Sujet 6	
Buts :	Héritage simple, polymorphisme, méthodes virtuelles, listes chaînées, attributs de classe, méthodes de classe.

Organisation d'un voyage (Evaluation juin 2007)

Dans ce problème on s'intéressera à l'organisation d'une `Escapade` touristique. Une `Escapade` étant composée de plusieurs étapes, la solution proposée devra permettre une mise à jour facile de l'escapade : ajout ou suppression d'une étape. En ajoutant une étape à une `Escapade` on devra augmenter le moins possible la longueur totale du voyage. L'utilisation des listes chaînées s'avère donc justifiée pour ce problème. Par ailleurs une `Escapade` devra pouvoir contenir plusieurs catégories d'étapes que l'on pourra assimiler à des `Lieux`. Dans un premier temps on se limitera à deux catégories spécifiques : les `Monuments` et les `Villes`. Chacun des `Lieux` sera par ailleurs caractérisé par sa `Position`.

Afin de pouvoir proposer une solution à ce problème, on se propose de travailler en deux temps :
- modélisation des classes `Position`, `Lieu`, `Monument` et `Ville` ;
- modéliser les classes `Noeud` et `Escapade`.

Attention : dans le cadre du polymorphisme certaines fonctions peuvent devenir virtuelles.

Le fichier *makefile*

On développera les 6 classes citées ci-dessus en appliquant le principe de la compilation séparée (fichiers **.h* et **.cc* pour chaque classe développée) en prenant compte des contraintes suivantes :
- la classe `Lieu` contient un champ de type `Position`,
- la classe `Monument` hérite de la classe `Lieu`,
- la classe `Ville` hérite de la classe `Lieu`,
- la classe `Noeud` contient un pointeur sur un `Lieu`,
- la classe `Escapade` contient un pointeur sur un `Noeud`,
- le programme principal se trouve dans le fichier *principal.cc*,

Ecrivez le fichier *makefile* correspondant à ce sujet.

Autour des étapes d'un voyage

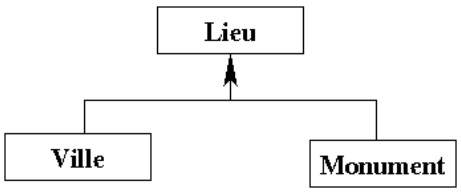


Figure 1: Diagramme de classes : Lieu, Ville, Monument

L'objectif de cette partie est de modéliser la gestion d'un ensemble de `Lieux` (`Ville` ou `Monument`). Ces `Lieux` composent naturellement les étapes d'une `Escapade`. On se propose

de travailler en respectant l'arbre d'héritage (page 1).

La classe `Lieu` est une classe abstraite dont dérivent les classes `Ville` et `Monument`. Elle est caractérisée par ses coordonnées (`Position`), ainsi que son nom. La classe `Monument` contient en plus sa description, son année de construction et le temps nécessaire pour le visiter. La classe `Ville` est caractérisée en plus par son nombre d'habitants, sa surface, son nombre de curiosités à visiter et le temps moyen de la visite d'une curiosité (champ commun à toutes les `Ville`s).

Les coordonnées d'un Lieu étant composées d'une latitude et d'une longitude on propose de gérer les problèmes liées à celles-ci dans une classe `Position`.

La classe Position

La `Postion` géographique d'un `Lieu` est connue à travers une latitude et à une longitude.

La latitude est une chaîne de caractères composée de 5 caractères dont :
- les 2 premiers correspondent aux degrés (nombre compris entre 0 et 90 compris),
- les 2 suivants correspondent aux minutes (nombre compris entre 0 et 59 compris),
- le dernier caractère permet de spécifier l'orientation : N ou S pour nord ou sud.

La longitude est une chaîne de caractères composée de 6 caractères dont :
- les 3 premiers correspondent aux degrés (nombre compris entre 0 et 180 compris),
- les 2 suivants correspondent aux minutes (nombre compris entre 0 et 59 compris),
- le dernier caractère permet de spécifier l'orientation : E ou W pour est ou ouest.

Ainsi on se propose de définir une classe `Position` comme suit :

`Position`
`string latitude` `string longitude`

Exemple : Les coordonnées de Paris sont : (4851N,00220E)

Important ! Les fichiers `Position.h` et `Position.cc` sont disponnibles sur l'intranet à l'adresse :
`https://iut-info.univ-reims.fr/users/romaniuk/restricted/S3M3106C1BasesDuCpp/TP6/`

La classe Lieu

Un `Lieu` est caractérisé par ses coordonnées (`Position`) ainsi que son nom. On propose de définir la classe `Lieu` comme suit :

`Lieu`
`string nom` `Position coord`

Attention : la classe `Lieu` est polymorphe.

Important ! Les fichiers incomplets `Lieu.h` et `Lieu.cc` sont disponnibles à l'adresse : `https://iut-info.univ-reims.fr/users/romaniuk/restricted/S3M3106C1BasesDuCpp/TP6/`

As-t-on besoin d'un destructeur dans la classe `Lieu` ? Si oui donnez la déclaration et/ou la définition du destructeur.

Complétez la classe `Lieu` avec une méthode `affiche` utilisée dans l'opérateur `<<`.

Complétez la classe `Lieu` avec une méthode permettant de calculer la `distance` entre deux `Lieux`.

On souhaite s'assurer de l'existence d'une méthode `tempsVisite` dans les classes dérivées `Ville` et `Monument`. Cette méthode retournera le temps nécessaire pour visiter un `Lieu` sous forme d'un réel. Les informations nécessaires à la définition de cette méthode sont cependant absentes dans la classe `Lieu`. Complétez la classe `Lieu` avec la méthode `tempsVisite`.

Afin de pouvoir comparer des objets polymorphes de type `Lieu`, `Monument` ou `Ville` on souhaite définir un opérateur `==`. Cette opérateur prendra en paramètre un objet de type `Lieu*`. La classe `Lieu` étant polymorphe et abstraite cet opérateur sera déclaré virtuel pur dans cette classe. Complétez la classe `Lieu` avec la déclaration de l'opérateur `==`.

Peut-on instancier un objet de type `Lieu` ? Pourquoi ?

La classe Monument

La classe `Monument` a pour caractéristiques outre celles de la classe `Lieu` :
- la description,
- l'année de construction,
- le temps nécessaire pour la visite.

On propose de définir la classe `Monument` comme suit :

Monument
string description int annee float temps

On souhaite disposer d'un constructeur avec arguments pour la classe `Monument`. Ce constructeur prendra en paramètres l'ensemble des informations nécessaires à l'initialisation des champs de l'objet. Seule la `description` y sera initialisée par défaut à `""`.

La définition d'un opérateur `<<` pour un objet de type `Monument` implique la surcharge de la méthode `affiche` dans la classe `Monument`.

Définissez la classe `Monument`. Testez les fonctionnalités développées dans la classe `Monument`.

La classe Ville

La classe `Ville` a pour caractéristiques outre celles de la classe `Lieu` :
- le nombre d'habitants,
- la surface,
- le nombre de curiosités à visiter,
- le temps moyen de la visite d'une curiosité (champ commun à toutes les villes).

On propose de définir la classe `Ville` comme suit :

Ville

`unsigned int population`
`float surface`
`unsigned int nbCuriosites`
`static float visiteMoyenne`

On souhaite disposer d'un constructeur avec arguments pour la classe `Ville`. Ce constructeur prendra en paramètres l'ensemble des informations nécessaires à l'initialisation des champs de l'objet (si ceci est nécessaire).

Le champ `static visiteMoyenne` sera initialisé à 1,25. Un accesseur `getVisiteMoyenne` sera proposé.
La définition d'un opérateur `<<` pour un objet de type `Ville` implique la surcharge de la méthode `affiche` dans la classe `Ville`.

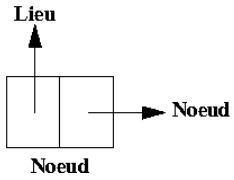
Définissez la classe `Ville`. Testez les fonctionnalités développées dans la classe `Ville`.

Organiser un voyage

L'objectif de cette partie est de modéliser la planification d'une `Escapade`. Une `Escapade` est caractérisée par une liste d'étapes : les `escales`. Chacune de ces `escales` est un `Lieu`.

La classe Noeud

La classe `Noeud` représente un élément de la liste chaînée.



Un `Noeud` contient un pointeur sur un `Lieu` (pour pouvoir bénéficier du polymorphisme) ainsi qu'un pointeur sur le `Noeud` suivant.

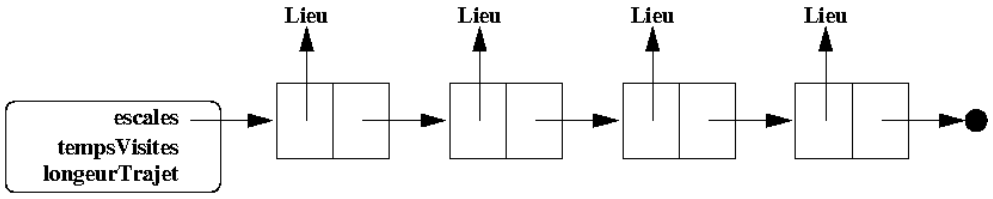
Noeud
`Lieu * etape` `Noeud * suivant`

Important ! Le fichier incomplet `Noeud.h` est disponnible à l'adresse :
`https://iut-info.univ-reims.fr/users/romaniuk/restricted/S3M3106C1BasesDuCpp/TP6/`

Définissez la classe `Noeud`. Complétez la d'une méthode `affiche` ainsi que de *méthodes* indispensables à son bon fonctionnement.

La classe Escapade

La classe `Escapade` contient l'ensemble des `escales`, ainsi que le temps total des visites et la longueur totale du trajet. Ces deux dernières informations devront être mises à jour à chaque insertion et suppression d'escale.



Une `Escapade` contient un pointeur sur un `Noeud` correspondant au premier `Lieu` du voyage.

Escapade
`Noeud * escales`
`float tempsVisites`
`float longueurTrajet`

Important ! Le fichier incomplet `Escapade.h` est disponnible à l'adresse :
`https://iut-info.univ-reims.fr/users/romaniuk/restricted/S3M3106C1BasesDuCpp/TP6/`

La méthode `appartient` permet de savoir si un `Lieu` fait partie de l'`Escapade`. Cette méthode retourne vrai si le `Lieu` est présent, faux sinon.

La méthode `enleve` permet d'enlever un `Lieu` dans la liste des `escales` connaissant sa position dans la liste.

Afin de faciliter l'écriture de la fonction d'insertion, on souhaite disposer de la méthode : `int meilleurePosition(const Lieu *, float &)const;`. Cette méthode permet de déterminer la meilleure position à laquelle une escale (`Lieu *`) peut être insérée dans une `Escapade` existante, sachant que cette position permet de minimiser la longueur du voyage. Si l'escale prise en paramètre est déjà présente dans l'`Escapade`, la position retournée sera -1 (aucune insertion ne sera alors effectuée). Cette méthode prendra en paramètre (par référence) la nouvelle longueur du voyage après insertion.

La méthode `insere` permet d'insérer de façon intelligente un `Lieu *` dans la liste d'`escales`. L'insertion sera effectuée de façon à minimiser la longueur (en kilomètres) du voyage.

Définissez la classe `Escapade`. Complétez la d'une méthode `affiche` ainsi que de *méthodes* indispensables à son bon fonctionnement. Testez les fonctionnalités développées dans la classe `Escapade`.