TP AP4A

Gabriel FROSSARD - Jean-Charles CREPUT

Automne 2025

1 Présentation du projet

1.1 Introduction

Le TP de AP4A va prendre la forme d'un mini-projet, que vous allez devoir réaliser en solitaire, qui va vous permettre de mettre en pratique les connaissances acquises lors du cours ; il sera également un aperçu du projet Java, ainsi que ces attentes, que vous devrez rendre en fin de semestre.

Vous allez disposer de trois séances de trois heures chacune pour réaliser votre projet et étant donné la nature de l'UE, la difficulté de ce dernier sera croissante.

1.2 Mise en contexte

Votre projet a pour but la réalisation d'un simulateur, permettant de modéliser et simuler un système de badges et lecteurs de badges pour le contrôle d'accès sécurisé à des batiments et salles de travail sur, par exemple, un campus universitaire. Le système est modélisé par un programme orienté objet C++ et les communications entre objets simulées par appel de méthodes conformément à la métaphore objet.

Ce système sera composé d'un serveur communiquant avec divers types de lecteurs de badges, associés à des portes d'entrée, répartis au sein du campus, tels que des lecteurs de badges pour les portes d'entrée des batiments, et des lecteurs de badges pour l'accès à certaines salles de travail. Une demande d'accès à un bâtiment ou salle est effectué par l'appariement avec le lecteur de badge, d'un badge nominatif associé à une personne, étudiant, enseignant, personnel sécurité, administratif, etc.. Les droits d'accès et priorités diffèrent selon le type de personnel et/ou le badge dont ils disposent. Ainsi, un enseignant peut disposer des accès de type sécurité, ou encore des accès spécifiques à son laboratoire s'il est également chercheur.

Il s'agira de modéliser et simuler l'activité du système par exemple au cours d'une journée type. Un ordonnanceur permettra d'animer les différents objets en interactions simulant les demandes d'accès sur la journée. De plus, les informations associées aux entrées pourront être stockées sous forme de fichiers de logs ou encore partagées dans une console afin de les visualiser. Seront reportés les autorisations d'accès accordées avec le détail des informations utiles telles que l'heure d'accès, le nom de la personne, son statut, ainsi que les tentatives d'accès infructueuses dans deux fichiers de log respectivement.

2 Les outils de développement

Vous utiliserez le standard C++17, qui est largement supporté et facile à compiler à la fois sur Linux et sur Windows. Concernant la compilation, vous utiliserez le compilateur GNU pour le langage C++, "g++". Sur Windows, il s'appelle MinGW.

Après avoir compilé un premier programme de type "hello world" par ligne de commande à la console, vous créerez un programme plus élaboré, avec un main.cpp, et un squelette succinct des classes principales de l'application (voir ci-dessous) implémenté dans les classes Scheduler, Server, LecteurBadge, Badge, pour la mise en place et le test de vos outils de développement, compilation et exécution. L'utilisation des PC personnels est à privilégier sur lesquels vous pourrez installer vos IDE favoris. Eventuellement des PC portables sont fournis pour la séance de TP

Parmi les IDE que vous pourrez utiliser, on trouve par exemple qtcreator (par défaut sur Linux), visual studio code, devC++, clion, visual studio, etc. Cependant, pour des raisons de rendu du travail et donc de compilation et exécution multi-plateforme de vos programmes, à la fois sur Linux, Windows ou MacOS, il sera demandé d'utiliser un utilitaire de gestion de projet multi-plateforme, soit l'outil cmake associé à la plupart des IDE, dont le fichier de description de projet est CMakeLists.txt, soit l'outil qmake associé à l'IDE qtCreator, dont le fichier de description de projet est <projet>.pro. Le principe consiste en la génération automatique de Makefile pour la plateforme cible à partir d'une description du projet. Des exemples d'utilisation et de configuration seront présentés en TP. À vous de choisir l'IDE le plus simple et commode à utiliser permettant un transfert aisé des sources et une reconfiguration rapide en cas de changement de système. Dans tous les cas de figure, **un fichier CMakeLists.txt devra être intégré au rendu final, ou sinon un fichier <projet>.pro**.

3 Première séance

Lors de cette première séance, vous allez commencer à mettre en pratique les fondamentaux de la programmation objet de par son architecture, son écriture ainsi que sa compilation.

3.1 Votre premier programme

Pour votre premier programme, vous allez rédiger un grand classique de la programmation, le bien connu "Hello World !". Une fois ce dernier écrit, vous devrez le compiler, puis l'exécuter.

Concernant la compilation, vous utiliserez le compilateur GNU pour le langage C++, "g++". Vous devrez savoir nommer votre fichier de sortie, séparer la phase de compilation de la phase d'édition de liens, avec les options de compilation qui conviennent, puis vous devrez afficher les warnings. Pour savoir comment rédiger la commande g++, vous utiliserez la documentation Linux présente dans la console, le "man".

Une fois le travail effectué, vous le ferez constater par un professeur. Pour le reste de cette UE, vous pouvez faire appel aux IDE précédemment évoqués pour la compilation.

3.2 Le simulateur

Nous allons maintenant nous pencher sur le développement de notre simulateur. Il conviendra de procéder par prototypage, selon l'approche agile, en raffinant les structures de classes au fur et à mesure de l'avancement. Il conviendra de bien prévoir l'organisation de la gestion des droits d'accès selon les différents types de priorité ou d'autorisation d'accès afin d'en fournir une descrition générique au serveur. Dans une première approche, notre simulateur est composé d'un serveur (classe Serveur), d'un ensemble de lecteurs de badges associés à des portes d'entrée (classe LecteurBadge), ainsi que de badge associés à des personnes (classe Badge). Il comporte aussi une classe pour l'ordonnancement des exécutions (classe Scheduler).

Votre première tâche consistera à créer un squelette succinct de programme avec ces différentes classes, un main, et à simuler une demande d'accès au serveur par appariement d'un badge avec un lecteur de badge. Par exemple la classe Serveur reçoit une demande d'accès et permet de l'analyser, de la formater, de stocker les informations utiles dans des fichiers de logs ou encore de les visualiser dans la console, puis de renvoyer l'autorisation avec ouverture de la porte d'entrée. Pour cela, vous allez commencer par implémenter la classe avec ses attributs, méthodes et diverses fonctions vues comme des services, pour la gestion centralisée des informations reçues des lecteurs de badges. On prendra des conventions standards pour les noms des fichiers, classes et variables, et la mise en forme du code.

Travail en autonomie.

À la fin de cette première séance, vous aurez normalement fini d'implémenter votre classe Serveur et un squelette de programme. Cependant, pour que vous ne restiez pas sans pratiquer jusqu'à la prochaine séance, il vous est conseillé d'organiser votre projet de manière structuré afin d'assurer une compréhension claire de son architecture. Cela inclut la gestion des fichiers et la mise en place de la structure de base des autres classes, comme Scheduler, Badge, LecteurBadge, etc... Il sera particulièrement important de bien réfléchir à l'organisation du système de description des autorisations d'accès, par groupes ou types de personnes suivant leurs fonctions, et groupes de portes associés aux différents batiments et salles de travail. Pour les différentes classes créées vous ajouterez des opérateurs friend d'entrée/sortie << en commençant par la classe Serveur. Deux modes d'utilisation devront être testés vous permettant de rediriger le flux de sortie, soit vers la console, soit vers un fichier de log.

4 Deuxième séance

Lors de cette deuxième séance, vous allez mettre en place l'architecture de votre projet. Vous pourrez commencer par implémenter le système global sans référence aux mécanismes d'héritage dans un premier temps, notamment si ceux-ci n'ont pas été encore abordés en cours et en TD. Vous considerez dans ce cas les classes Serveur, Scheduler, Badge, LecteurBadge en leur attribuant un fonctionnement par défaut. Etant donné l'association des badges aux différents types de personnels, il conviendra de préciser l'organisation de vos données. Faudra–t-il créer une classe d'objets représentant les types de personnels ? la réponse est laissée au libre choix du concepteur. Les mécanismes d'héritage et de polymorphisme seront introduits dans un deuxième temps, donnant toute sa généricité au système complet.

Vous commencerez votre séance par une introduction à l'architecture logicielle et une mise en pratique, afin de commencer à prévoir le fonctionnement global et le programme

final. Il vous sera ensuite proposé un schéma UML que vous aurez à compléter, de manière individuelle. Cela vous permettra de mieux visualiser l'architecture du projet demandé. Vous allez maintenant affiner vos structures de données et classes en tenant compte des recommendations ci-dessous.

4.1 Scheduler

L'ordonnanceur, implémenté dans la classe Scheduler, gère l'animation des processus, leur donne une activation suivant une base de temps prédéfinie. Dans cette classe, vous allez définir une fonction *simulation* qui organise le séquencement de l'exécution des demandes d'accès. La fonction *simulation* doit permettre aux personnes d'initier une demande d'accès via leur badge sur un lecteur de badge. Ainsi, une demande d'accès est initiée par une personne représentée par son badge. Il est laissé au concepteur le choix d'organiser la séquence d'appel de méthodes appropriée lors de la boucle d'ordonnancement.

4.2 Serveur

Le code de cette classe doit être complété en conséquence des choix d'organisation. Notez que le serveur est considéré jusqu'à présent suivant un mode de fonctionnement passif : ses services sont invoqués de l'extérieur par les badges/lecteur de badges. Le serveur devra gérer des demandes d'accès et donc fournir le service approprié, méthode demandeAcces(Badge, LecteurBadge). Il devra également récupérer toutes les informations utiles pour gérer la demande. Pour cela on considère que le serveur "connaît" les types d'objets appropriés (lecteur, badge, personne) et peut ainsi communiquer avec eux via appel de méthodes. Il s'agira d'utiliser la surcharge et le polymorphisme de manière appropriée à ce niveau. Également, la configuration des droits d'accès est spécifiée au niveau du serveur, il sera opportun de fournir un mécanisme générique de description des associations de droits d'accès entre personnels et lecteurs de badges, bâtiments et salles de travail, via un fichier de description chargé au démarrage.

4.3 LecteurBadge

La classe "LecteurBadge" représente une classe mère pour les différents types de lecteurs de badge associés à des portes de batiments ou portes de salles de travail. Les lecteurs de badge peuvent être organisés selon une hiérarchie d'heritage selon leurs spécificités.

4.4 Badge

Les badges sont associés à des personnes qui initient des demandes d'accès dont on va simuler l'exécution via l'utilisation de la classe Scheduler (ordonnanceur). La classe Badge permet de retrouver les informations utiles pour une demande d'accès, telles que la personne concernée, son statut, ses droits d'accès. Elle pourra faire l'objet d'une hiérarchie d'héritage selon le type d'accès autorisé. Dans notre approche, la question se pose de définir les responsabilités de cette classe concernant la simulation des demandes d'accès. Comment répartir les choix de simulation des demandes d'accès (heure, fréquences, portes) entre les différentes classes du système est une question ouverte.

5 Troisième séance

On implémente tous les mécanismes d'héritage et de polymorphisme pas encore implémentés ou définis. Les classes de base deviennent des classes abstraites, et/ou des interfaces. Les

mécanismes de généricité sont poussés à leur maximum et les dernières fonctionnalités manquantes du projet sont réalisées. Les concepteurs qui sont le plus avancés peuvent imaginer des fonctions d'analyse des demandes d'accès, exécutées en autonomie par le serveur qui au besoin, devient aussi un processus actif. Une liberté de choix est laissée au programmeur pour faire la différence avec le minimum requis.

6 Synthèse

Voici la liste des logiciels que vous avez pu mettre en place pour votre projet :

- g++
- man
- make pour faire des makefile (compilation automatisée)
- gdb (débeuger)
- valgrind (détection fuite mémoire)
- éditeur de code basique : VIM, EMACS, Gedit, atom, nano, imax, sublimeText
- IDE : qtcreator, visual studio code, clion, devC++.
- éditeur d'architecture logicielle (UML): le seul et unique "ASTAH"

Parmi les notions orientées objet du C++ que vous avez mis en oeuvre on aura, sans être exhaustif :

- classes avec constructeurs, destructeur, liste d'initialisation, forme de Coplien
- surcharge de méthode et d'opérateur
- utilisation de l'héritage et du polymorphisme, redéfinition de méthode
- concept d'interface, méthodes virtuelles
- conversions ascendantes, voir dans certains cas conversion descendante
- variable statique pour la gestion des identifiants uniques
- utilisation de la STL (conteneurs, itérateurs, algorithmes)
- utilisation des bibliothèques standards pour la gestion des dates, temporisations, nombres aléatoires

La notation lors de l'évaluation du projet tiendra compte des critères suivants :

- réalisation des objectifs à chaque TP et rendu final dans les temps
- lisibilité et structuration standard du code, compilation multi-plateforme
- implémentation correcte des principaux mécanismes du C++ (voir ci-dessus)
- compilation et exécution sans erreur
- gestion correcte de la mémoire
- niveau de sophistication de l'implémentation

7 Rendu de projet

Un rapport de TP et le code sont à rendre individuellement pour le **7 novembre 2025** avant minuit. Vous déposez le rapport et le code dans la section "devoir" dédiée à ce TP sur **Moodle**. Seuls les codes en C++ sont acceptés et les rapports en format PDF. Ne pas oublier le fichier CMakeLists.txt ou le fichier <projet>.pro pour la compilation multi-plateforme. Donc, vous avez deux fichiers à télécharger :

1. **Rapport_AP4A_Groupe_Nom_Prenom.pdf** : Rapport + le diagramme UML.

2. **Code_AP4A_Groupe_Nom_Prenom.zip** : Code en c++.

Concernant le nom de groupe, vous prenez le nom de votre groupe TP auquel vous ajoutez la lettre A ou B selon la semaine où vous êtes inscrit. Il varie selon votre créneau de TP AP4A:

1. TP1A: Mercredi semaine A
2. TP1B: Mercredi semaine B
3. TP2A: Lundi semaine A
4. TP2B: Lundi semaine B
5. TP3A: Vendredi semaine A
6. TP3B: Vendredi semaine B