Algorithmique
Cours 8

IUT Informatique de Lens, 1ère Année
Université d'Artois

Frédéric Koriche
koriche@cril.fr
2011 - Semestre 1

Sommaire

L'objectif de ce cours est d'étudier les **tableaux dynamiques et multi-dimensionnels** en algorithmique et programmation C++.

1. **Tableaux Statiques (Rappel)**
2. **Tableaux Dynamiques**
3. **Tableaux Multi-Dimensionnels**

Sommaire

L'objectif de ce cours est d'étudier les **tableaux dynamiques et multi-dimensionnels** en algorithmique et programmation C++.

1. **Tableaux Statiques (Rappel)**
2. **Tableaux Dynamiques**
3. **Tableaux Multi-Dimensionnels**

Tableaux unidimensionnels statiques

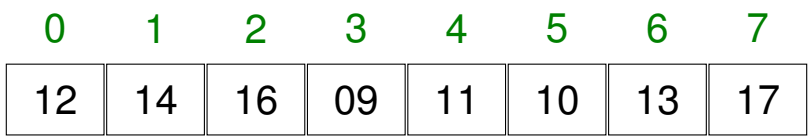
Un **tableau unidimensionnel statique** est une séquence de données du même type accessibles par leur index.

- la taille de la séquence est constante ; elle est spécifiée une fois pour toute lors de la déclaration du tableau,
- le premier index d'un tableau de N éléments est 0 et le dernier index est $N-1$.

Tableaux unidimensionnels statiques

Un **tableau unidimensionnel statique** est une séquence de données du même type accessibles par leur index.

- la taille de la séquence est constante ; elle est spécifiée une fois pour toute lors de la déclaration du tableau,
- le premier index d'un tableau de N éléments est 0 et le dernier index est $N-1$.

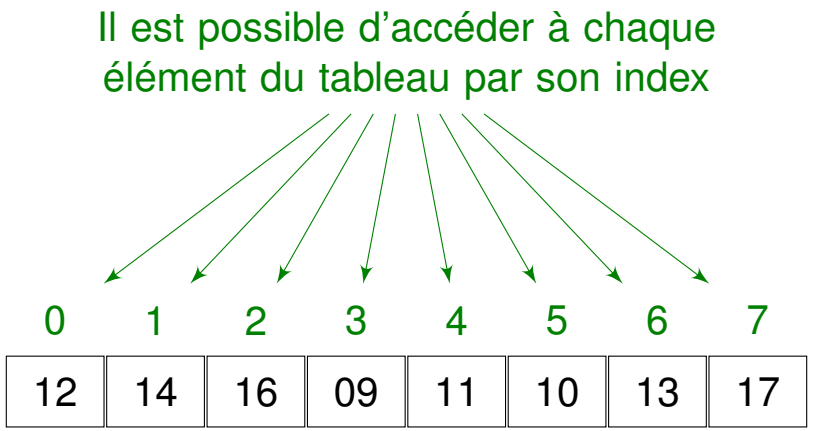


Un tableau de 8 entiers

Tableaux unidimensionnels statiques

Un **tableau unidimensionnel statique** est une séquence de données du même type accessibles par leur index.

- la taille de la séquence est constante ; elle est spécifiée une fois pour toute lors de la déclaration du tableau,
- le premier index d'un tableau de N éléments est 0 et le dernier index est $N-1$.

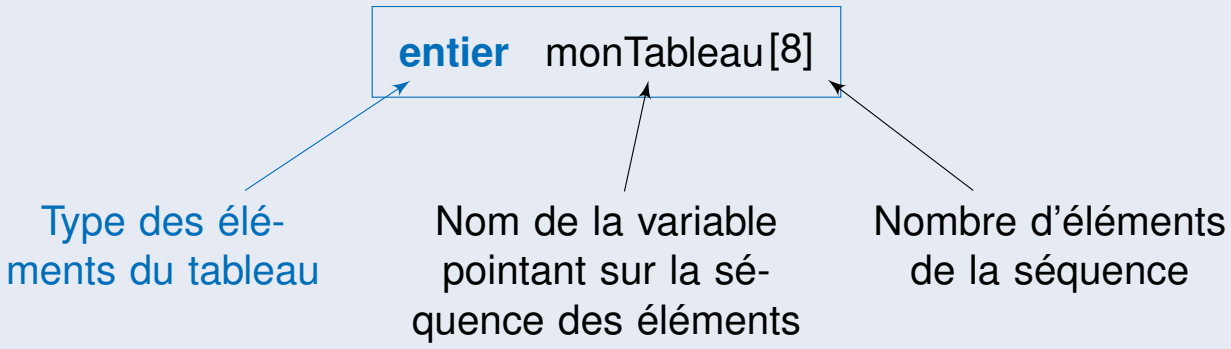


Un tableau de 8 entiers

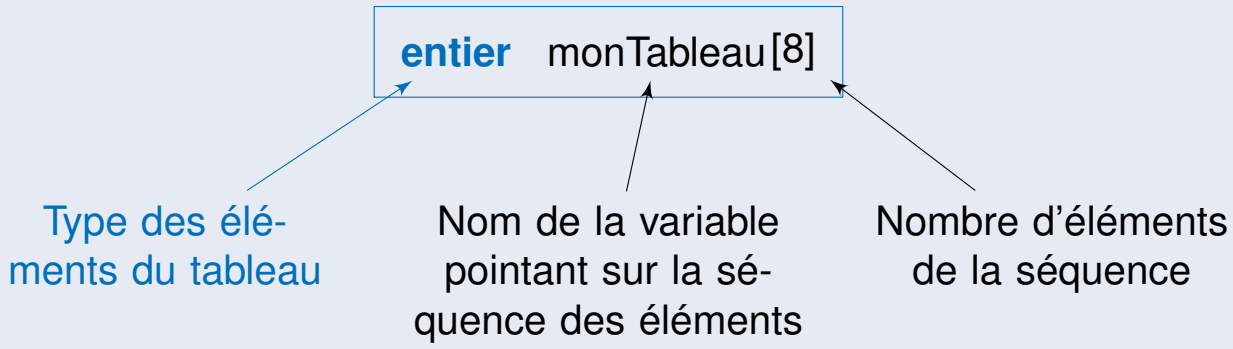
Déclaration d'une variable tableau en pseudo-code

```
entier monTableau[8]
```

Déclaration d'une variable tableau en pseudo-code



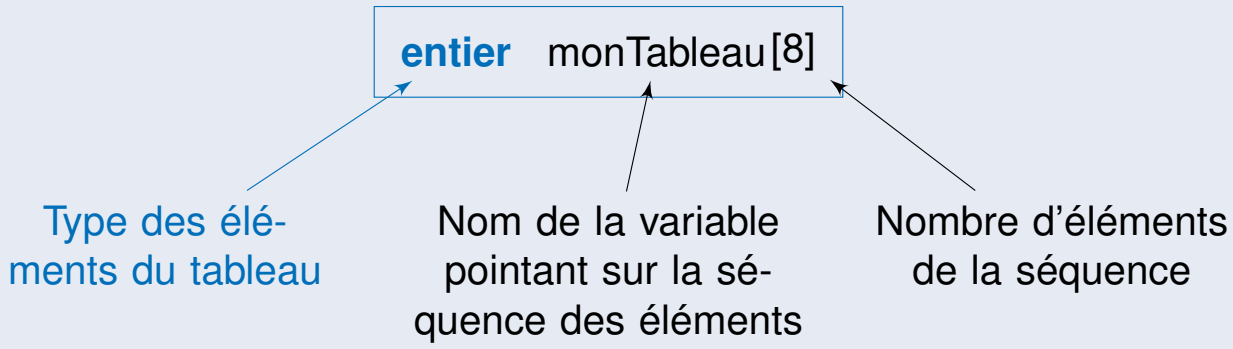
Déclaration d'une variable tableau en pseudo-code



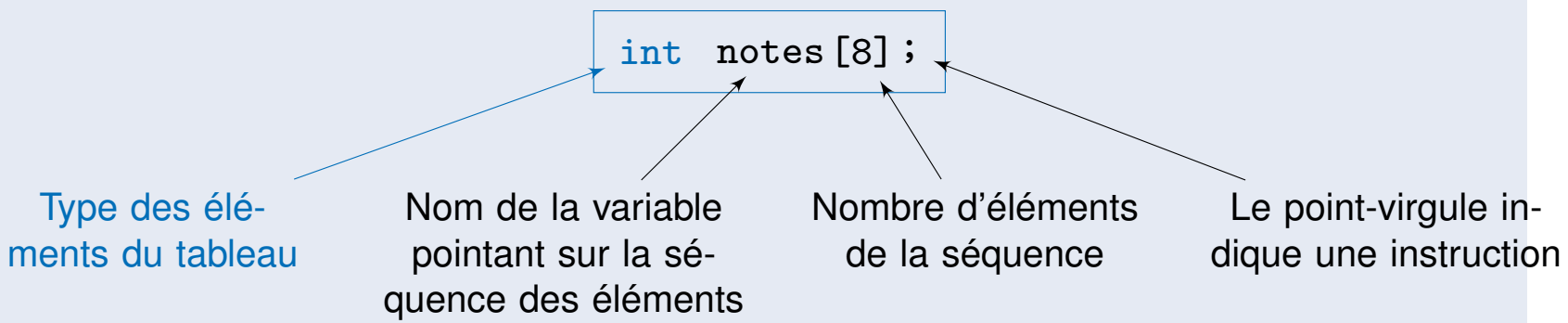
Déclaration d'une variable tableau en C++

```
int notes[8];
```

Déclaration d'une variable tableau en pseudo-code



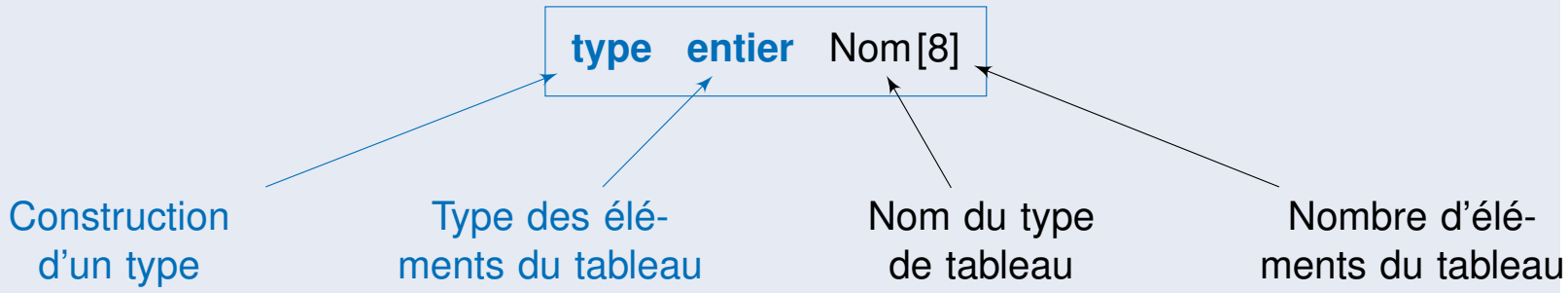
Déclaration d'une variable tableau en C++



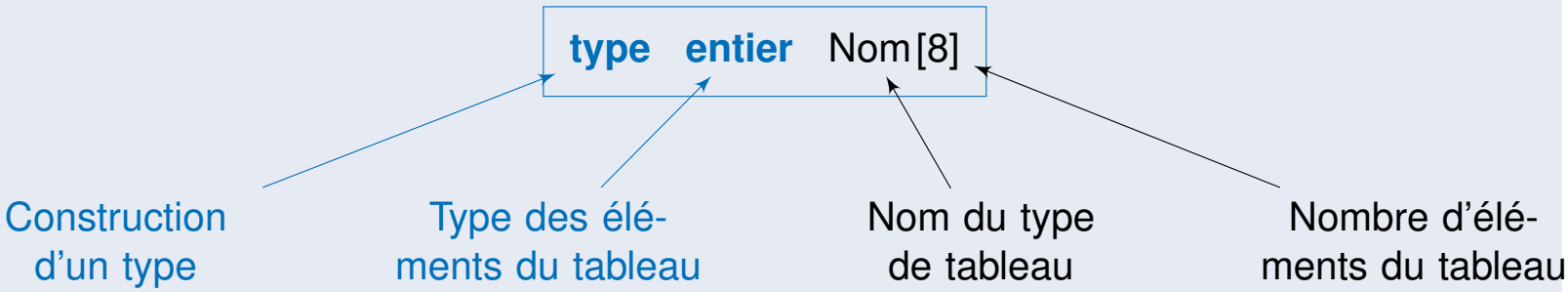
Déclaration d'un type de tableau en pseudo-code

```
type entier Nom[8]
```

Déclaration d'un type de tableau en pseudo-code



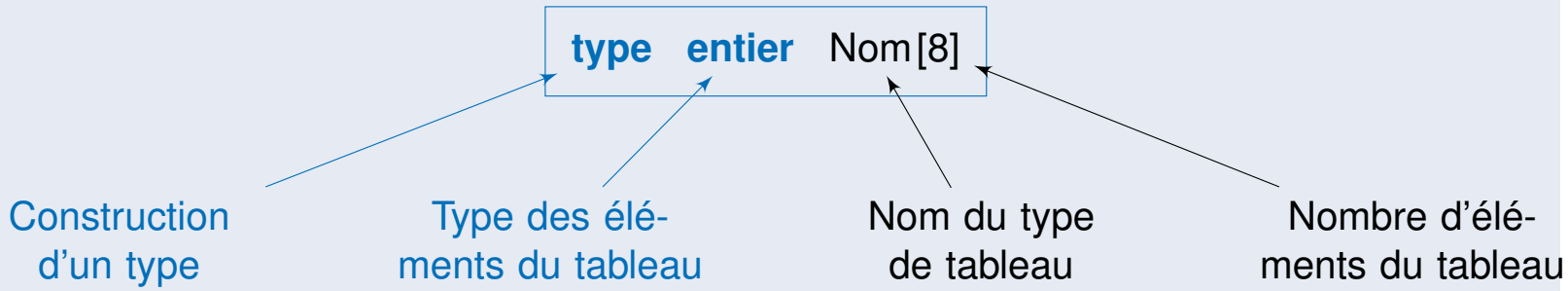
Déclaration d'un type de tableau en pseudo-code



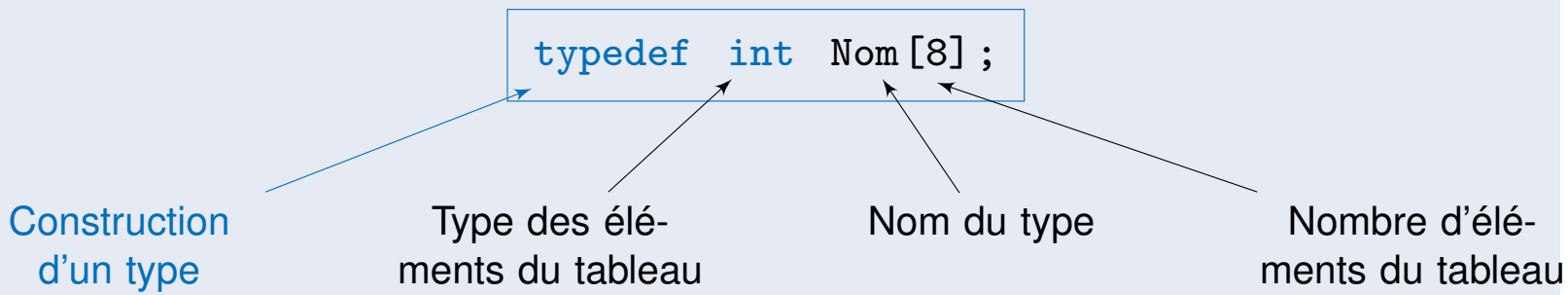
Déclaration d'une variable tableau en C++

```
typedef int Nom[8];
```

Déclaration d'un type de tableau en pseudo-code



Déclaration d'une variable tableau en C++



Exemple d'utilisation d'un type tableau en C++

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


typedef int TableauDeNotes[8];


int maximum(const TableauDeNotes& tab)
{
int max = 0;
for (int i = 0; i < 8; i++)
   if (tab[i] > max) max = tab[i];
return max;
}


int main()
{
TableauDeNotes notes = {12,14,16,09,11,13,10,17};
cout << maximum(notes);
return 0;
}
```

Exemple d'utilisation d'un type tableau en C++

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


typedef int TableauDeNotes[8];


int maximum(const TableauDeNotes& tab)
{
int max = 0;
for (int i = 0; i < 8; i++)
   if (tab[i] > max) max = tab[i];
return max;
}


int main()
{
TableauDeNotes notes = {12,14,16,09,11,13,10,17};
cout << maximum(notes);
return 0;
}
```

Déclaration du type `TableauDeNotes`

Utilisation du type en argument de fonction

Déclaration de la variable `notes`

Affichage de la valeur 17

Opérations sur les tableaux statiques

- **Opérateurs de construction**

Opération	Type	Pseudo-code	C++
Initialisation	**pointeur**	← et {}	= et { }

Opérations sur les tableaux statiques

- **Opérateurs de construction**

Opération	Type	Pseudo-code	C++
Initialisation	**pointeur**	← et {}	= et { }

- **Opérateur d'accès**

Opération	Type	Pseudo-code	C++
Accès par l'index	type de l'élément	[ ]	[ ]

Opérations sur les tableaux statiques

- **Opérateurs de construction**

Opération	Type	Pseudo-code	C++
Initialisation	**pointeur**	← et {}	= et { }

- **Opérateur d'accès**

Opération	Type	Pseudo-code	C++
Accès par l'index	type de l'élément	[ ]	[ ]

Limitations

Une fois le tableau déclaré, seule l'opération d'accès est possible ! En particulier,

- Il est impossible de ré-affecter tout un tableau une fois qu'il a été déclaré
- Il est impossible de changer sa taille

Limitations des tableaux statiques en C++

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


int maximum(const & int tab[8])
{
int max = 0;
for (int i = 0; i < 8; i++)
   if (tab[i] > max) max = tab[i];
return max;
}


int main()
{
int notes[8];
notes = {12,14,16,09,11,13,10,17};
cout << maximum(notes);
return 0;
}
```

Limitations des tableaux statiques en C++

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


int maximum(const & int tab[8])
{
int max = 0;
for (int i = 0; i < 8; i++)
   if (tab[i] > max) max = tab[i];
return max;
}


int main()
{
int notes[8];
notes = {12,14,16,09,11,13,10,17};
cout << maximum(notes);
return 0;
}
```

Erreur ! On ne peut pas envoyer un tableau avec un nombre fixé d'éléments en argument de fonction. Il faut passer par son type (ou son pointeur)

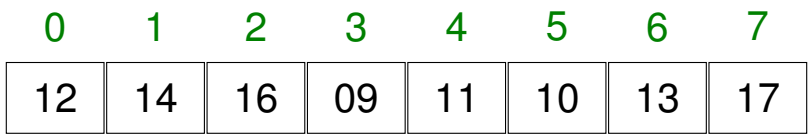
Erreur ! On ne peut pas ré-affecter tout un tableau déjà déclaré

Tableaux unidimensionnels dynamiques

Un **tableau unidimensionnel dynamique** ou **vecteur** est une séquence de données du même type ; la taille de la séquence est variable (elle peut changer au cours de l'exécution du programme).

Tableaux unidimensionnels dynamiques

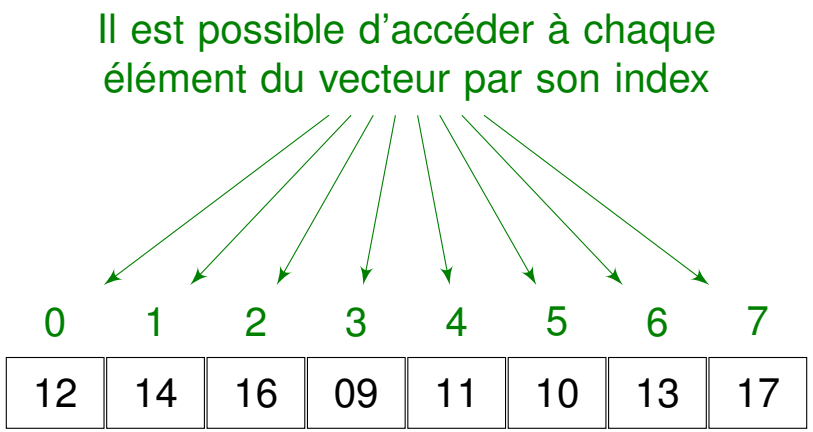
Un **tableau unidimensionnel dynamique** ou **vecteur** est une séquence de données du même type ; la taille de la séquence est variable (elle peut changer au cours de l'exécution du programme).



Un vecteur initial de 8 entiers

Tableaux unidimensionnels dynamiques

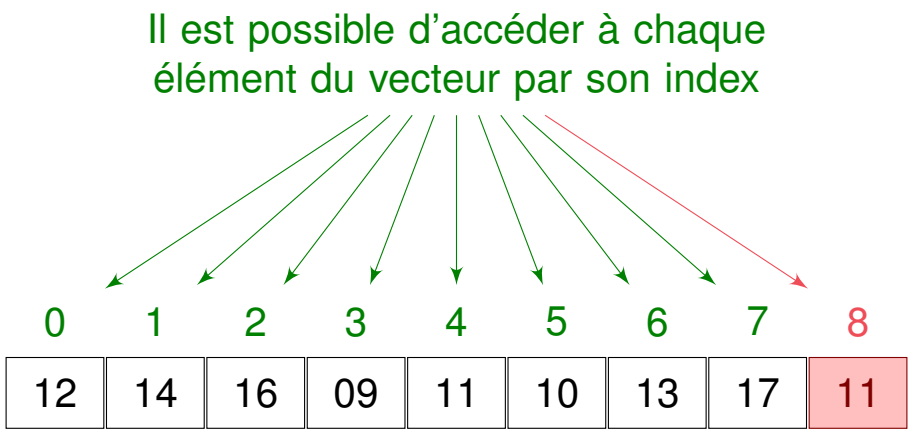
Un **tableau unidimensionnel dynamique** ou **vecteur** est une séquence de données du même type ; la taille de la séquence est variable (elle peut changer au cours de l'exécution du programme).



Un vecteur initial de 8 entiers

Tableaux unidimensionnels dynamiques

Un **tableau unidimensionnel dynamique** ou **vecteur** est une séquence de données du même type ; la taille de la séquence est variable (elle peut changer au cours de l'exécution du programme).

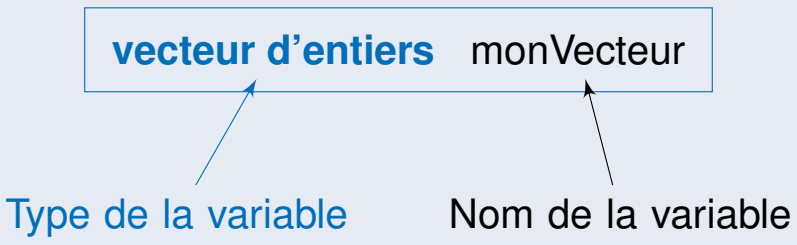


Le vecteur avec un élément supplémentaire ajouté en fin de séquence

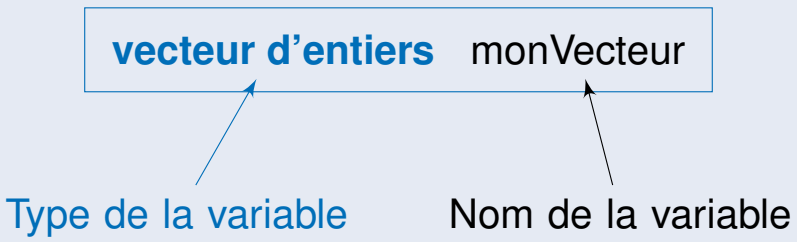
Déclaration d'une variable de type vecteur en pseudo-code

```
vecteur d'entiers monVecteur
```

Déclaration d'une variable de type vecteur en pseudo-code



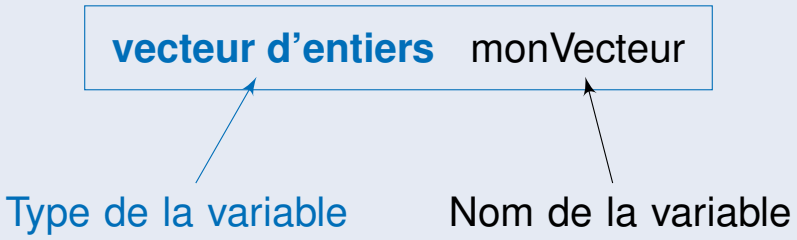
Déclaration d'une variable de type vecteur en pseudo-code



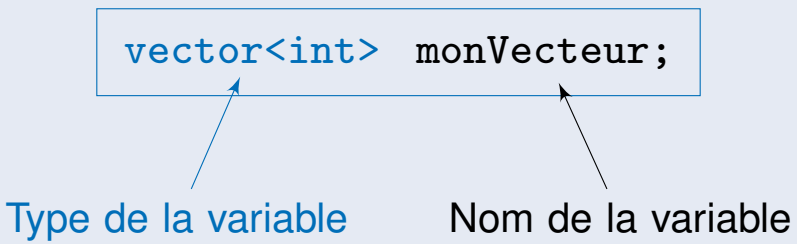
Initialisation d'un vecteur en C++

```
vector<int> monVecteur;
```

Déclaration d'une variable de type vecteur en pseudo-code



Initialisation d'un vecteur en C++



Opérations sur les vecteurs

Opérateurs de construction

Opération	Type	Pseudo-code	C++
Initialisation	**vecteur**	()	()
Copie	**vecteur**	←	=

Opérations sur les vecteurs

Opérateurs de construction

Opération	Type	Pseudo-code	C++
Initialisation	**vecteur**	()	`()`
Copie	**vecteur**	←	=

Opérateurs de taille

Opération	Type	Pseudo-code	C++
Longueur d'un vecteur	**entier**	longueur()	`.size()`
Test du vecteur vide	**booléen**	estVide()	`.empty()`

Opérations sur les vecteurs

Opérateurs de construction

Opération	Type	Pseudo-code	C++
Initialisation	**vecteur**	()	`()`
Copie	**vecteur**	←	=

Opérateurs de taille

Opération	Type	Pseudo-code	C++
Longueur d'un vecteur	**entier**	longueur()	`.size()`
Test du vecteur vide	**booléen**	estVide()	`.empty()`

Opérateur d'accès

Opération	Type	Pseudo-code	C++
Accès par l'index	type de l'élément	[]	`[]`

Opérations sur les vecteurs

Opérateurs de construction

Opération	Type	Pseudo-code	C++
Initialisation	**vecteur**	()	`()`
Copie	**vecteur**	←	=

Opérateurs de taille

Opération	Type	Pseudo-code	C++
Longueur d'un vecteur	**entier**	longueur()	`.size()`
Test du vecteur vide	**booléen**	estVide()	`.empty()`

Opérateur d'accès

Opération	Type	Pseudo-code	C++
Accès par l'index	type de l'élément	[]	`[]`

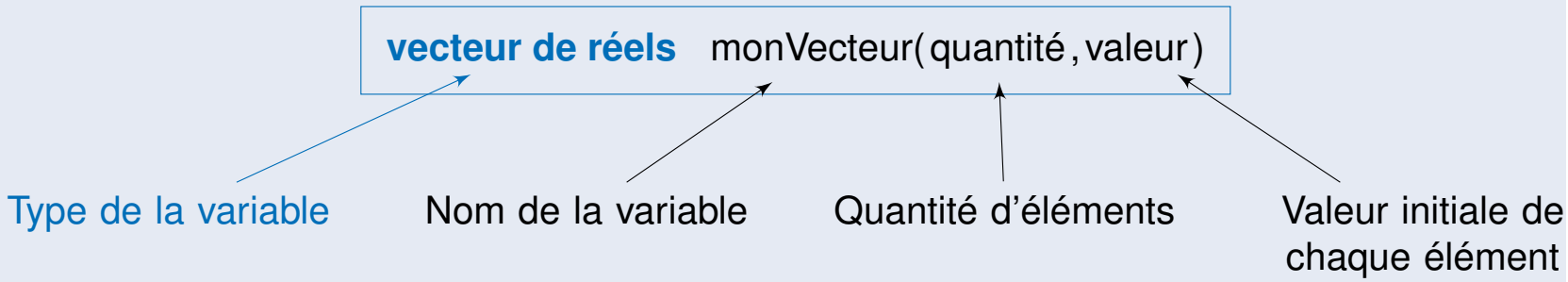
Opérateurs de modification

Opération	Type	Pseudo-code	C++
Ajout d'un élément à la fin	vecteur	+	`.push_back()`
Retrait du dernier élément	vecteur	−	`.pop_back()`
Vider le vecteur	vecteur	vider()	`.clear()`

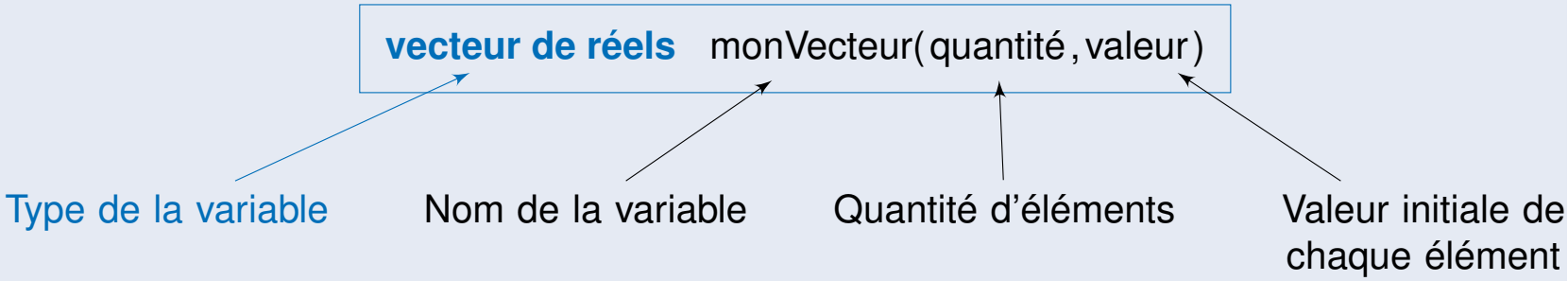
Initialisation d'un vecteur en pseudo-code

vecteur de réels monVecteur(quantité, valeur)

Initialisation d'un vecteur en pseudo-code



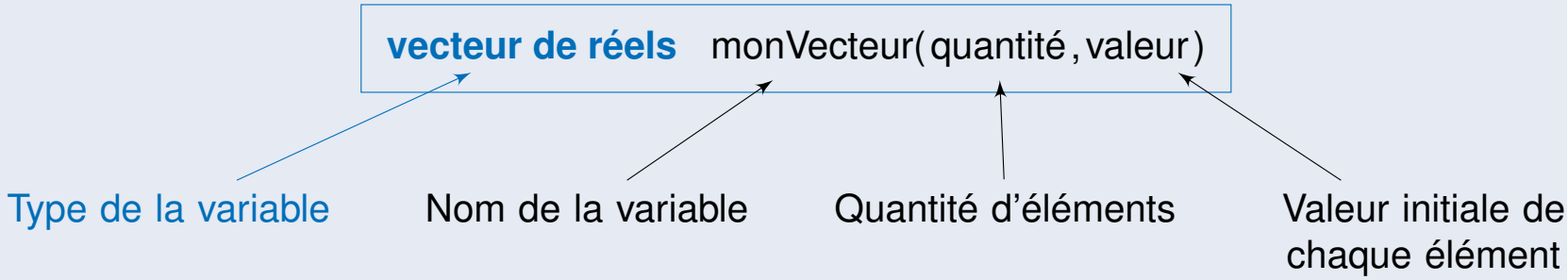
Initialisation d'un vecteur en pseudo-code



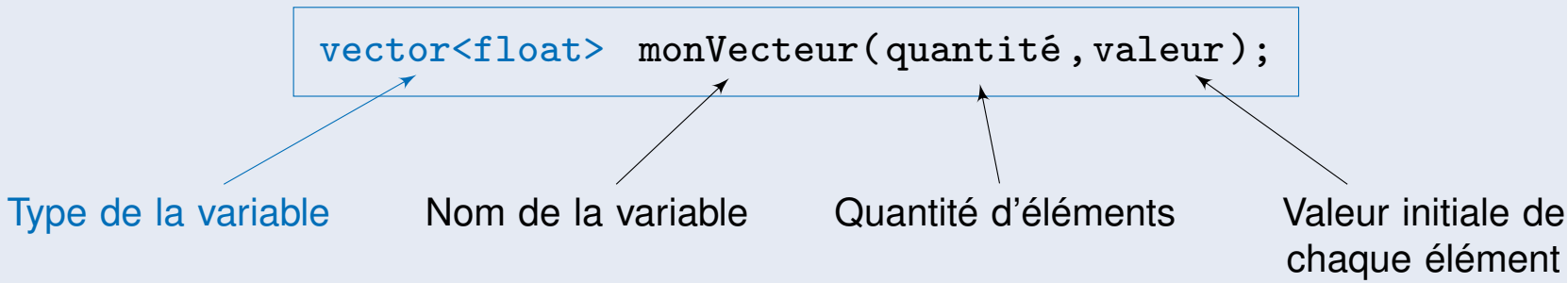
Initialisation d'un vecteur en C++

```
vector<float> monVecteur(quantité,valeur);
```

Initialisation d'un vecteur en pseudo-code



Initialisation d'un vecteur en C++



Exemple d'initialisation de vecteur en C++

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


int main()
{
int n;
cout << "Entrez la quantité d'éléments";
cin >> n;
vector<int> monVecteur(n,0);
return 0;
}
```

Exemple d'initialisation de vecteur en C++

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


int main()
{
int n;
cout << "Entrez la quantité d'éléments";
cin >> n;
vector<int> monVecteur(n,0);
return 0;
}
```

Variable entière `n` dont la valeur est saisie par l'utilisateur

Variable vecteur d'entiers `monVecteur` dont la taille n'est connue qu'à l'exécution !

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

L'état de la variable `monVecteur` pour `n = 8`

Exemple de copie de vecteurs en C++

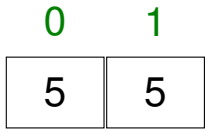
```
#include <iostream>
using namespace std;


int main()
{
vector<int> premier(2,5);
vector<int> deuxième(3,1);

deuxième = premier;


return 0;
}
```

Exemple de copie de vecteurs en C++

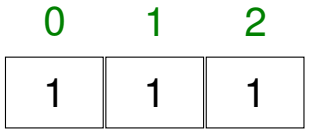
```
#include <iostream>
using namespace std;


int main()
{
vector<int> premier(2,5);
vector<int> deuxième(3,1);

deuxième = premier;


return 0;
}
```

Initialisation du premier vecteur

Initialisation du second vecteur



Etat de la variable `premier`



Etat initial de la variable `deuxième`

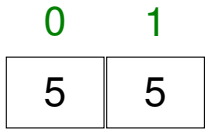
Exemple de copie de vecteurs en C++

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

int main()
{
vector<int> premier(2,5);
vector<int> deuxième(3,1);

deuxième = premier;

return 0;
}
```

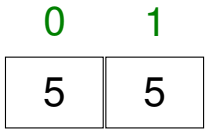
Initialisation du premier vecteur

Initialisation du second vecteur

Tous les éléments du deuxième vecteur sont éliminés, et remplacés par les éléments du premier



Etat de la variable `premier`



Etat de la variable `deuxième` après affectation

Exemple de modification de vecteurs en C++

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


int main()
{
vector<string> mesAmis;


mesAmis.push_back("Anne");
mesAmis.push_back("Bob");
mesAmis.push_back("Jean");


cout << mesAmis[1] << endl;
return 0;
}
```

Exemple de modification de vecteurs en C++

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


int main()
{
vector<string> mesAmis;


mesAmis.push_back("Anne");
mesAmis.push_back("Bob");
mesAmis.push_back("Jean");


cout << mesAmis[1] << endl;
return 0;
}
```

Déclaration d'un vecteur de chaînes

Ajout successif de chaînes

Affichage du deuxième élément

Etat initial de la variable `mesAmis`

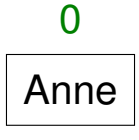
Exemple de modification de vecteurs en C++

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


int main()
{
vector<string> mesAmis;


mesAmis.push_back("Anne");
mesAmis.push_back("Bob");
mesAmis.push_back("Jean");


cout << mesAmis[1] << endl;
return 0;
}
```

Déclaration d'un vecteur de chaînes

Ajout successif de chaînes

Affichage du deuxième élément



Etat de `mesAmis` après le premier ajout

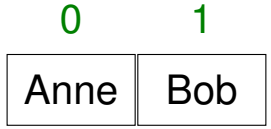
Exemple de modification de vecteurs en C++

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


int main()
{
vector<string> mesAmis;


mesAmis.push_back("Anne");
mesAmis.push_back("Bob");
mesAmis.push_back("Jean");


cout << mesAmis[1] << endl;
return 0;
}
```

Déclaration d'un vecteur de chaînes

Ajout successif de chaînes

Affichage du deuxième élément



Etat de mes `mesAmis` après le second ajout

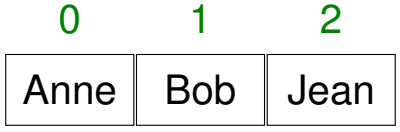
Exemple de modification de vecteurs en C++

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


int main()
{
vector<string> mesAmis;


mesAmis.push_back("Anne");
mesAmis.push_back("Bob");
mesAmis.push_back("Jean");


cout << mesAmis[1] << endl;
return 0;
}
```

Déclaration d'un vecteur de chaînes

Ajout successif de chaînes

Affichage du deuxième élément



Etat de mes `mesAmis` après le troisème ajout

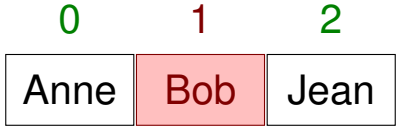
Exemple de modification de vecteurs en C++

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


int main()
{
vector<string> mesAmis;


mesAmis.push_back("Anne");
mesAmis.push_back("Bob");
mesAmis.push_back("Jean");


cout << mesAmis[1] << endl;
return 0;
}
```

Déclaration d'un vecteur de chaînes

Ajout successif de chaînes

Affichage du deuxième élément



Accès à l'index 1 de `mesAmis`

Exemple de fonction de recherche dans un vecteur en C++

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


bool appartient(const string & nom, const vector<string> & dico)
{
bool trouve = false;
int i = 0;


while (! trouve  &&  i < dico.size())
    {
     trouve = (dico[i] == nom);
     i++;
    }


return trouve;
}
```

La fonction `appartient` retourne `vrai` ssi la chaîne `nom` appartient au vecteur de chaînes `dico`

Exemple de procédure de construction d'un vecteur en C++

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


void construire(vector<string> & dico)
{
string nom;


dico.clear();
while (nom != "fin")
    {
     cin >> nom;
     if(nom != "fin")
       dico.push_back(nom);
    }


}
```

La procédure `construire` démarre avec le vecteur `dico` vide et insère au fur et à mesure des chaînes saisies par l'utilisateur

Tableaux multi-dimensionnels

Un **tableau de dimension** d est une séquence de tableaux de dimension $d-1$. En particulier, une **matrice** est un tableau de dimension 2.

Tableaux multi-dimensionnels

Un **tableau de dimension** *d* est une séquence de tableaux de dimension $d-1$. En particulier, une **matrice** est un tableau de dimension 2.

	0	1	2	3	4	5	6	7
0	12	14	16	09	11	10	13	17
1	01	04	17	11	08	02	06	19
2	00	18	05	03	01	00	19	16
3	17	08	19	13	07	04	01	14

Une matrice appelée `maMatrice` comprenant 4 rangées et 8 colonnes d'entiers

Tableaux multi-dimensionnels

Un **tableau de dimension** *d* est une séquence de tableaux de dimension $d-1$. En particulier, une **matrice** est un tableau de dimension 2.

Il est possible d'accéder à chaque élément du vecteur par sa rangée et sa colonne. Par exemple, la valeur de `maMatrice[1][2]` est 17

	0	1	2	3	4	5	6	7
0	12	14	16	09	11	10	13	17
1	01	04	17	11	08	02	06	19
2	00	18	05	03	01	00	19	16
3	17	08	19	13	07	04	01	14

Une matrice appelée `maMatrice` comprenant 4 rangées et 8 colonnes d'entiers

Matrices

Les matrices sont très utilisées en informatique :

- Pour représenter des relations entre objets (villes d'une carte, pièces d'un jeu, etc.)
- Pour appliquer des opérations linéaires sur des objets (physique, graphisme, etc.)

X		X
	O	X
O	X	O

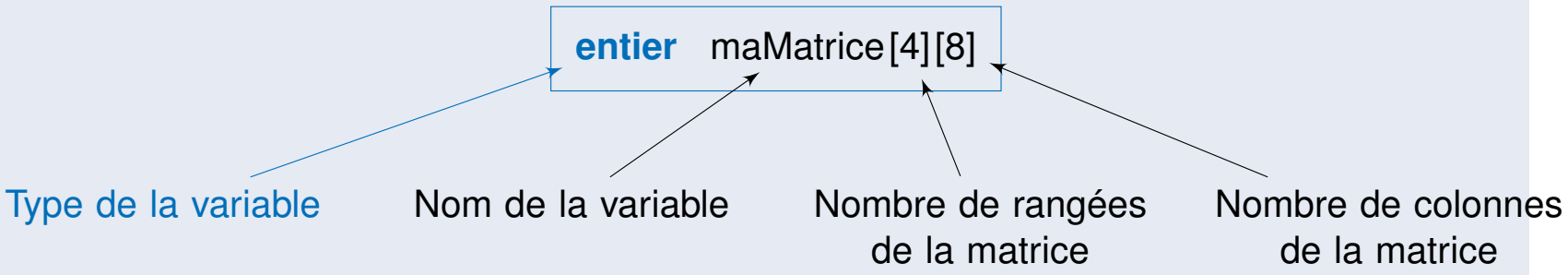
	0	1	2
0	2	0	2
1	0	1	2
2	1	2	1

Le jeu de tic-tac-toe (morpion) codé par une matrice d'entiers : le joueur **X** est codé par 2, le joueur **O** est codé par 1, et la case vide est codée par 0

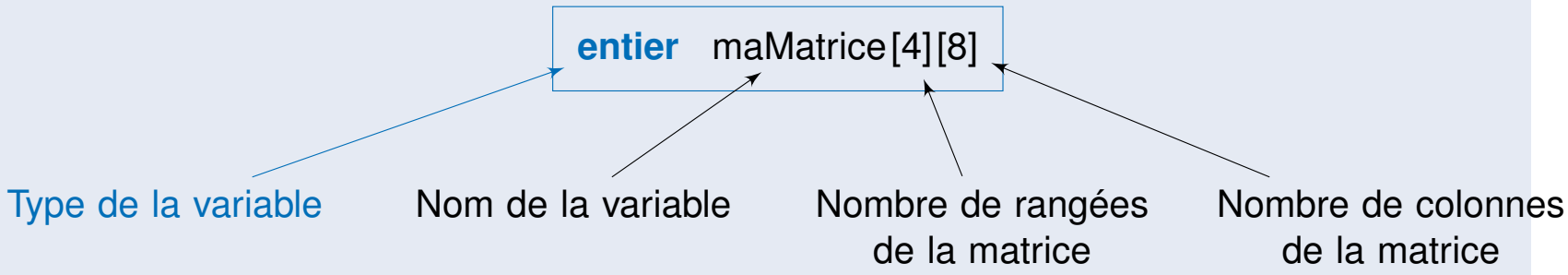
Déclaration d'une matrice statique en pseudo-code

```
entier maMatrice[4][8]
```

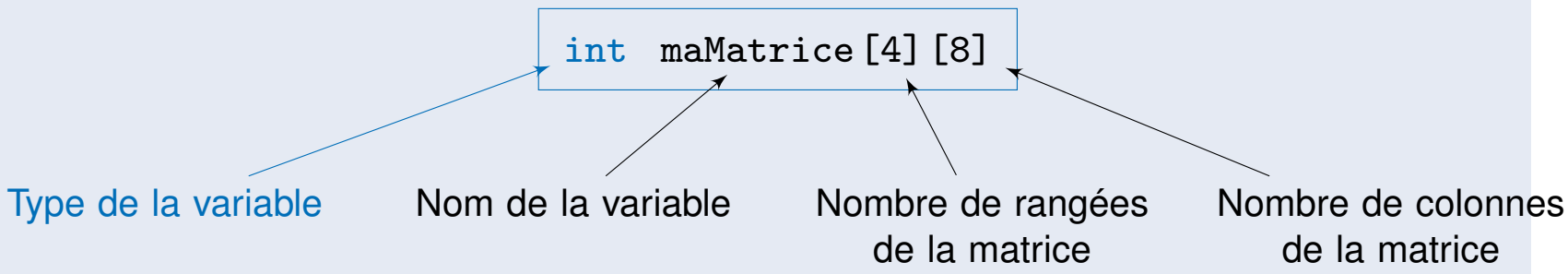
Déclaration d'une matrice statique en pseudo-code



Déclaration d'une matrice statique en pseudo-code



Déclaration d'une matrice statique en C++



Exemple d'utilisation de matrices statiques en C++

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


typedef int Matrix3by3 [3][3];


void somme(const Matrix3by3& A, const Matrix3by3& B, Matrix3by3& C)
{
for (int i=0; i < 3; i++)
  for (int j=0; j < 3; j++)
    C[i][j] = A[i][j] + B[i][j];
}
```

	0	1	2
0	3	0	6
1	4	2	2
2	1	5	1

+

	0	1	2
0	7	1	5
1	2	1	4
2	2	1	3

=

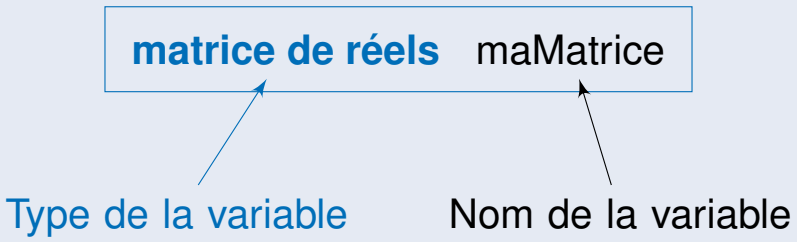
	0	1	2
0	10	1	11
1	6	3	6
2	3	6	4

La procédure `somme` calcule la somme C de deux matrices A et B

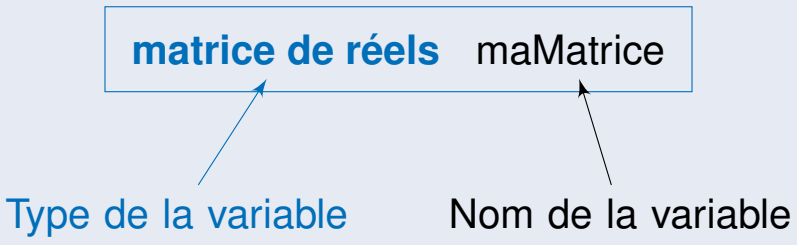
Déclaration d'une matrice dynamique en pseudo-code

```
matrice de réels maMatrice
```

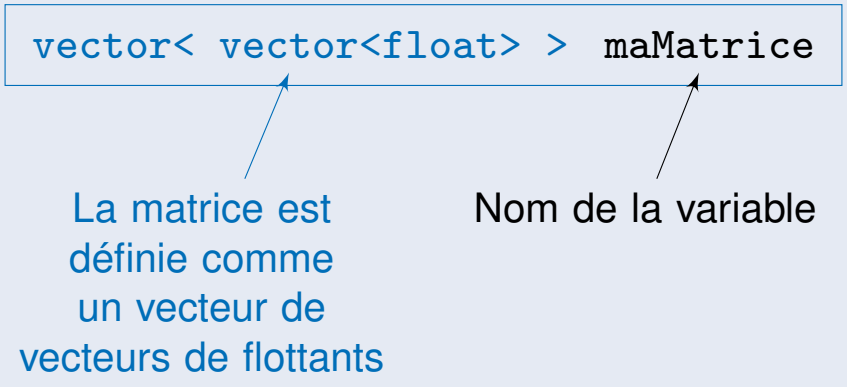
Déclaration d'une matrice dynamique en pseudo-code



Déclaration d'une matrice dynamique en pseudo-code



Déclaration d'une matrice statique en C++



Tableaux de dimension supérieure

Ils peuvent être représentés de manière statique ou dynamique en étendant simplement les définitions précédentes :

- Un tableau 3D statique : `float monTableau[10][8][6];`
- Un tableau 3D dynamique : `vector< vector< vector<float> > > monTableau;`

Attention ! L'espace mémoire augmente exponentiellement avec le nombre de dimensions (ex : $10 \times 8 \times 6 = 480$ éléments flottants, soit 1920 octets)

Tableaux de dimension supérieure

Ils peuvent être représentés de manière statique ou dynamique en étendant simplement les définitions précédentes :

- Un tableau 3D statique : `float monTableau[10][8][6];`
- Un tableau 3D dynamique : `vector< vector< vector<float> > > monTableau;`

Attention ! L'espace mémoire augmente exponentiellement avec le nombre de dimensions (ex : $10 \times 8 \times 6 = 480$ éléments flottants, soit 1920 octets)

Tableaux statiques versus Tableaux dynamiques

- La représentation statique peut être utilisée lorsque la taille du tableau est connue à l'avance et reste fixe pendant tout le déroulement du programme (elle est quelquefois préférable pour certaines applications).
- La représentation dynamique est préférable (voir indispensable) lorsque la taille du tableau n'est pas connue à l'avance ou peut changer au cours du temps.