7 Les tableaux

Peter Schlagheck

Université de Liège

7 Les tableaux

7.1 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
```

7.1 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
```

7.1 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 . . .

7.1 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
```

7.1 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
```

7.1 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
```

7.1 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
```

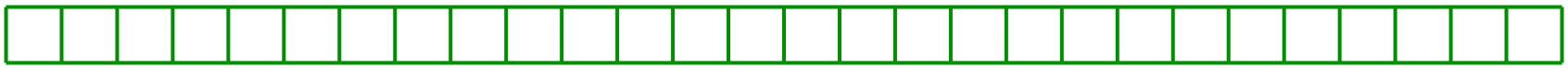
7.1 Les tableaux conventionnels

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
```



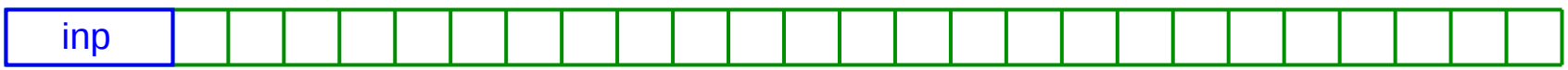
7.1 Les tableaux conventionnels

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
```



7.1 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
```

7.1 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
```

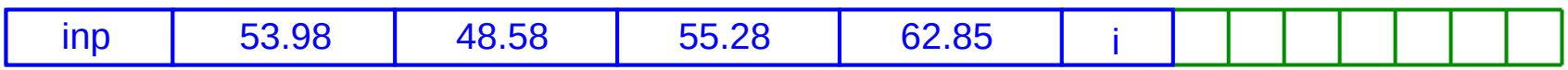
7.1 Les tableaux conventionnels

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
```



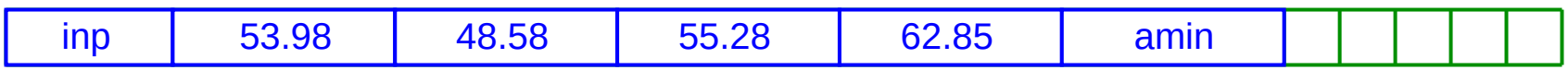
7.1 Les tableaux conventionnels

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
```



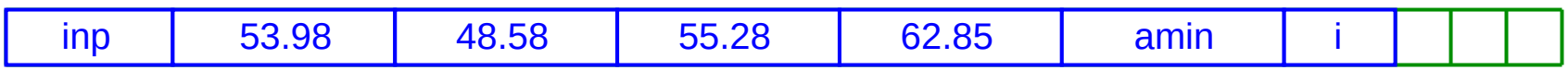
7.1 Les tableaux conventionnels

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
```



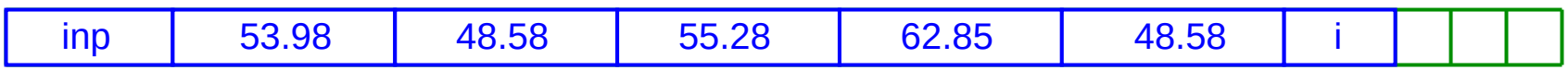
7.1 Les tableaux conventionnels

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
```



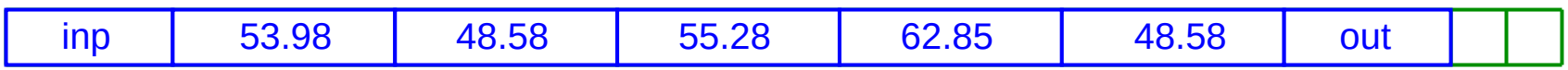
7.1 Les tableaux conventionnels

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
```



7.1 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

7.1 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⟩

7.1 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
```

7.1 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
```

7.1 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
```

7.1 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
```

7.1 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
```

7.1 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
```

7.1 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 !

7.1 Les tableaux conventionnels

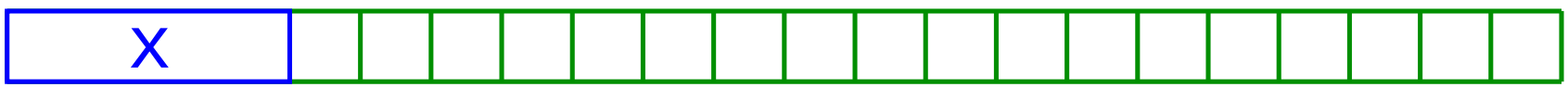
```
#include <iostream>
#include <fstream>
using namespace std;

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
```

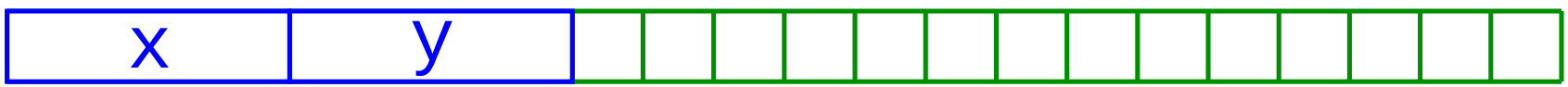
7.1 Les tableaux conventionnels

```
#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;
}
```



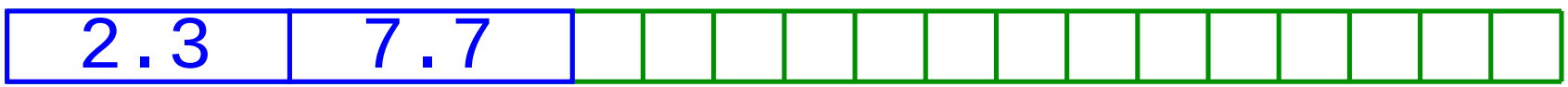
7.1 Les tableaux conventionnels

```
#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;
}
```



7.1 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;
}
```



7.1 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;
}
```

7.1 Les tableaux conventionnels

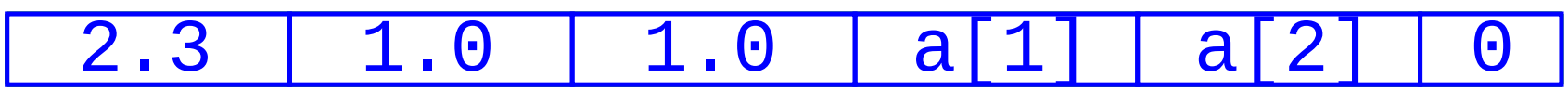
```
#include <iostream>
#include <fstream>
using namespace std;

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;
}
```

7.1 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;
}
```

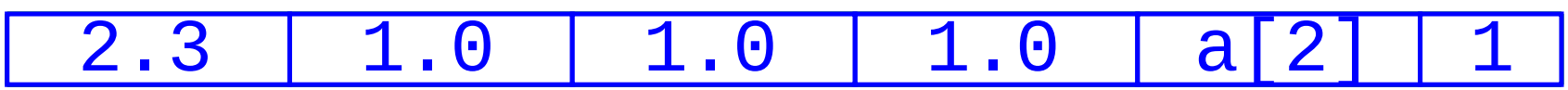
7.1 Les tableaux conventionnels

```
#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;
}
```



7.1 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;
}
```



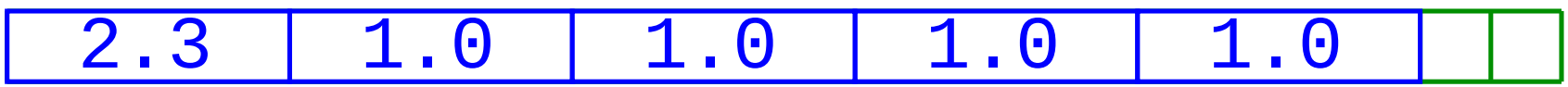
7.1 Les tableaux conventionnels

```
#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

7.1 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

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

⟨type_f⟩ ⟨nom_f⟩ (... , ⟨type_t⟩ ⟨nom_t⟩ [ ], ...)
{ ... }

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

⟨nom_f⟩ (... , ⟨nom⟩, ...)

⟶ on transmet l'adresse du premier élément du tableau à la fonction

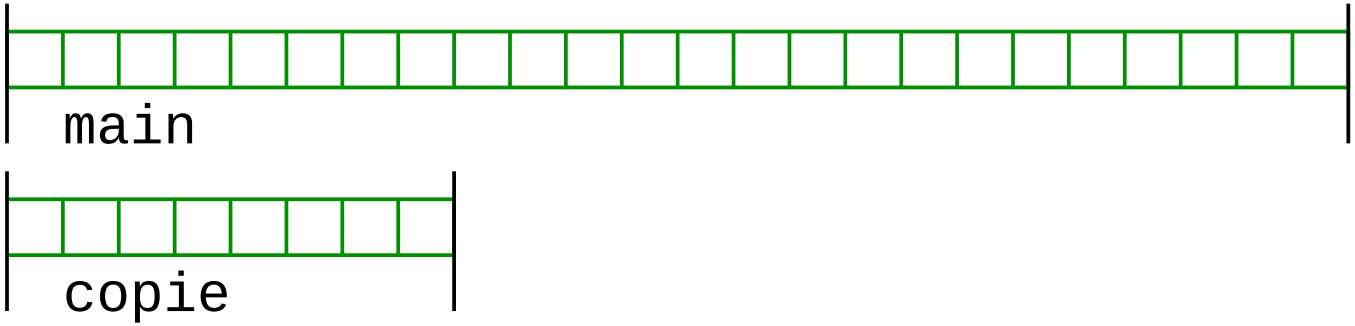
⟶ la fonction travaille donc forcément avec des variables originales du tableau

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

Les tableaux comme arguments des fonctions

Une routine de copie de deux tableaux:

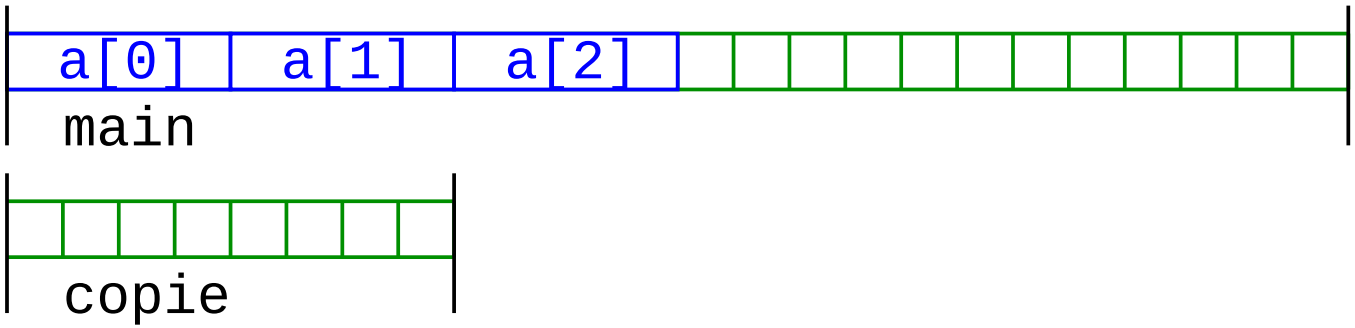
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

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:

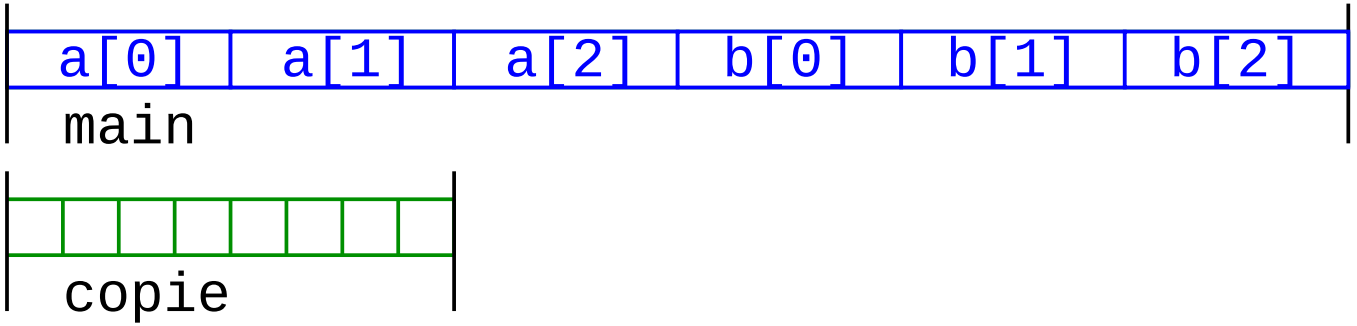
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

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:

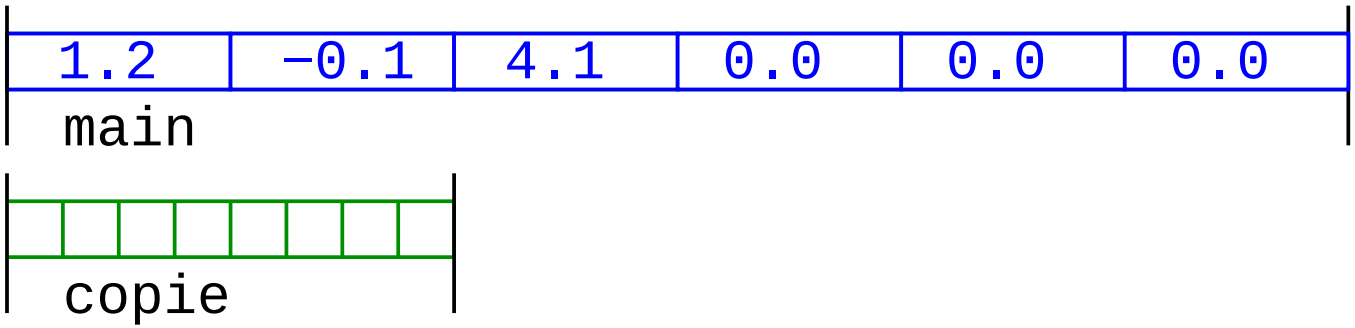
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

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:

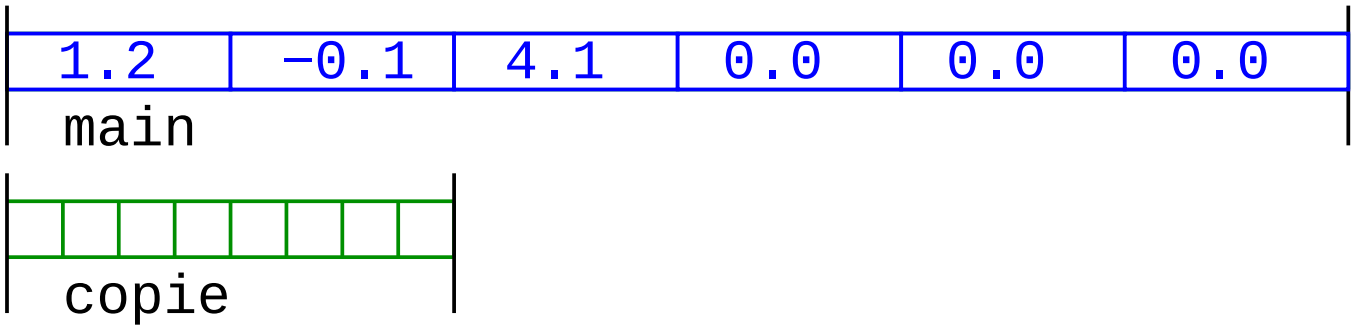
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

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:

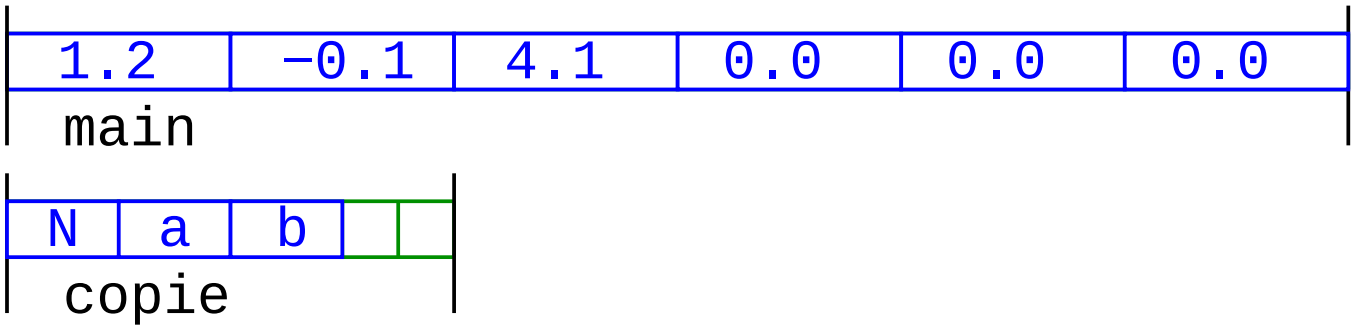
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

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:

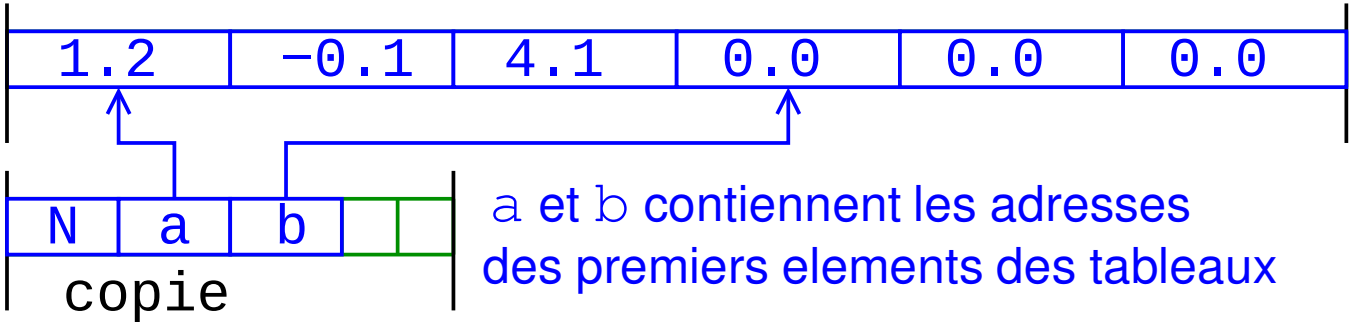
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

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:

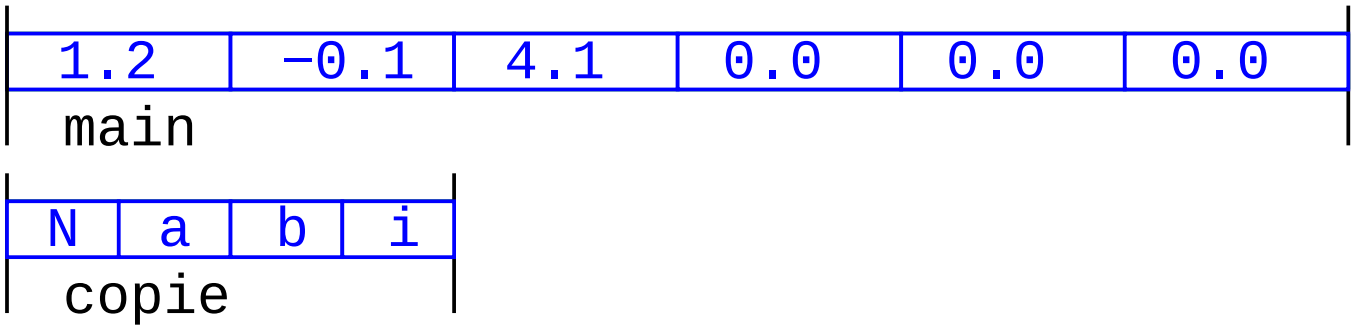
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

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:

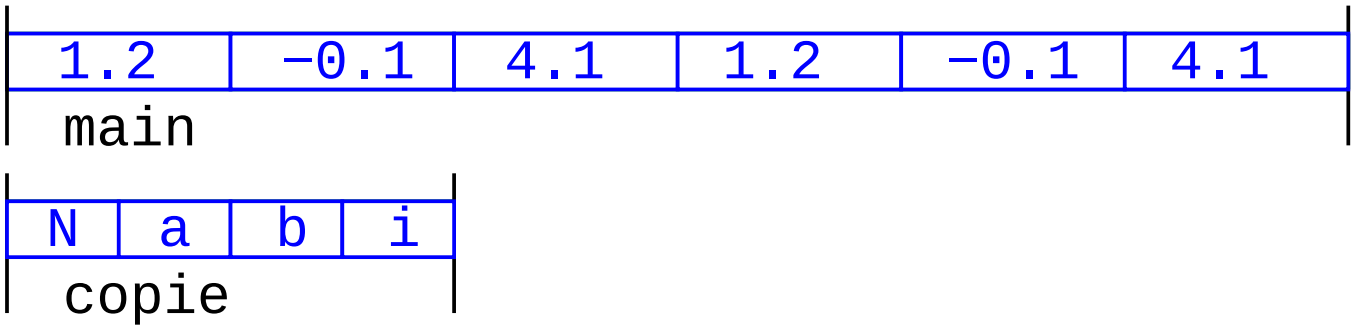
```
void copie( int N, double a[], double b[] )
{
  for ( int i = 0; i < N; i++ )
    b[ i ] = a[ i ];
}

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"

7.2 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⟩ [ ⟨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⟩

7.2 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`

7.2 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⟩ [ ] [ ⟨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

7.2 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⟩ [ ] [ ⟨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

7.2 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⟩] =

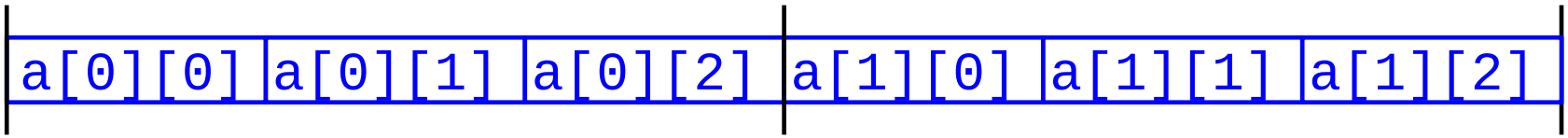
```
{ { { ... }, { ... }, ..., { ... } },
  { { ... }, { ... }, ..., { ... } },
      ...
  { { ... }, { ... }, ..., { ... } } };
```

Accès aux éléments:

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

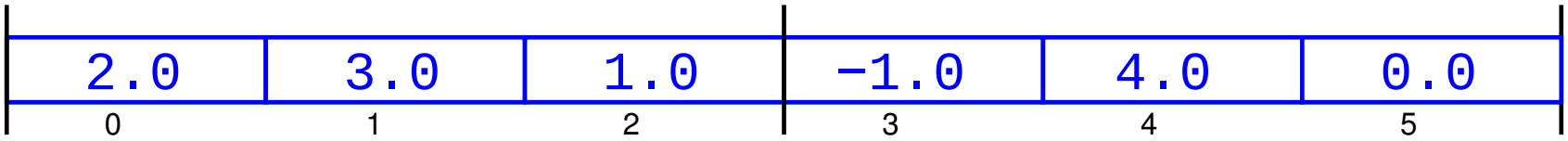
7.2 Les tableaux multidimensionnels

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



7.2 Les tableaux multidimensionnels

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



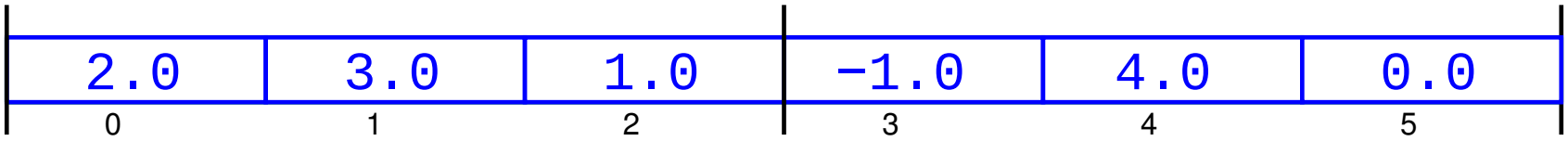
Exécution:

```
4
```

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

7.2 Les tableaux multidimensionnels

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



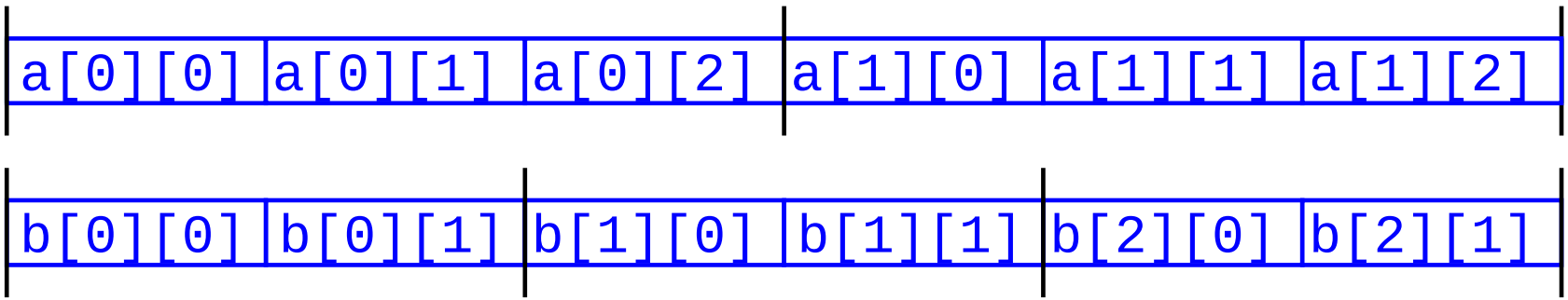
Exécution:

```
-1
```

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

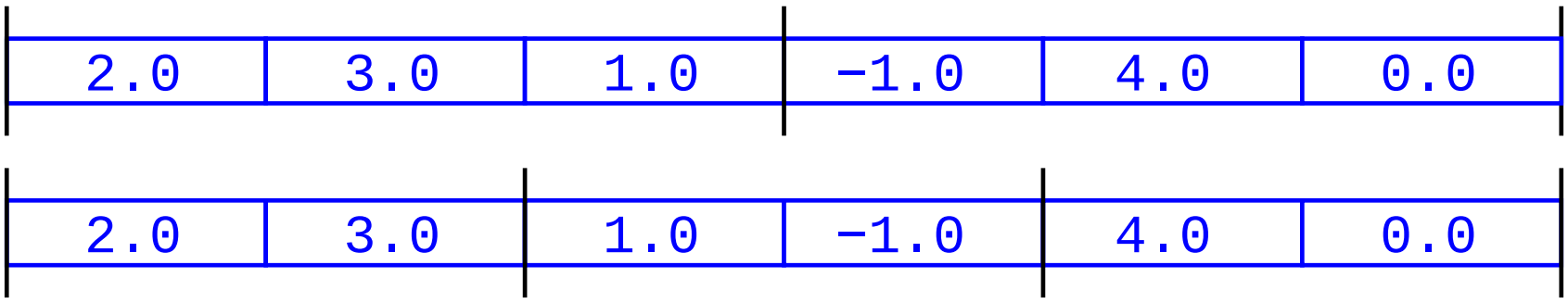
7.2 Les tableaux multidimensionnels

```
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;
}
```



7.2 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;
}
```

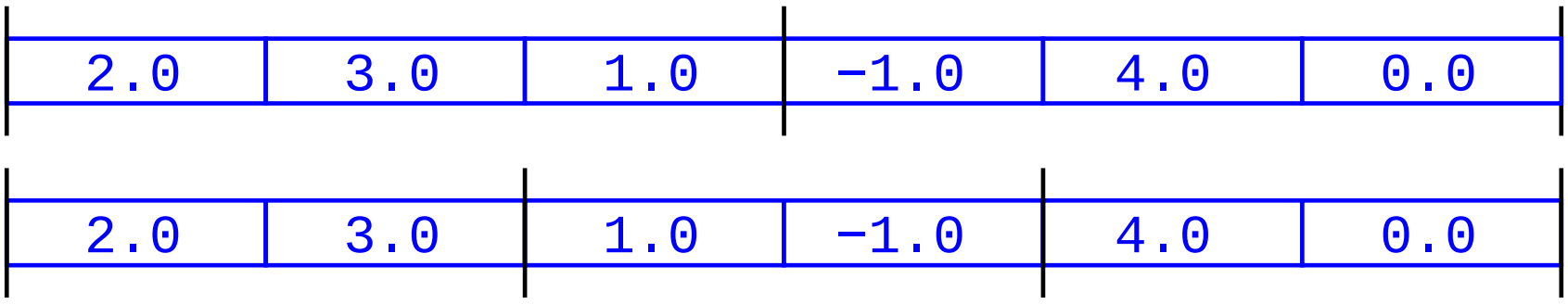


Exécution:

```
4
-1
```

7.2 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
```

7.2 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;
}
```

7.2 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
```

7.2 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
```

7.2 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

7.2 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;
}
```

7.2 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
```

7.2 Les tableaux multidimensionnels

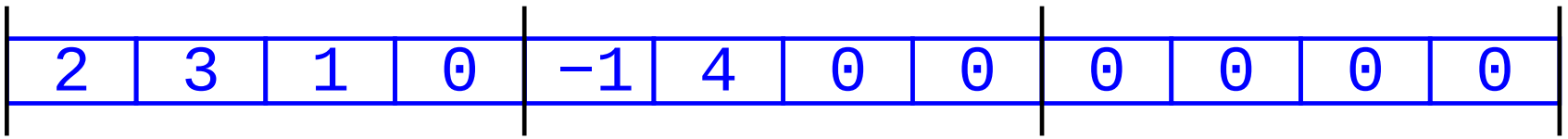
```
const int M = 4;

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

7.2 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;
}
```



7.2 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 ] )
{
  ...
}
```

7.2 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 ] )
{
  ...
}
```

7.2 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 $M \times M$

7.2 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[] );

...
```

7.2 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[] )
{
  ...
}
```

7.2 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

7.3 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
```

7.3 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;
```

7.3 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;
```

7.3 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

7.3 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
```

7.3 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
```

7.3 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
```