INF1010

Programmation Orientée-Objet

Travail pratique #5
Fonctions et classes génériques, librairies STL

Objectifs :	Permettre à l'étudiant de se familiariser avec le concept de fonctions et des classes générique ainsi qu'introduction à la librairie STL.
Remise du travail :	Mardi 4 avril 2017, 8h00
Références :	Notes de cours sur Moodle & Chapitres 11 à 14, 16 et 20 du livre Big C++ 2e éd.
Documents à remettre :	La solution ainsi que les fichiers .cpp et .h complétés réunis sous la forme d'une archive au format .zip.
Directives :	Directives de remise des Travaux pratiques sur Moodle **Les en-têtes (fichiers, fonctions) et les commentaires sont obligatoires.** Les travaux dirigés s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe. **Veuillez suivre le guide de codage**

Informations préalables

Le travail à faire sera divisé en deux parties. Dans la première, vous aurez à implémenter un conteneur de type Pile pouvant accueillir des types de données quelconques. Dans la deuxième, on vous demande d'implémenter un système de gestion de base de données de gènes.

Documentation : le site officiel de la STL se trouve à l'adresse : www.sgi.com/tech/stl

Veuillez porter une attention particulière aux points suivants :

- **Sauf mention explicite du contraire, c'est à vous de déterminer la visibilité de vos attributs (protected, private, public).**
- **L'un des principes du polymorphisme étant de limiter la duplication du code, pensez à utiliser au maximum les méthodes des classes mères.**
- **Lors de l'utilisation d'itérateurs, pensez à mettre à profit le mot clé *auto*!**
- **ATTENTION : Tout au long du TP, faites attention de toujours vérifier que vos itérateurs ne sont pas nuls et que vos conteneurs ne sont pas vides.**
- **ATTENTION : L'utilisation de boucles for ou while de la forme for(int i; i <vec.size(); i++) est interdit pour ce TP. Vous devez utiliser les algorithmes lorsque possible, ou les boucles for/while en utilisant les itérateurs.**
- **ATTENTION : Afin d'éviter de créer des fichiers .cpp et .h en grande quantité, tous les foncteurs de ce TP doivent être implémentés dans un même fichier nommé Foncteur.h.**

Partie 1 - Travail à réaliser

Dans cette première partie, vous allez devoir implémenter le fonctionnement d'une pile d'éléments de **type quelconques**.

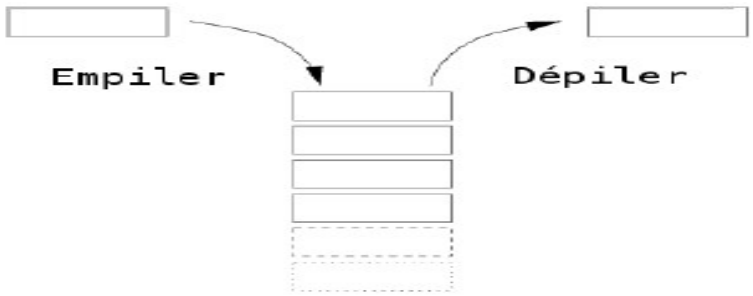


Fig1 : Schéma d'une pile

La pile peut être représentée comme une **structure de données générique** dans laquelle on peut insérer des éléments dans un ordre et les récupérer ensuite dans l'ordre du dernier entré, premier sorti (voir schéma ci-haut).

Dans votre implémentation, la pile sera un tableau dynamique qui permettra de réaliser les actions suivantes sur son contenu :

- « Empiler » : ajoute un élément sur la pile. Terme anglais correspondant : « Push ».
- « Dépiler » : enlève un élément de la pile et le renvoie. Terme anglais correspondant : « Pop ».
- « La pile est-elle vide ? » : renvoie vrai si la pile est vide, faux sinon.
- « La pile est-elle pleine ? » : renvoie vrai si la pile est pleine, faux sinon.

La classe Pile contient les attributs suivants :

- capacite (entier) : représente la capacité du tableau dynamique
- nombreElements (entier) : représente le nombre d'éléments actuellement dans la pile
- elements : tableau dynamique de type générique

La classe Pile contient les méthodes suivantes :

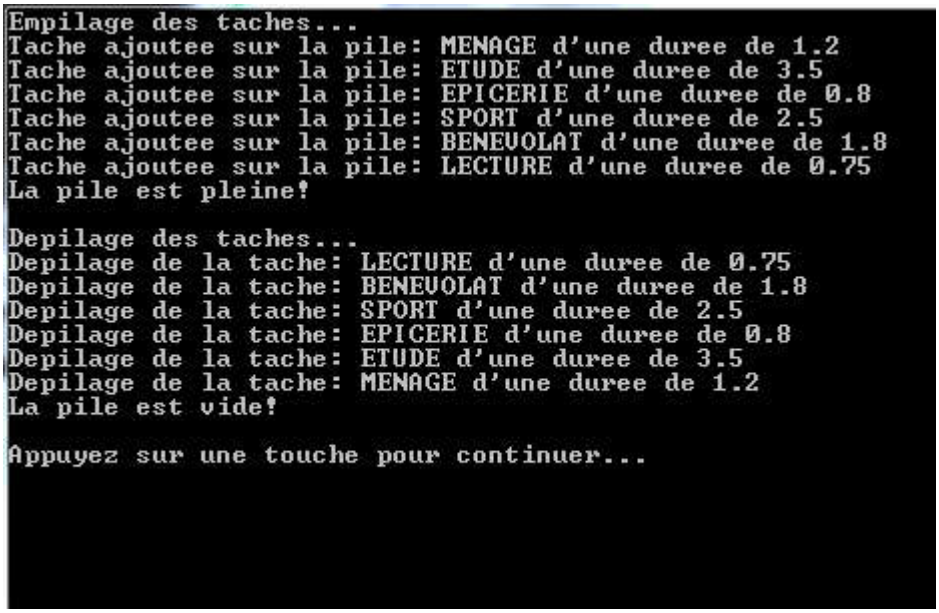
- Un constructeur par défaut qui initialise le tableau dynamique à une capacité par défaut de 6. Pour simplifier l'implémentation de la pile, la capacité du tableau restera fixe.
- Un destructeur, si nécessaire.
- Une méthode empiler() qui reçoit un élément de type générique en paramètre et l'ajoute sur la pile. Cette méthode retourne *true* si l'élément peut être ajouté, *false* sinon.
- Une méthode depiler() qui effectue un retour par paramètre de l'élément sur le dessus de la pile. Cette méthode retourne *true* si le dépilage est réussi, *false* sinon.
- Une méthode estVide() qui retourne une valeur booléenne indiquant si la pile est vide.

- Une méthode estPleine() qui retourne une valeur booléenne indiquant si la pile est pleine.
- Une méthode obtenirSommet() ne prenant aucun paramètre et qui retourne une référence sur l'élément du dessus de la pile.
- Une méthode obtenirTaille() qui retourne un entier indiquant le nombre d'éléments actuellement dans la pile.

Votre pile est sensée accepter **n'importe quel type de donnée**.

Le fichier main qui vous est fourni doit pouvoir fonctionner avec vos implémentations.

Exemple d'exécution :



Partie 2 - Travail à réaliser

Nous voulons faire des recherches dans une base de données (BD) de gènes. Le code génétique étant fait d'acides nucléiques, un gène est donc composé d'une série de nucléotides. Pour simplifier le stockage des informations, on représente le contenu d'un gène par une chaîne de caractères faite de A (pour adénine), de G (pour guanine), de T (pour thymine) et de C (pour cytosine). Dans notre dictionnaire, la classe Gène contient les caractéristiques suivantes : l'identificateur du gène, son nom officiel, sa description, le nom de l'espèce et son contenu en nucléotides.

Vous devez donc implémenter des classes et des méthodes permettant de faire des recherches dans un dictionnaire de gènes. Faites attention à bien vous familiariser avec les caractéristiques de chacune des recherches.

Le dictionnaire se remplit dès le début du programme à partir du fichier genes.txt. Pour vous faciliter la tâche, le code vous est fourni dans le fichier main. Bien que ce fichier s'inspire de données réelles, le contenu en nucléotides des gènes a été réduit pour ne pas que le fichier soit trop gros.

Il y aura deux façons de gérer les données concernant les gènes :

- En utilisant des *list* de la bibliothèque standard STL, donc un conteneur séquentiel.
- En utilisant des *map* de la bibliothèque STL, donc un conteneur associatif.

Vous devez implémenter les deux !

Le déroulement typique de l'application demande à l'usager quelle version il désire utiliser. Peu importe la version (*list* ou *map*) que l'usager choisit, toutes les commandes doivent fonctionner et donner le même résultat. Votre code doit fonctionner avec le programme principal (main.cpp) et la classe Gene qui vous est fournie.

La version avec *list*

Les listes séquentielles représentent des conteneurs d'objets de même type. On remarque qu'un certain nombre de méthodes, comme par exemple *clear()* et *size()*, sont communes à tous les conteneurs STL.

Pour la version avec des listes STL, les méthodes doivent absolument utiliser des algorithmes de STL. Pour ce faire, vous aurez un certain nombre de foncteurs à implémenter. **Il ne faut donc pas retrouver de boucles pour parcourir les listes.**

La version avec *map*

Les listes d'associations, ou *map*, sont des conteneurs très pratiques de la STL. Dans la STL, un *map* est défini comme étant un ensemble de paires dont le premier élément représente la clé et le deuxième élément la valeur.

Pour la version avec le *map* STL, on vous demande d'utiliser des boucles utilisant les itérateurs et de réutiliser les foncteurs préalablement définis lorsque possible. Il est conseillé d'utiliser un « multimap » pour faciliter certaines recherches.

Classes *ConteneurGenesListe* et *ConteneurGenesMap*

Les classes ConteneurGenesMap et ConteneurGenesListe héritent de la classe de base abstraite conteneurGenes et doivent implémenter toutes les méthodes virtuelles de celle-ci.

Dans chacune des classes, un attribut doit être créé :

- Un conteneur STL approprié de pointeurs de Gene

Les méthodes suivantes doivent être implémentées :

- Un constructeur par défaut
- Un destructeur si nécessaire
- Une méthode *inserer* qui reçoit en paramètres les informations nécessaires sur un gène à ajouter dans le conteneur. Le gène doit être créé dynamiquement.
- Une méthode *trouver* qui permet de rechercher un gène à partir de son id et retourne un pointeur sur celui-ci. La méthode doit retourner un pointeur nul si le gène n'est pas trouvé.
- Une méthode *retirer* qui permet de retirer un gène du conteneur à partir de son id. La méthode retourne une valeur booléenne indiquant si un gène a été retiré.
- Une méthode *retirerEspece* qui permet de retirer tous les gènes d'une certaine espèce. La méthode retourne le nombre de gènes retirés.
- Une méthode *vider* qui permet de retirer tous les gènes du conteneur.
- Une méthode *afficherParLongueur* qui affiche dans un flux de sortie passé en paramètre les gènes en ordre croissant de leur longueur (contenu).
- Une méthode *afficherParEspeceEtNom* qui affiche, dans un flux de sortie passé en paramètre, les gènes en ordre croissant d'espèce et pour chaque espèce en ordre croissant de nom.
- Une méthode *afficherEspece* qui affiche, dans un flux de sortie passé en paramètre, tous les gènes associés à une espèce donnée.
- Une méthode *modifierNoms*, qui permet de modifier le nom d'un ou plusieurs gènes d'une certaine espèce passée en paramètre. Cette méthode reçoit en paramètre un objet de type *map* dont la clé représente le nom d'un gène à modifier et la valeur associée le nouveau nom désiré. Plusieurs paires de noms peuvent ainsi être fournies. La méthode retourne le nombre de noms modifiés. Attention : des espèces différentes peuvent avoir des gènes ayant le même nom.

Pour toutes ces méthodes, c'est à vous de déterminer les foncteurs appropriés à utiliser.

Pour les méthodes d'affichage, pensez à utiliser la surcharge de l'opérateur << déjà implémentée dans Gene.

Rappel : il est interdit d'utiliser des boucles pour parcourir la liste de gènes dans *ConteneurGenesListe*.

Foncteur MemeId

Ce foncteur prend en argument un pointeur de Gene et renvoie *true* si ce gène possède le même id que l'attribut de la classe correspondante. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Il possède l'attribut suivant :

- Un entier désignant le id de référence auquel sera comparé celui du gène passé en argument.

Il possède les méthodes suivantes :

- Un constructeur par paramètres qui initialise l'attribut id

Foncteur MemeEspece

Ce foncteur prend en argument un pointeur de Gene et renvoie *true* si ce gène appartient à la même espèce que l'attribut de la classe correspondante. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Il possède l'attribut suivant :

- Une chaîne de caractères désignant l'espèce de référence auquel sera comparée celle du gène passé en argument.

Il possède les méthodes suivantes :

- Un constructeur par paramètres qui initialise l'attribut espèce

Foncteur MemeNom

Ce foncteur prend en argument un pointeur de Gene et renvoie *true* si ce gène possède le même nom que l'attribut de la classe correspondante. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Il possède l'attribut suivant :

- Une chaîne de caractères désignant le nom de référence auquel sera comparée celui du gène passé en argument.

Il possède les méthodes suivantes :

- Un constructeur par paramètres qui initialise l'attribut nom

Foncteur TriParLongueur

Ce foncteur prend en argument deux pointeurs de gènes et renvoie *true* si la longueur du contenu du premier gène est plus courte que celle du deuxième gène. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Foncteur TriParEspeceEtNom

Ce foncteur prend en argument deux pointeurs de gènes et renvoie *true* lorsque le premier est alphabétiquement inférieur au deuxième, en termes d'espèce et de nom. Ce foncteur doit permettre le tri des gènes par espèce d'abord, puis par nom pour chaque espèce. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Foncteur DetruireGenes

Ce foncteur reçoit un pointeur de Gene en paramètre. Lorsque appelé par un algorithme de STL, il permet de détruire tous les gènes du conteneur. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Foncteur DetruireEspece

Ce foncteur reçoit un gène en paramètre. Lorsque appelé par un algorithme de STL, il permet de détruire tous les gènes associés à une certaine espèce. **ATTENTION : La déclaration de ce foncteur se fait dans le fichier Foncteur.h**

Il possède l'attribut suivant :

- Une chaîne de caractères désignant l'espèce de référence auquel sera comparée celle du gène passé en argument.

Il possède les méthodes suivantes :

- Un constructeur par paramètres qui initialise l'attribut espèce

Main.cpp

Le programme principal contient des directives à suivre pour instancier différents objets et essayer les différentes méthodes implémentées.

Le résultat final devrait être similaire à ce qui suit :

```
---  Voulez-vous utiliser la version liste (1) ou la version map (2) ?
1
Utilisation de la version LISTE
------------------------------------------------------------
AFFICHAGE PAR ESPECE ET NOM

        552     Avpr1a
        Homo sapiens
        arginine vasopressin receptor 1A
        taattgcttg aaggattttt tccagacagg tggtctggaa acctttttacc tattaccttc
        catccctgaa ccatttcaat cttctgcctc

        2290    FOXG1
        Homo sapiens
        forkhead box G1
        aggggtggtt gctgcttttg ctacatgact tgccagcgcc cgagcctgcg gtccaactgc
        gctgctgccg

        4857    NOVA1
        Homo sapiens
        neuro-oncological ventral antigen 1
        ctctcccttc tccactctct ccccctgtct cctttcttct tcttctttca ccctccgtct

        9341    Vamp3
        Homo sapiens
        vesicle-associated membrane protein 3
        agtgacgtct ttgccccgcg ccgcgccgtc ccacccatct ccctggcctc cggtcccaac
        ttcgcttctc tgctgaccct ctctcgtcgc cgctgccgcc gccgcagctg

        54140   Avpr1a
        Mus musculus
        arginine vasopressin receptor 1A
        gacttggagg ggttgggtgt aggggacagt ctccagccac taagatgaga aggagcgcat
        cctgaggcag cctaagactg ctttgggagc agggagagtc cgctcccttg cctcggacaa

        14265   Fmr1
        Mus musculus
        fragile X mental retardation syndrome 1 homolog
        aggaggcgca gcggagccct tggcctcagt cagtcaggcg ctggggagcg tttcggtttc
        acttccggtg aggggccgcg cctgagaggg cgggcagtga agcaaacgga cggcgagcgc
        gggcggtggc agtgacggcg

        22319   Vamp3
        Mus musculus
        vesicle-associated membrane protein 3
        gcccatctcc tcagcctcgg ttgcctgcag cgcgagtccg tcgaccctcg gccattcgcc
        gccgccaccg ccgccaaaat gtctacaggt gtgccttcgg

        25107   Avpr1a
        Rattus norvegicus
        arginine vasopressin receptor 1A
        aatctaagac tgcgctggga gctcagagag tgggcccccc tgcctcagga ccagacagaa
```

```
        gtagggacat gtttgaatat cccatttaaa accactctgc acgcaccgga ttgcgcacgc

------------------------------------------------------------
RECHERCHE DU GENE 25107
        25107   Avpr1a
        Rattus norvegicus
        arginine vasopressin receptor 1A
        aatctaagac tgcgctggga gctcagagag tgggcccccc tgcctcagga ccagacagaa
        gtagggacat gtttgaatat cccatttaaa accactctgc acgcaccgga ttgcgcacgc
------------------------------------------------------------
RETRAIT DU GENE 2290
Le gene a ete retire
------------------------------------------------------------
RETRAIT DE L'ESPECE HOMO SAPIENS
Nombre de genes retires = 3
------------------------------------------------------------
RETRAIT DU GENE 552
Le gene 552 n'a pas ete trouve
------------------------------------------------------------
MODIFICATION DU NOM DES GENES DE Mus musculus: Avpr1a-->XXXXX, Fmr1-->YYYYY et UwtA1-->ZZZZZ
Modification de 2 nom(s) effectuee
------------------------------------------------------------
AFFICHAGE PAR LONGEUR

        22319   Vamp3
        Mus musculus
        vesicle-associated membrane protein 3
        gcccatctcc tcagcctcgg ttgcctgcag cgcgagtccg tcgaccctcg gccattcgcc
        gccgccaccg ccgccaaaat gtctacaggt gtgccttcgg

        25107   Avpr1a
        Rattus norvegicus
        arginine vasopressin receptor 1A
        aatctaagac tgcgctggga gctcagagag tgggcccccc tgcctcagga ccagacagaa
        gtagggacat gtttgaatat cccatttaaa accactctgc acgcaccgga ttgcgcacgc

        54140   XXXXX
        Mus musculus
        arginine vasopressin receptor 1A
        gacttggagg ggttgggtgt aggggacagt ctccagccac taagatgaga aggagcgcat
        cctgaggcag cctaagactg ctttgggagc agggagagtc cgctcccttg cctcggacaa

        14265   YYYYY
        Mus musculus
        fragile X mental retardation syndrome 1 homolog
        aggaggcgca gcggagccct tggcctcagt cagtcaggcg ctggggagcg tttcggtttc
        acttccggtg aggggccgcg cctgagaggg cgggcagtga agcaaacgga cggcgagcgc
        gggcggtggc agtgacggcg

Appuyez sur une touche pour continuer...
```

Correction

La correction du TP5 se fera sur 20 points. Voici les détails de la correction :

- (1.5+1.5 points) Compilation des programmes
- (1.5+1.5 points) Exécution des programmes
- (1.25+2.75 points) Comportement exact des méthodes
- (2 points) Utilisation adéquate des templates
- (3 points) Utilisation adéquate des conteneurs et des algos STL
- (3 point) Utilisation adéquate des foncteurs et des boucles.
- (1 point) Gestion correcte de la mémoire
- (1 point) Documentation du code et normes de codage