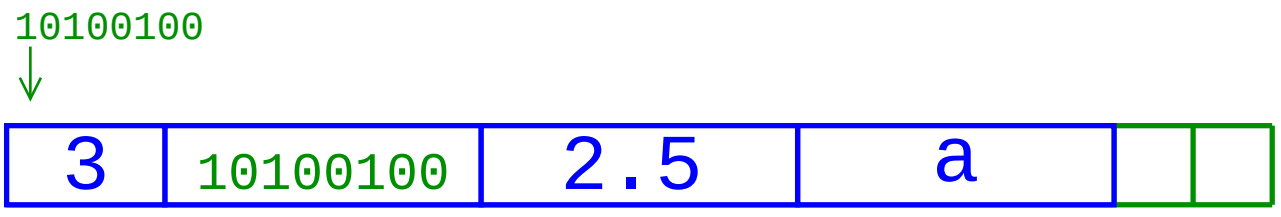
8.3 Les Pointeurs

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



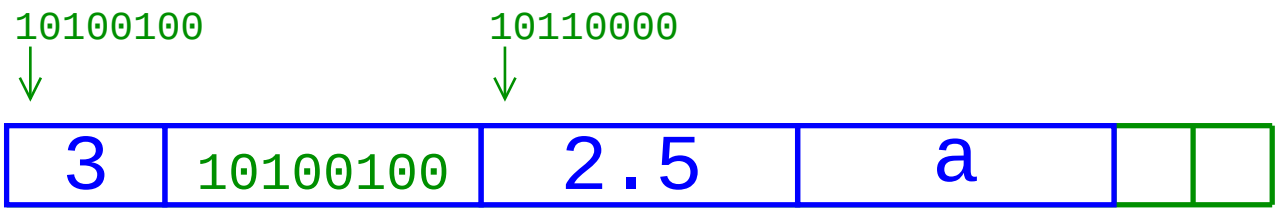
8.3 Les Pointeurs

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



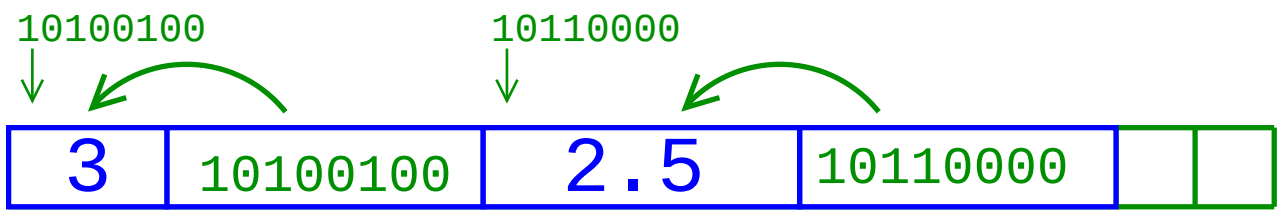
8.3 Les Pointeurs

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



8.3 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)

8.3 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⟩

8.3 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)

8.3 Les Pointeurs

⟨type⟩ ∗⟨nom⟩;

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

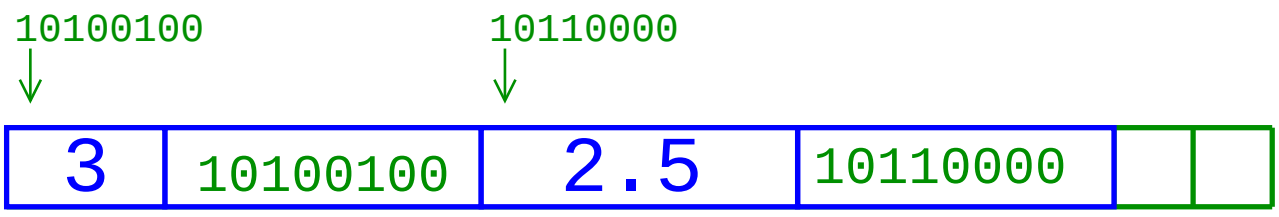
opérateur d'adresse: &⟨variable⟩
⟶ rend l'adresse en mémoire de la variable ⟨variable⟩

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

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

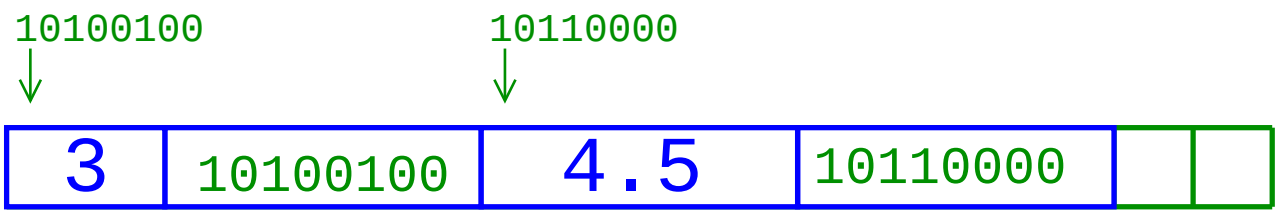
8.3 Les Pointeurs

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



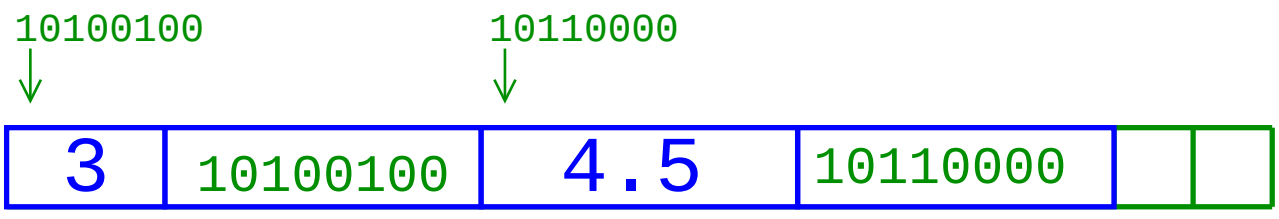
8.3 Les Pointeurs

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



8.3 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 !!!

8.3 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:

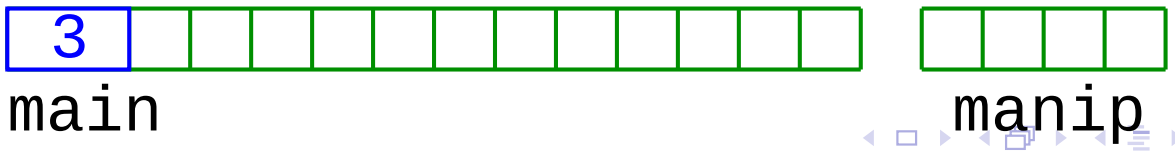
```
3  8  5
```

8.3 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
```

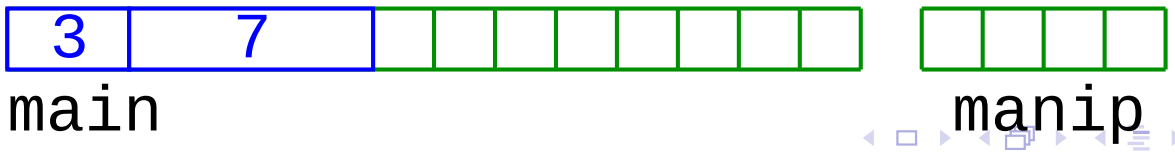


8.3 Les Pointeurs

```
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
```

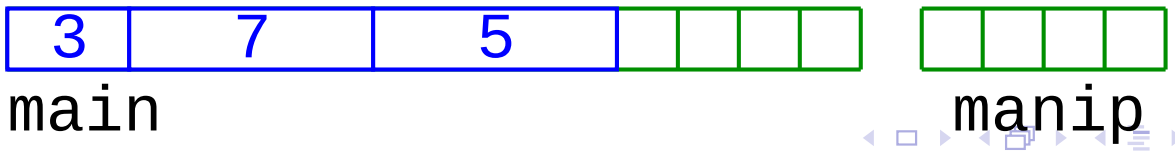


8.3 Les Pointeurs

```
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
```

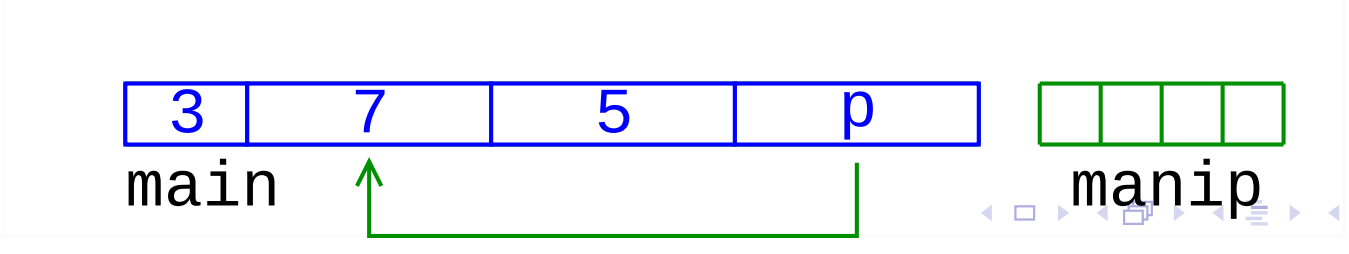


8.3 Les Pointeurs

```
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
```

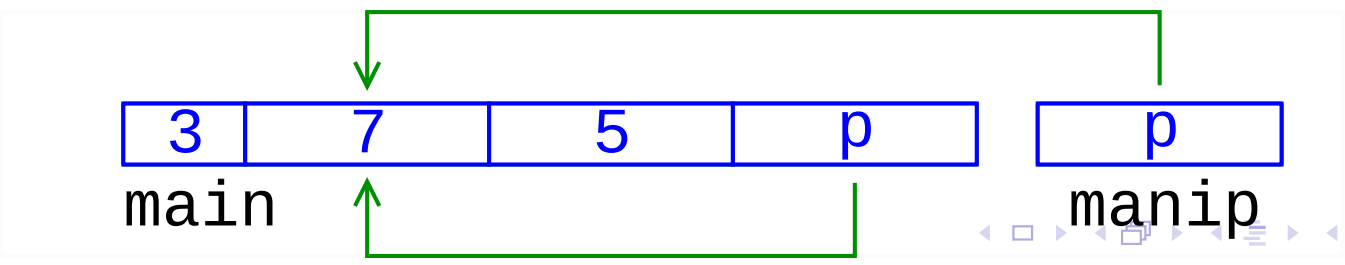


8.3 Les Pointeurs

```
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
```

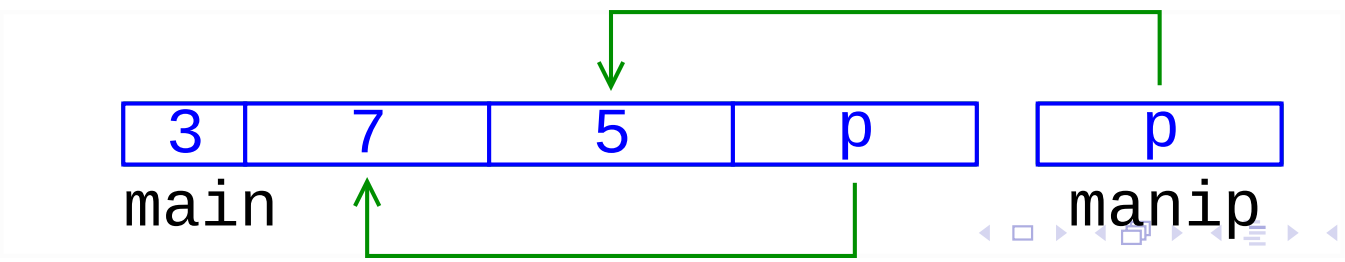


8.3 Les Pointeurs

```
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
```

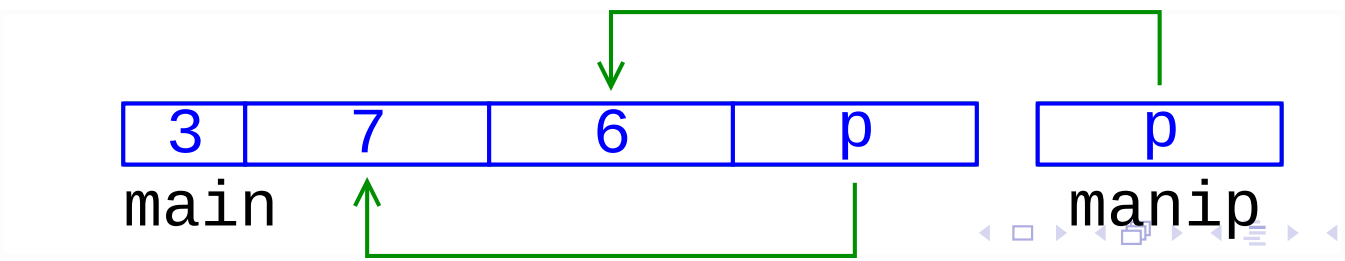


8.3 Les Pointeurs

```
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
```

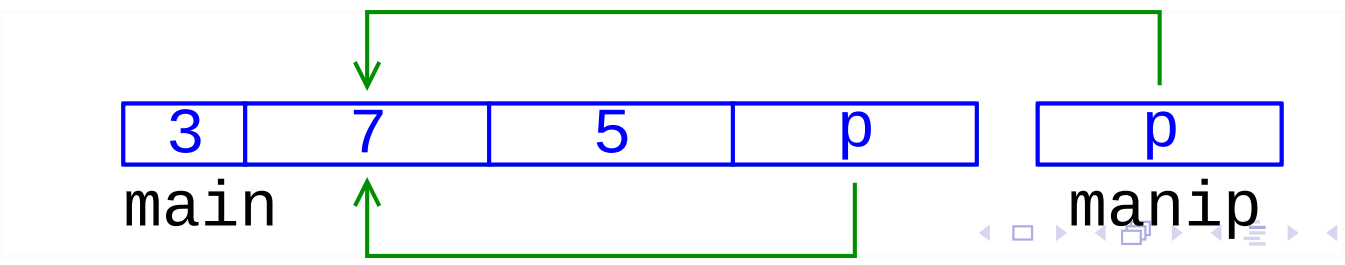


8.3 Les Pointeurs

```
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
```

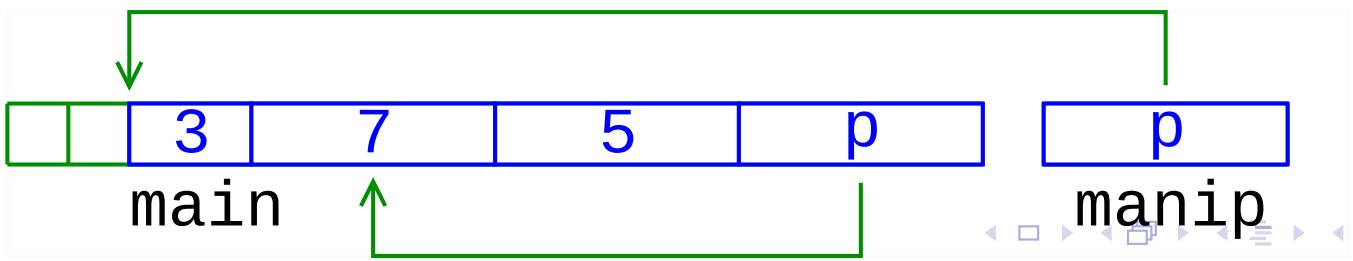


8.3 Les Pointeurs

```
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
```

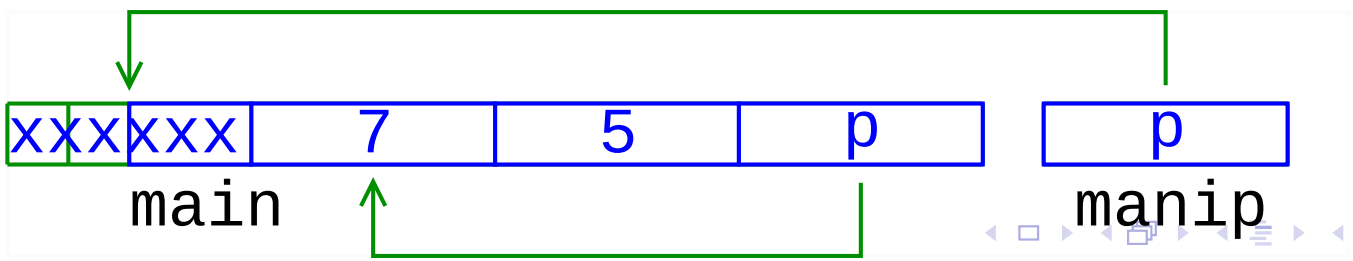


8.3 Les Pointeurs

```
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
```



8.3 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⟩

8.3 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⟩ . . .)

8.3 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⟩;

8.3 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;

8.3 Les Pointeurs

Opérations élémentaires pour des pointeurs:

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

$\longrightarrow$ décrément de l'adresse dans ⟨nom⟩

on peut également écrire ⟨nom⟩ `-= 1;`

8.3 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

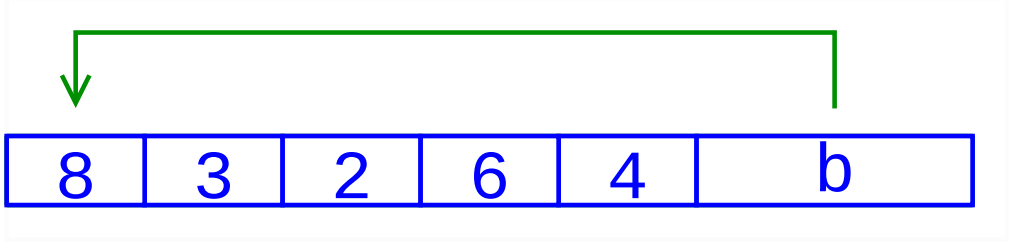
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

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

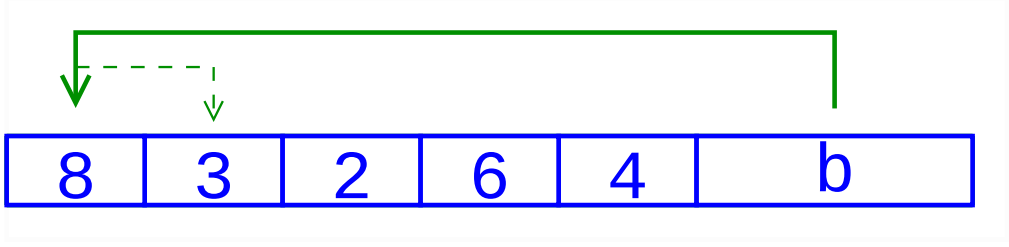
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

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

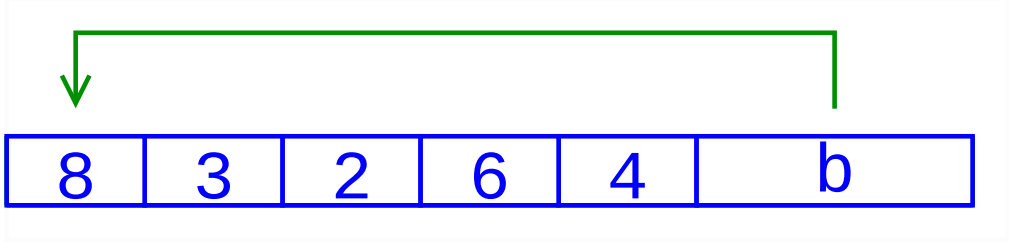
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

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

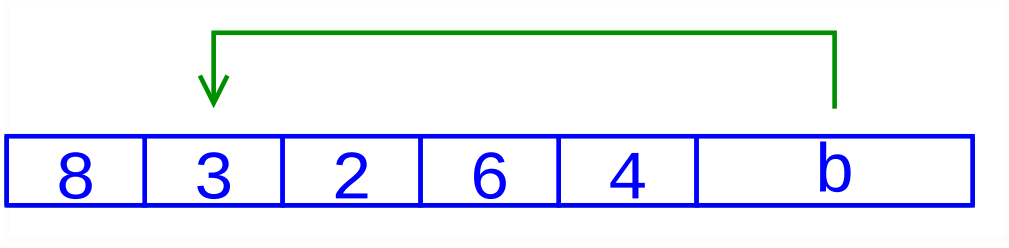
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

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

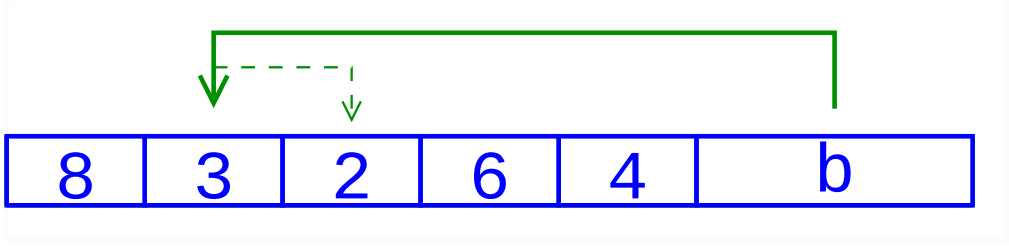
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

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

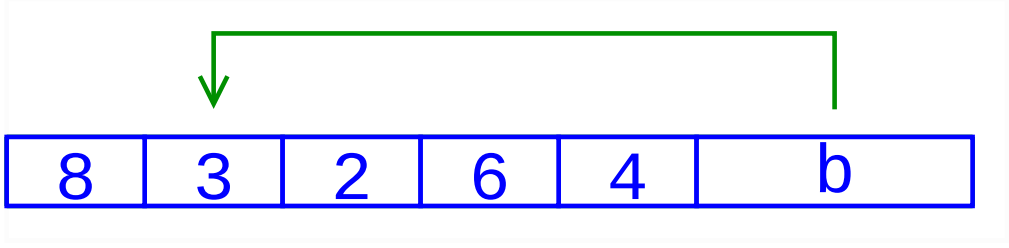
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

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

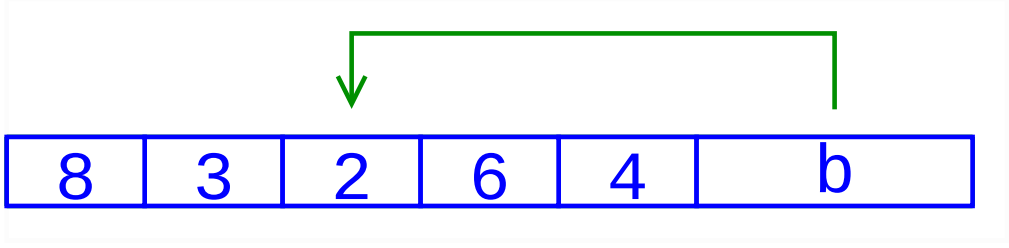
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

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

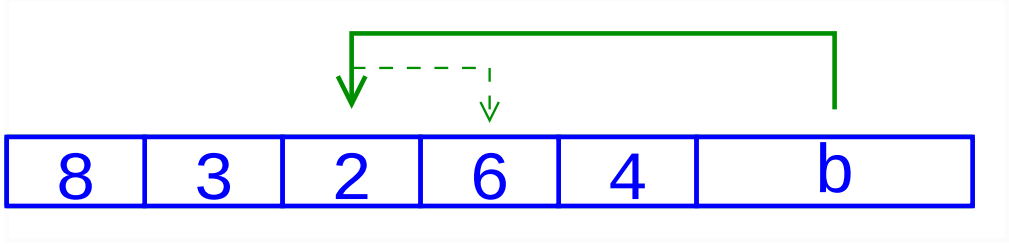
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

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

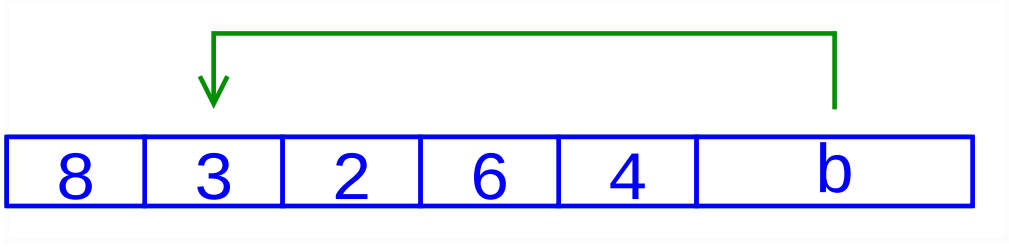
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

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

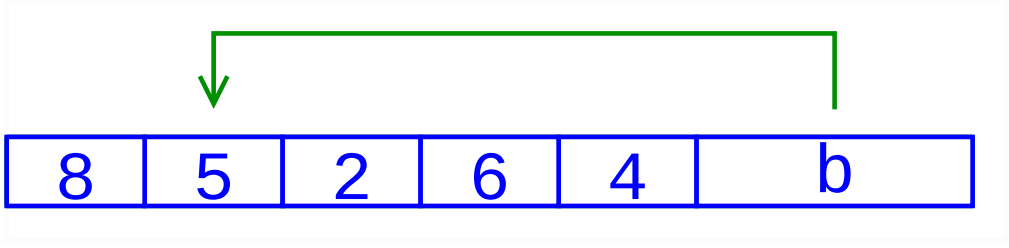
Les variables de tableau se comportent comme des pointeurs qui contiennent l'adresse du premier élément du tableau.

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⟩`;`

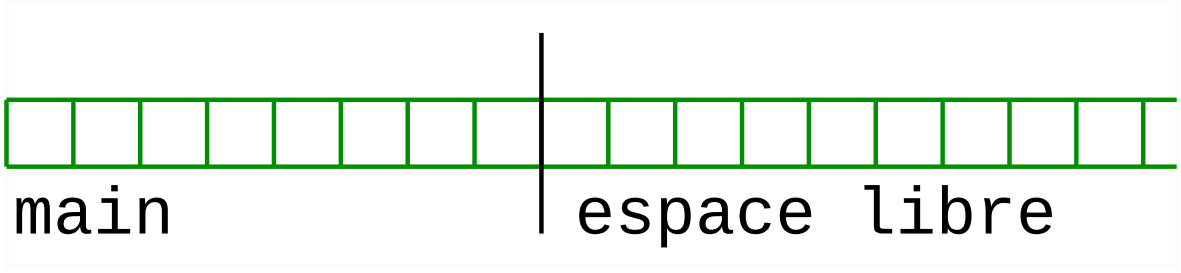
`delete` ⟨nom⟩`;`

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

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

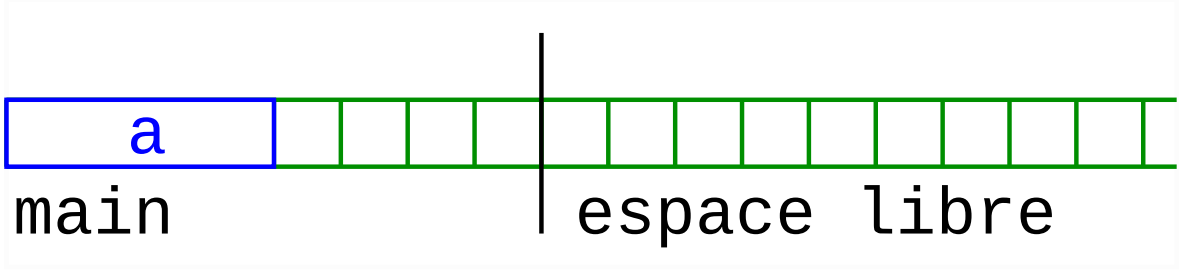
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
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.

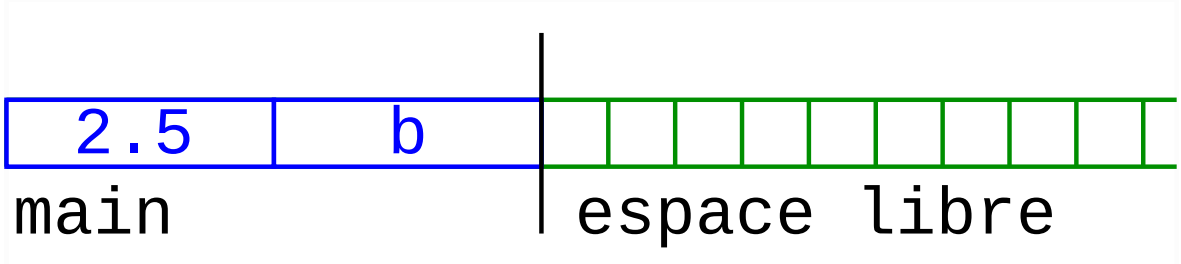
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
}
```



new et delete

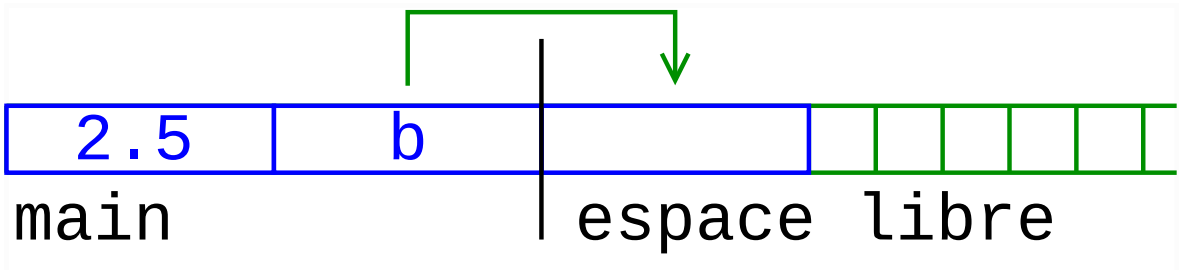
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

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



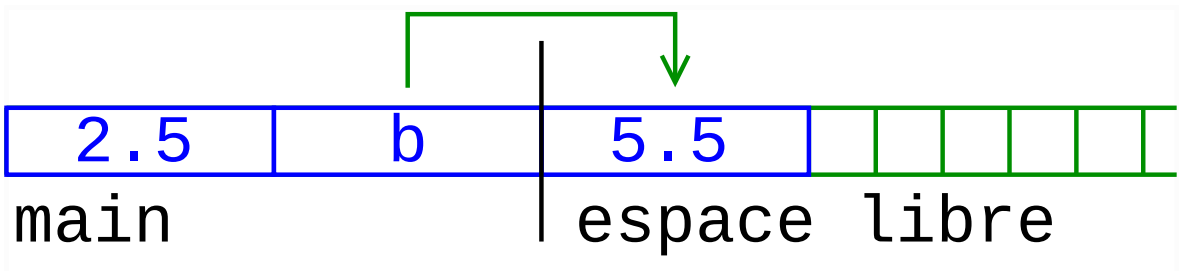
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

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



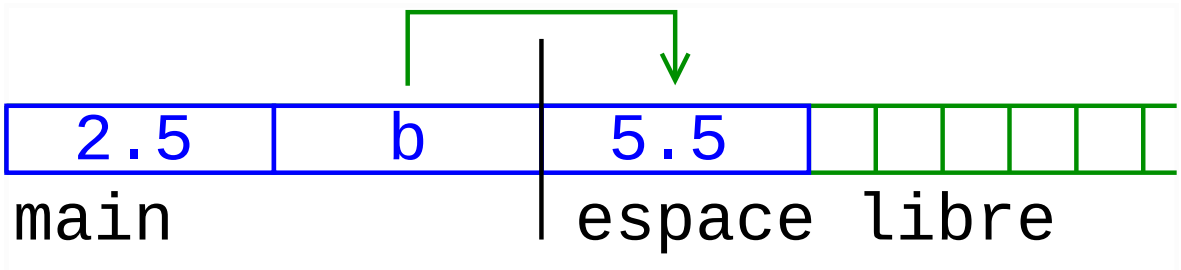
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
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.

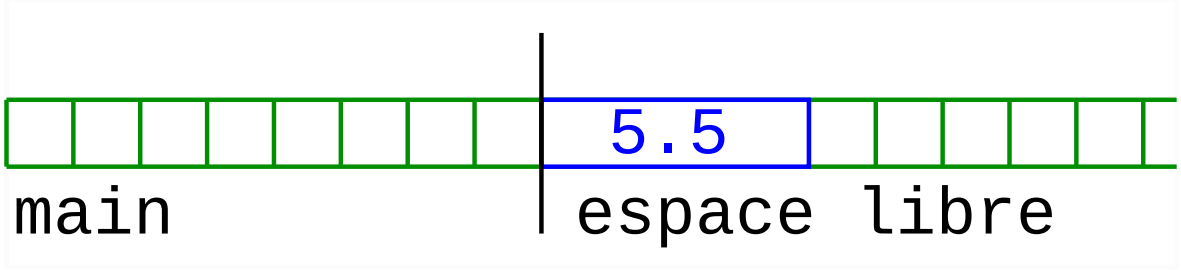
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
}
```

Exécution:

```
5.5
```

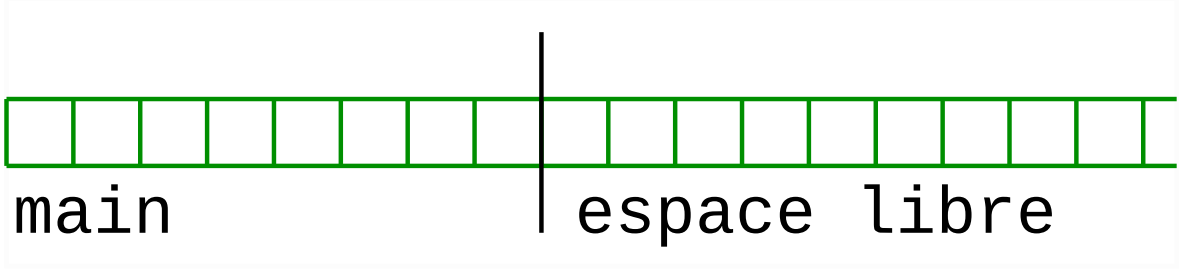
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

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



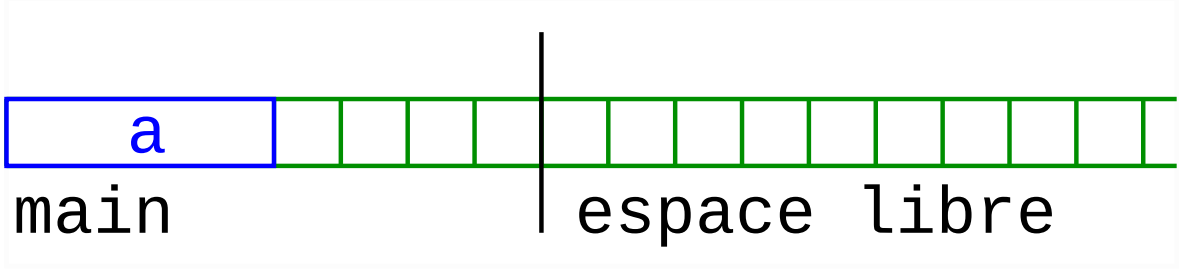
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
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.

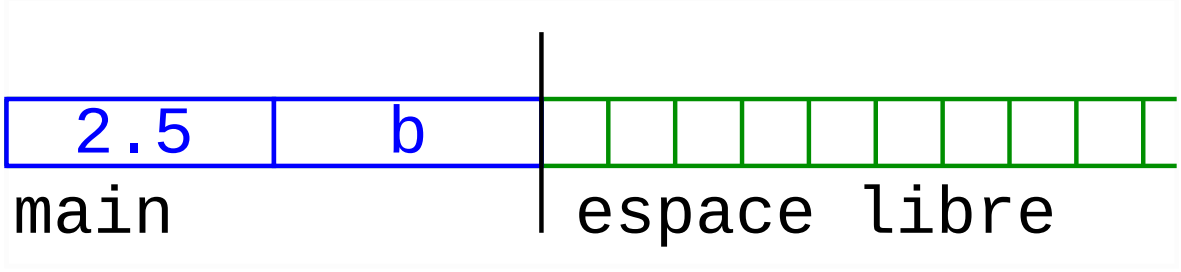
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



new et delete

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

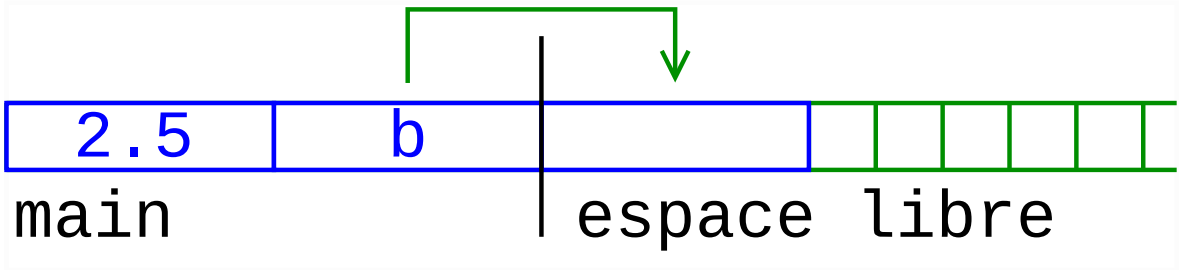
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



new et delete

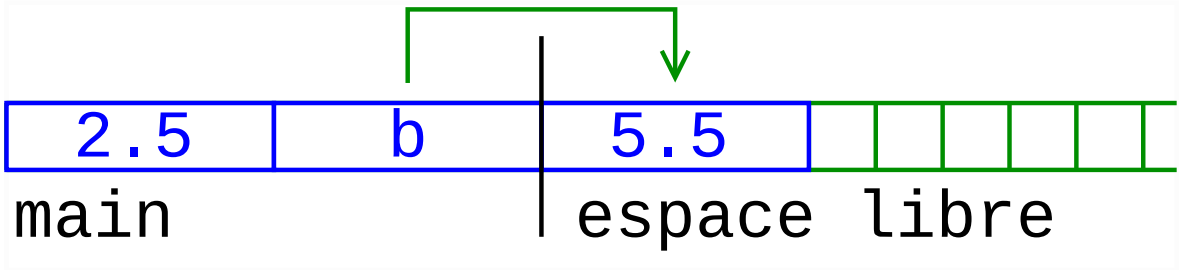
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
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.

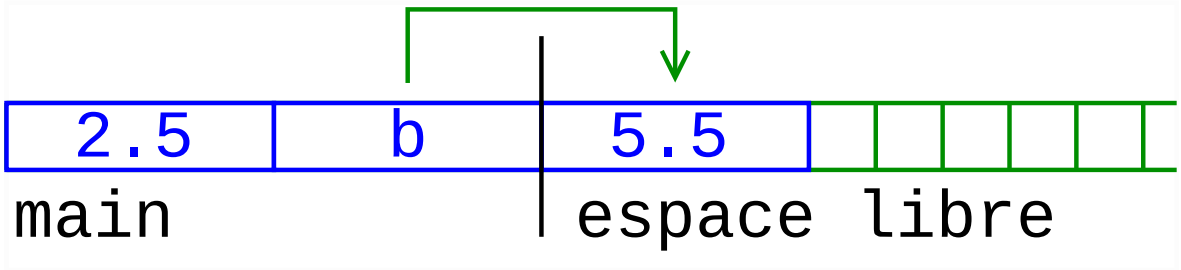
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



new et delete

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

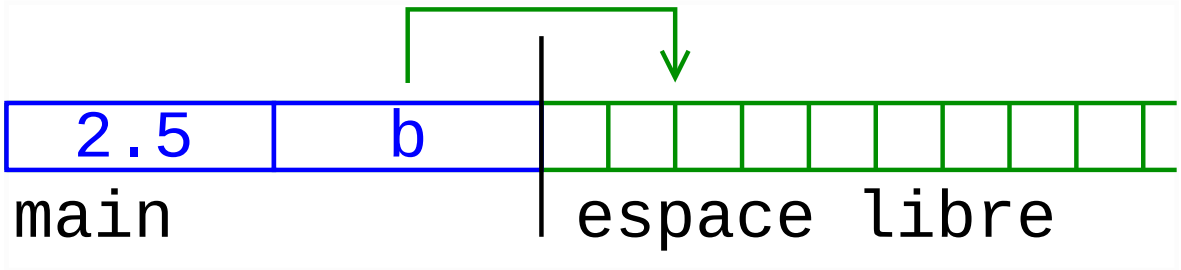
```
int main()
{
  double a = 2.5;
  double *b = new double;
  *b = a + 3;
  cout << *b << endl;
  delete b;
}
```



new et delete

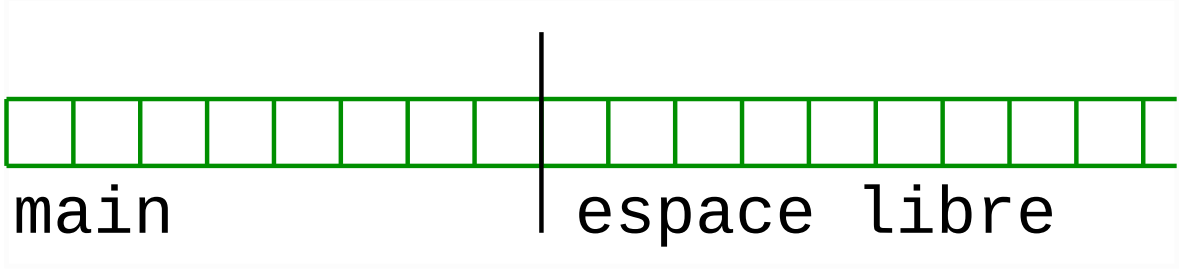
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

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



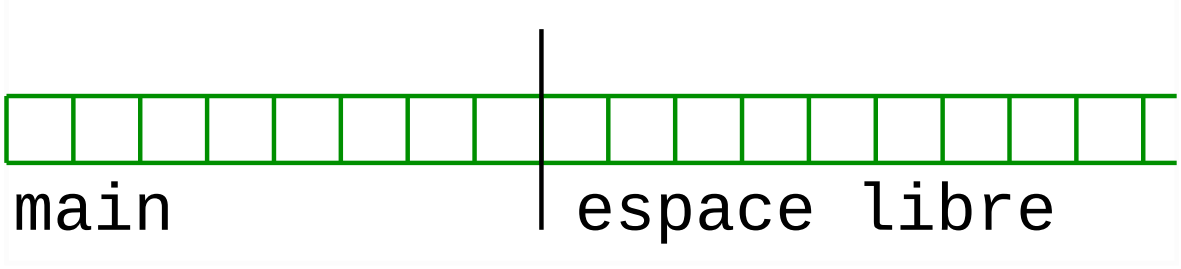
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

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



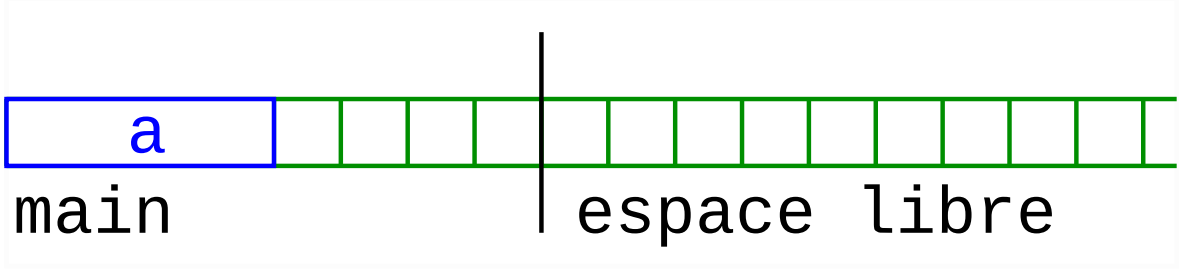
Des pointeurs peuvent aussi être utilisés pour créer des nouvelles variables sur l'espace libre de la mémoire.

```
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.

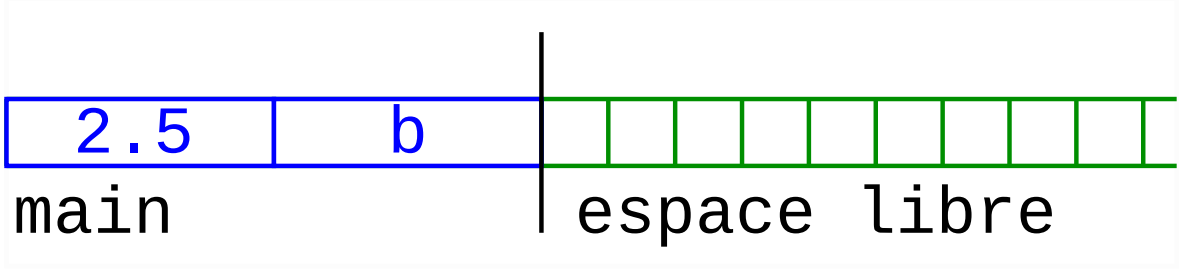
```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



new et delete

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

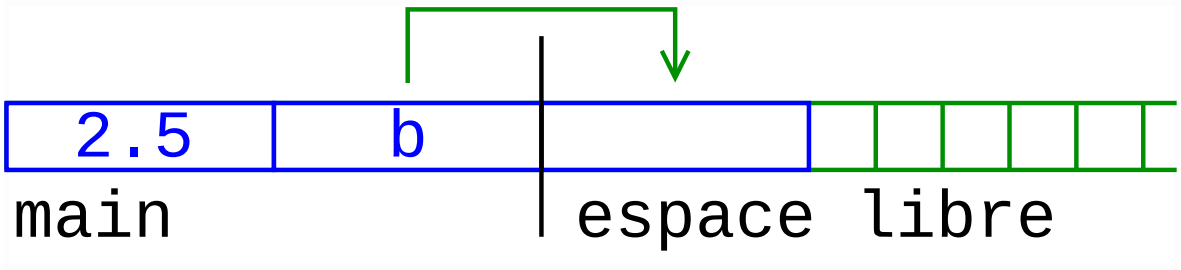
```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



new et delete

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

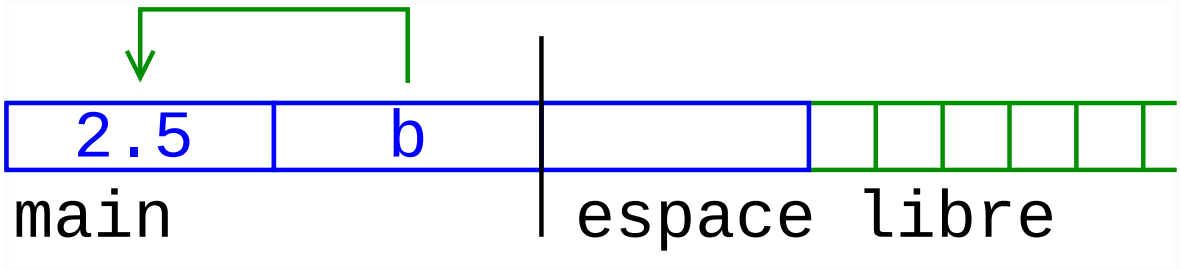
```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



new et delete

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

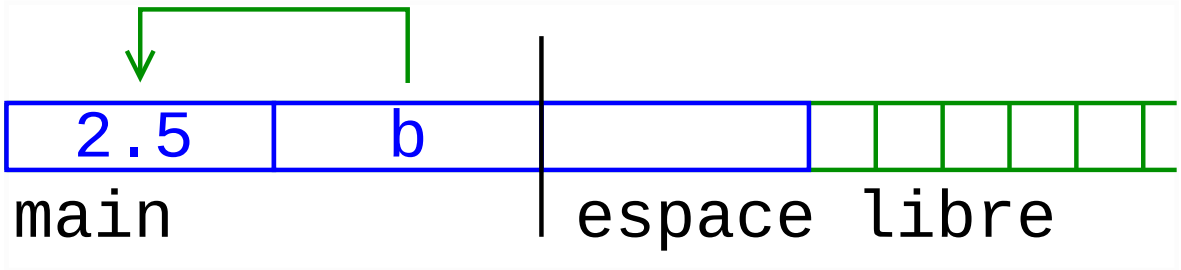
```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



new et delete

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

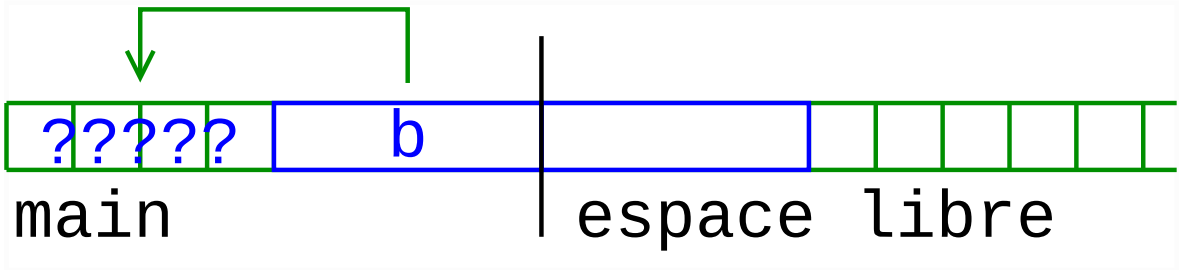
```
int main()
{
  double a = 2.5;
  double *b = new double;
  b = &a;
  cout << *b << endl;
  delete b;
}
```



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 ??

8.4 Les tableaux dynamiques

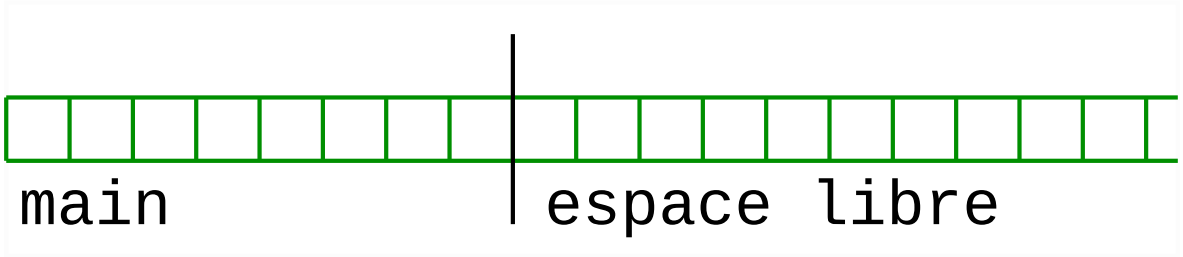
`new` et `delete` peuvent aussi être utilisés pour créer des tableaux “dynamiques” sur l’espace libre

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

8.4 Les tableaux dynamiques

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

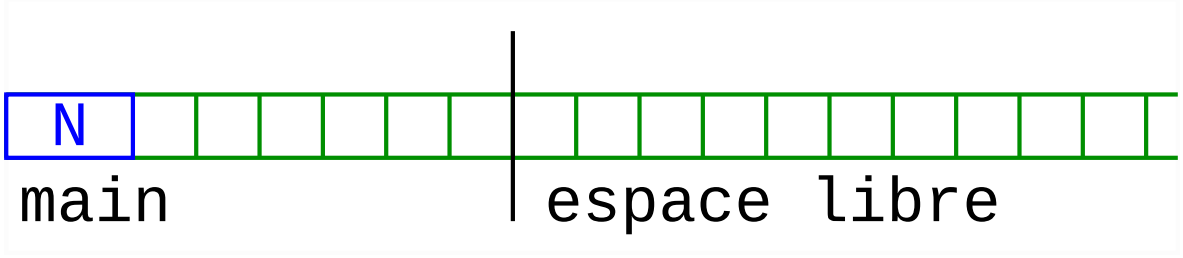
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



8.4 Les tableaux dynamiques

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

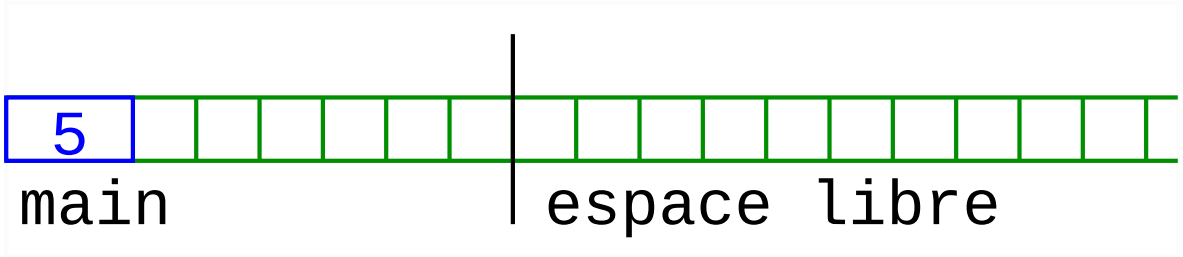
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



8.4 Les tableaux dynamiques

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

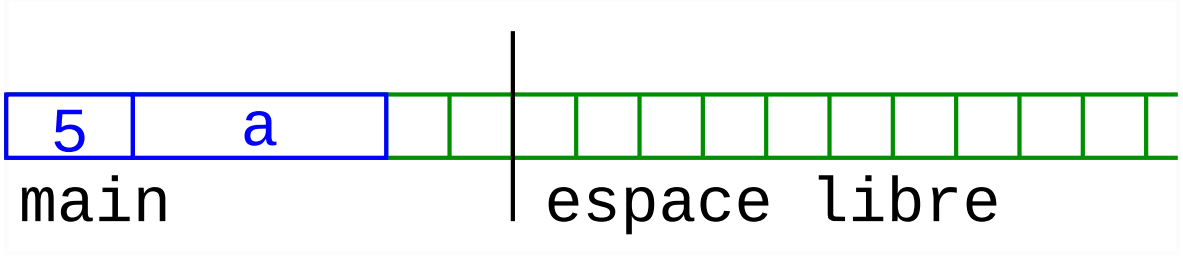
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



8.4 Les tableaux dynamiques

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

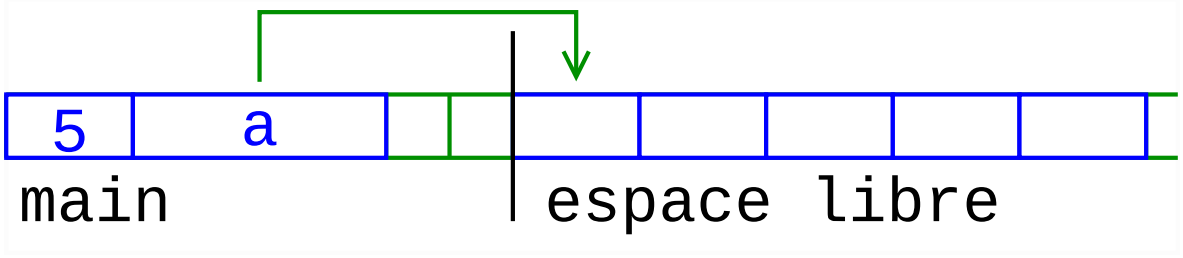
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



8.4 Les tableaux dynamiques

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

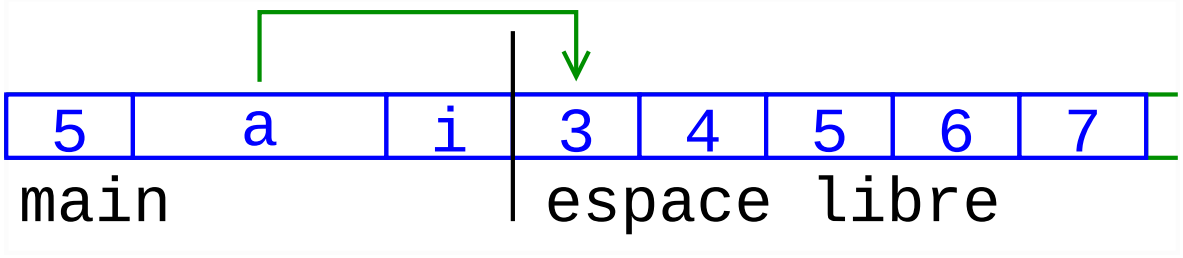
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



8.4 Les tableaux dynamiques

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

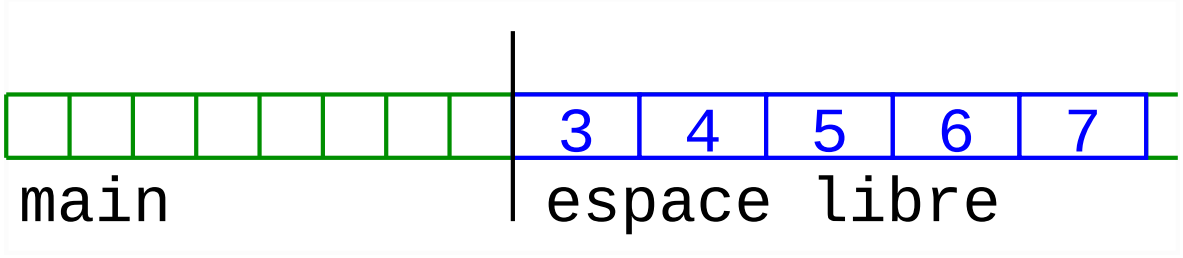
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



8.4 Les tableaux dynamiques

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

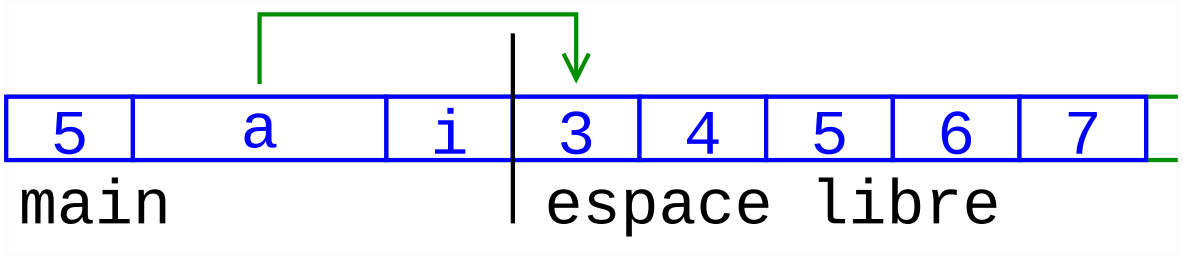
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
}
```



8.4 Les tableaux dynamiques

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

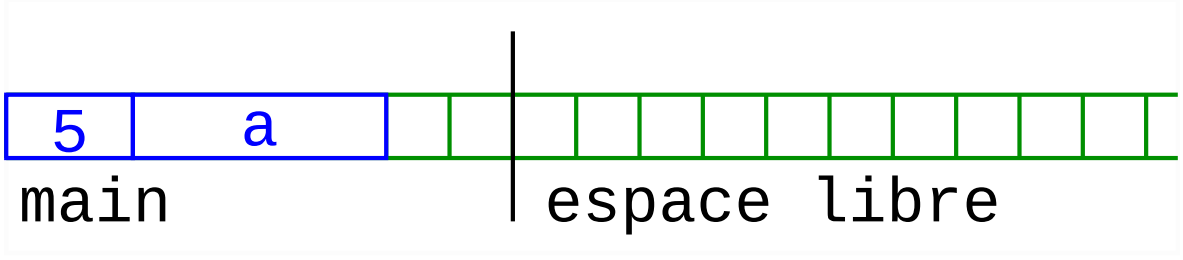
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
  delete[] a;
}
```



8.4 Les tableaux dynamiques

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

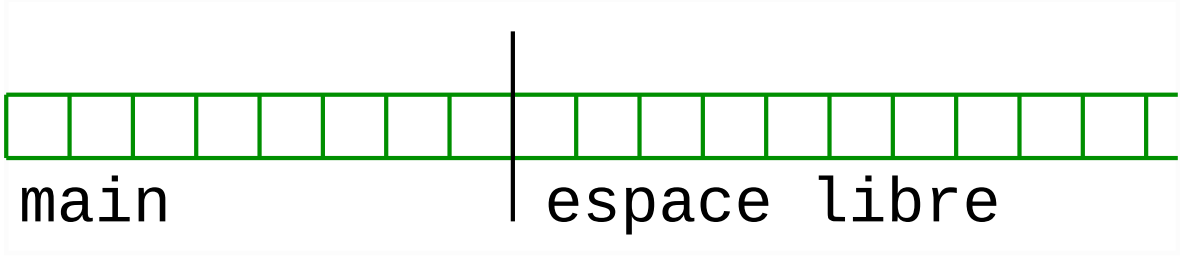
```
int main()
{
  int N;
  cin >> N;
  int *a = new int[ N ];
  for ( int i = 0; i < N; i++ )
    a[ i ] = i + 3;
  delete[] a;
}
```



8.4 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;
}
```



8.4 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⟩ `]`.

8.4 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`

8.4 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 ] );
}
```

8.4 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`

8.4 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`

8.4 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 );
```

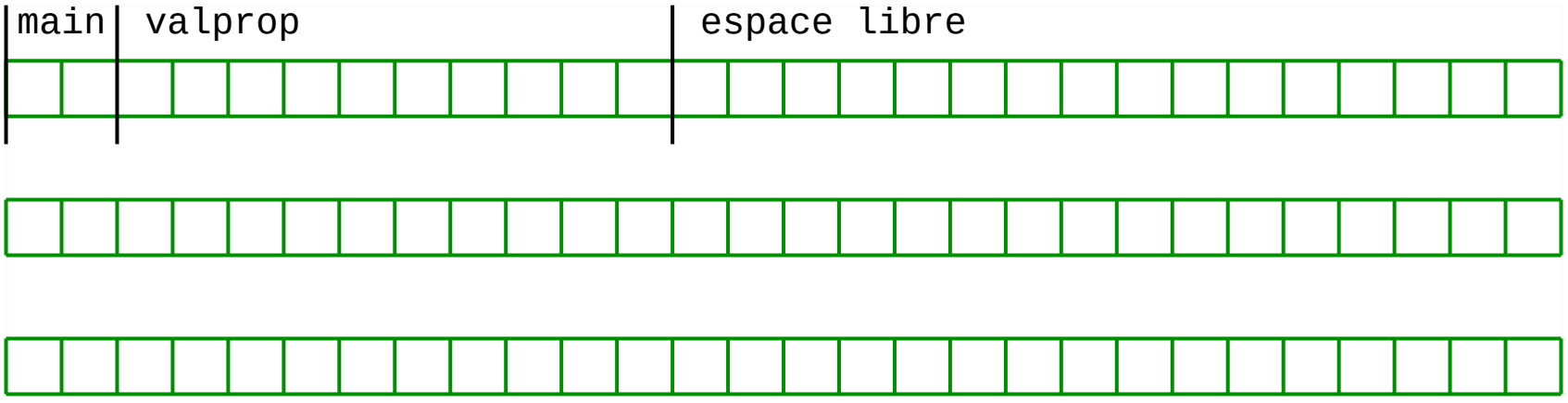
$\longrightarrow$ déterminer la taille optimale de la matrice

8.4 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;
}
```

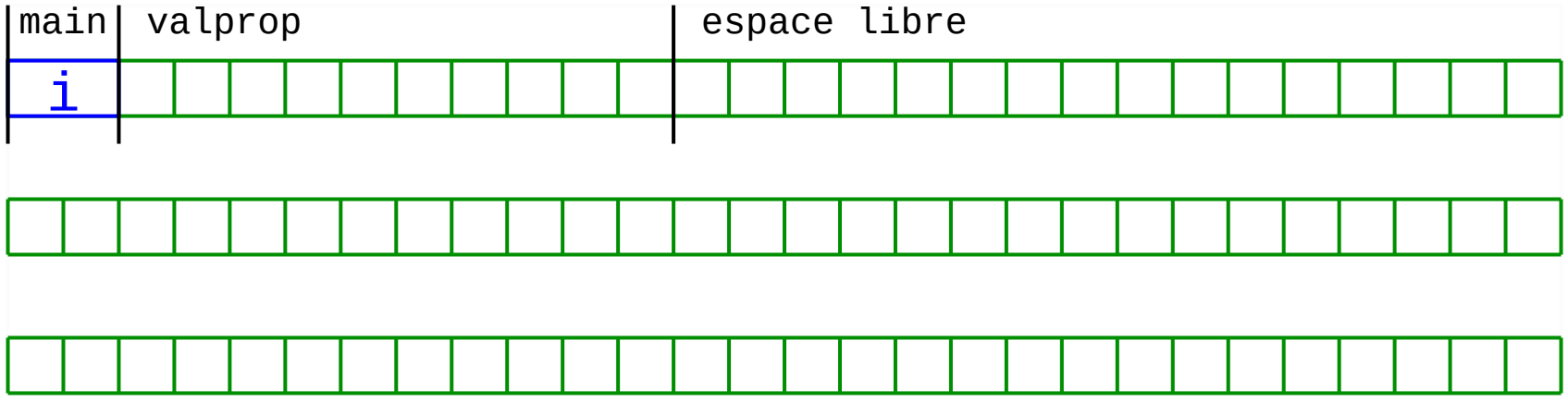
8.4 Les tableaux dynamiques

```
#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;
}
```



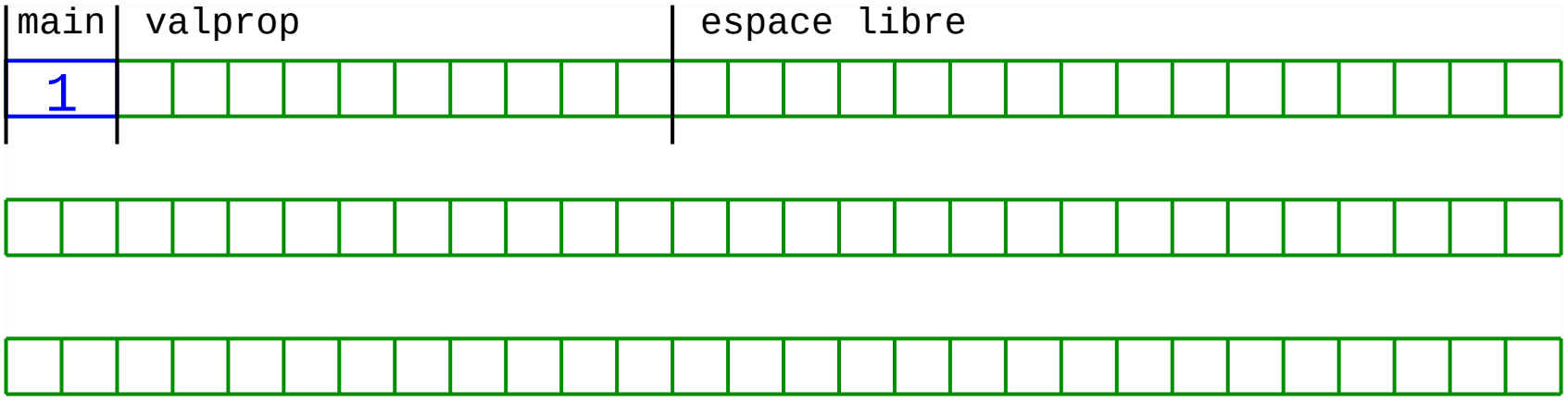
8.4 Les tableaux dynamiques

```
#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;
}
```



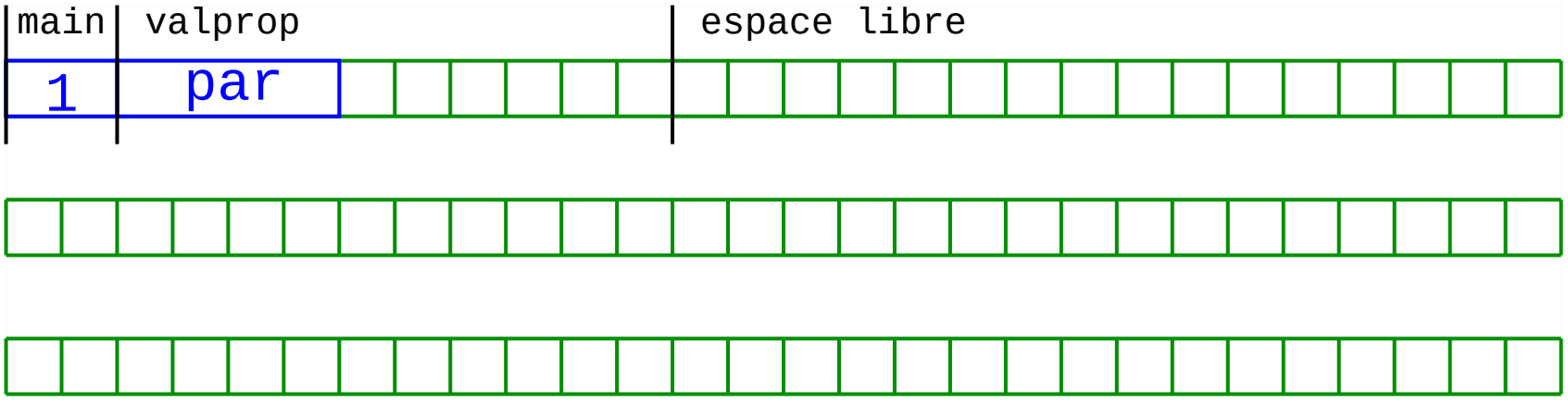
8.4 Les tableaux dynamiques

```
#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;
}
```



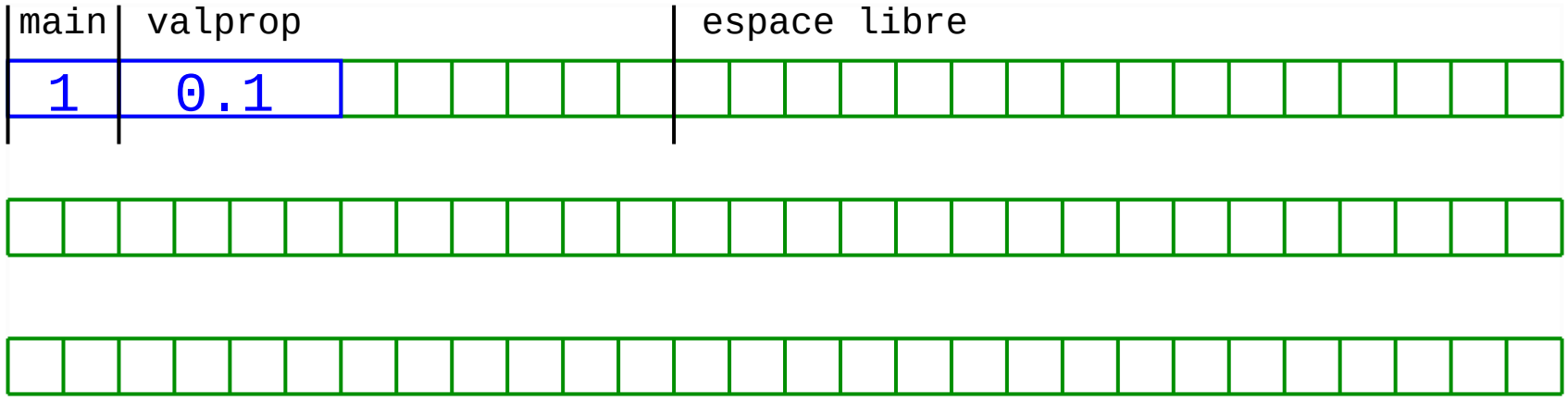
8.4 Les tableaux dynamiques

```
#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;
}
```



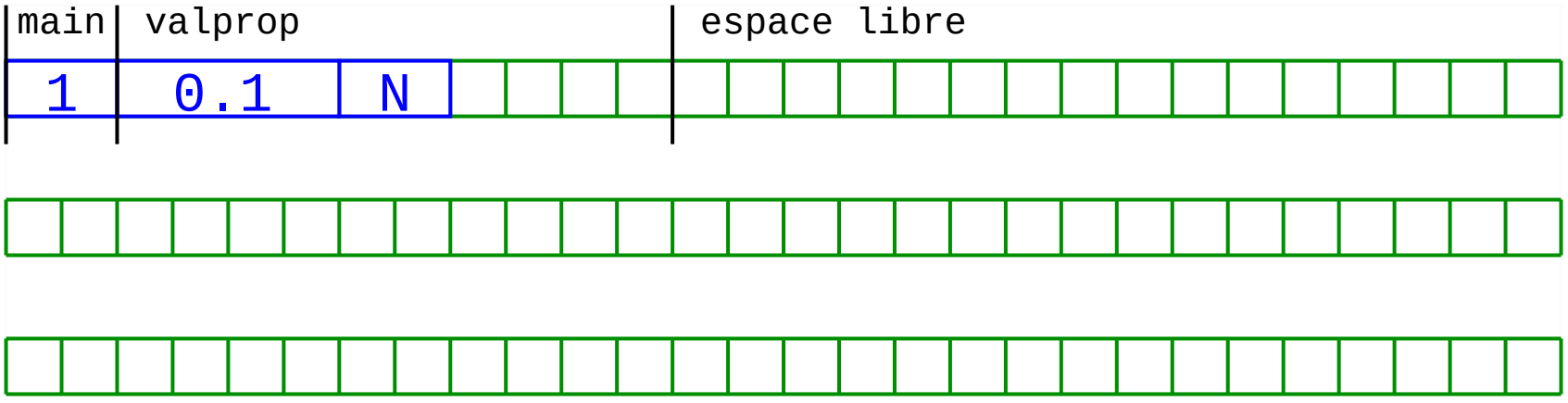
8.4 Les tableaux dynamiques

```
#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;
}
```



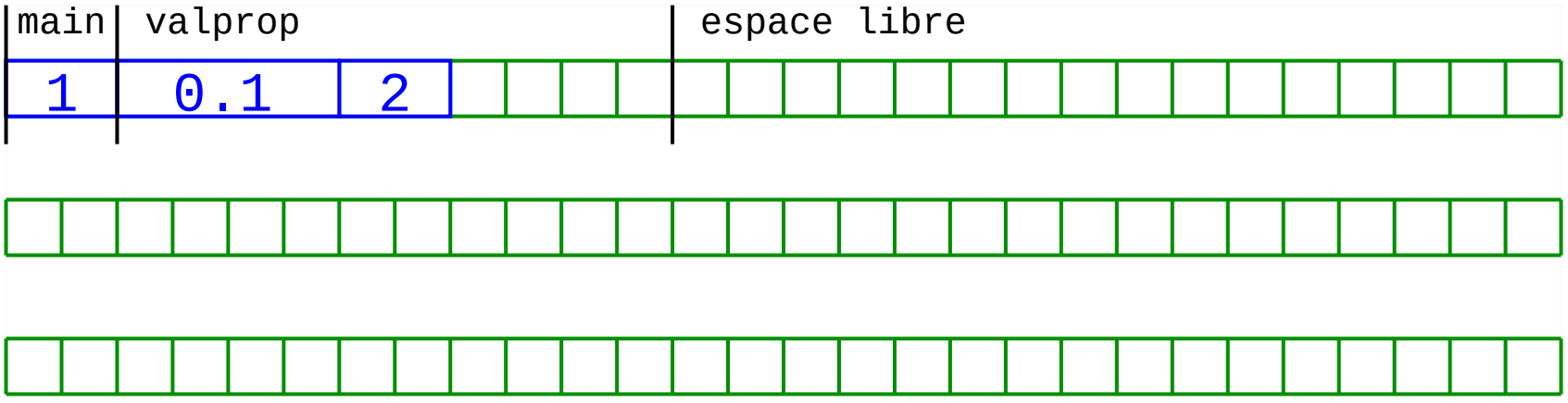
8.4 Les tableaux dynamiques

```
#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;
}
```



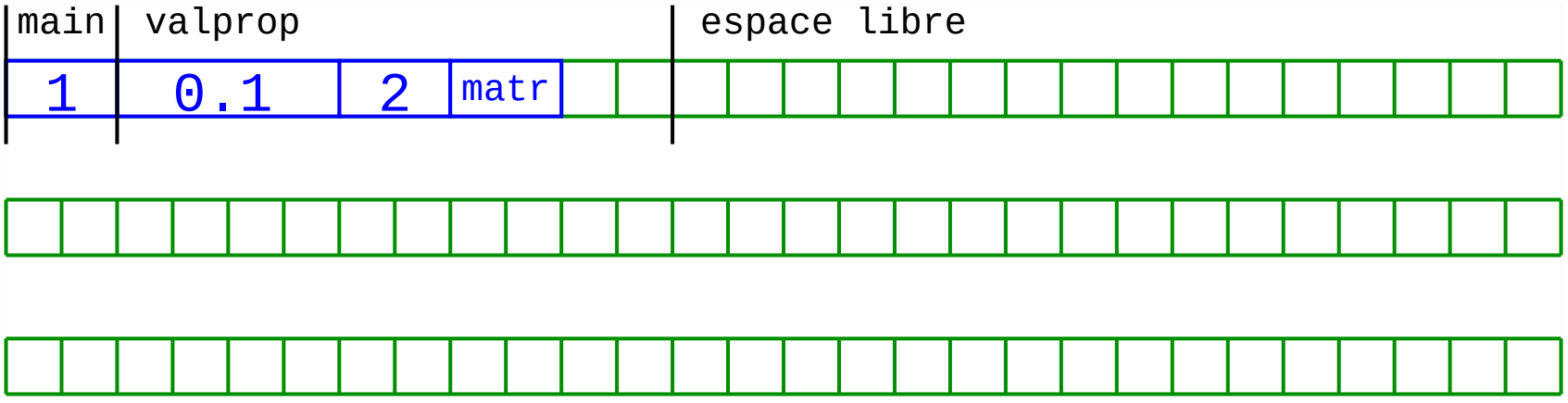
8.4 Les tableaux dynamiques

```
#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;
}
```



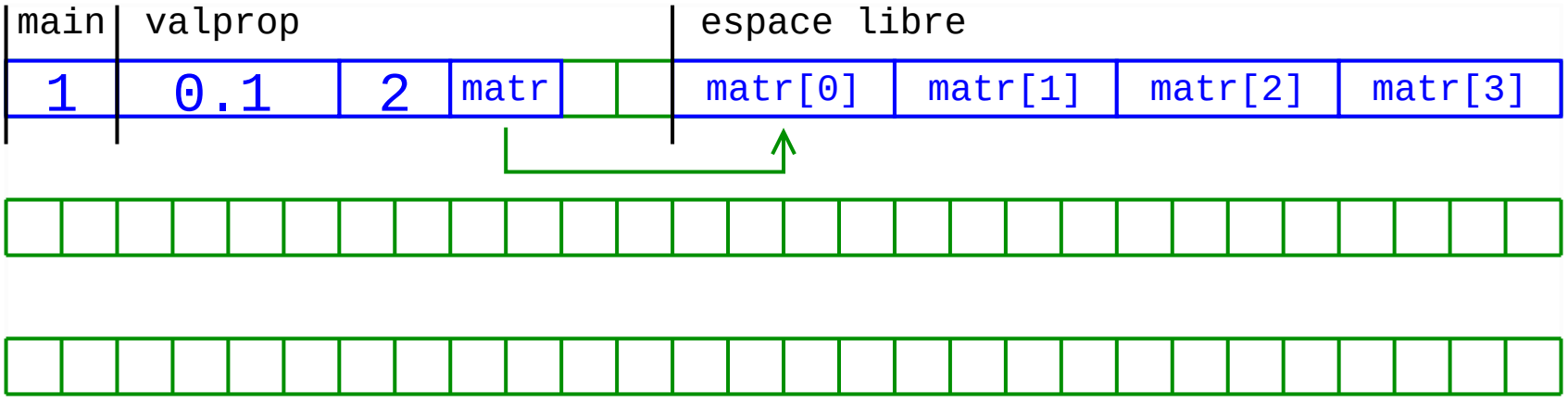
8.4 Les tableaux dynamiques

```
#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;
}
```



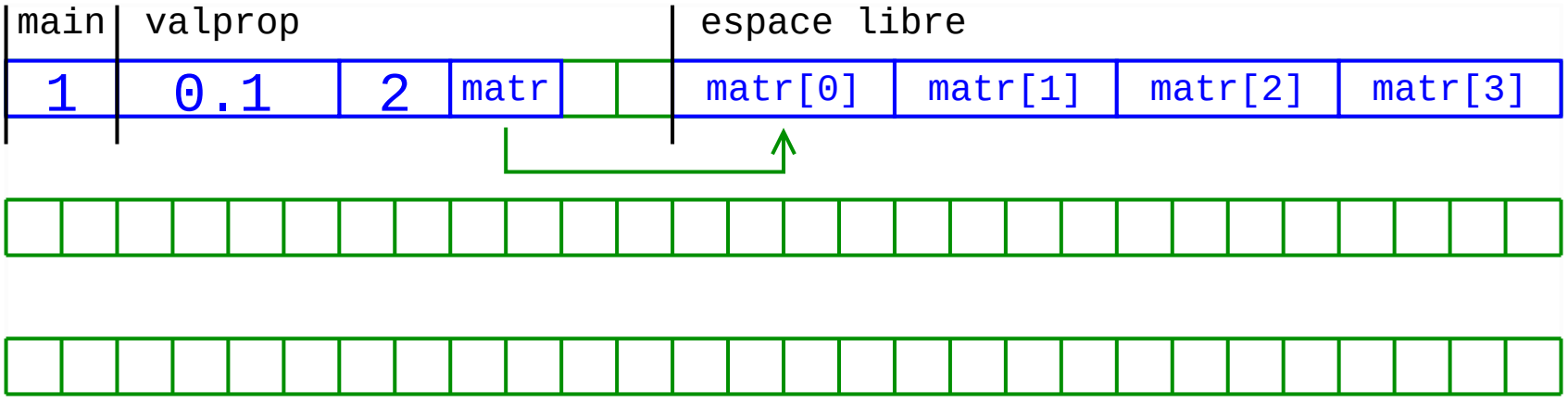
8.4 Les tableaux dynamiques

```
#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;
}
```



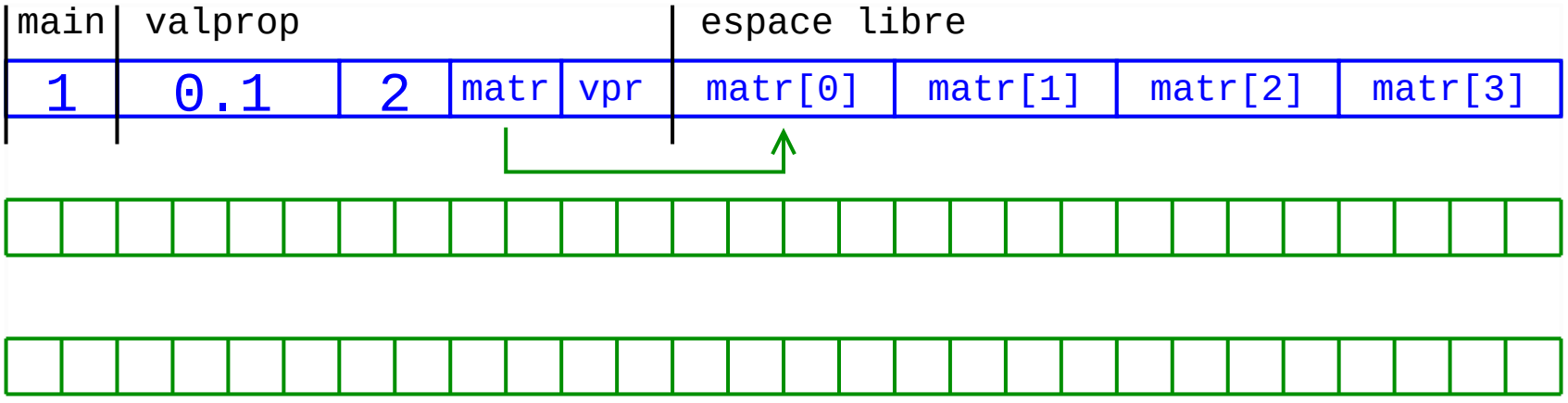
8.4 Les tableaux dynamiques

```
#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;
}
```



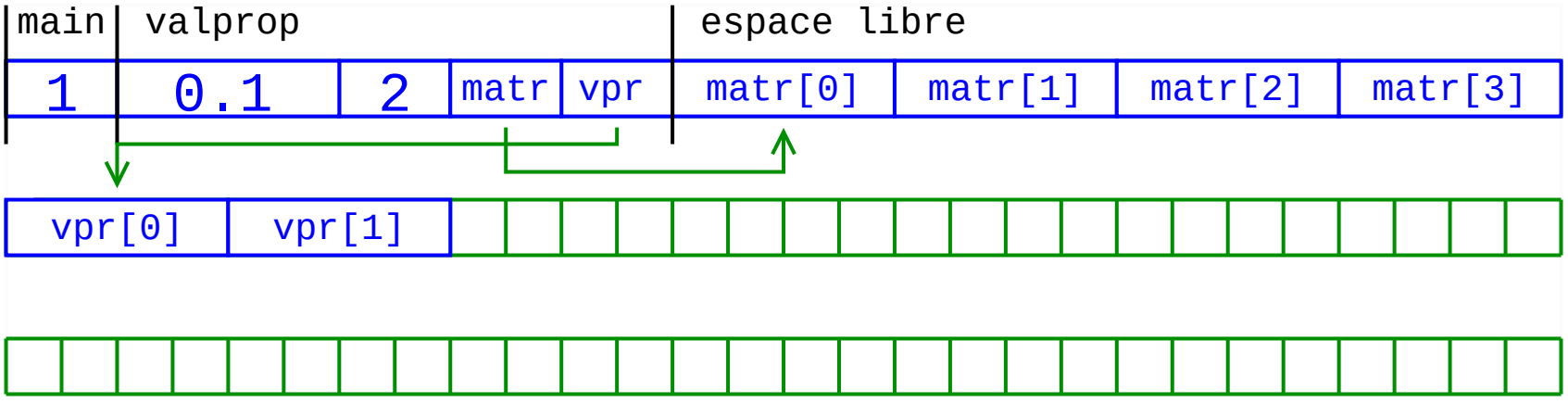
8.4 Les tableaux dynamiques

```
#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;
}
```



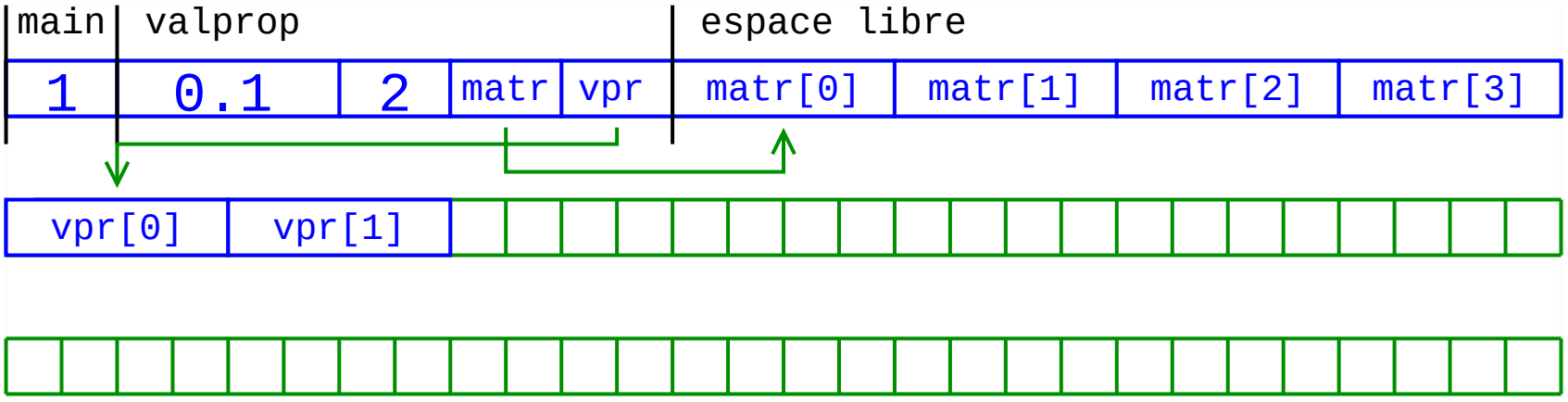
8.4 Les tableaux dynamiques

```
#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;
}
```



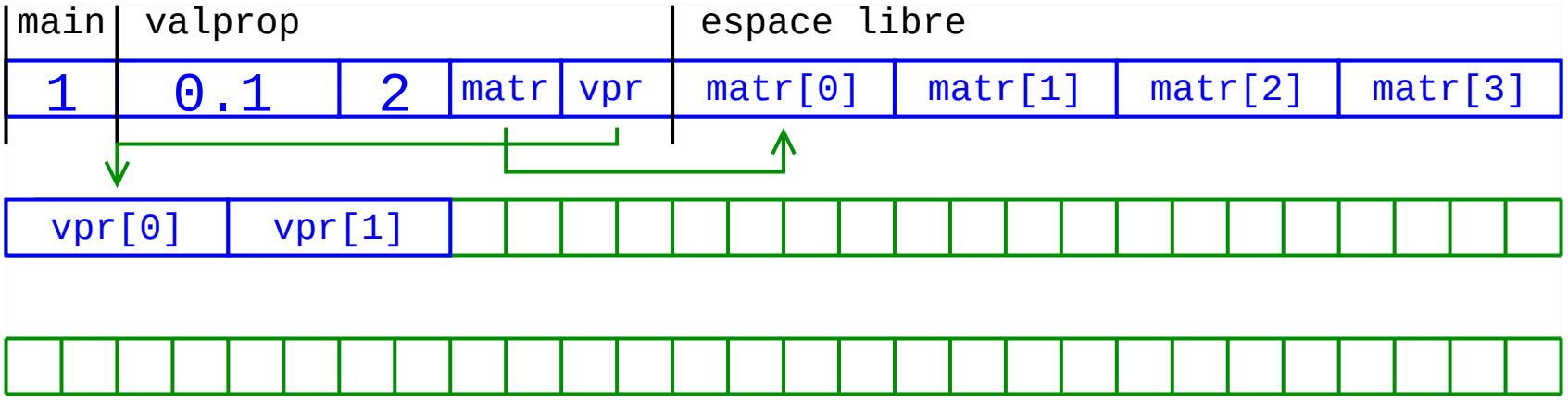
8.4 Les tableaux dynamiques

```
#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;
}
```



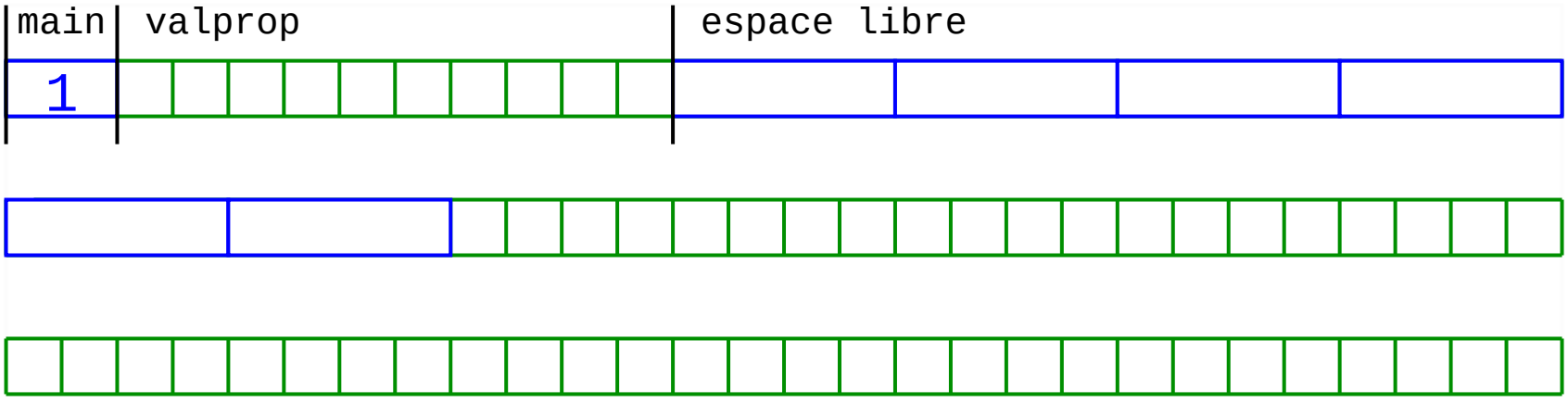
8.4 Les tableaux dynamiques

```
#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;
}
```



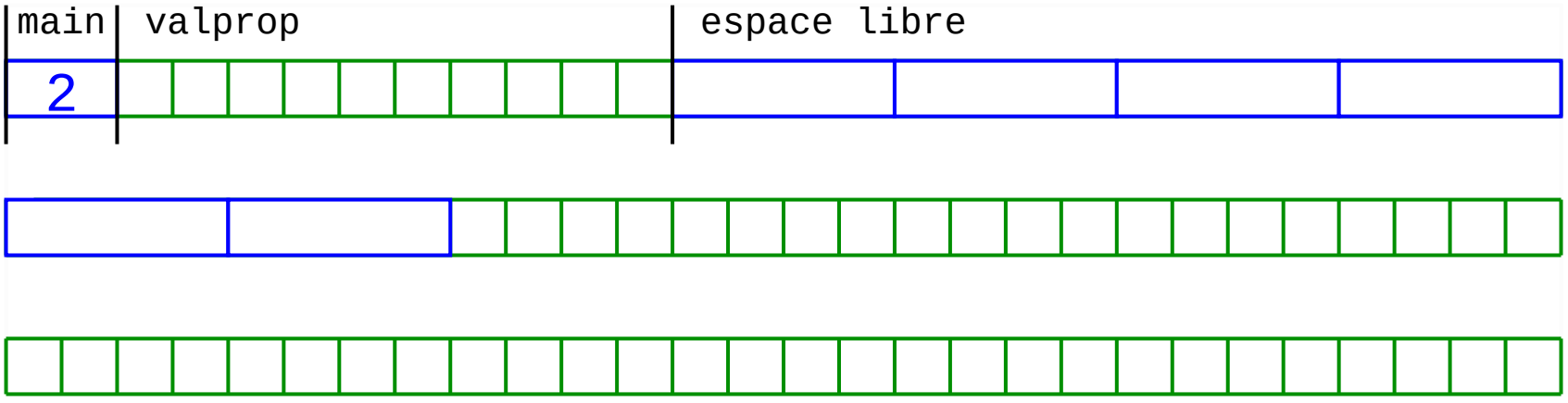
8.4 Les tableaux dynamiques

```
#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;
}
```



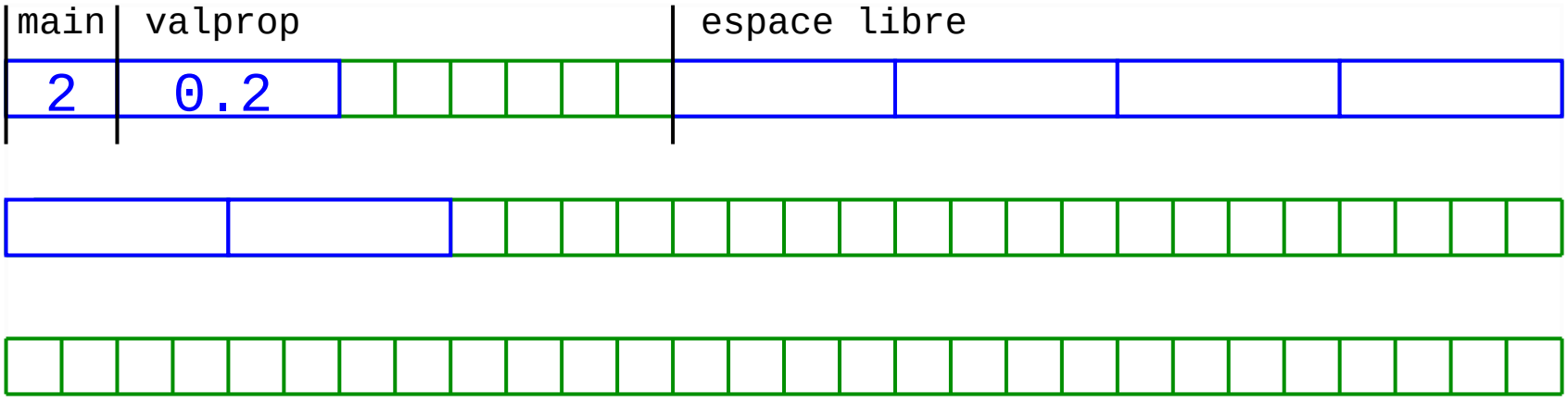
8.4 Les tableaux dynamiques

```
#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;
}
```



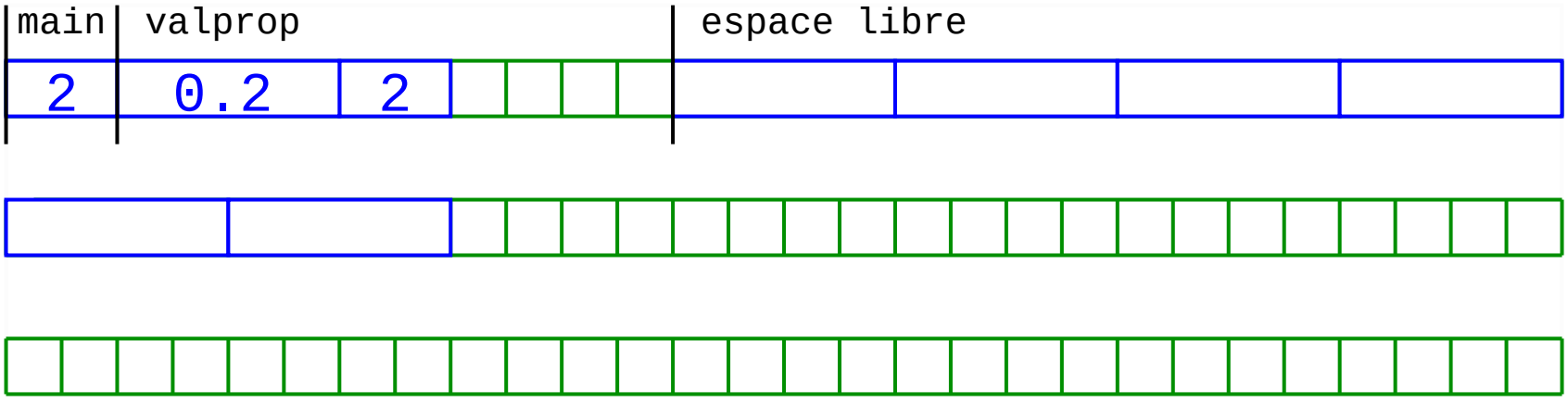
8.4 Les tableaux dynamiques

```
#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;
}
```



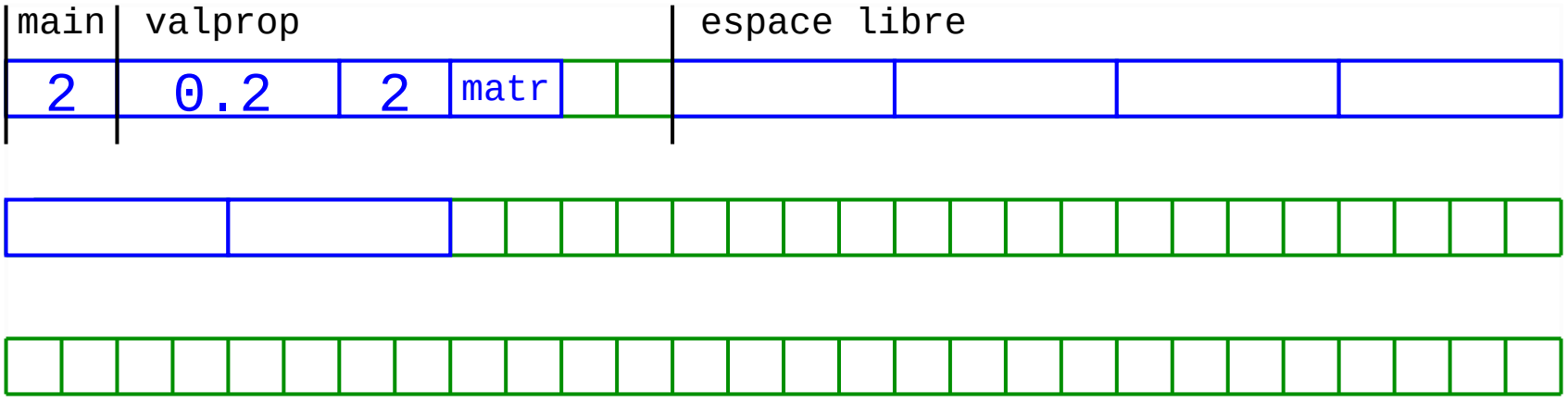
8.4 Les tableaux dynamiques

```
#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;
}
```



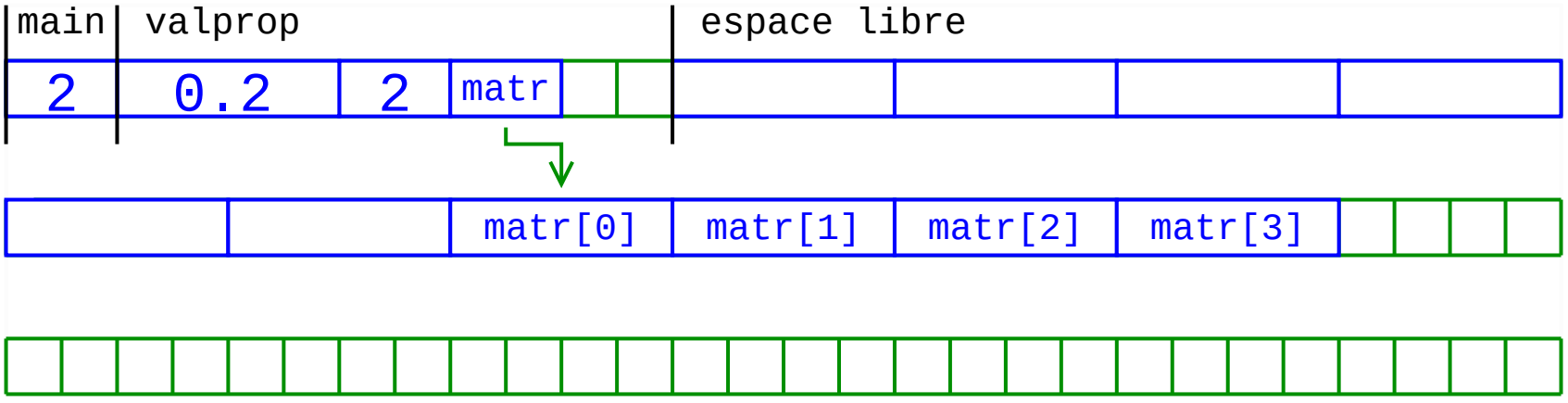
8.4 Les tableaux dynamiques

```
#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;
}
```



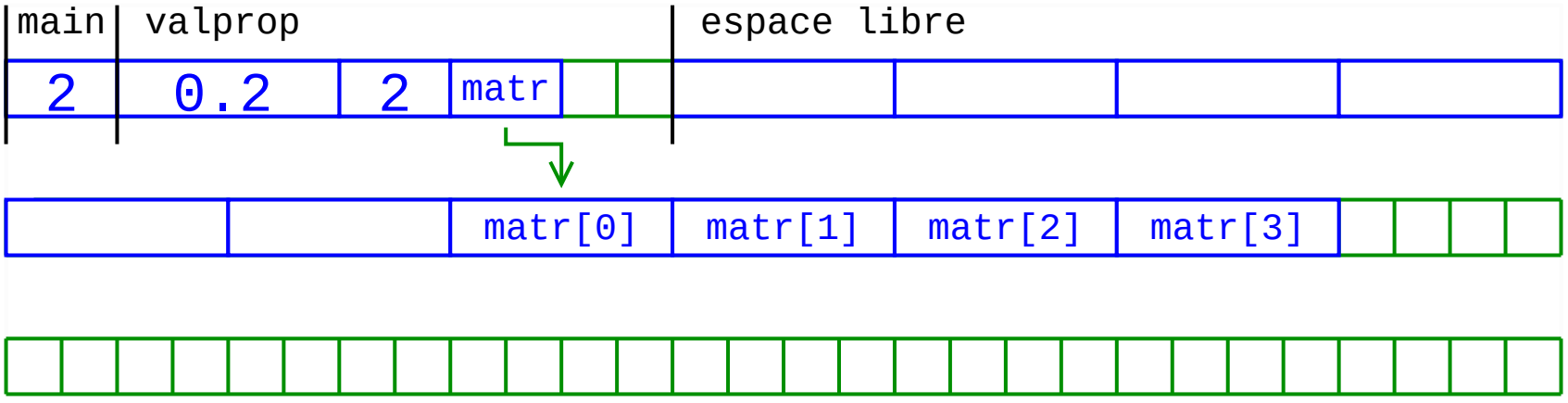
8.4 Les tableaux dynamiques

```
#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;
}
```



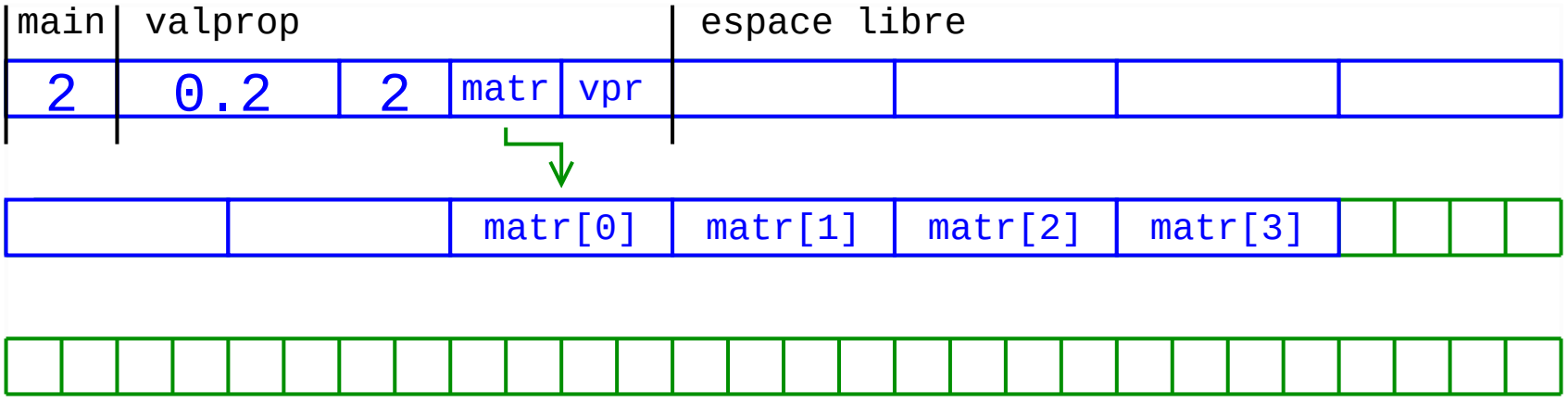
8.4 Les tableaux dynamiques

```
#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;
}
```



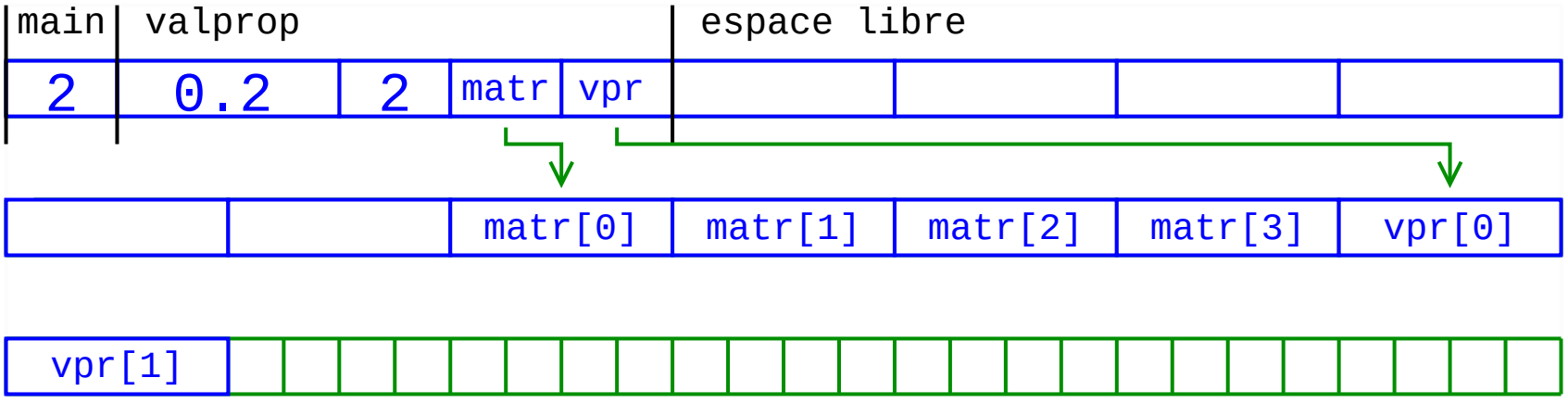
8.4 Les tableaux dynamiques

```
#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;
}
```



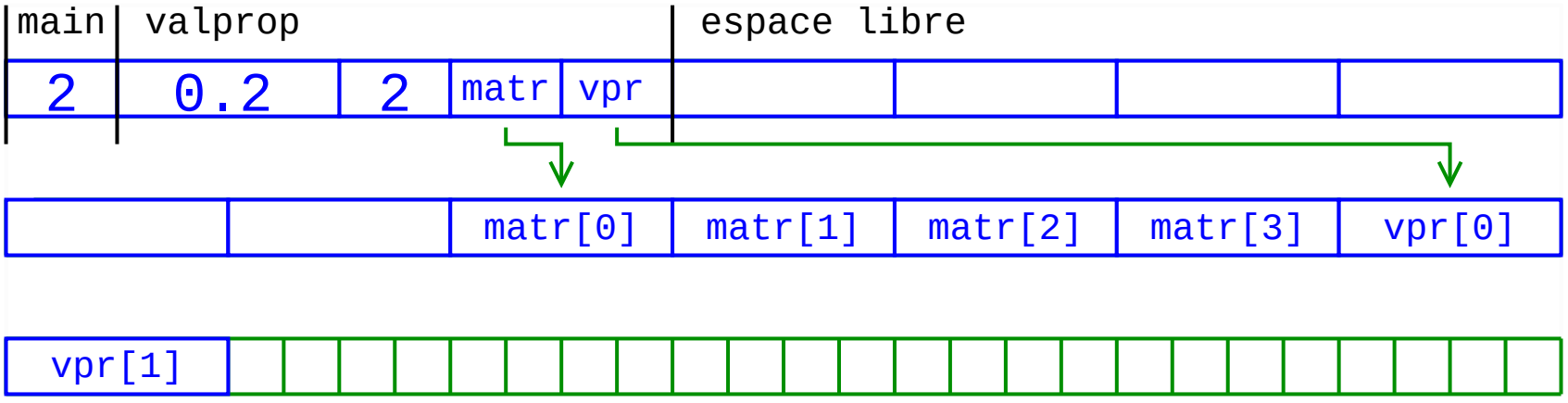
8.4 Les tableaux dynamiques

```
#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;
}
```



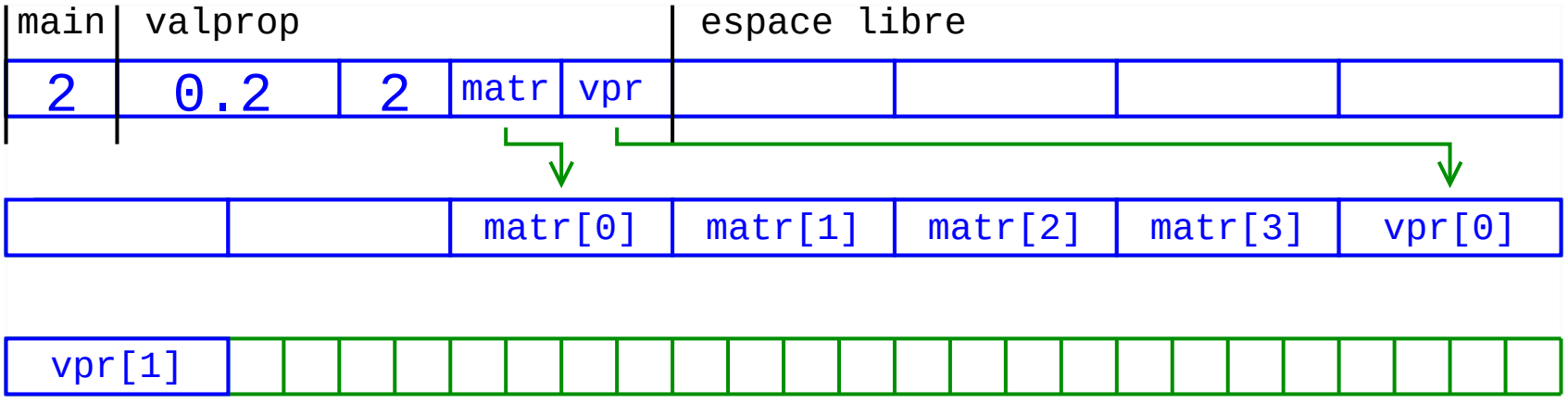
8.4 Les tableaux dynamiques

```
#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;
}
```



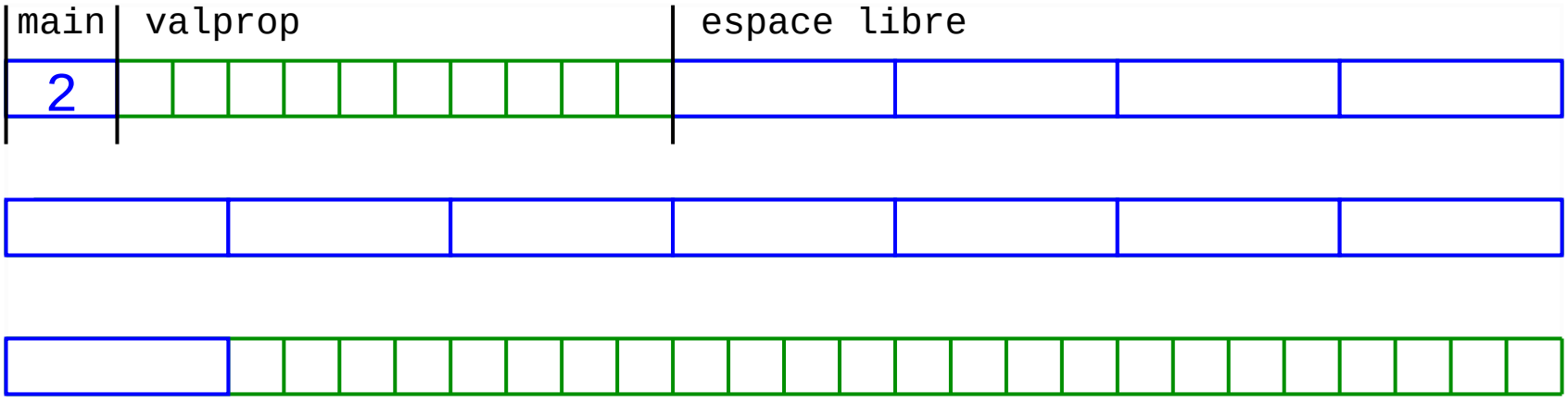
8.4 Les tableaux dynamiques

```
#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;
}
```



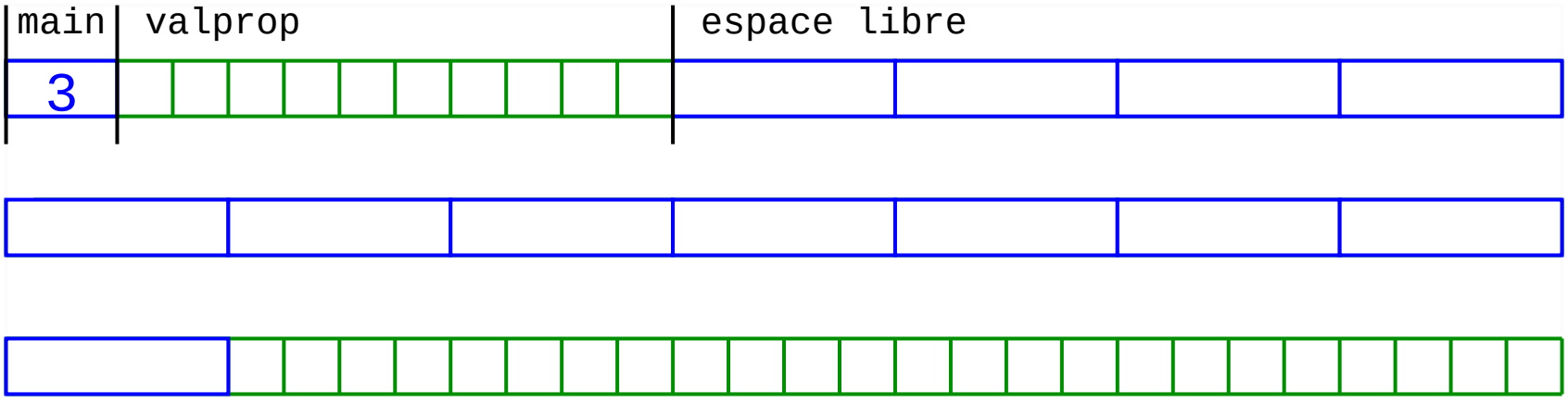
8.4 Les tableaux dynamiques

```
#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;
}
```



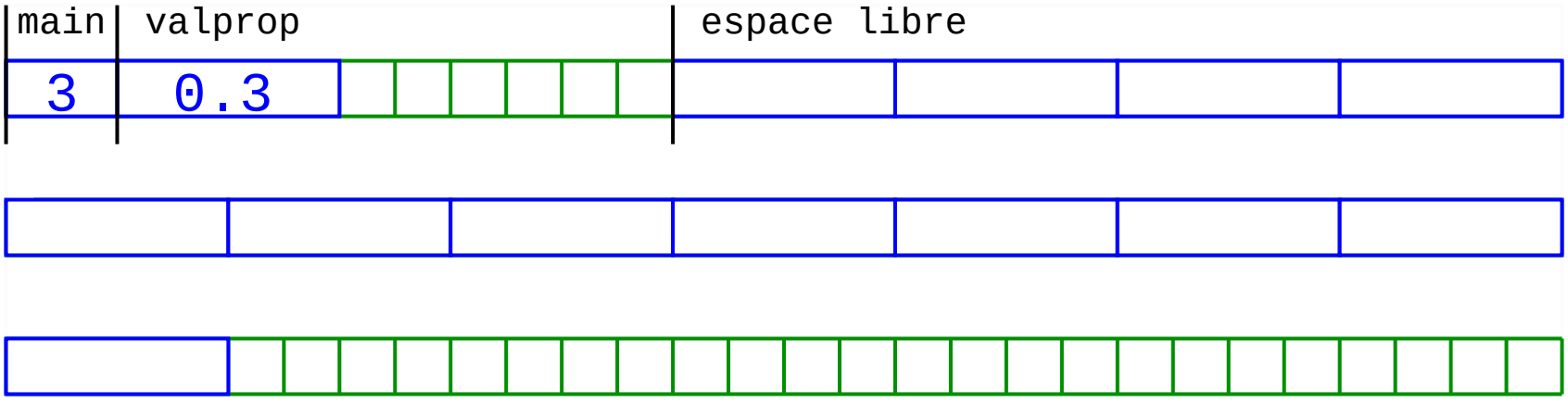
8.4 Les tableaux dynamiques

```
#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;
}
```



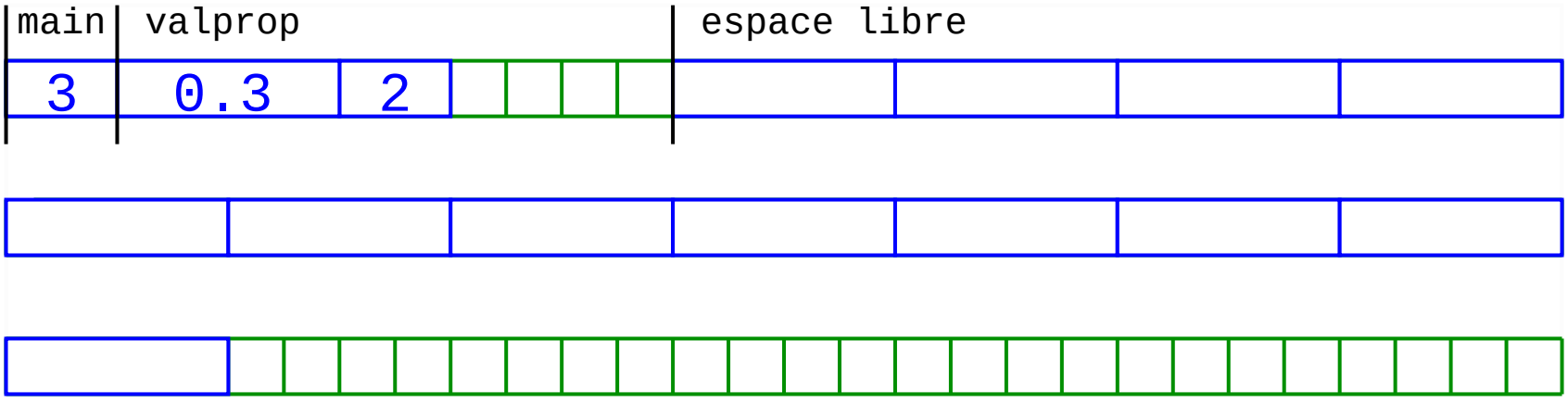
8.4 Les tableaux dynamiques

```
#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;
}
```



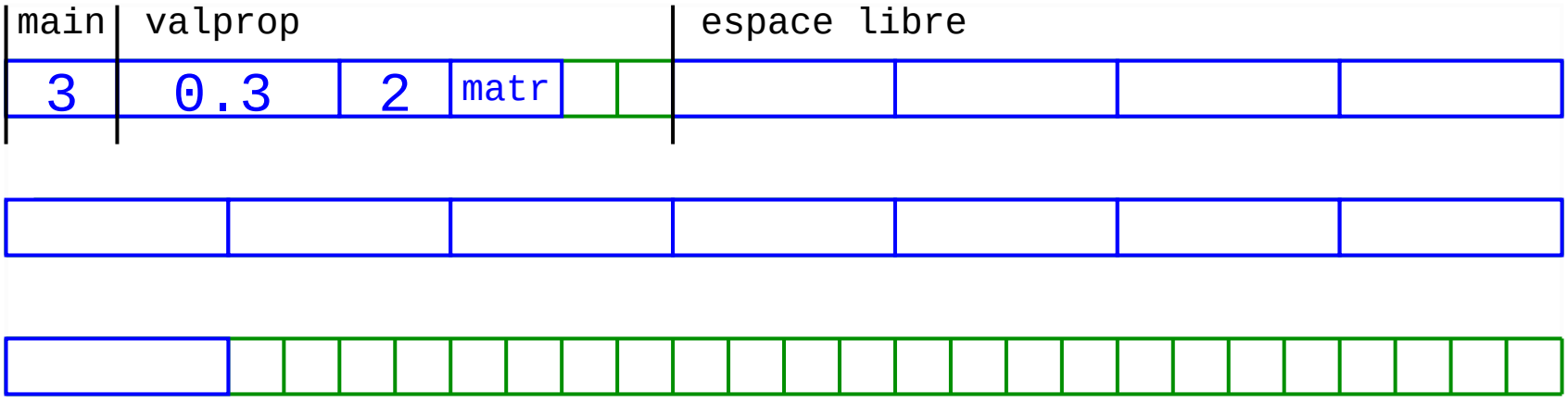
8.4 Les tableaux dynamiques

```
#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;
}
```



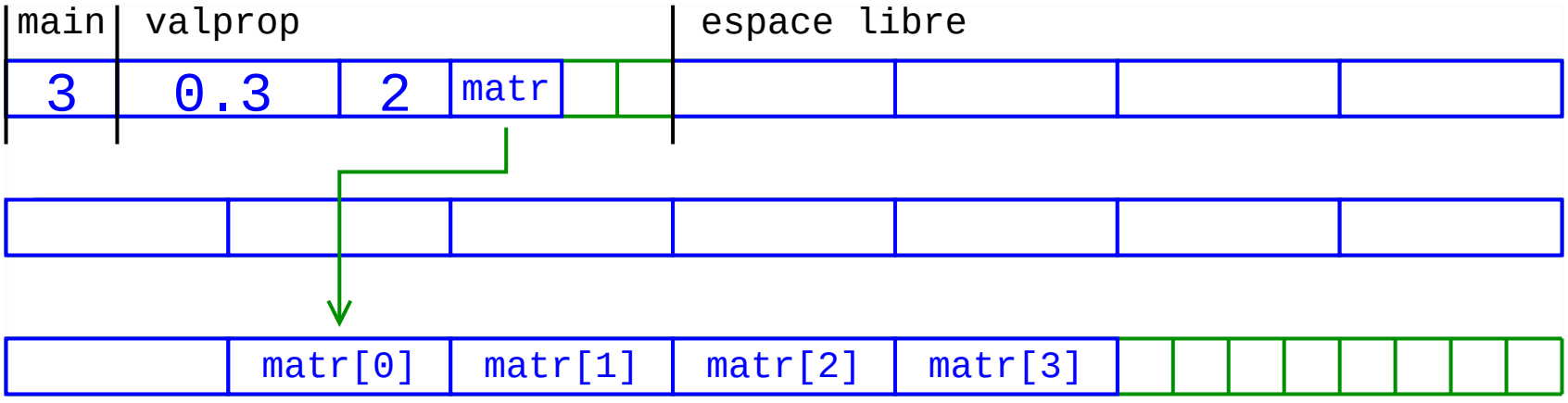
8.4 Les tableaux dynamiques

```
#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;
}
```



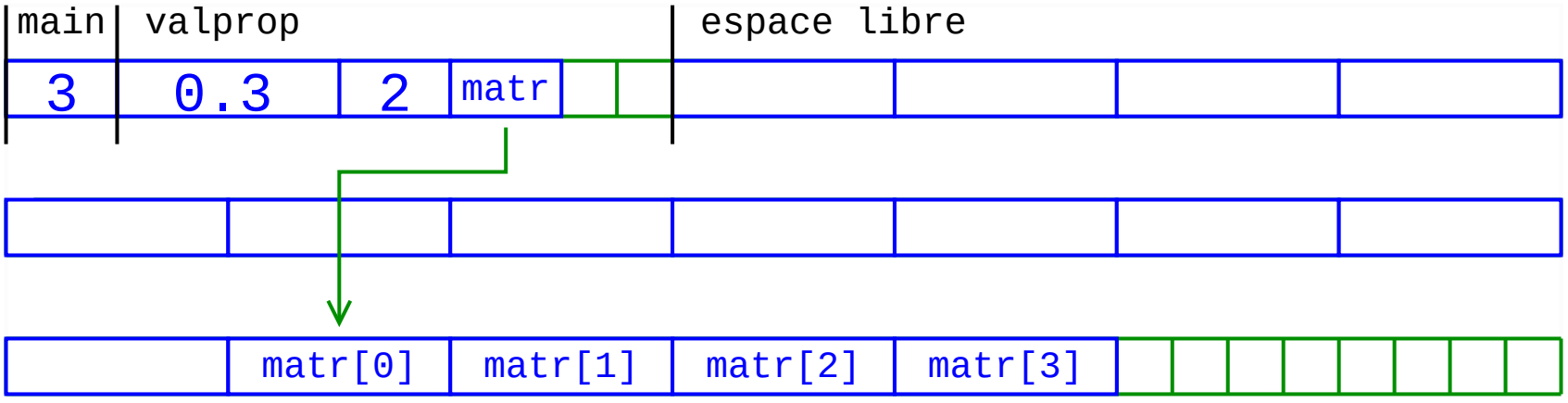
8.4 Les tableaux dynamiques

```
#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;
}
```



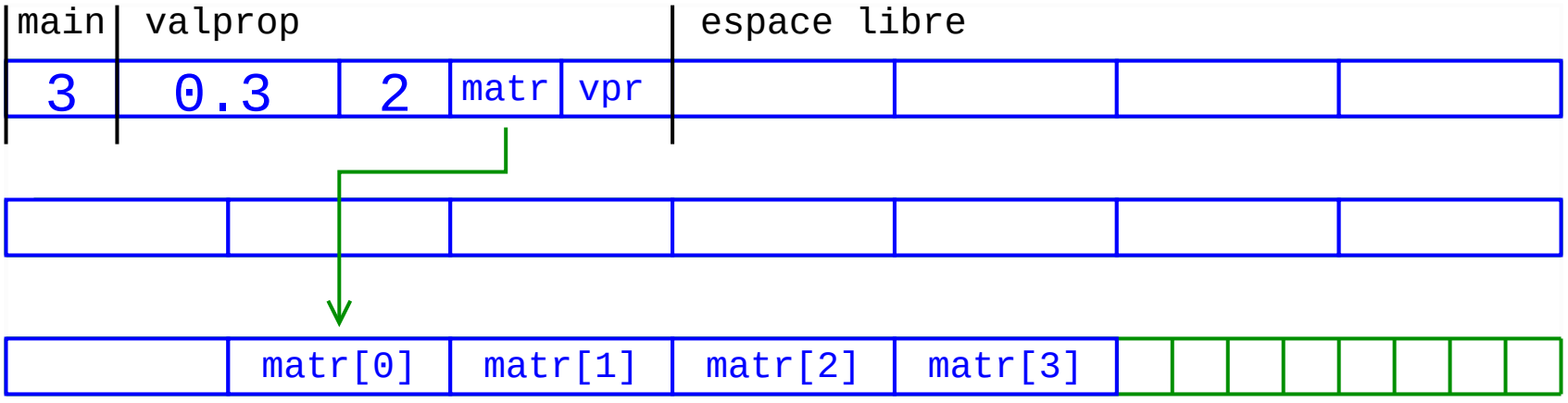
8.4 Les tableaux dynamiques

```
#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;
}
```



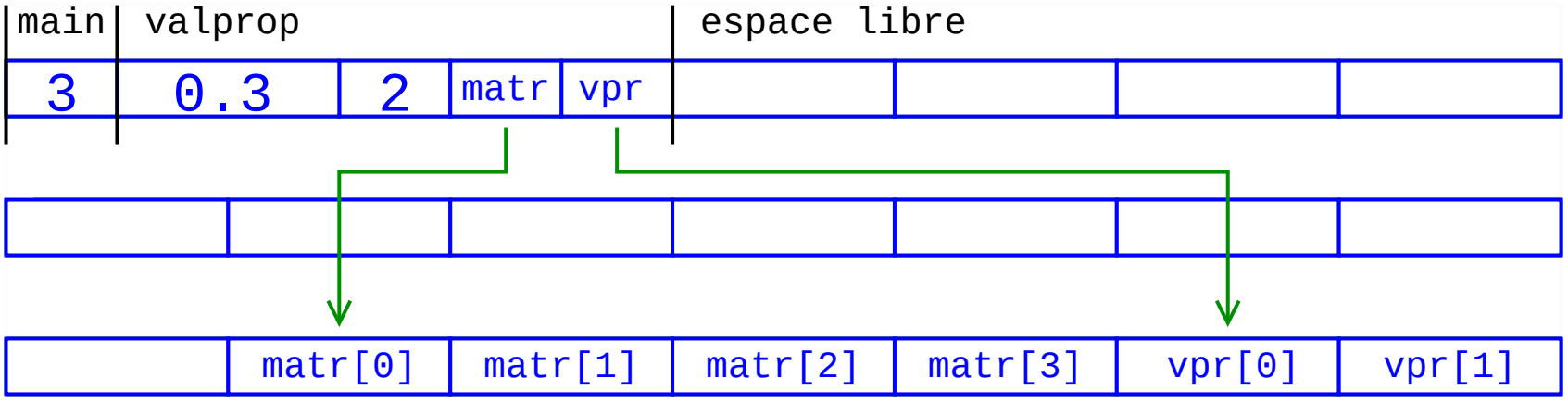
8.4 Les tableaux dynamiques

```
#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;
}
```



8.4 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;
}
```



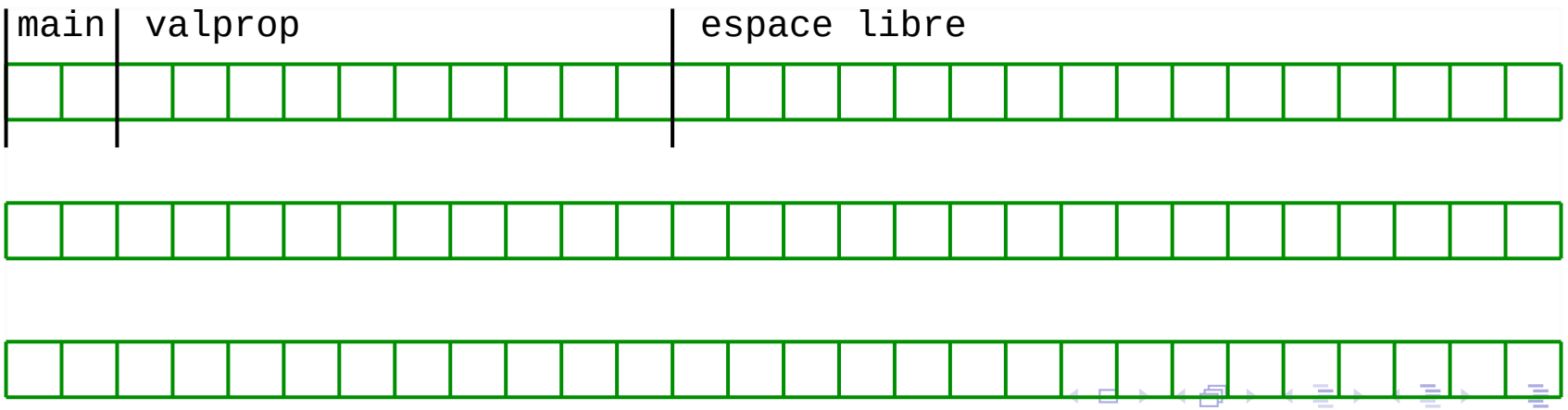
8.4 Les tableaux dynamiques

```
#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`

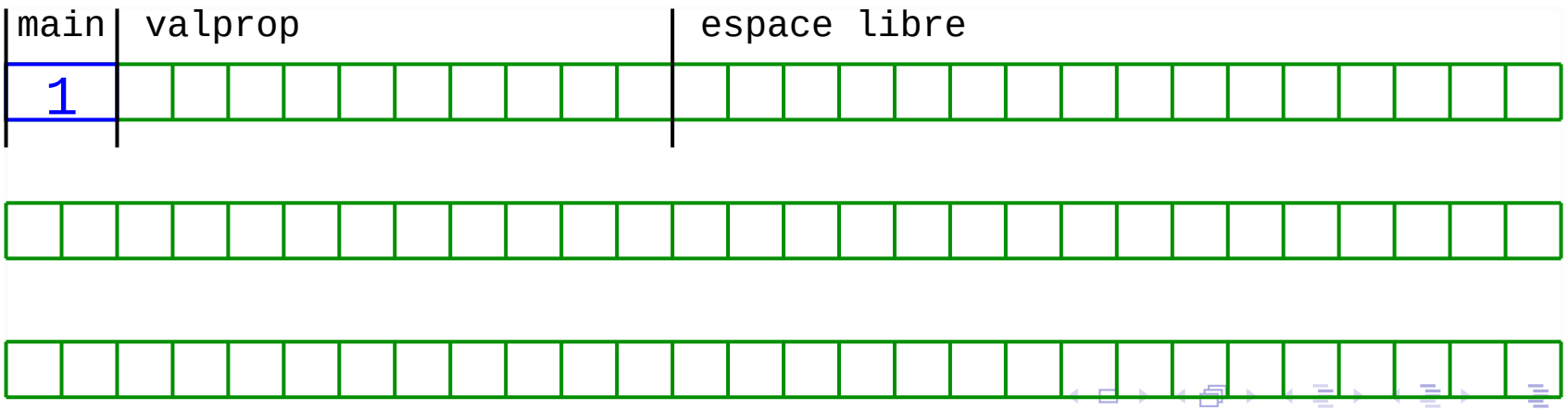
8.4 Les tableaux dynamiques

```
#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;
}
```



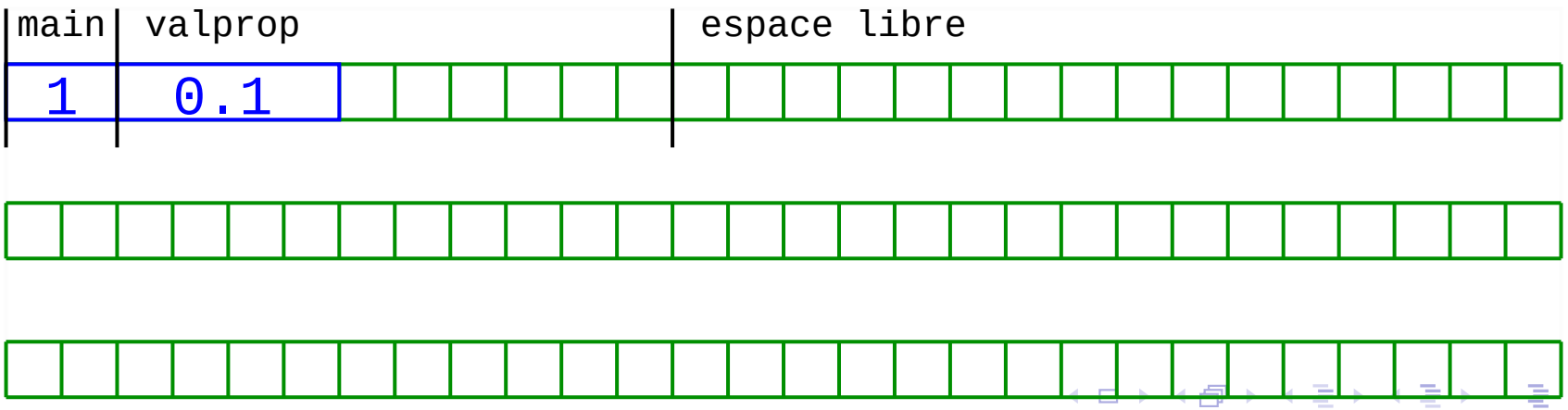
8.4 Les tableaux dynamiques

```
#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;
}
```



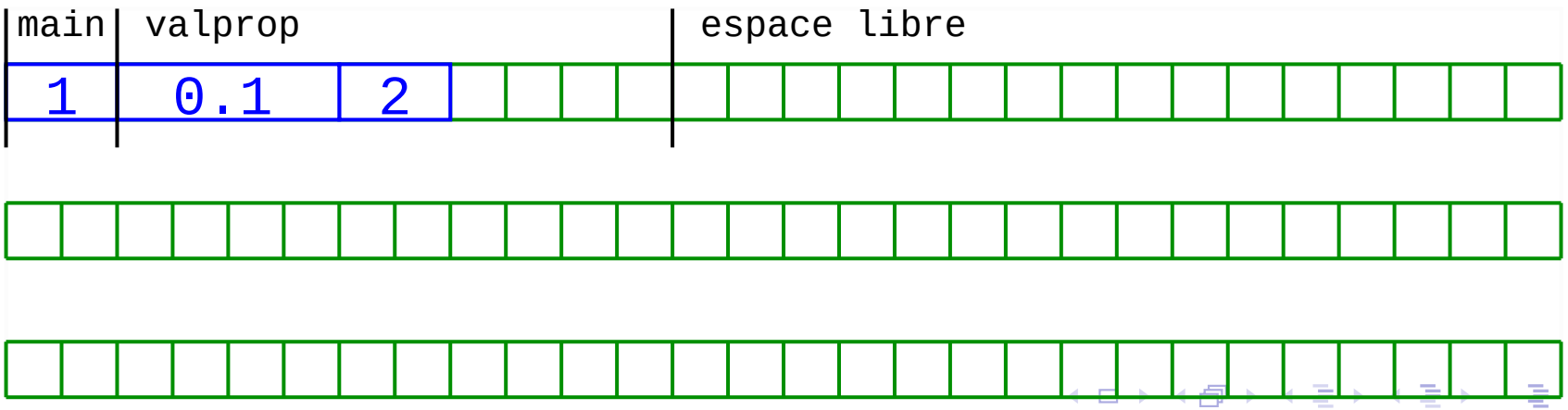
8.4 Les tableaux dynamiques

```
#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;
}
```



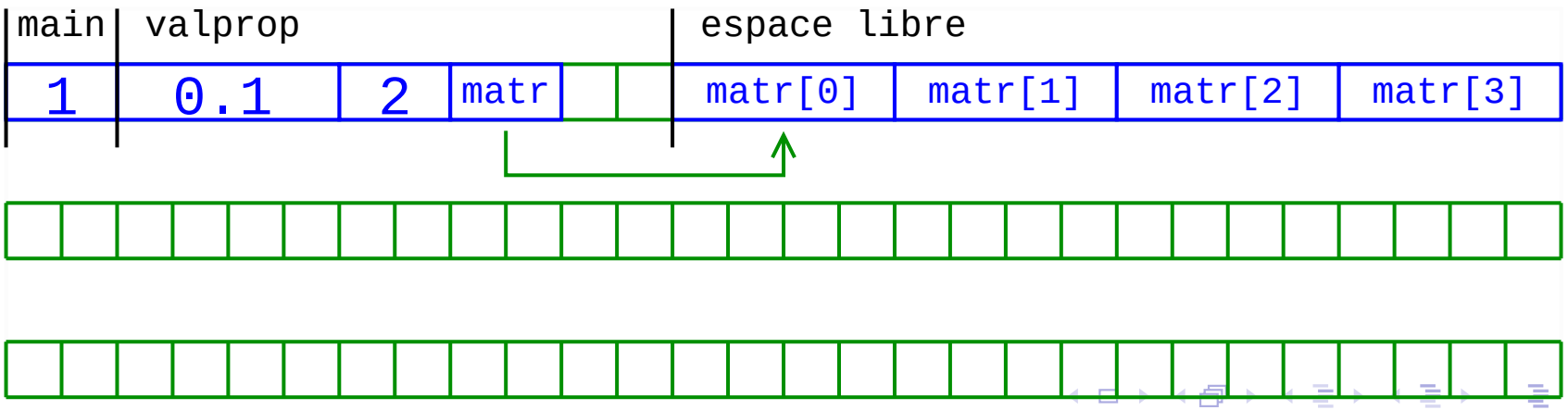
8.4 Les tableaux dynamiques

```
#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;
}
```



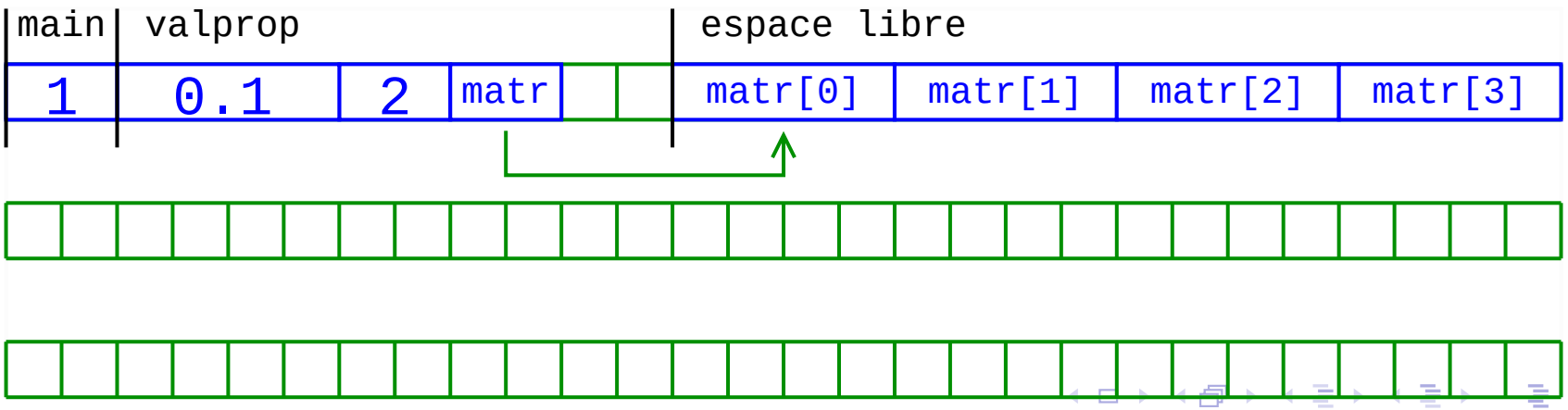
8.4 Les tableaux dynamiques

```
#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;
}
```



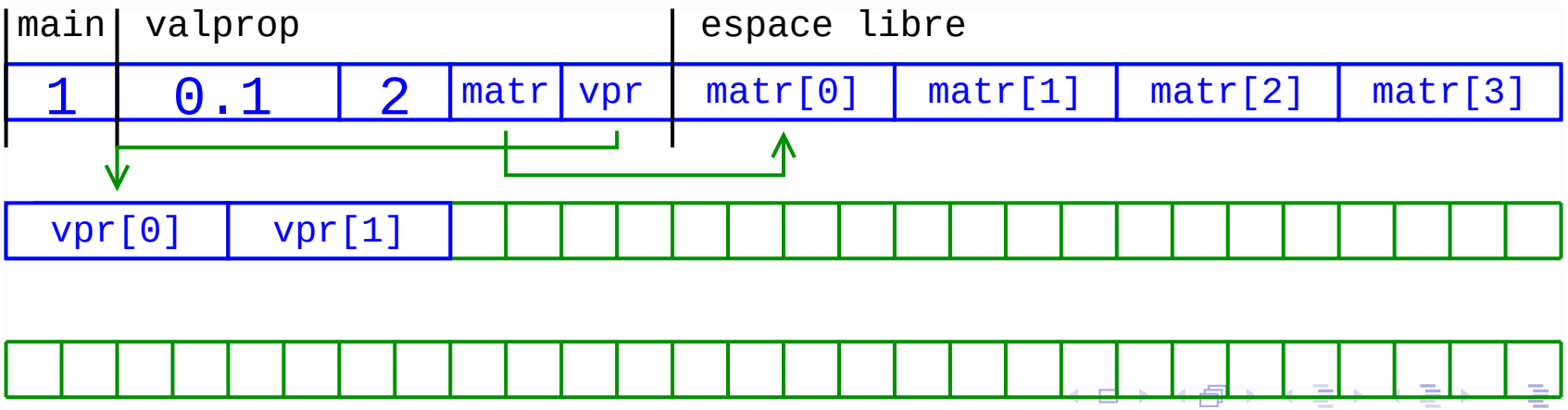
8.4 Les tableaux dynamiques

```
#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;
}
```



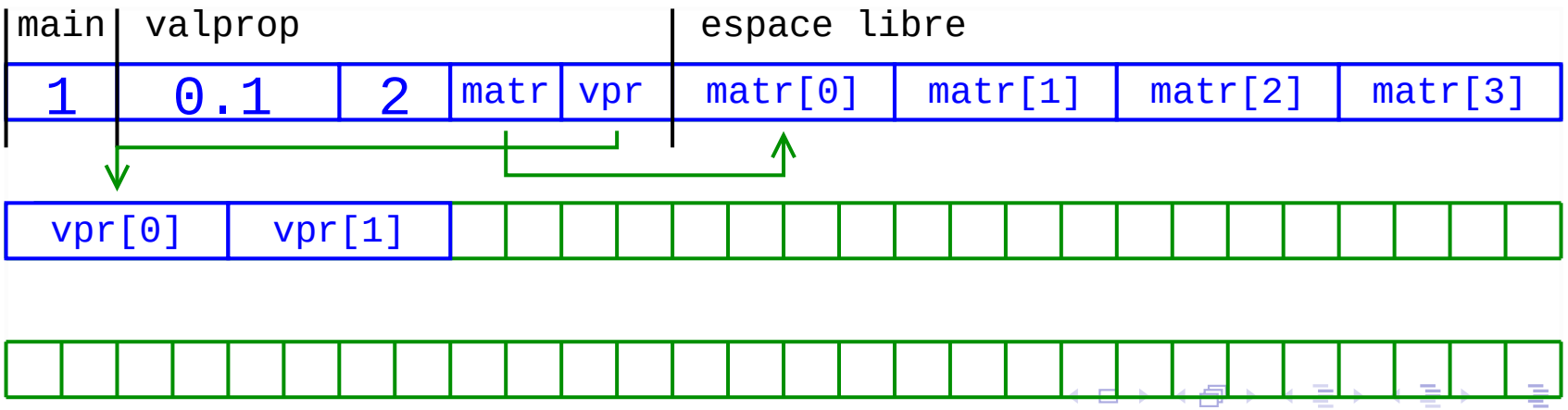
8.4 Les tableaux dynamiques

```
#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;
}
```



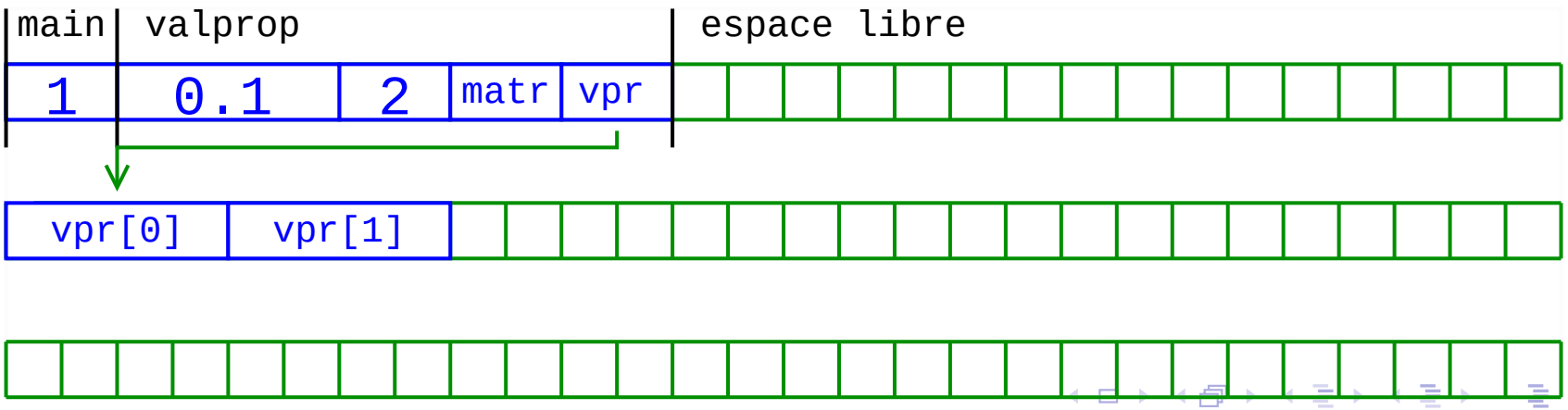
8.4 Les tableaux dynamiques

```
#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;
}
```



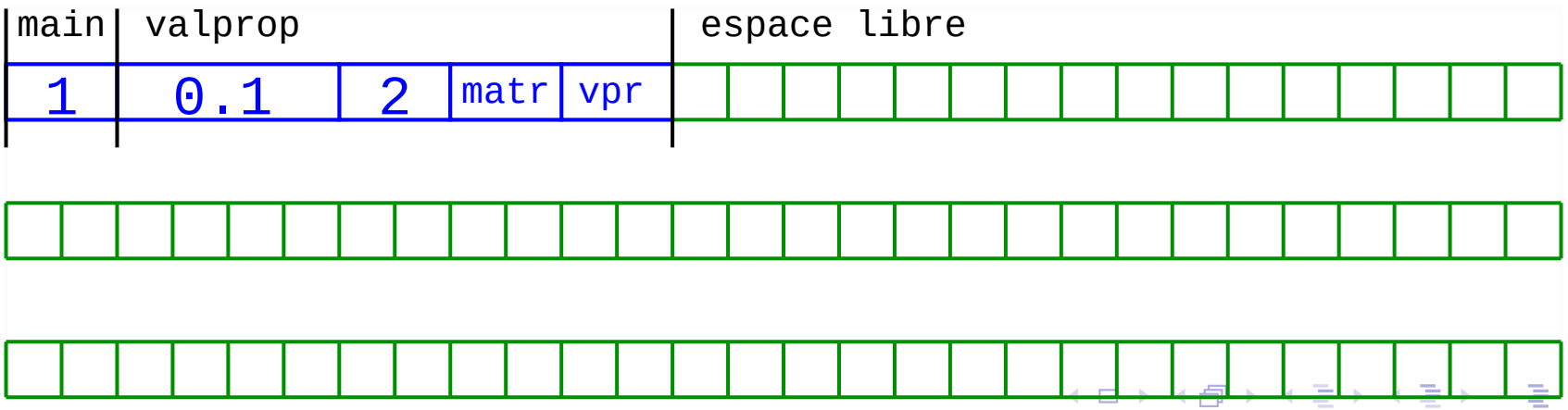
8.4 Les tableaux dynamiques

```
#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;
}
```



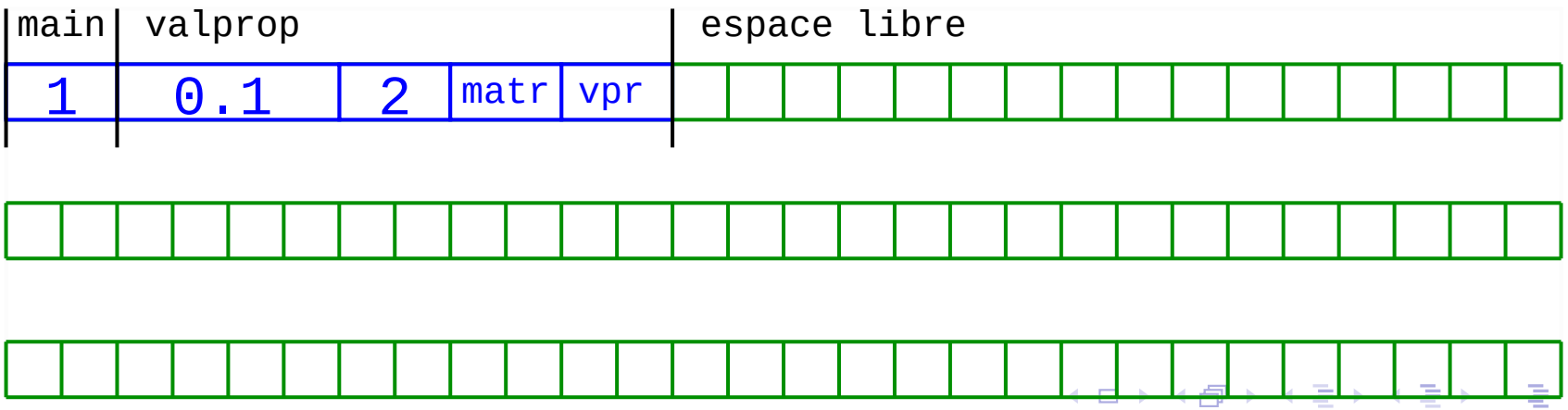
8.4 Les tableaux dynamiques

```
#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;
}
```



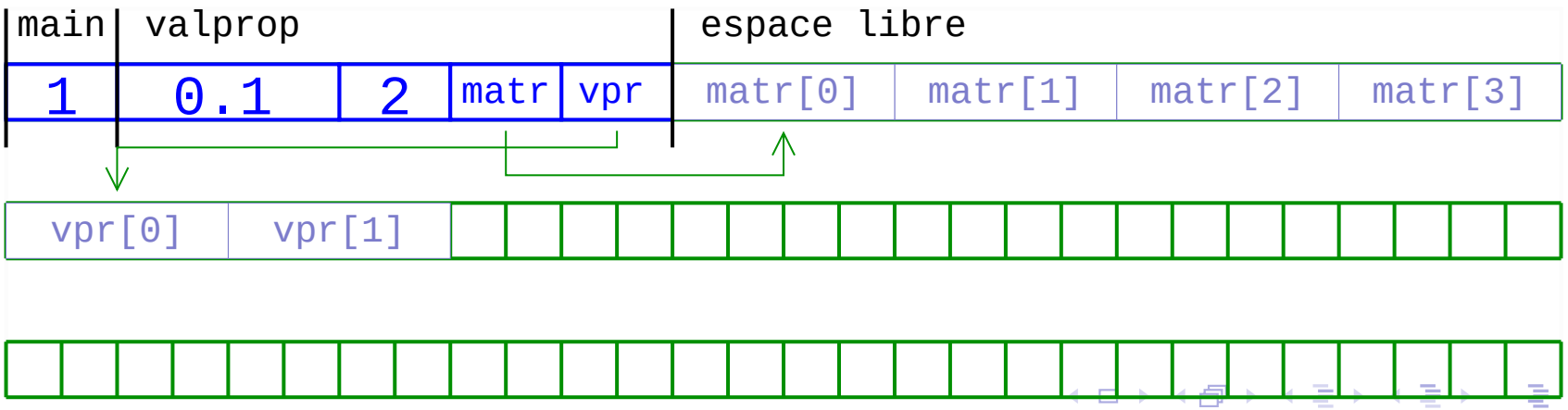
8.4 Les tableaux dynamiques

```
#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;
}
```



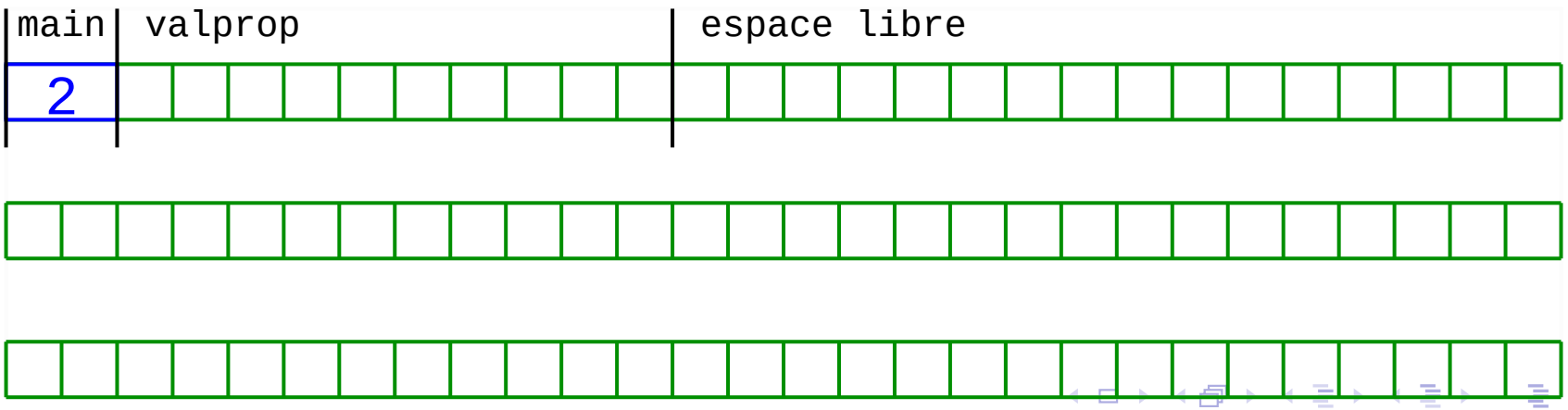
8.4 Les tableaux dynamiques

```
#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;
}
```



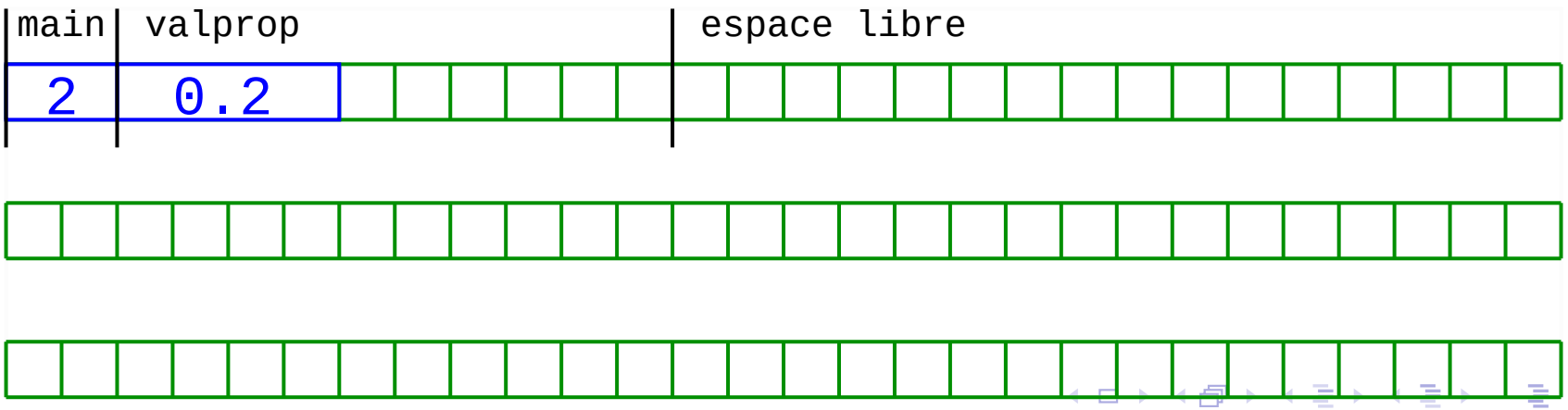
8.4 Les tableaux dynamiques

```
#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;
}
```



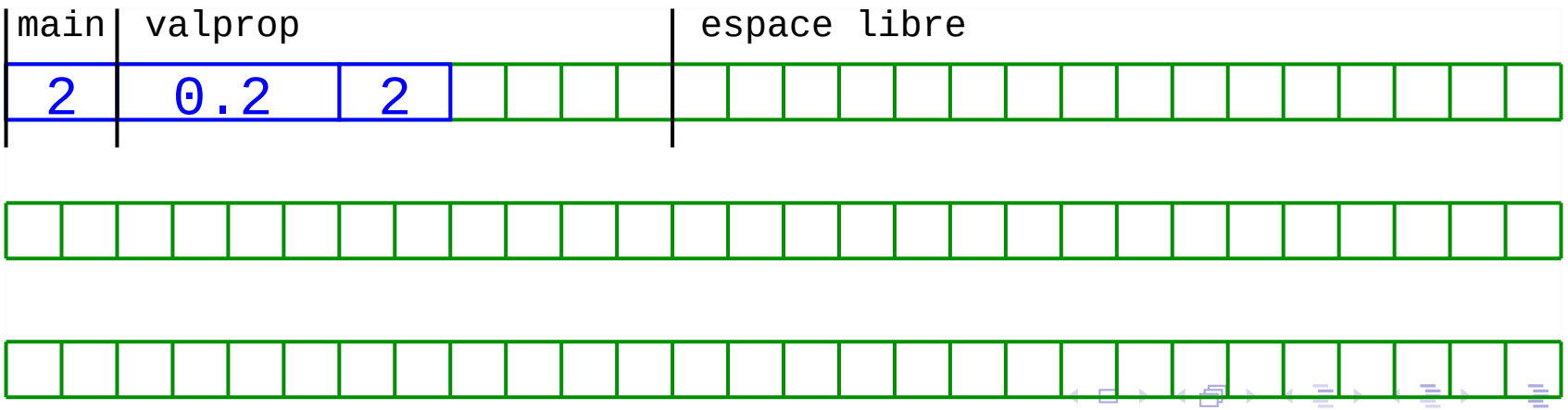
8.4 Les tableaux dynamiques

```
#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;
}
```



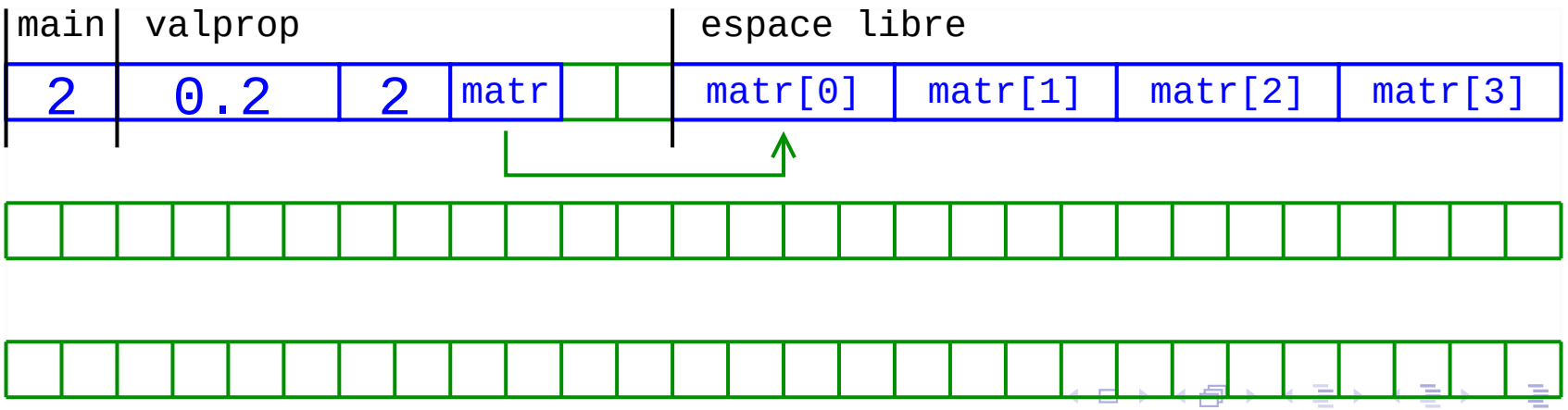
8.4 Les tableaux dynamiques

```
#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;
}
```



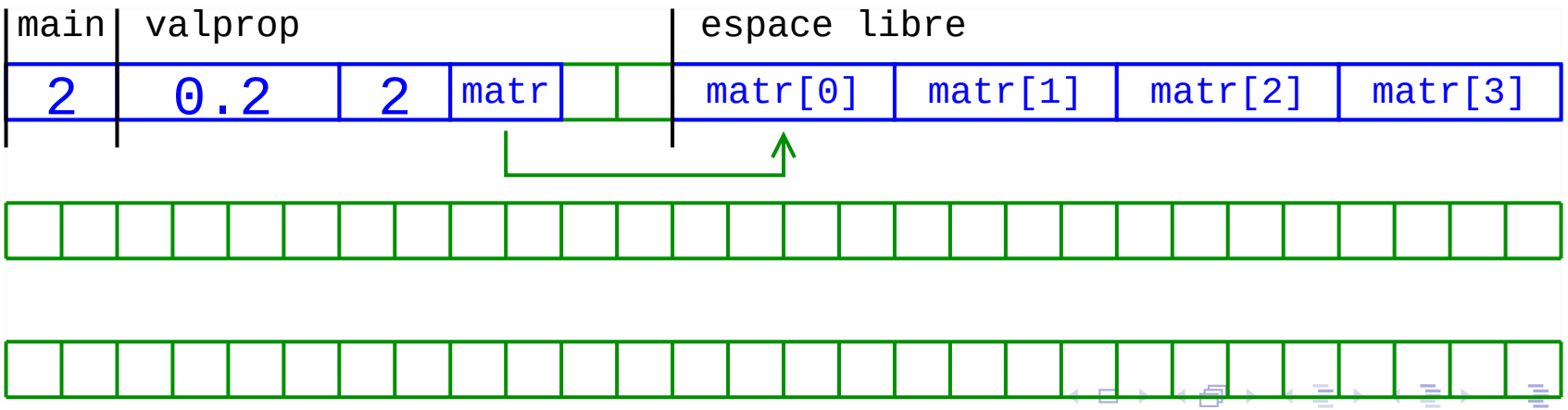
8.4 Les tableaux dynamiques

```
#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;
}
```



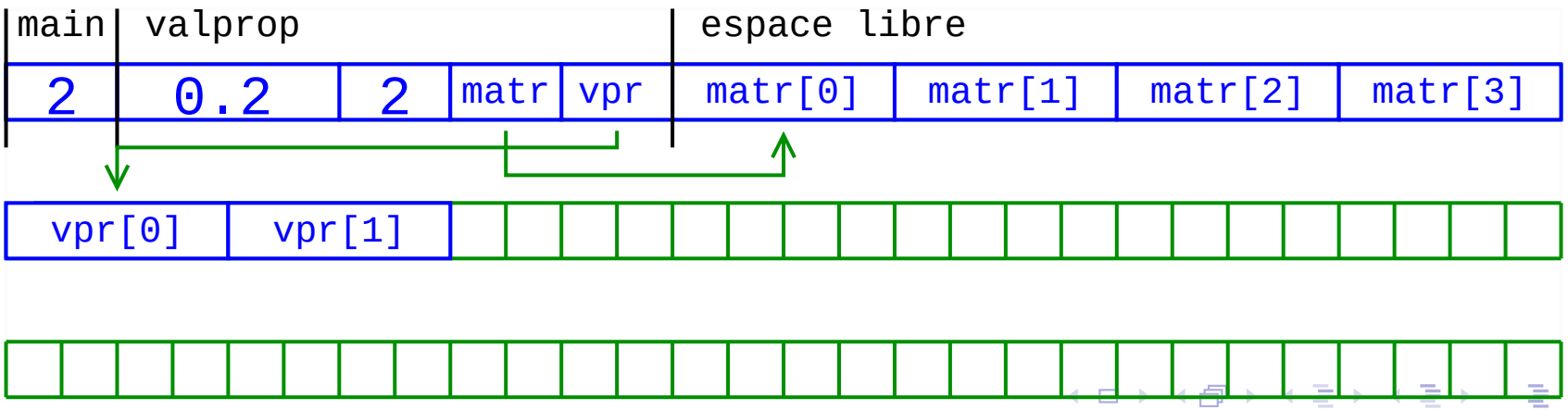
8.4 Les tableaux dynamiques

```
#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;
}
```



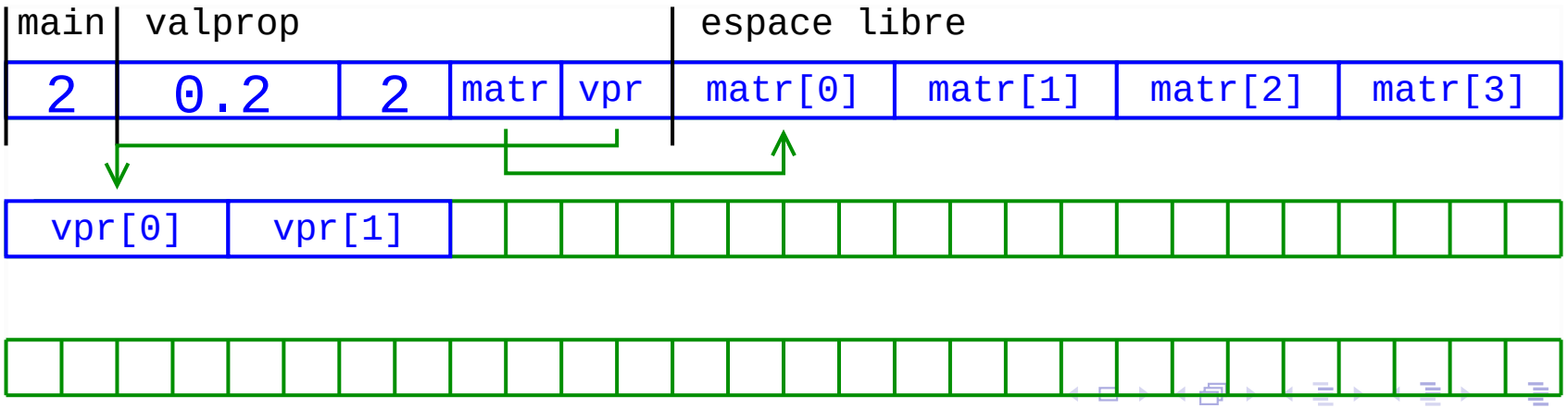
8.4 Les tableaux dynamiques

```
#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;
}
```



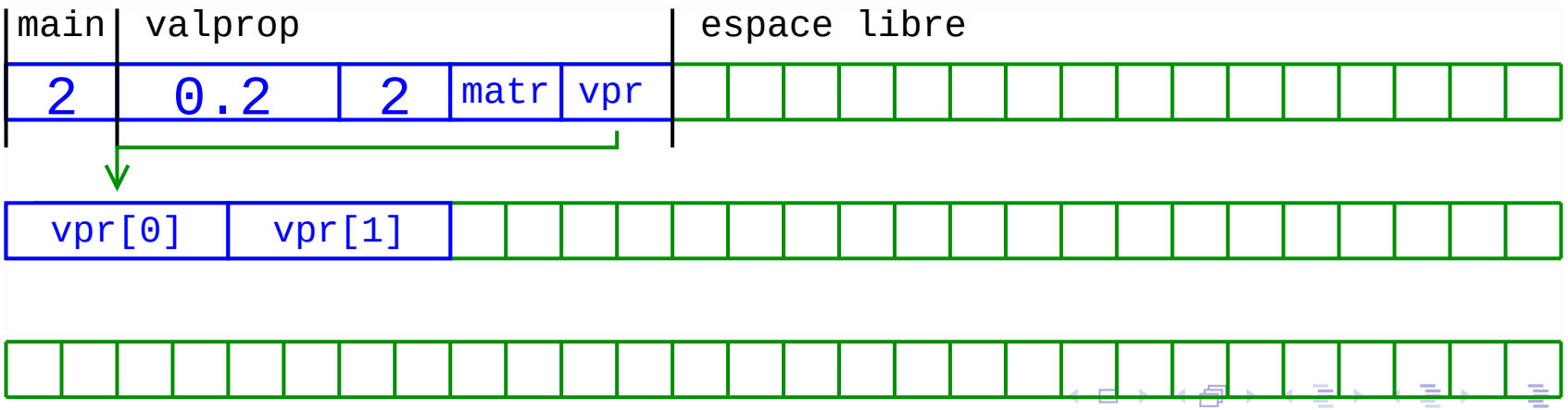
8.4 Les tableaux dynamiques

```
#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;
}
```



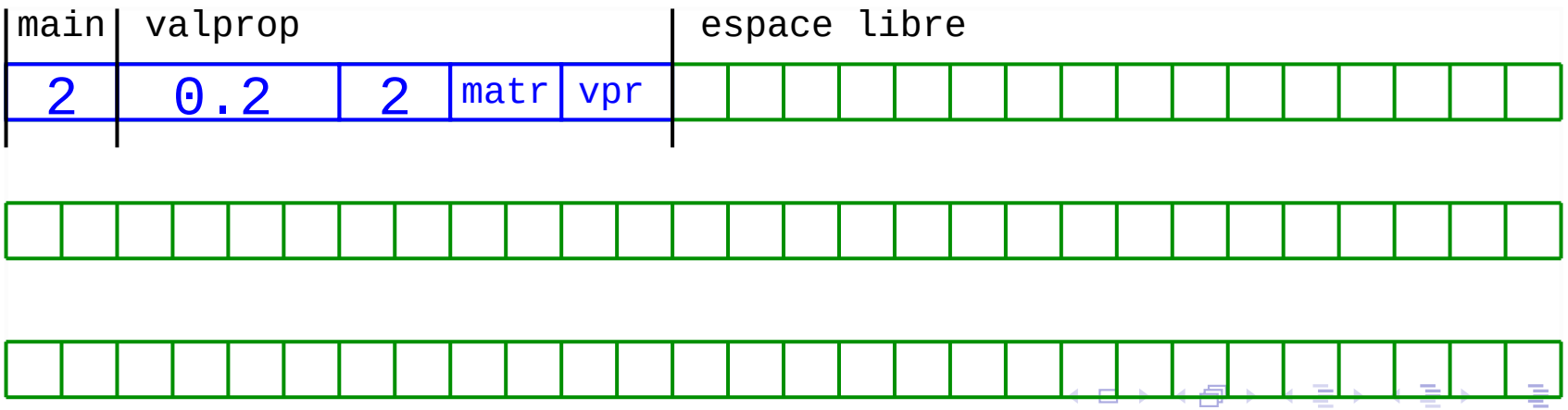
8.4 Les tableaux dynamiques

```
#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;
}
```



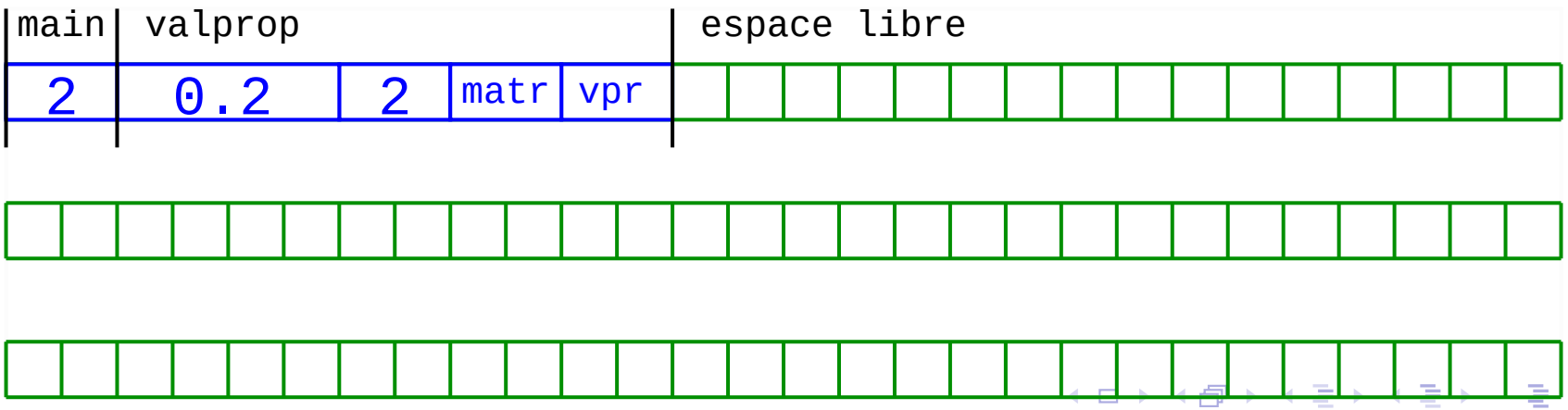
8.4 Les tableaux dynamiques

```
#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;
}
```



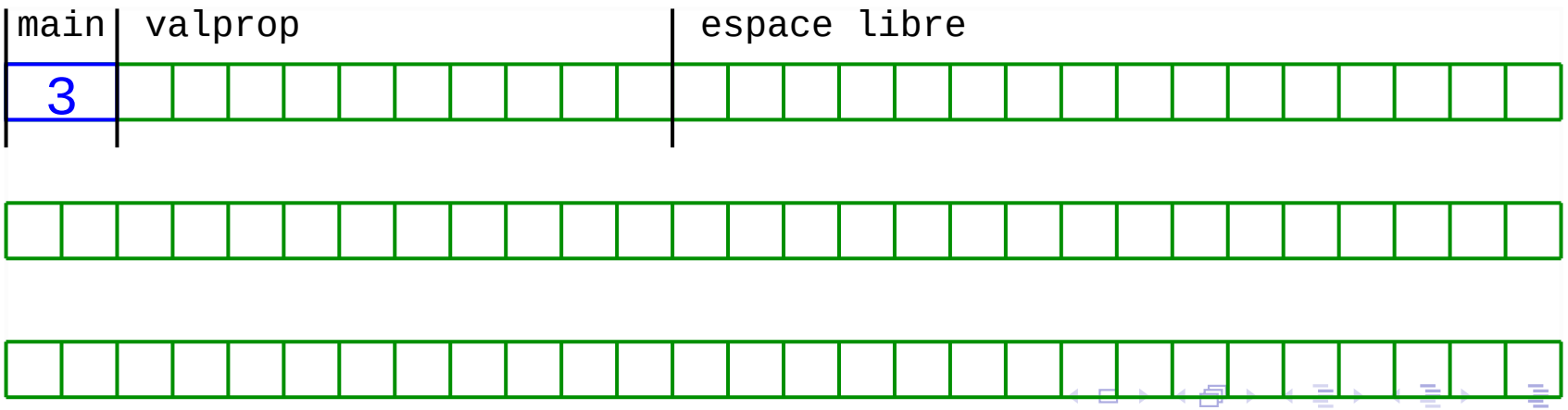
8.4 Les tableaux dynamiques

```
#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;
}
```



8.4 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;
}
```



8.4 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.

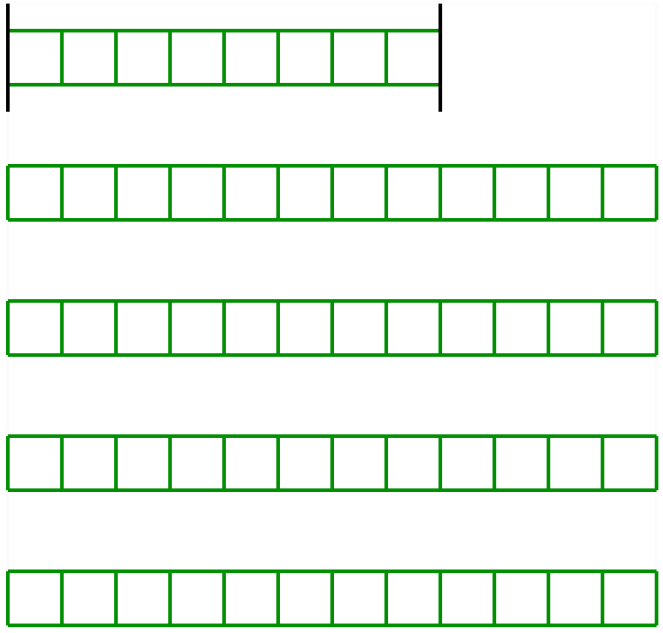
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```

`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.

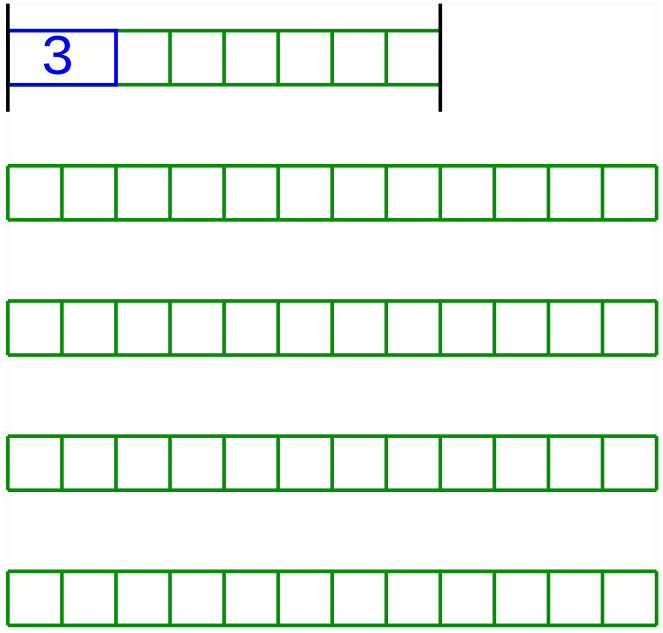
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

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

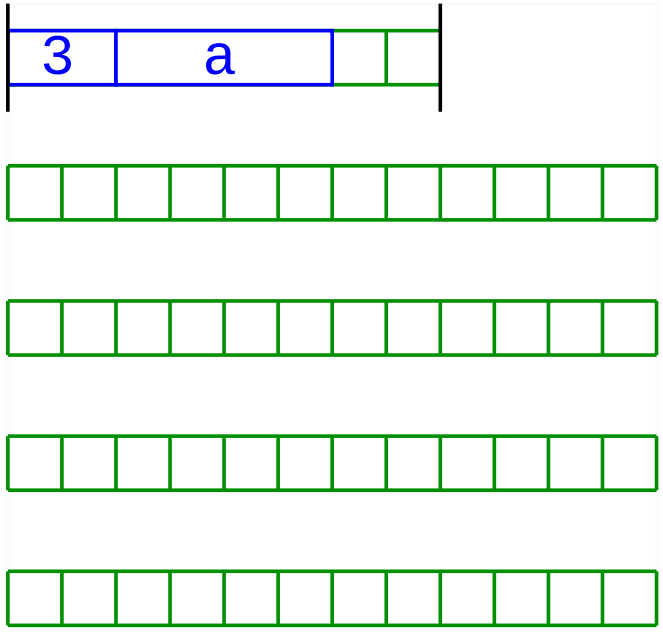
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

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

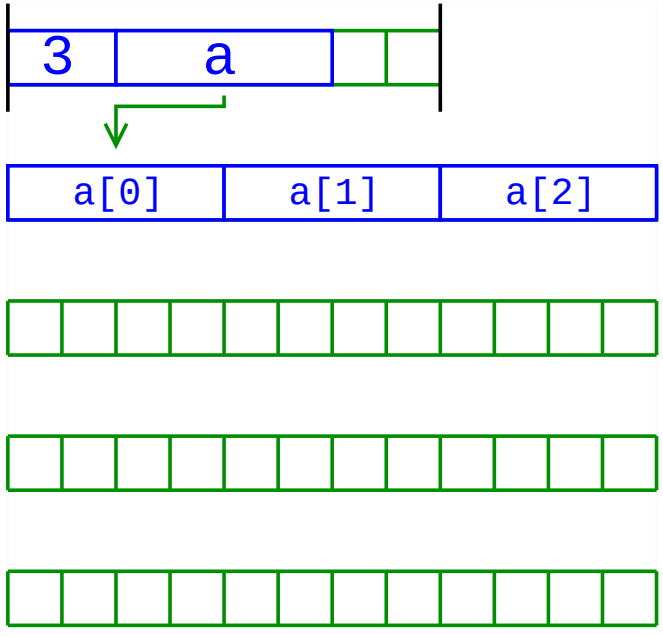
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

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

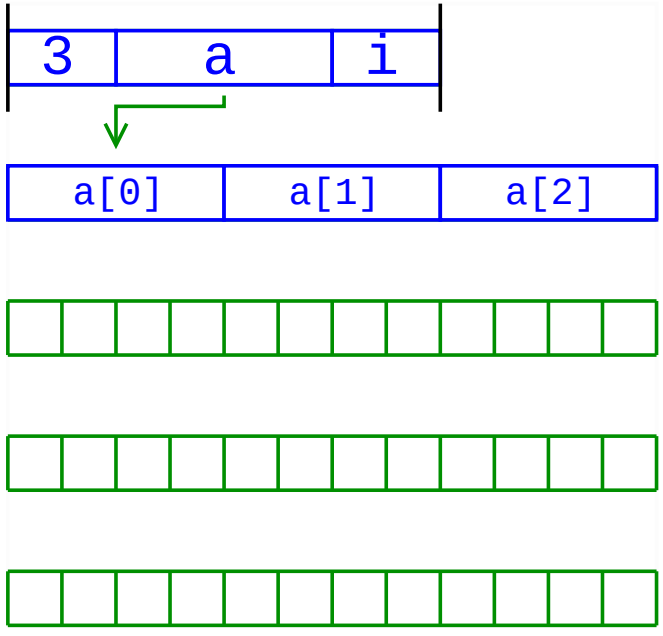
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

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

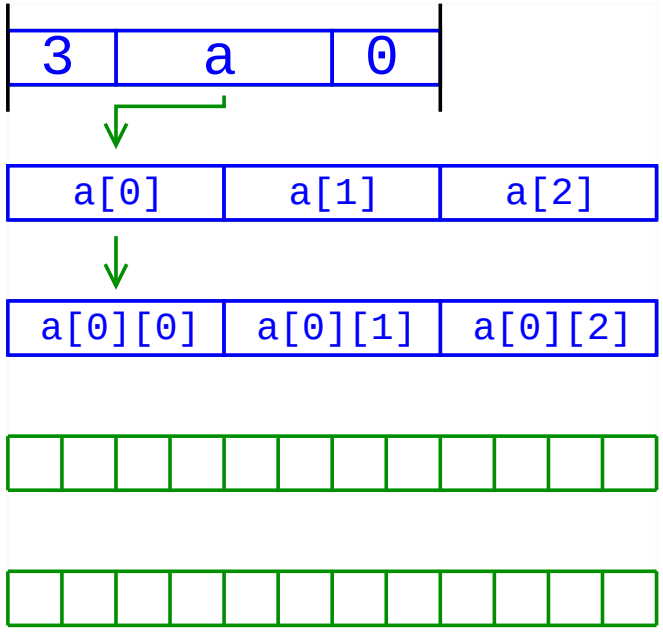
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

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

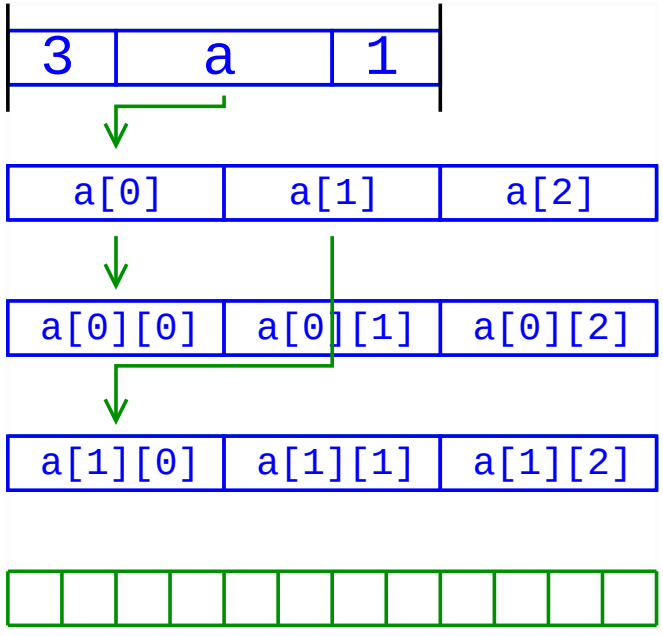
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

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

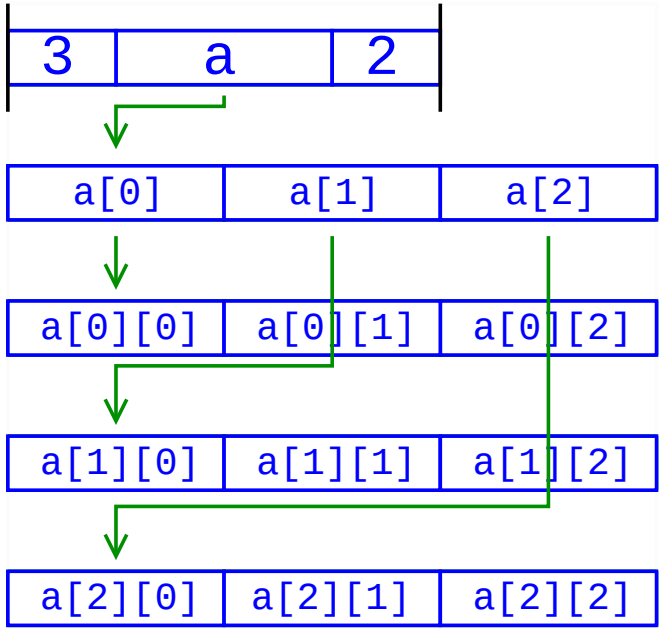
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

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

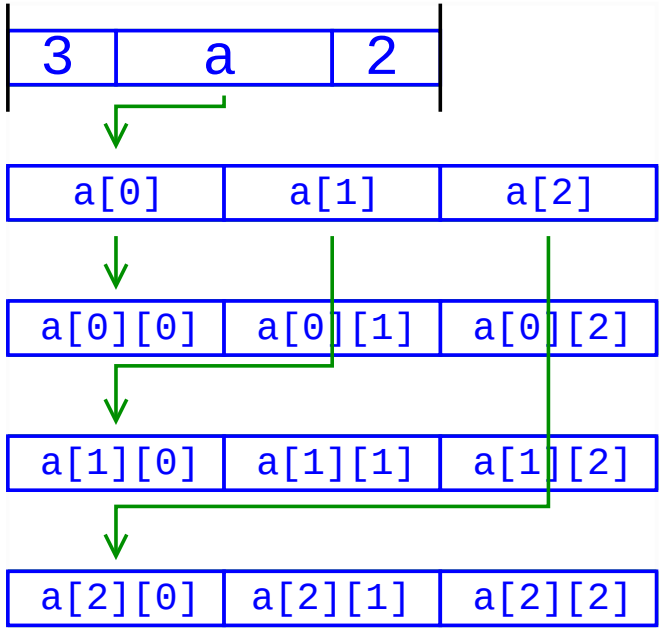
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];
```



Les tableaux dynamiques multidimensionnels

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

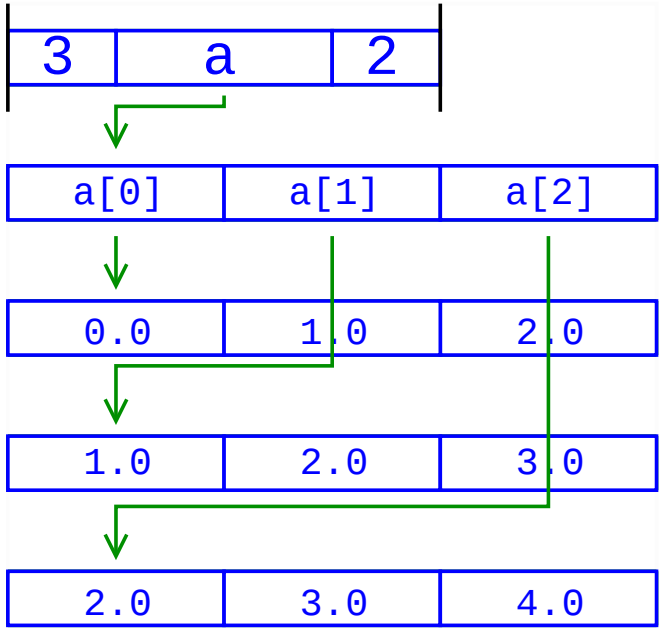
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

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.

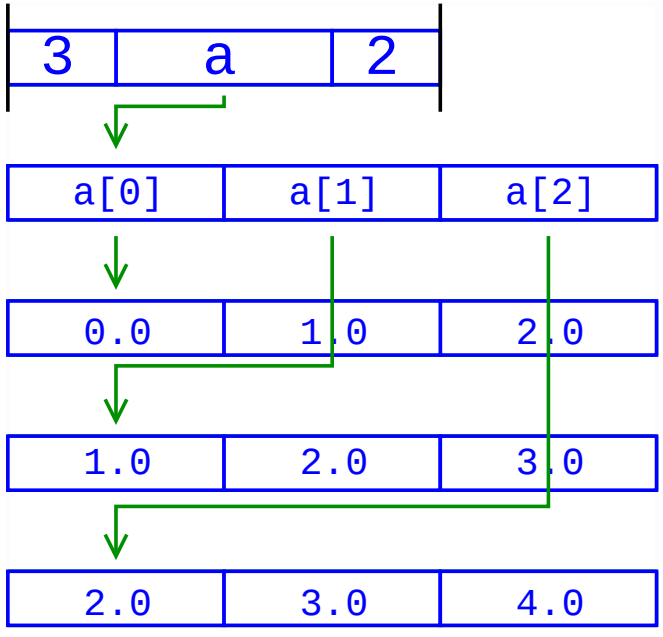
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

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.

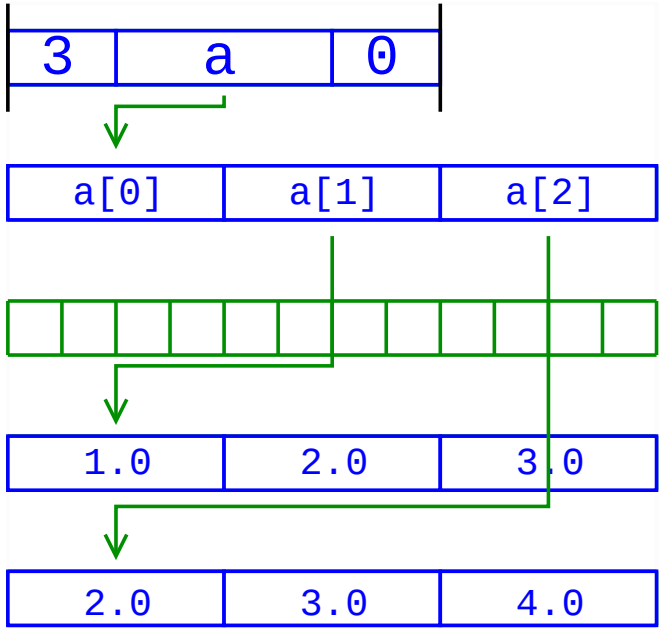
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

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.

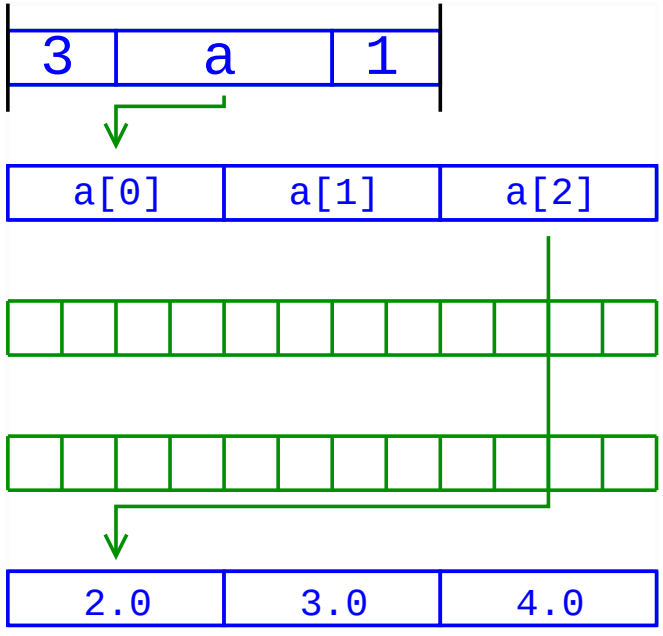
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

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.

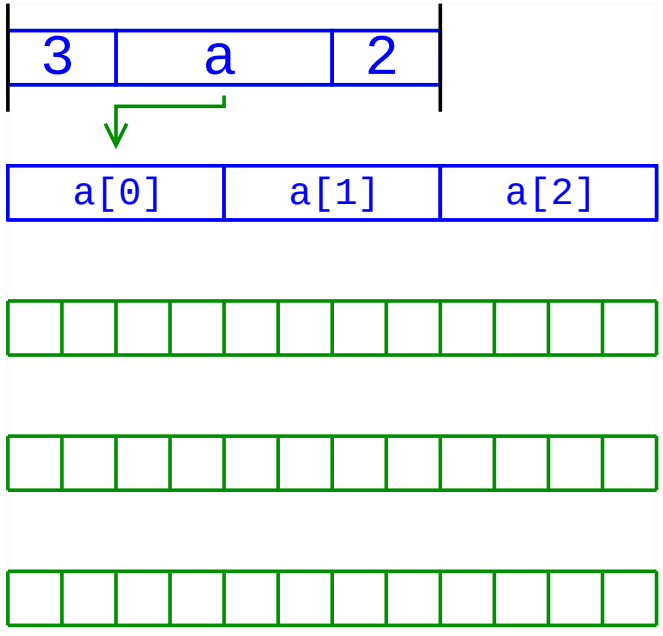
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

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.

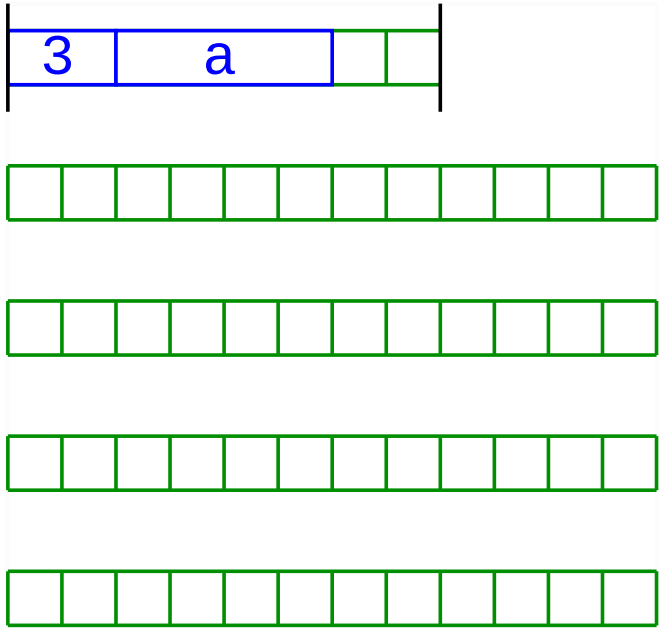
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ N ];

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 :

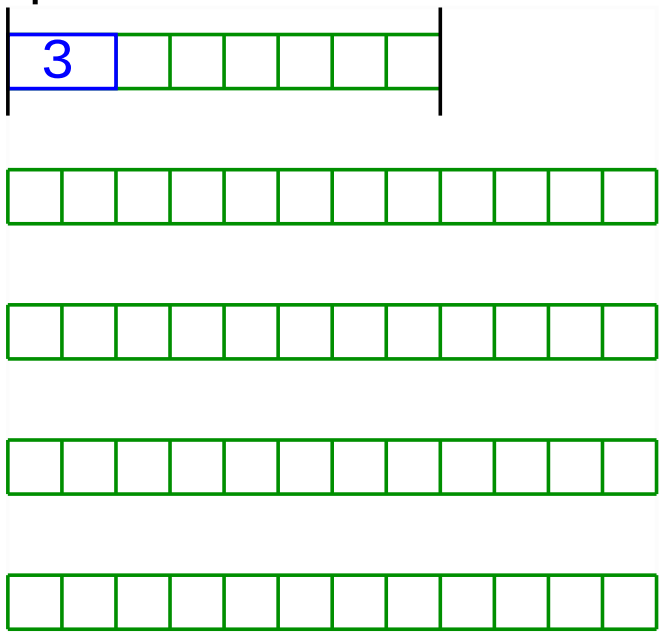
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

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

Les tableaux dynamiques multidimensionnels

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

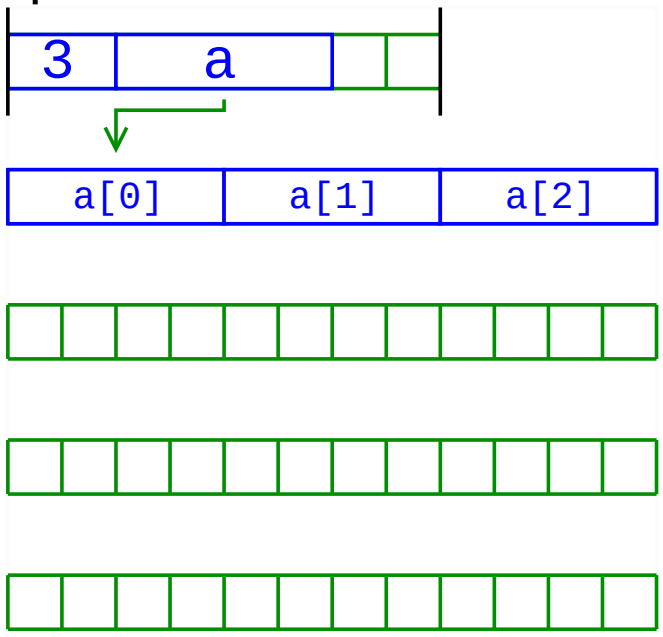
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];
```



Les tableaux dynamiques multidimensionnels

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

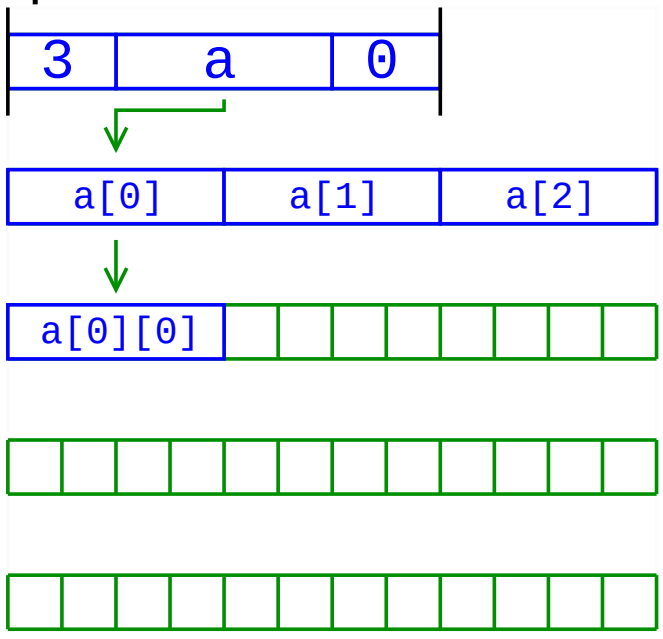
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];
```



Les tableaux dynamiques multidimensionnels

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

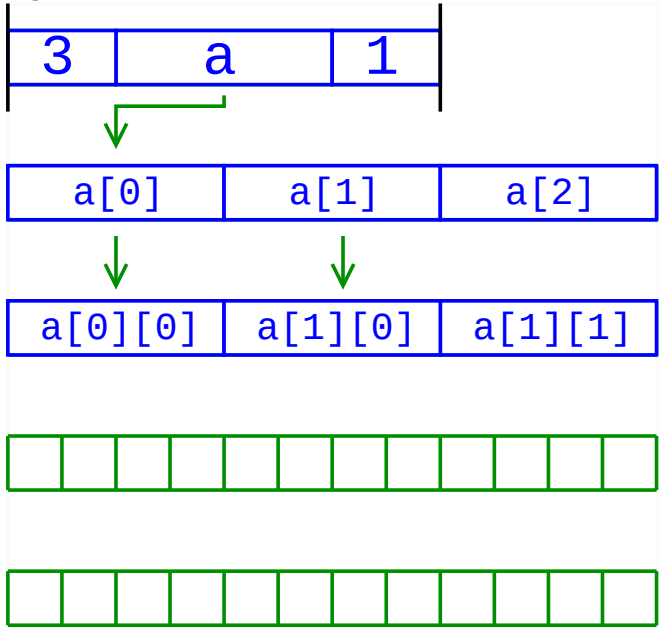
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];
```



Les tableaux dynamiques multidimensionnels

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

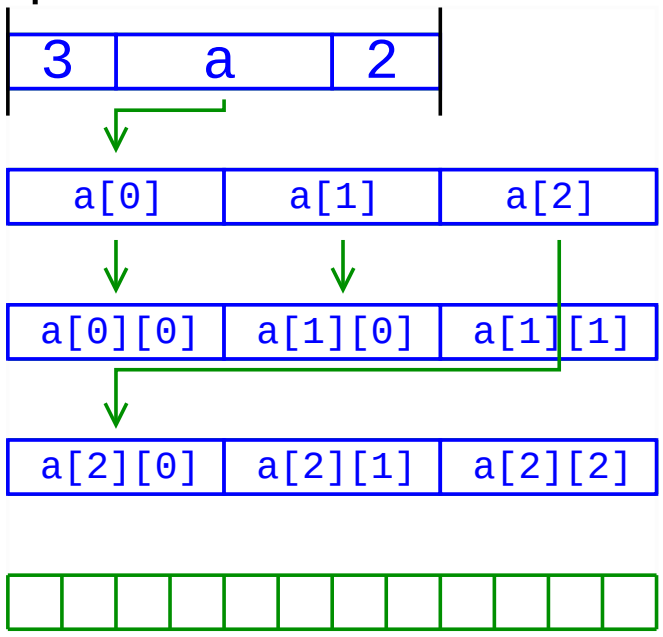
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];
```



Les tableaux dynamiques multidimensionnels

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

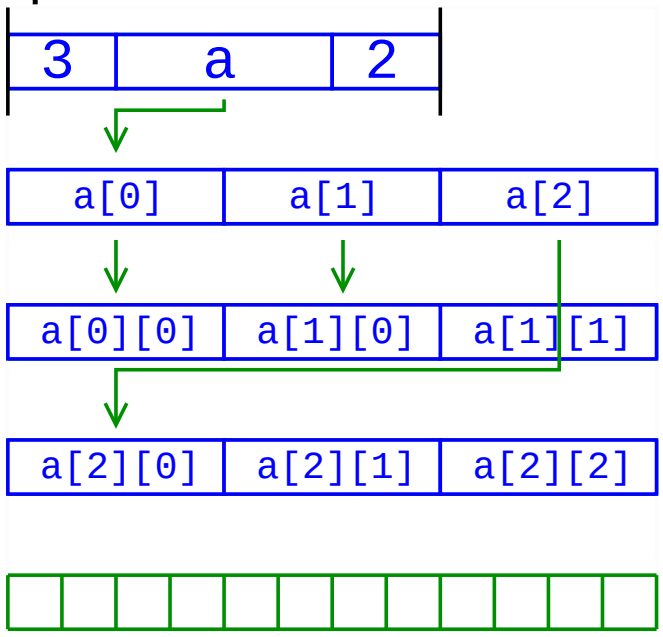
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];
```



Les tableaux dynamiques multidimensionnels

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

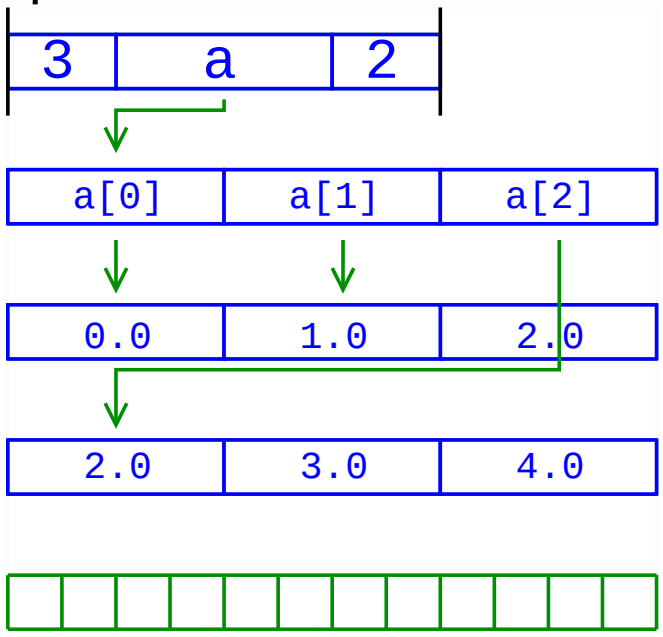
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

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 :

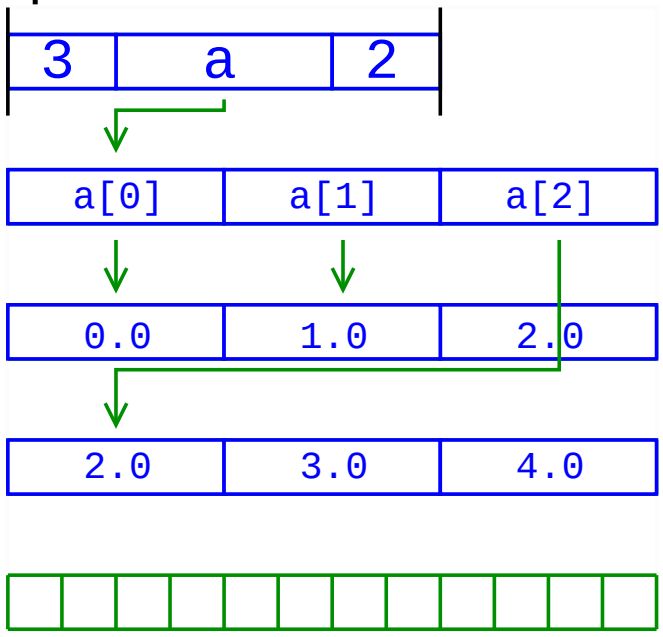
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

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 :

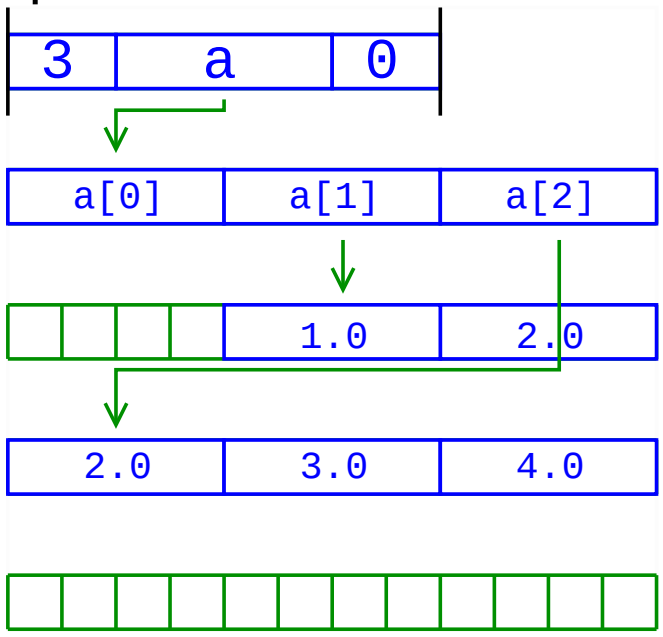
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

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 :

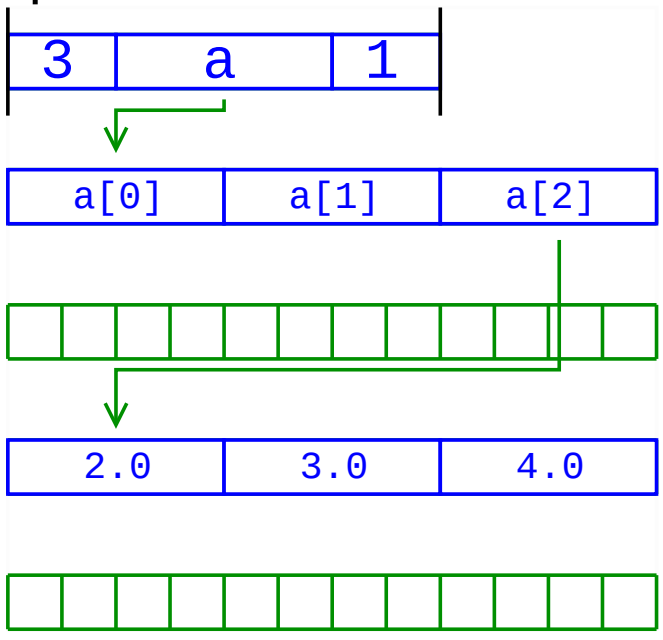
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

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 :

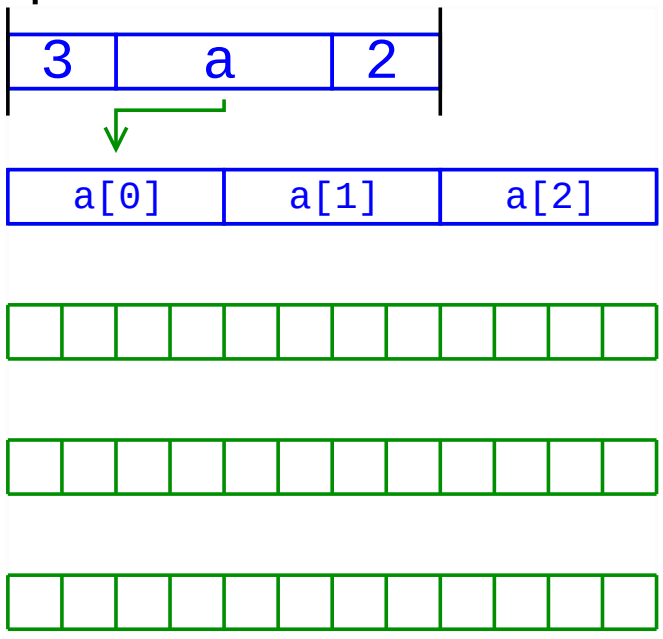
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

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 :

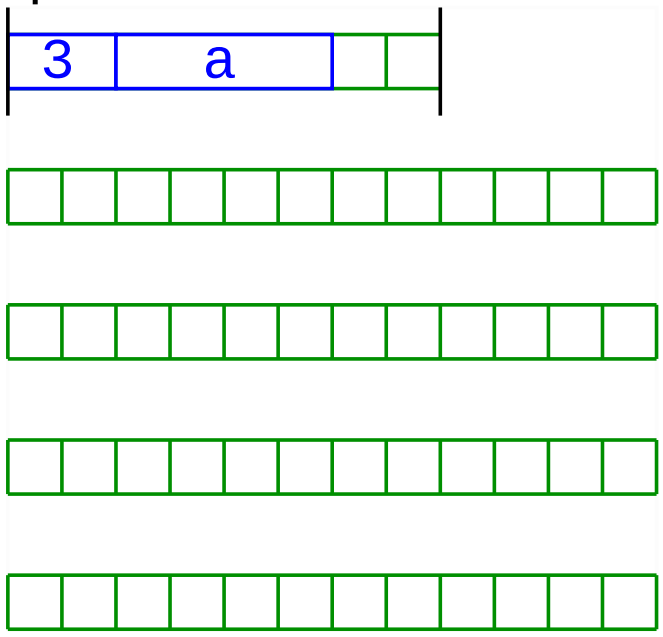
```
int N = 3;
double **a = new double*[ N ];
for ( int i = 0; i < N; i++ )
  a[ i ] = new double[ i + 1 ];

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
```