Référence/type[1]	LOO C++ - TD3		☒ TD ☐ AP
Thématique principale	Gestion des exceptions (standard) en C++		
Thématique(s) secondaire(s)	Poursuite de l'exploration de la STL		
Durée	2h	***Niveau[2]***	A
Prérequis	✓ LOO C++ CM et TDs précédents		
Compétences opérationnelles et niveau cible[3]	✓ Lever et intercepter des exceptions standard (***M***) ✓ Découvrir et mettre en œuvre quelques conteneurs de la STL (***A***) ✓ Appliquer les bonnes pratiques du C++ (***N***)		
Modalités et critères d'évaluation	Auto-évaluation		
Matériel(s), équipement(s), composant(s) nécessaires			
Logiciel(s) nécessaire(s)[4]			
Ressources			

Avant la séance…

Compléter, si ce n'a pas déjà été fait, la classe « De » du TD précédent pour y ajouter la levée d'une exception au cas où la création d'un dé à 1 face serait tentée.

Mettre en place l'architecture minimale du travail à réaliser lors de la séance (projet, main de test, fichiers de la classe…).

Le cœur du travail consiste à mettre en place (partiellement) une classe destinée à générer automatiquement du code de configuration de registre pour MCU. L'exemple traité part du principe que le MCU est un PIC24 (16 bits). Un objet cette classe doit permettre de stocker des « paires » de type Nom Registre : Valeur.

- Rechercher parmi les conteneurs de la STL C++ le conteneur qui semble le plus adapté. La séance débutera par une discussion sur le sujet.

Spoiler alert : la suite du sujet donne une réponse à cette question, il vaut mieux ne pas aller plus loin dans la lecture de ce document avant d'avoir essayé de répondre à la question…

Travail encadré en séance

Apports de cours

- Quelques éléments complémentaires sur la STL, si besoin.
- Retour sur la notion de template (pour la STL), là encore si besoin.

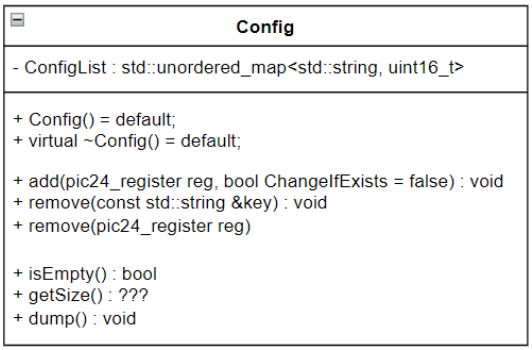
[1] TD : ½ groupe, apports de cours, mise en pratique guidée – TP : ½ groupe, travail autonome majoritaire.
[2] Le niveau se rapporte à la thématique principale. Il peut ici être entendu comme un indicateur de la difficulté du travail (N-Facile, A-Sans grande difficulté, M-Quelques points complexes, MA-Difficile)
[3] Les niveaux cible correspondent pour chaque compétence opérationnelle au niveau d'acquisition de la celle-ci à l'issue du travail dans son intégralité (en incluant les phases préparatoires et de synthèse).
[4] En plus d'un environnement de développement C++

Activités

La classe « Config » - Présentation & commentaires

Le diagramme de classe partiel de la classe « Config » est fourni ci-contre.

Ce diagramme est partiel car il ne comporte que les éléments clés du travail du jour. Il sera complété si le travail va plus loin que le minimum exigé.



La configuration du MCU est contenue dans un conteneur de la STL de type *unordered_map*.

Les éléments de cette map sont de type *std::pair<std ::string, uint16_t>*.

- Justifer ces choix de conteneurs / types.
- Indiquer le moyen de « créer » un nouveau type appelé pic24_register qui serait un simple alias vers la paire.

La règle du zéro s'applique concernant les constructeurs, le destructeur est lui-même celui par défaut.

Plusieurs méthodes permettent de gérer la base de configuration (map) :

- ✓ ***add*** : ajoute, ou tente d'ajouter, une valeur (pic24_register) dans la base de configuration (map). Cette méthode prend comme paramètres le pic24_register à ajouter et un booléen dont la valeur par défaut est false. Ce boolée définit le comportement si le registre existe déjà dans la base. Si le registre existe et que ChangeIfExists est vrai, la valeur est modifiée. Si le booléen vaut faux, une exception de type *std::invalid_argument("Register already exists.")* est levée.
- ✓ ***remove*** : supprime une valeur de la base de configuration. Cette méthode reçoit une référence constante sur une chaîne de caractères. Cette chaîne correspond au nom du registre à supprimer. Si aucun registre de ce nom n'existe, une exception de type *std::invalid_argument("Register doesn't exist.")* est levée.
  La méthode remove est surchargée de manière à ce que cette opération soit possible aussi en passant directement un pic24_register.
- ✓ ***isEmpty*** : retourne vrai si la base de configuration est vide, faux dans le cas contraire.
- ✓ ***getSize*** : retourne le nombre d'éléments (paires) présents dans la base de configuration. *Il sera intéressant de se poser la question du type de la valeur de retour, ainsi que des différentes manières de l'exposer.*
- ✓ ***dump*** : Affiche le contenu de la base de configuration sous la forme : string = Valeur. Une ligne par élément, et la valeur est affichée sous forme fixe de 4 digits hexadécimaux avec préfixe (exemple : ***LATB : 0x5A0F***).

Travail à réaliser

- Mettre en place rapidement l'architecture du projet (2 fichiers pour la classe et 1 main de test).
- Déclarer la classe Config, mettre en place les premiers éléments (définition du type pic24_register et de la base de configuration notamment). Ne pas oublier de « defaulter » explicitement constructeur par défaut et destructeur.
- Analyser, coder et tester de manière « concurrente » les méthodes add et dump.
- Finaliser la validation de la méthode ***add*** (vérification de la levée d'exception…).
- Analyser, coder et tester les méthodes ***remove***.
- Finir le travail avec la définition et le test des méthodes ***isEmpty*** et ***getSize***.

Synthèse

Faire une synthèse rapide :

- ✓ Des exceptions standard spécifiques qui semblent les plus récurrentes et les cadre dans lesquels elles seraient adaptées.
- ✓ Sur les conteneurs de la STL croisés/rencontrés au cours de ce TD ainsi que la situation justifiant leur utilisation.

Après la séance…

Continuer à renseigner votre document « pense-bête C++ » avec les informations glanées au cours de la séance.

Annexe 1 – Les exceptions standard

Ref : https://en.cppreference.com/w/cpp/error/exception