ECE
PARIS ÉCOLE D'INGÉNIEURS

INITIATION AUX CONCEPTS

OBJETS EN C++

Cahier d'exercices

2020-2021

MAJ le 30/09/2020

DIEDLER Florent
Enseignant / Ingénieur
Créateur de la société CODEANALYSIS

Table des exercices

TP 1 – Analyse et conception objet

Nécessite la maîtrise du cours :

CHAPITRE 1 – CONCEPTS, DIAGRAMME DE CLASSES ET VOCABULAIRE

A. Bref historique
B. Vocabulaire et concepts objets
C. Diagrammes de classe
D. Relations entre classes

Vous n'êtes pas encore ingénieur ni même informaticien mais vous savez réfléchir.

Etre ingénieur, c'est quoi ? Apporter une solution à un besoin.

Comprendre le besoin : nécessite la compréhension du haut niveau
Élaborer la solution : nécessite la maîtrise du bas niveau.

Entre les 2, les phases **d'analyse**, de **conception** et de **test**.
But de la formation : acquérir ces notions.

Commençons donc par réfléchir en français (avant de coder en C++) !!

Un client vous exprime ses attentes. Il est rarement clair, rarement complet…

1. Identifiez ses attentes, faites préciser les points flous, posez des hypothèses…
2. Déterminez les ressources qui vous seront nécessaires (charge mémoire)
3. Déterminez les traitements (actions) à exécuter et leur ordre d'exécution (charge processeur)

Rappelez-vous, ce qui nous intéresse ici c'est l'analyse d'un CDC et non le code !

A VOUS DE JOUER…

Exercice 100 Etude d'un cahier des charges (CDC)

Voici un exemple de cahier des charges proposé par un directeur d'établissement scolaire (cf site www.cairn.info) qui joue ici le rôle de client.

Identifier les éléments suivants dans le texte ci-bas : classes, objets, méthodes, et attributs

Tableau 1 - *Descriptif du déroulement des activités*

Descriptif du déroulement des activités tel qu'il apparaît dans le cahier des charges

Un cours commence par une séance présentielle à l'occasion de laquelle le professeur présente, outre le cours proprement dit, les travaux qui seront à réaliser durant la période qui sépare ce cours du suivant, appelée « période intercours ». Cette période est mise à profit par l'étudiant pour revoir les notions abordées durant le cours précédent et préparer la séance suivante. Les travaux qu'il doit préparer sont remis et évalués à partir de la plate-forme internet.

A l'occasion du cours suivant, le professeur réalise un bilan des difficultés rencontrées par les étudiants au vu des travaux qu'il a eu l'occasion de recevoir durant la période intercours. Il met à profit le début du cours pour répondre aux questions que les étudiants se sont posées au cours de la période d'étude en autonomie avant d'entamer le deuxième cours.

Les cours et périodes de travail à distance se succèdent à intervalle régulier jusqu'à la dernière séance consacrée habituellement à la préparation de l'examen.

Par ailleurs, l'évaluation des travaux intermédiaires réalisés tout au long du cours permettra au professeur d'adapter son cours présentiel à son public. Les difficultés rencontrées par les étudiants dans leurs travaux indiqueront au professeur les points de la matière sur lesquels il est important de revenir.

.

Exercice 101 Etude d'un jeu vidéo simplifié

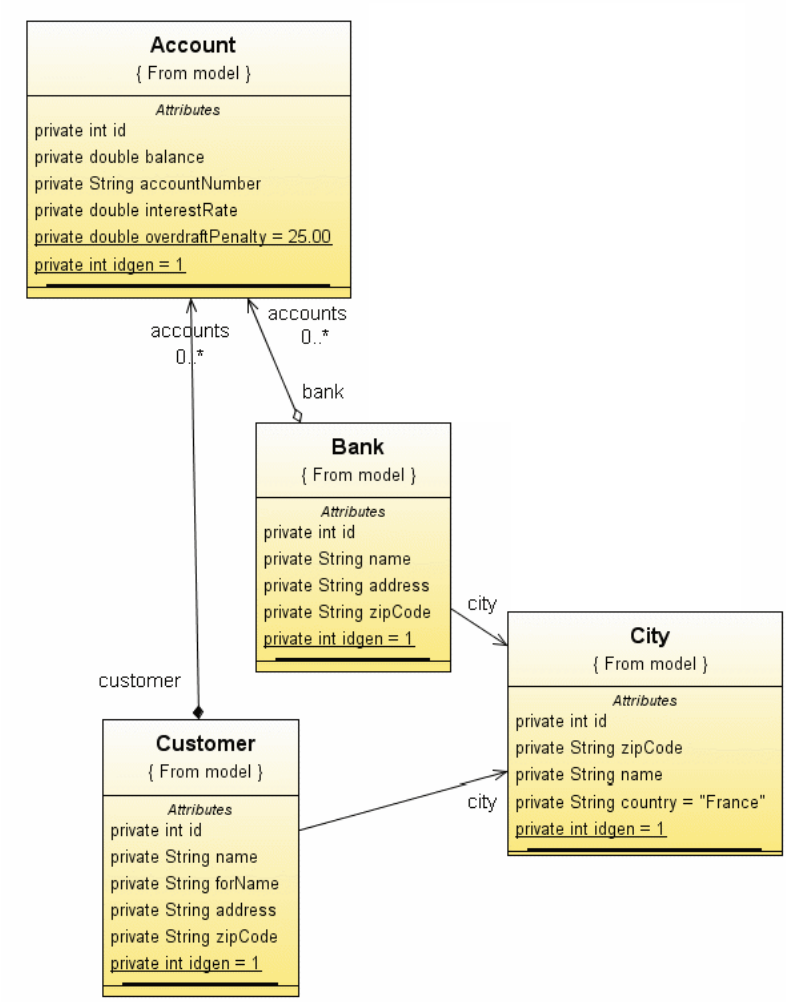
Soit le cahier des charges suivant pour la conception d'un mini RPG :

« Vlad (le héros) évolue dans un monde nommé Weyard où règne l'alchimie : une puissante magie capable d'anéantir le monde. Mais cette puissance a été libérée par les méchants Salamandar et Phoenixia. Vlad part donc à l'aventure avec son ami Garet pour récupérer les 4 étoiles élémentaires renfermant cette puissance. Chaque personnage a des pouvoirs spéciaux nommés psynergie. Au cours du jeu, on pourra rencontrer 3 types d'ennemies : des rats, des squelettes et des chauves-souris ☺. Weyard est composé de 3 villes nommées Val, Vault et Imil. Dans chaque ville, on trouve une auberge et un magasin de potions… »

1) Identifier les classes et les relations entre ces classes.
2) Faites un diagramme de classe simplifié montrant les classes et leurs relations

Exercice 102 Etude d'un diagramme de classe

Soit le diagramme de classe suivant :



1) Identifier les classes
2) Identifier les cardinalités et toutes les relations entre ces classes et les expliciter en français
3) Que modélise ce diagramme de classe ?

Exercice 103 Etude d'un texte

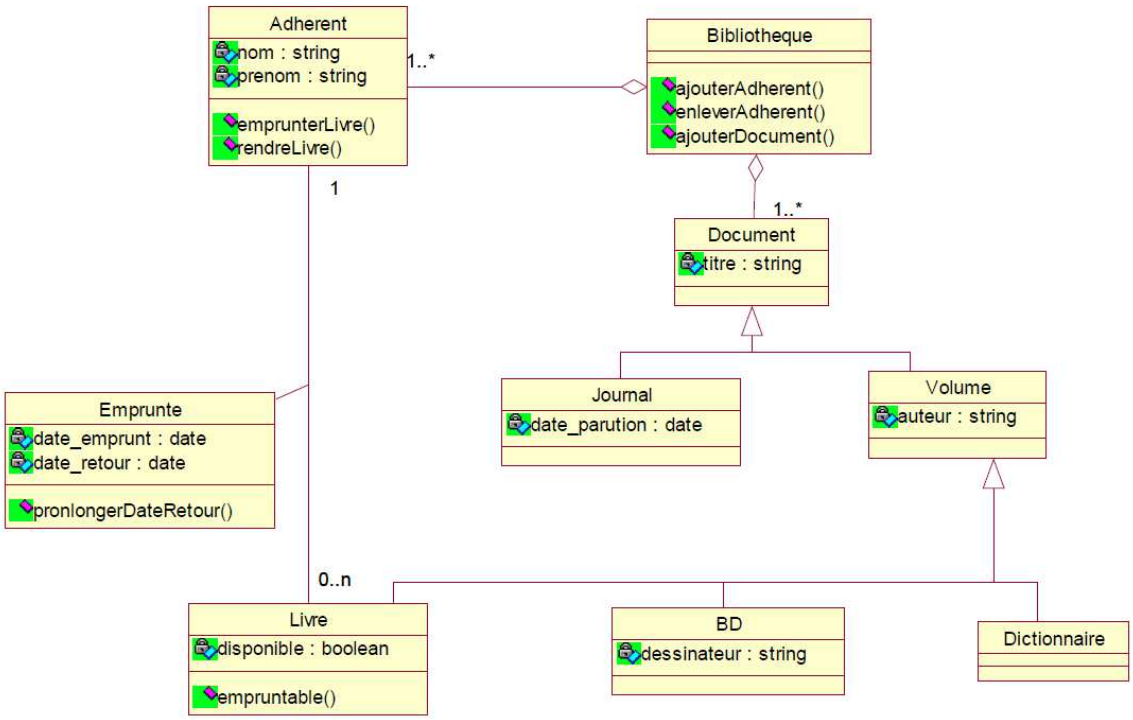
Un hôtel est composé d'au moins deux chambres. Chaque chambre dispose d'une salle d'eau : douche ou baignoire. Un hôtel héberge des personnes. Il peut employer du personnel et il est impérativement dirigé par un directeur. On ne connaît que le nom et le prénom des employés, des directeurs et des occupants. Certaines personnes sont des enfants et d'autres des adultes (faire travailler des enfants est interdit).

Un hôtel a les caractéristiques suivantes : une adresse, un nombre de pièces et une catégorie. Une chambre est caractérisée par le nombre et de lits qu'elle contient, son prix et son numéro. On veut pouvoir savoir qui occupe quelle chambre à quelle date. Pour chaque jour de l'année, on veut pouvoir calculer le loyer de chaque chambre en fonction de son prix et de son occupation (le loyer est nul si la chambre est inoccupée). La somme de ces loyers permet de calculer le chiffre d'affaires de l'hôtel entre deux dates.

Donner une proposition de diagramme de classe modélisant cet énoncé.

Exercice 104 Etude d'un diagramme de classe

Soit le diagramme de classe suivant :



1) Lister les différentes entités / classes
2) Expliciter en français toutes les relations entre ces entités
3) Que peut-on dire de la classe « *Emprunte* » ?
4) Que modélise ce diagramme de classe ?

TP 2 – Introduction au langage C++

Nécessite la maîtrise du cours :

CHAPITRE 2 – PRINCIPALES DIFFERENCES ENTRE LE C ET C++

A. Les flux d'entrée et sortie
B. Les types scalaires
C. La déclaration des variables
D. La gestion de la mémoire
E. Les paramètres par défaut
F. La surcharge de fonctions
G. Les fonctions « inline »
H. Les chaines de caractères

Attention !

Pour chaque exercice, vous DEVEZ commenter vos programmes.
Chaque sous-programme doit être commenté de la façon suivante :

```
/* scan_dir : permet de récupérer les sous répertoires dans un répertoire donné en excluant certains répertoires.
ENTREE :
        dir = le répertoire à scanner
        tabexlu = un tableau contenant le nom des répertoires à exclure du scan
SORTIE :
        Un tableau contenant le nom des répertoires
*/
String* scan_dir(string dir, string* tabexclu);
```

Les prototypes des sous-programmes doivent être mis dans les fichiers « headers » et les implémentations dans les fichiers sources (.cpp)

Vous devez coder entièrement en C++ ! Il n'est pas autorisé d'utiliser des fonctions de la librairie standard du C.
Exemple de fonctions interdites : malloc, free, strlen, strcpy, strcat, printf, scanf, puts, gets, fgets, fopen, fclose… Exceptions : srand() et rand() sont autorisées en C++

Exercice 200 Entrées/sorties

Ecrivez un programme qui demande la saisie d'un nombre entier et d'un nombre réel à l'utilisateur. Ces deux nombres seront ensuite affichés à l'écran.

Exercice 201 Utilisation du booléen / blindage

Ecrivez un programme qui autorise l'accès à un utilisateur dont le login/mot de passe est correct. Le programme demande donc à l'utilisateur de saisir son login puis son mot de passe. Si les informations de connexion sont correctes, le programme afficher « accès autorisé » sinon il affichera « accès refusé » et redemandera la saisie des informations.

L'utilisation du booléen est obligatoire !

Exercice 202 Chaine de caractères (String)

Ecrivez un programme proposant à l'utilisateur les choix suivants :

1. Saisie d'une chaine de caractères
2. Calcul de la taille de la chaine de caractères
3. Compter le nombre de voyelles et consonnes de la chaine saisie
4. Supprimer un caractère précis. Exemple : si la chaine saisie est « bonjour » et que je souhaite supprimer le caractère « o », la chaine devient « bnjur »
5. Concaténation avec une autre chaine de caractères. Exemple : si j'ai les chaines « hello » et « world », la chaine finale sera « hello world » ***(attention à séparer avec un espace s'il n'est pas déjà présent !)***
6. Décalage avec un pas de 1 de chaque caractère de la chaine. Le « a » devient « b » et le « b » devient « c » …

Il est obligatoire d'utiliser l'objet String et ses méthodes pour la manipulation des chaines ! Chaque sous-programme devra se trouver dans un fichier « fonctions.cpp » et les prototypes dans un fichier header « fonctions.h »

Exercice 203 Boucles et allocations dynamiques

Ecrivez un programme qui demande la saisie de 10 nombres réels. Stocker ces nombres dans un tableau alloué dynamiquement (ne pas utiliser le « vector » ni les autres conteneurs). Calculer ensuite la moyenne des nombres et ajouter cette moyenne au tableau (dans la dernière case). Le programme affichera le contenu du tableau.

Attention aux fuites mémoires… Rappel : « malloc » et « free » pas autorisés…

Exercice 204 Tri dans un tableau monodimensionnel / aléatoire

Ecrivez un programme qui génère aléatoirement 20 entiers dans l'intervalle [-10, 10]. Stocker ces valeurs dans un tableau dynamique (ne pas utiliser le « vector » ni les autres conteneurs) puis faites un tri croissant de ce tableau avec la méthode de votre choix (tri à bulles, tri par fusion, quicksort…). Afficher le tableau trié pour vérifier votre algorithme.

Votre tri devra se faire obligatoirement dans un sous-programme. Il est évidemment interdit d'utiliser les fonctions de tri offertes par les bibliothèques C++

Exercice 205 Surcharge de fonction et paramètres par défaut

Ecrivez deux sous-programmes « *findMin* » permettant de trouver le minimum de deux ou trois nombres. Le premier sous-programme travaillera sur des entiers et le second sur des réels quelconques. *(Evidemment les deux fonctions doivent avoir le même nom « findMin » !)*

Exercice 206 Récursivité et suite mathématiques

Ecrivez un programme qui demande à l'utilisateur de taper un entier N et qui calcule u(N) défini par : ***u(n+1) = u(n) + u(n-1)*** avec ***u(0) = 1*** **et u(1) = 1**.

Exercice 207 Structures, aléatoire, allocation dynamique, tableaux 2D

Ecrivez un programme permettant d'effectuer le produit matriciel d'un vecteur V avec une matrice M de taille n x m. Les valeurs seront générées aléatoirement et les dimensions de la matrice et du vecteur seront saisies par l'utilisateur.

Vous devez utiliser une structure *Matrice* et une structure *Vecteur*.
Si le produit matriciel n'est pas possible, vous afficherez un message d'erreur et vous redemanderez une nouvelle saisie valide.

Exercice 208 Entrées / sorties formatées

Ecrivez un programme qui demande la saisie d'un réel X et effectue les actions suivantes :

- affiche le carré et la racine carrée de X
- affiche le périmètre et l'aire du cercle de rayon X
- affiche la longueur de la diagonale d'un carré de côté X

Vos résultats devront être précis à 2 chiffres après la virgule. En cas d'impossibilité d'effectuer une opération, vous afficherez un message d'erreur.

Exercice 209 Tableau multi dimensionnels

Ecrivez un programme qui demande à l'utilisateur de saisir le contenu d'un tableau de réels de N lignes et M colonnes et qui effectue les actions suivantes :

- affiche la matrice saisie
- calcul la moyenne de cette matrice
- calcul le nombre d'occurrence d'un nombre donné
- affiche uniquement les nombres pairs de cette matrice. Les nombres impairs seront masqué par le symbole 'X'

TP 3 – Classes et objets

Nécessite la maîtrise du cours :

CHAPITRE 3 – CLASSES ET OBJETS

A. **Vocabulaire (rappels)**
B. **Définition et structure d'une classe**
C. **Constructeurs**
D. **Destructeurs**
E. **Accesseurs**
F. **Méthodes**
G. **Pointeur *this***

Attention !

Pour chaque exercice, vous DEVEZ commenter vos programmes.
Chaque sous-programme doit être commenté de la façon suivante :

```
/* scan_dir : permet de récupérer les sous répertoires dans un répertoire donné en excluant certains répertoires.
ENTREE :
        dir = le répertoire à scanner
        tabexlu = un tableau contenant le nom des répertoires à exclure du scan
SORTIE :
        Un tableau contenant le nom des répertoires
*/
String* scan_dir(string dir, string* tabexclu);
```

Les prototypes des sous-programmes doivent être mis dans les fichiers « headers » et les implémentations dans les fichiers sources (.cpp)

Vous devez coder entièrement en C++ ! Il n'est pas autorisé d'utiliser des fonctions de la librairie standard du C.
Exemple de fonctions interdites : malloc, free, strlen, strcpy, strcat, printf, scanf, puts, gets, fgets, fopen, fclose… Exceptions : srand() et rand() sont autorisées en C++

Pour chaque classe, on DOIT séparer l'interface (fichier header) de l'implémentation (fichier source).
Exemple : Soit une classe « Foo », on doit donc avoir un fichier « foo.h » qui contient l'interface (déclaration de la classe, constantes, …) et un fichier « foo.cpp » pour l'implémentation (le code)

Exercice 300 Création d'une classe simple

Ecrivez une classe *Rectangle* qui comporte les informations suivantes :

1. Deux attributs caractérisant le rectangle (n'oubliez pas de les préfixer de « m_ »)
2. Un constructeur par défaut
3. Un constructeur surchargé
4. Un destructeur
5. Deux méthodes pour calculer le périmètre et l'aire du rectangle
6. Une méthode pour l'affichage des informations sur le rectangle (hauteur, largeur, aire, périmètre)
7. Les accesseurs nécessaires pour respecter le principe d'encapsulation

Faites un « main » qui utilise la classe *Rectangle*.

Exercice 301 Création d'un compteur

Ecrivez une classe *Compteur* qui comporte les méthodes suivantes :

1. un constructeur par défaut
2. un constructeur surchargé
3. une méthode d'incrémentation du compteur
4. une méthode de décrémentation du compteur
5. un accesseur pour récupérer la valeur du compteur

Faites un « main » qui utilise la classe *Compteur* en créant un compteur puis on incrémente 10 fois et on le décrémente 20 fois puis on affiche sa valeur.

Exercice 302 Création d'une classe *Complexe*

Ecrivez une classe *Complexe* définie par une partie réelle et une partie imaginaire qui comporte les méthodes suivantes :

1. un constructeur par défaut
2. un constructeur surchargé
3. une méthode pour affiche ce nombre complexe sous forme cartésienne et polaire
4. une méthode qui calcule le module
5. une méthode qui calcule le conjugué
6. une méthode qui calcule la somme de deux nombres complexes
7. une méthode qui calcule le produit de deux nombres complexes
8. une méthode qui calcul le produit d'un nombre complexe avec un scalaire
9. les accesseurs nécessaires

Faites un « main » qui utilise la classe *Complexe*.

Exercice 303 Classes et composition

Ecrivez une classe *Point* permettant de gérer un point dans le plan euclidien (à vous de trouver les attributs et méthodes…). Ecrivez ensuite une classe *Ligne* en respectant le fait qu'une ligne est composée de deux points. La classe *Ligne* aura une méthode « affiche » qui affichera les coordonnées des deux points.

Faites un « main » qui utilise les classes *Ligne* et *Point*.

Exercice 304 Classes et agrégation

Ecrivez une classe *Personne* permettant de décrire une personne selon la phrase suivante : *« M. Dupond Pierre est né en 1980 et est célibataire »* (on utilisera un booléen pour savoir si une personne est célibataire ou non)

Ajouter maintenant la possibilité à une personne d'avoir un conjoint. Réfléchissez !

Faites une méthode permettant de marier une personne à une autre (peu importe le sexe des deux personnes… ☺) puis une autre méthode permettant de divorcer…

Faites un « main » respectant les consignes suivantes :

- Deux personnes p1 et p2 se marient.
- Finalement, p2 divorce.
- Puis p2 se marie à nouveau avec une nouvelle personne p3.

Implémenter ces instructions en C++.

TP 4 – Conteneurs de la STL (vector, stack, and queue)

Nécessite la maîtrise du cours :

CHAPITRE 3 – CLASSES ET OBJETS

A. Vocabulaire (rappels)
B. Définition et structure d'une classe
C. Constructeurs
D. Destructeurs
E. Accesseurs
F. Méthodes
G. Pointeur *this*

CHAPITRE 4 – CONTENEURS

A. Classes « templates » et itérateurs
B. Vecteurs
C. Piles
D. Files
E. Autres conteneurs

Attention !

Pour chaque exercice, vous DEVEZ commenter vos programmes.
Chaque sous-programme doit être commenté de la façon suivante :

```
/* scan_dir : permet de récupérer les sous répertoires dans un répertoire donné en excluant certains répertoires.
ENTREE :
        dir = le répertoire à scanner
        tabexlu = un tableau contenant le nom des répertoires à exclure du scan
SORTIE :
        Un tableau contenant le nom des répertoires
*/
String* scan_dir(string dir, string* tabexclu);
```

Les prototypes des sous-programmes doivent être mis dans les fichiers « headers » et les implémentations dans les fichiers sources (.cpp)

Vous devez coder entièrement en C++ ! Il n'est pas autorisé d'utiliser des fonctions de la librairie standard du C.
Exemple de fonctions interdites : malloc, free, strlen, strcpy, strcat, printf, scanf, puts, gets, fgets, fopen, fclose… Exceptions : srand() et rand() sont autorisées en C++

Pour chaque classe, on DOIT séparer l'interface (fichier header) de l'implémentation (fichier source).
Exemple : Soit une classe « Foo », on doit donc avoir un fichier « foo.h » qui contient l'interface (déclaration de la classe, constantes, …) et un fichier « foo.cpp » pour l'implémentation (le code)

Exercice 400 Découverte du vecteur ☺

Ecrivez un programme qui génère aléatoirement 20 entiers dans l'intervalle [-10, 10]. Stocker ces valeurs dans un vecteur puis faite un tri croissant de ce vecteur. Afficher le tableau trié pour vérifier votre algorithme.

N'oubliez pas de faire les bons « includes » notamment pour pouvoir utiliser les vecteurs.

Vous voyez que les vecteurs sont une alternative très puissante aux tableaux classiques du C.

Exercice 401 Vecteurs, classes et pointeurs

Ecrivez un programme permettant de manipuler une collection d'étudiants. Commencez par créer une classe *Etudiant*. Un étudiant sera caractérisé par un identifiant, un nom et une année de promotion. Faites une fonction qui affichera les différentes informations de l'étudiant.

Faites un « main » qui gère une collection de pointeurs sur étudiants. N'oubliez pas de libérer correctement la mémoire allouée.

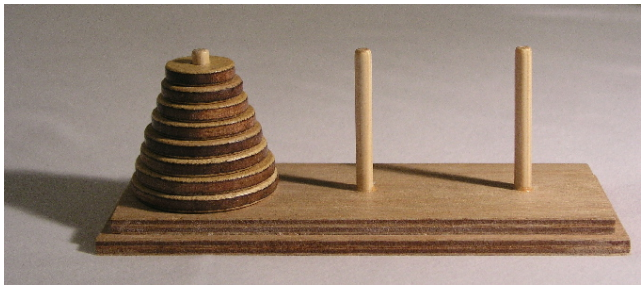
On souhaite maintenant inscrire un étudiant dans une école. Implémenter les classes et méthodes nécessaires pour cela et tester votre implémentation dans le « main ».

Exercice 402 Utilisation d'une pile – Tours de Hanoï

Les tours de Hanoï *sont un jeu de réflexion consistant à déplacer des disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire », et ceci en un minimum de coups, tout en respectant les règles suivantes :*

- *on ne peut déplacer plus d'un disque à la fois ;*
- *on ne peut placer un disque que sur un autre disque plus grand que lui ou sur un emplacement vide.*

On suppose que cette dernière règle est également respectée dans la configuration de départ.(source : wikipedia)



Ecrivez un programme permettant de résoudre ce problème pour un nombre quelconque de disques. On pourra tester l'algorithme avec n = 3 dans un premier temps.

Exercice 403 Utilisation d'une file – File d'impression de documents

On souhaite créer un programme permettant de gérer une file d'attente d'impression de documents. Par simplification, un document sera caractérisé par un nom et un identifiant.

Ecrivez un programme permettant les actions suivantes :

- Ajouter un document
- Supprimer un document par son identifiant
- Supprimer un document par son nom
- Visualiser l'état de la file d'impression
- Vider la file d'impression

Attention, il est interdit d'ajouter deux fois le même document. Deux documents sont identitiques si et seulement si ils ont le même identifiant et le même nom. Vous afficherez un message d'erreur si on tente d'ajouter un document déjà existant.

Exercice 404 Utilisation d'un conteneur associatif

Ecrivez un programme permettant de gérer une mini base de données de login et de mot de passe. Chaque login est associé à un seul mot de passe. Le programme devra être en mesure de retrouver le mot de passe à partir d'un login et vice-versa.
Vous entrerez des logins et mot de passe « bidons » pour créer votre mini base de données.

Exercice 405 Utilisation d'un conteneur associatif

Ecrivez un programme permettant de gérer un terrain de jeu de 200 cases. Chaque case peut contenir 0 ou plusieurs personnages caractérisés par un nom. Il n'est pas autorisé d'utiliser le vecteur dans cet exercice. Vous devez utiliser un tableau associatif de votre choix.
Le programme doit être capable d'afficher le nombre de personnage sur une case donnée.

Exercice 406 Fichier et Table de hachage

Ecrivez un programme qui charge le fichier de mots de la langue française « words.txt ». Stocker tous les mots dans une table de hachage. Ecrivez ensuite une fonction permettant de chercher un mot dans la table de hachage.

On utilisera la fonction de hash des « String » par défaut offerte par le conteneur « unordered_set »

Pour rappel, une table de hachage est un tableau qui associe une clé à une valeur.
La *clé* est calculée à partir d'une fonction de hachage (basé sur un calcul du code ASCII pour les « string » par exemple). La *valeur* est la donnée que vous souhaitez stocker.

Aides : Pour déclarer une table de hachage qui utilise la fonction de hash par défaut :
std::unordered_set<std::string> hashTable;

Pour lire un fichier, on utilise la classe « std :: ifstream » dans la bibliothèque <fstream>.

Exercice 407 Fonction de hachage et performance (challenge facultatif)

Télécharger le code « exo 407 » et compléter la fonction de hash qui se trouve au début du fichier « main.cpp ». Ne pas toucher au reste du code !

Le but de l'exercice est d'écrire la fonction de hash **la plus performante possible**. Cela permettra de charger les quelques 650 000 mots du dictionnaire de manière rapide. De même une fonction de recherche nommée « testSpeed » a été écrite afin de tester la rapidité en recherche de la table de hachage.

Pour rappel, une fonction de hachage calcule l'empreinte d'un objet et retourne donc un entier non signé (*size_t* équivalent à un *unsigned long*). Vous pouvez faire ce que vous voulez pour calculer cet entier…

Le meilleur étudiant (celui dont la fonction de hachage sera la plus performante) aura un bonus… Les fonctions de hachage seront testées sur une même machine afin de respecter l'équité ☺

Exercice 408 Snake et listes chainées

Le but de cet exercice est d'implémenter un snake en utilisant les classes et à l'aide du conteneur liste. Voici quelques étapes pour vous aider données à titre indicatif :

- Créer une classe *Pomme* et des méthodes permettant de « manger » une pomme et d'afficher une pomme sur le terrain de jeu
- Créer une classe *Serpent* permettant et des méthodes pour gérer la taille dynamique du serpent qui grossit d'une case à chaque fois qu'il mange une pomme
- Créer une classe *Jeu* qui gère le terrain de jeu, les scores, le serpent et les pommes

Avant de vous lancer dans la programmation tête baissée, faite un diagramme de classe et expliciter en français les relations entre les classes *Pomme*, *Serpent* et *Jeu*.

Commencer ensuite par implémenter les petites classes (la classe *Pomme* par exemple). Vérifier que vos méthodes fonctionnent bien à l'aide de tests simples.

TP 5 – Héritage et polymorphisme

Nécessite la maîtrise du cours :

CHAPITRE 3 – CLASSES ET OBJETS

A. **Vocabulaire (rappels)**
B. **Définition et structure d'une classe**
C. **Constructeurs**
D. **Destructeurs**
E. **Accesseurs**
F. **Méthodes**
G. **Pointeur *this***

CHAPITRE 4 – CONTENEURS

A. **Classes « templates » et itérateurs**
B. **Vecteurs**
C. **Piles**
D. **Files**
E. **Autres conteneurs**

CHAPITRE 5 – HERITAGE ET POLYMORPHISME

A. **Concepts et vocabulaire**
B. **Héritage simple – Implémentation**
C. **Héritage multiple – Implémentation**
D. **Polymorphisme**
E. **Classe abstraite**

Attention !

Pour chaque exercice, vous DEVEZ commenter vos programmes.
Chaque sous-programme doit être commenté de la façon suivante :

```
/* scan_dir : permet de récupérer les sous répertoires dans un répertoire donné en excluant certains répertoires.
ENTREE :
        dir = le répertoire à scanner
        tabexlu = un tableau contenant le nom des répertoires à exclure du scan
SORTIE :
        Un tableau contenant le nom des répertoires
*/
String* scan_dir(string dir, string* tabexclu);
```

Les prototypes des sous-programmes doivent être mis dans les fichiers « headers » et les implémentations dans les fichiers sources (.cpp)

Vous devez coder entièrement en C++ ! Il n'est pas autorisé d'utiliser des fonctions de la librairie standard du C.
Exemple de fonctions interdites : malloc, free, strlen, strcpy, strcat, printf, scanf, puts, gets, fgets, fopen, fclose… Exceptions : srand() et rand() sont autorisées en C++

Exercice 500 Découverte de l'héritage

Ecrivez un programme qui manipule des formes géométriques 2D (rectangle, triangle et losange). Un carré n'est qu'un cas particulier du rectangle… A vous d'implémenter les méthodes de votre choix. Vous pouvez aussi représenter les formes graphiquement avec Allegro ☺

Faites un « main » qui manipule toutes ces formes.

Exercice 501 Héritage simple

Considérons les classes *Vehicule*, *Moto*, *Camion*, *Voiture*. Tous ces véhicules sont caractérisés par un numéro de plaque d'immatriculation, une vitesse maximale de 130 km/h, une couleur et une marque. Les motos et camions peuvent transporter une maximum de 2 personnes alors qu'une voiture peut en transporter 4. Les camions peuvent transporter une cargaison et leur vitesse est limitée à 70 km/h lorsque qu'ils sont chargés (sinon il roule à 90km/h).

Faites un programme permettant de faire les instructions suivantes dans le « main » :

- Créer 1 camion, 1 voiture et 1 moto
- Ajouter un passager dans le camion
- Ajouter un autre passager dans le camion
- Ajouter 3 passagers dans la voiture
- Ajouter 1 passager sur la moto
- Faire rouler les 3 véhicules à la vitesse de 150 km/h
- Afficher les vitesses pour chacun des véhicules et la cargaison du camion
- Faire rouler les 3 véhicules à la vitesse de 90 km/h
- Afficher les vitesses pour chacun des véhicules et la cargaison du camion
- Ajouter une cargaison dans le camion
- Afficher à nouveau les vitesses
- Décharger la cargaison du camion
- Afficher le contenu de la cargaison du camion

Essayer d'ajouter une cargaison à la voiture ou à la moto. Que se passe t-il et pourquoi ?

Exercice 502 Héritage multiple

Vous souhaitez programmer la gestion des ennemies dans un jeu vidéo.
Considérons des ennemis de différents types : terrestre, volant et aquatique. Le jeu vidéo comporte 5 ennemis ayant les caractéristiques suivantes :

- triceratops : ne peut se déplacer que sur le sol
- ichtyosaure : ne peut se déplacer que dans l'eau
- ptérodactyle : ne peut se déplacer qu'en volant
- velociraptor : peut se déplacer sur le sol et en volant
- plésiosaure : peut se déplacer sur le sol et dans l'eau

Chaque ennemi peut se déplacer, attaquer et se défendre. On modélisera ces actions par de simples affichages (std ::cout).

Faites un diagramme de classes qui modélise la gestion de tous les ennemis.

Implémenter ensuite votre solution en C++

Exercice 503 Héritage et polymorphisme

Considérons un RPG caractérisé par les personnages suivants : magicien, guerrier et elfe. Tous les personnages ont 100 points de vie au départ. Les caractéristiques des personnages sont les suivantes :

- Un magicien attaque et se protège en lançant de la magie. Il a donc un compteur de magie. L'attaque lui coute 10 points de magie et la défense 5 points. Au départ, le magicien possède 50 points de magie et il ne peut pas regagner de magie en cours de combat. Son attaque a une puissance de 20 points de dégâts.
- Un guerrier attaque avec son épée mais ne peut pas se protéger (pas de bouclier). Son attaque a une puissance de 15 points de dégâts.
- Un elfe attaque en utilisant un arc et donc des flèches. Il possède 20 flèches au départ. Il ne peut pas se recharger en flèches. Son attaque a une puissance de 10 points de dégâts.

On ne peut rencontrer qu'un seul ennemi qui a 100 points de vie, qui inflige 13 points de vie par attaque et ne peut pas se défendre. A chaque tour, l'ennemie ne fait donc qu'attaquer. C'est toujours le joueur qui commence le combat.

1) Implémenter en C++ la gestion des personnages.
2) Implémenter en C++ la gestion de l'ennemi
3) Implémenter en C++ un combat au tour par tour entre un personnage choisi au hasard (parmi magicien, guerrier ou elfe) et l'ennemi sachant que les entités ne peuvent qu'attaquer.

Maintenant, le joueur possède une équipe de deux personnages (un magicien et un guerrier par exemple ou alors un magicien et un elfe…) qu'il choisit au départ. Quand un personnage meurt, il ne peut plus faire d'actions. L'ennemi possède 200 points de vie dans ce scénario.

4) Implémenter en C++ un mini-système de combat en console (basé sur du std ::cout) au tour par tour. Le combat oppose alors deux personnages contre l'ennemi. A chaque tour, le joueur peut choisir entre « attaquer » ou « defendre ». L'ennemi (contrôlé par l'IA) choisi aléatoirement les actions (attaquer ou défendre) et choisi aléatoirement le personnage attaqué.

Extensions possibles : Pour ceux qui sont à l'aise et veulent aller plus loin, vous pouvez ajouter une couche graphique avec Allegro ou une autre librairie afin de rentre votre jeu plus conviviale. Sinon, vous pouvez ajouter des nouveaux sorts ou nouveaux personnages (amis ou ennemis) ainsi que des spécificités intrinsèques à ces nouvelles entités.

TP 6 – Templates

Nécessite la maîtrise du cours :

CHAPITRE 3 – CLASSES ET OBJETS

A. **Vocabulaire (rappels)**
B. **Définition et structure d'une classe**
C. **Constructeurs**
D. **Destructeurs**
E. **Accesseurs**
F. **Méthodes**
G. **Pointeur *this***

CHAPITRE 4 – CONTENEURS

A. **Classes « templates » et itérateurs**
B. **Vecteurs**
C. **Piles**
D. **Files**
E. **Autres conteneurs**

CHAPITRE 6 – TEMPLATES

A. **Template de fonctions**
B. **Template de classes**
C. **Spécialisation**

Attention !

Pour chaque exercice, vous DEVEZ commenter vos programmes.
Chaque sous-programme doit être commenté de la façon suivante :

```
/* scan_dir : permet de récupérer les sous répertoires dans un répertoire donné en excluant certains répertoires.
ENTREE :
        dir = le répertoire à scanner
        tabexlu = un tableau contenant le nom des répertoires à exclure du scan
SORTIE :
        Un tableau contenant le nom des répertoires
*/
String* scan_dir(string dir, string* tabexclu);
```

Les prototypes des sous-programmes doivent être mis dans les fichiers « headers » et les implémentations dans les fichiers sources (.cpp)

Vous devez coder entièrement en C++ ! Il n'est pas autorisé d'utiliser des fonctions de la librairie standard du C.
Exemple de fonctions interdites : malloc, free, strlen, strcpy, strcat, printf, scanf, puts, gets, fgets, fopen, fclose… Exceptions : srand() et rand() sont autorisées en C++

Exercice 600 Premier pas avec les templates

Ecrivez une fonction qui calcule la moyenne de deux entiers. Faites une surcharge de cette fonction pour des réels. On souhaite maintenant faire une fonction qui calcule la moyenne de deux caractères (basé sur leur code ASCII). Que feriez-vous ? Proposez une solution élégante pour répondre à ce problème.

Faites un « main » qui appelle ces fonctions pour en vérifier le bon fonctionnement.

Exercice 601 Template et tableaux

On souhaite réaliser une fonction de tri (choisissez le tri que vous voulez mais ne pas utiliser *std::sort* dans cet exercice !) par ordre décroissant pour des entiers, des réels et des caractères (basé sur le code ASCII). Comment faire ?

Faites un « main » qui teste votre fonction de tri avec un tableau d'entiers puis un tableau de réel et enfin un tableau de caractères.

On souhaite maintenant utiliser notre fonction de tri pour trier des chaines de caractères selon l'ordre lexicographique. Que doit-on faire ? Vérifier votre solution dans le « main » ☺

Exercice 602 Template et spécialisation

Créer une fonction template qui calcule la somme des éléments d'un tableau. Spécialiser cette fonction de manière à ce qu'elle fonctionne pour des chaines de caractères (*std::string*). Dans ce cas, la fonction calculera la somme des longueurs de chaque chaine.

Faites un « main » pour vérifier votre solution

TP 7 – Exceptions

Nécessite la maîtrise du cours :

CHAPITRE 3 – CLASSES ET OBJETS

- **H. Vocabulaire (rappels)**
- **I. Définition et structure d'une classe**
- **J. Constructeurs**
- **K. Destructeurs**
- **L. Accesseurs**
- **M. Méthodes**
- **N. Pointeur *this***

CHAPITRE 4 – CONTENEURS

- **F. Classes « templates » et itérateurs**
- **G. Vecteurs**
- **H. Piles**
- **I. Files**
- **J. Autres conteneurs**

CHAPITRE 6 – TEMPLATES

- **D. Template de fonctions**
- **E. Template de classes**
- **F. Spécialisation**

Attention !

Pour chaque exercice, vous DEVEZ commenter vos programmes.
Chaque sous-programme doit être commenté de la façon suivante :

```
/* scan_dir : permet de récupérer les sous répertoires dans un répertoire donné en excluant certains répertoires.
ENTREE :
        dir = le répertoire à scanner
        tabexlu = un tableau contenant le nom des répertoires à exclure du scan
SORTIE :
        Un tableau contenant le nom des répertoires
*/
String* scan_dir(string dir, string* tabexclu);
```

Les prototypes des sous-programmes doivent être mis dans les fichiers « headers » et les implémentations dans les fichiers sources (.cpp)

Vous devez coder entièrement en C++ ! Il n'est pas autorisé d'utiliser des fonctions de la librairie standard du C.
Exemple de fonctions interdites : malloc, free, strlen, strcpy, strcat, printf, scanf, puts, gets, fgets, fopen, fclose… Exceptions : srand() et rand() sont autorisées en C++

Exercice 700 Premier pas avec les exceptions

Ecrivez une fonction qui calcule la moyenne d'une série de nombre entiers positifs passés en paramètre. Lever une exception si l'un des nombres est négatif.

Exercice 701 Exception personnalisée

On souhaite créer une exception personnalisée permettant d'écrire dans un fichier une erreur. Pour cela, créer une classe MyException qui hérite de la classe std ::exception. Notez que vous aurez à redéfinir la méthode what() (regardez sur internet son prototype). A vous de trouver les méthodes que vous aurez besoin pour stocker les erreurs dans un fichier texte.

Tester votre exception avec un programme simple.

Exercice 702 Exception et classes

Créer une classe Point permettant de créer des points dont les coordonnées sont comprises entre [0.0, 1.0]. Vous devez empêcher la création de Points ne respectant pas cette règle. Pour cela, lever une exception de votre choix dans le constructeur de la classe Point.

Tester votre classe dans le main.

NB : On ne lève jamais une exception dans le destructeur !

Exercice 703 Exception, classes et pointeurs

On souhaite créer une classe *FileManager* dont le rôle est de récupérer les informations d'un fichier texte. Pour cet exercice, on stockera le contenu du fichier dans un char* (ne pas utiliser std ::string).

1) Créer le constructeur surchargé prenant en paramètre un chemin d'accès vers un fichier
2) Que faire si le fichier n'existe pas ? Et si son contenu est vide ?
3) Créer le destructeur libérant correctement la mémoire allouée
4) Créer une méthode permettant de récupérer l'extension du fichier (sans le « . »)
5) Créer une méthode permettant de récupérer le contenu du fichier
6) Créer une méthode permettant de récupérer le nom du fichier sans son chemin.

Faire un main permettant de vérifier votre classe.

Extras – Pour aller plus loin

Nécessite la maîtrise du cours :

CHAPITRE 1 A 6 – TOUS LES CHAPITRES !!

Extra 1 Arbre binaire de recherche

Considérons un arbre binaire de recherche (ABR) : *« En informatique, un **arbre binaire de recherche** (**ABR**) est un arbre binaire dans lequel chaque nœud possède une clé, telle que chaque nœud du sous-arbre gauche ait une clé inférieure ou égale à celle du nœud considéré, et que chaque nœud du sous-arbre droit possède une clé supérieure ou égale à celle-ci — selon la mise en œuvre de l'ABR, on pourra interdire ou non des clés de valeur égale. Les nœuds que l'on ajoute deviennent des feuilles de l'arbre. » (Source Wikipedia : https://fr.wikipedia.org/wiki/Arbre_binaire_de_recherche).*

Dans notre cas, on va interdire les clés doublons pour simplifier.

1) Créer une classe permettant de gérer un ABR
   a. Trouver les attributs nécessaires
   b. Créer un ou plusieurs constructeurs
   c. Créer le destructeur
2) Créer une méthode qui calcule le nombre de nœuds présent dans l'ABR.
3) Créer une méthode permettant d'afficher les valeurs des nœuds de l'ABR par ordre décroissant.
4) Créer une fonction de comparaison permettant de comparer deux arbres binaires (faite une surcharge de l'opérateur « == »)
5) Créer une méthode permettant d'ajouter un nœud dans l'ABR en respectant les propriétés de l'ABR !
6) Créer une méthode permettant de trouver un nœud particulier en fonction de sa valeur. Si le nœud n'est pas présent dans l'ABR, la fonction devra renvoyer « NULL ».
7) Créer une méthode permettant de vérifier que l'objet passé en paramètre est bien un ABR.
8) Créer une méthode permettant de supprimer un nœud tout en respectant les propriétés de l'ABR.

Extra 2 « Puissance 5 » - De la conception au produit fini !

Créer un jeu de puissance 5 entre deux joueurs humains. Les règles sont les mêmes que le jeu de puissance 4 sauf qu'il faut aligner (verticalement, horizontalement ou en diagonal) 5 pions de la même couleur pour gagner. La taille du tableau peut être extensible pour plus de plaisir ☺

1) Modéliser le jeu de « puissance 5 » avec un diagramme de classe. Il est très important de ne pas se lancer dans le code sans avoir une idée des classes nécessaires et surtout de leurs relations !
2) Programmer les règles du jeu de base en blindant bien les saisies utilisateur
3) Tester votre jeu avec deux joueurs humains
4) Pour aller plus loin, proposer un mode 1 joueur VS l'ordinateur. Vous pouvez implémenter une IA très intelligente en se basant sur les algorithmes de type « minimax » ou alors une IA très basique ne jouant qu'aléatoirement… Il est aussi possible de créer une IA intermédiaire.

Vous êtes libre d'utiliser la librairie graphique de votre choix : Allegro, SDL ou SFML.