Boucles sur intervalle

Dans le chapitre 1, nous avons vu la boucle « pour chaque », par exemple « pour chaque lettre de la phrase », ou « pour chaque entier de l'ensemble ». Ce type de boucle existe en C++ depuis C++11, et porte le nom anglais « range-based for loop » que nous traduisons ici par « boucle sur intervalle ».

La syntaxe de la boucle sur intervalle est :

```
for ( Type variable : tableau )
    // Faire qqch avec variable.
```

Ça fonctionne avec les tableaux normaux, des `string` et des `vector` :

```
int tableau[3] = {1, 2, 5};
for ( int valeur : tableau )
    cout << valeur << " ";

string lettres = "bla";
for ( char lettre : lettres )
    cout << lettre;

vector<int> valeurs = {1, 2, 5};
for ( int valeur : valeurs )
    cout << valeur << " ";
```

Et de la même manière aussi pour `span` (troisième partie de ce document).

Le type peut être une référence pour pouvoir modifier les éléments, ou référence constante pour ne pas faire de copie inutile lorsque les éléments peuvent être gros. Par exemple, pour ajouter 42 à chaque valeur :

```
for ( int& valeur : valeurs )
    valeur += 42;
```

Copier des strings peut être coûteux, alors on itère par référence constante si on n'a pas besoin de modifier :

```
for ( const string& valeur : valeurs )
    cout << valeur << endl;
```

Donc, comme il en est pour le passage de paramètres de fonction, on itère par référence modifiable (`Type&`) si on a besoin de modifier les valeurs. Si on n'a pas besoin de modifier, on itère par valeur pour les types primitifs et on passe par référence constante (`const Type&`) pour les autres types (comme `string`).

Utilisation de *cppitertools*

Nous avons aussi inclus la bibliothèque *cppitertools* qui ajoute des fonctions pour construire des intervalles pour ce type de boucles. Vous pouvez utiliser `range()`. Il s'utilise de trois manières possibles :

`range(n)`	donne les valeurs de 0 à *n* (*n* exclu) avec incrément de 1
`range(a, b)`	donne les valeurs de *a* à *b* (*b* exclu) avec incrément de 1
`range(a, b, inc)`	donne les valeurs de *a* à *b* (*b* exclu) avec incrément de *inc*

Par exemple, pour afficher les multiples de 3 de 0 à 12 (inclu) :

```
for ( int valeur : range(0, 12+1, 3) )
    cout << valeur;
```

Pour vous servir de *cppitertools*, vous devez mettre le dossier *cppitertools* dans le dossier de votre projet (au même endroit que les fichiers source), puis changez vos options de compilation pour compiler en C++17. Dans les propriétés de projet, allez dans **C/C++ → Langage → Norme du langage C++** (voir la capture d'écran ci-dessous). Ensuite, dans votre code, pour inclure *range*, faites :

```
#include "cppitertools/range.hpp"
using namespace iter;
```

Ici, vous faites le `using namespace iter` pour la même raison que vous faites `using namespace std` avec la librairie standard. Le `#include` doit être fait avec des guillemets, sans quoi vous aurez besoin de changer les options de dossier de votre projet.

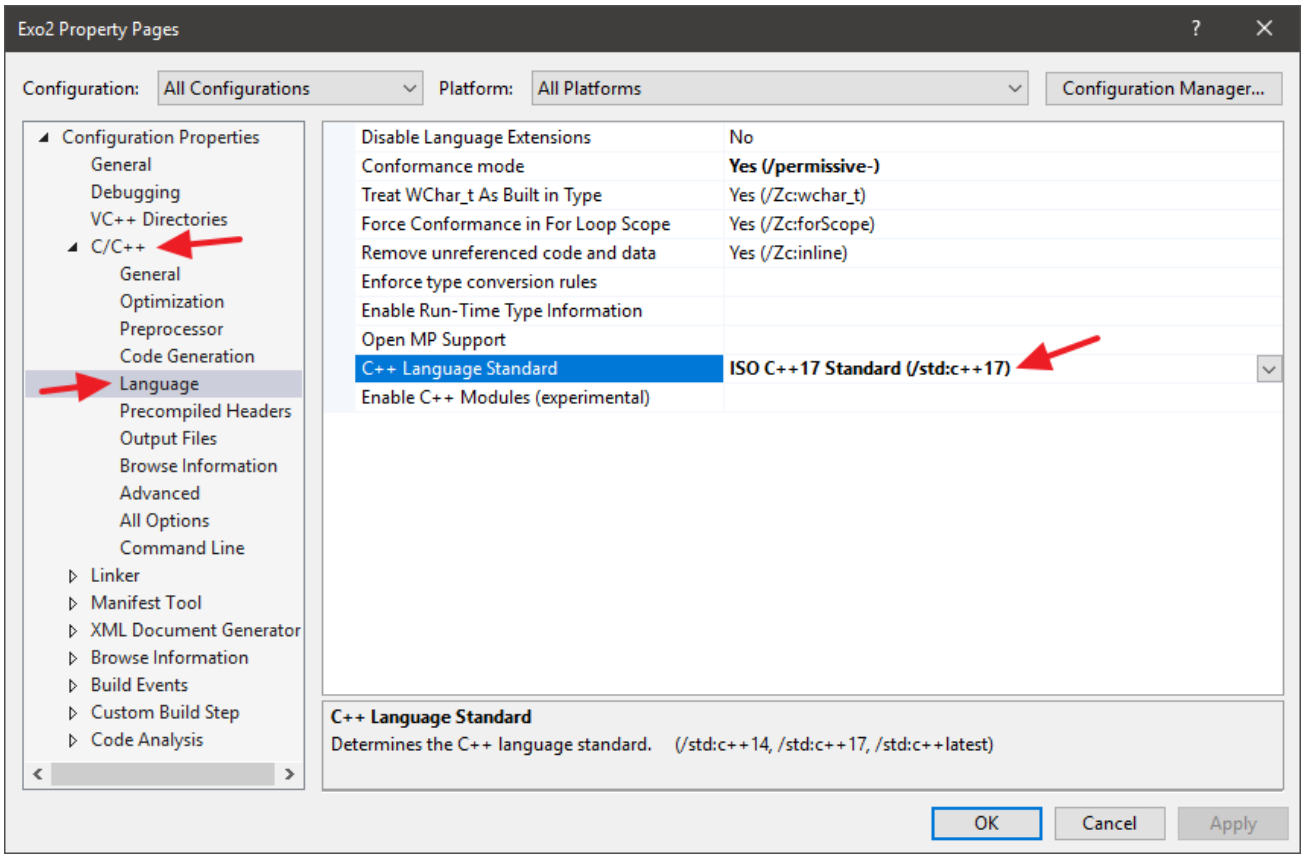

Figure 1 Menu pour changer la norme du langage

Utilisation des *span* de GSL (et de C++20)

Lorsqu'on veut passer des tableaux en paramètre à une fonction, il faut aussi spécifier la taille du tableau en paramètre, sinon la fonction ne peut pas savoir la taille du tableau. On a aussi vu en cours qu'on peut passer le tableau encapsulé dans une `struct`. Prenons l'exemple d'une fonction qui affiche les éléments d'un tableau d'entier :

```
void afficher_v1 ( const int tableau[], int nbElements ) {
    for (int i : range(nbElements))
        cout << tableau[i] << " ";
}

struct TableauInt {
    const int* elements;
    int        nbElements;
};
void afficher_v2 ( TableauInt tableau ) {
    for ( int i : range(tableau.nbElements) )
        cout << tableau.elements[i] << " ";
}

// Appels :
const int N_VALS = 5;
int foo[N_VALS] = {10, 20, 30, 40, 50};
afficher_v1(foo, N_VALS);
afficher_v2({foo, N_VALS});
TableauInt bar = {foo, N_VALS};
afficher_v2(bar);
```

Il y a plusieurs problèmes avec ces façons de faire : il faut répéter la taille du tableau et il faut créer une struct pour chaque type de valeurs de tableau.

Avec la classe `span` de la librairie GSL (qui fera partie du standard en 2020), vous pouvez passer un tableau (ou une partie d'un tableau) encapsulé dans un type qui vous permet entre autres d'utiliser les boucles sur intervalle. Reprenons l'exemple du tableau `foo` précédent :

```
void afficher_v3 ( span<const int> tableau ) {
    cout << "Taille : " << tableau.size() << endl; // .size() comme pour string.
    for ( int elem : tableau ) // On peut itérer comme avec les strings et les tableaux.
        cout << elem << " ";
}

afficher_v3(foo);                // Conversion en span implicite.
afficher_v3({foo, N_VALS});      // Peut construire span comme la struct.
span<const int> s = foo;         // Construction d'une variable de type span.
afficher_v3(s.subspan(1, 2));    // Prend une partie du span : .subspan(index_début, nb_elements)
afficher_v1(s.data(), s.size()); // Pour utiliser l'ancienne fonction.
```

Syntaxe dite *template* : le type des éléments entre `< >`. Si on veut modifier les éléments, on fait un `span<Type>`, sinon on fait un `span<const Type>`. On peut aussi créer des span de tableaux, mais avec les mêmes contraintes que les tableaux 2D réguliers (seulement première dimension variable). `span<int[42]>` donne l'équivalent de `int[][42]`.

Enfin, pour inclure la librairie, faites la même chose qu'avec *cppitertools* (mettre le dossier *gsl* dans votre dossier de projet), puis faites le `#include` dans votre code :

```
#include "gsl/span"
using namespace gsl;
```