Boucle Sur Intervalle

Avec Valeurs Entre Accolades

```
void avec_valeurs_entre_accolades () {
        for (int x : { 2, 5, 11 })  // La variable x aura la valeur 2 dans la
                                    première itération de la boucle, 5 dans la deuxième
                                    itération et 11 dans la troisième itération.
                cout << x << " ";
        cout << endl;
}
```

Boucle Sur Intervalle

Avec Tableau Par valeur

```
void avec_tableau_par_valeur () {
	int tableau[] = { 23, 45, 67, 89 };
	for (int x : tableau)  // La variable x aura la valeur 23 dans la première itération de la boucle, 45 dans la deuxième, etc.
		cout << x << " ";  // Si on modifie x, ça ne change pas la valeur dans le tableau car x est une copie de la valeur du tableau.
	cout << endl;
}
```

Boucle Sur Intervalle

Avec Tableau Par Référence

```
void avec_tableau_par_référence () {
    int tableau[] = { 23, 45, 67, 89 };
    for (int& x : tableau)  // La variable x va référer au 23 dans la première
                               itération de la boucle, 45 dans la deuxième, etc.
        x -= 10;  // Modifier x modifie la valeur dans le tableau car x réfère à la
                     valeur dans le tableau, ce n'est pas une copie.
    // Le tableau vaut { 13, 35, 57, 79 } après la boucle.
}
```

Boucle Sur Intervalle Avec Span

Fonctions utilisées pour **Avec Tableau En Paramètre**

```
void afficher(const int tableau[], int dim) {  // Comme diapos V.16 et V.36.
    // Comment faire la boucle sur intervalle avec 'tableau' ?
    //for (int x : tableau)  // Ne compile pas car il ne sait pas combien il y a
                             d'éléments dans le tableau; le compilateur ne sait pas que le
                             paramètre 'dim' indique ce nombre d'éléments.
    for (int x : span(tableau, dim))  // 'span' permet de lui dire le nombre
                             d'éléments; 'span' ne copie pas les valeurs, 'x' prendra ici une
                             copie des valeurs du tableau.
        cout << x << " ";
    cout << endl;
}
```

```
void réduireDe10(int tableau[], int dim) {  // Paramètres comme diapos V.16 et
                                               V.36.
    //for (int& x : tableau)  // Comme dans la fonction précédente, on ne peut pas
                                 directement faire cette boucle car il ne sait pas combien il y
                                 a d'éléments dans tableau.
    for (int& x : span(tableau, dim))
        x -= 10;  // Modifier x modifie la valur dans le tableau car 'span' ne
                     copie pas il réfère aux valeur du tableau original, et 'x' va
                     donc aussi référer aux valeurs du tableau et non à une copie.
}
```

Boucle Sur Intervalle Avec Span

Appel des fonctions pour **Avec Tableau En Paramètre**

```
void avec_tableau_en_paramètre () {
	const int tailleTableau = 4;
	int tableau[tailleTableau] = { 23, 45, 67, 89 };
	afficher(tableau, tailleTableau);
	réduireDe10(tableau, tailleTableau);
	// Le tableau vaut { 13, 35, 57, 79 }
	afficher(tableau, tailleTableau);
}
```

Fonctions utilisées pour

Avec span En Paramètre

```
void afficherSpan(span<const int> tableau) {  // Comme diapo V.38.  Le 'const'
          pour dire qu'on ne veut pas modifier les valeurs du tableau doit être à
          côté du 'int', pas avant 'span'.  Le 'int' dit que les éléments du
          'tableau' sont des 'int'.  On ne prend généralement pas le 'span' par
          référence car on peut considérer que c'est déjà un genre de référence
          puisqu'il ne copie pas les valeurs du tableau.
    for (int x : tableau)
        cout << x << " ";
    cout << endl;
}
```

```
void réduireDe10Span(span<int> tableau) {
    for (int& x : tableau)
        x -= 10;  // Comme le cas de la fonction 'réduireDe10', 'x' réfère bien à
                  la valeur du tableau original et non à une copie ('span' ne fait
                  pas de copie).
}
```

```
void avec_span_en_paramètre () {
    int tableau[] = { 31, 53, 75, 97 };
    afficherSpan(tableau);  // Ici, il sait combien il y a d'éléments dans le
           tableau, donc il arrive à faire le 'span' à partir du tableau sans
           qu'on ait à lui dire le nombre d'éléments; sinon on aurait pu mettre
           span(tableau, taille) comme dans un exemple précédent.
    réduireDe10Span(tableau);
    // Le tableau vaut { 21, 43, 65, 87 }
    afficherSpan(tableau);
```

```
    // Un autre avantage de span est qu'il fonctionne aussi directement avec
       d'autres types de tableaux vus en INF1010:
    array arr = {-14, 55, 24, 67};
    afficherSpan(arr);


    vector vec = {1, 3, 5, 8, 10};
    afficherSpan(vec);
}
```