UNIVERSITÉ LIBRE DE BRUXELLES ULB

FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

Langages de programmation 2 : Projet C++

Yves Roggeman*

INFO-F202 — Année académique 2019–2020 (2e session)

Résumé

Ceci est un des deux énoncés de l'épreuve de seconde session qui se déroule en août. Il sert de base à l'épreuve orale ; l'évaluation finale porte sur l'acquisition des concepts démontrée à cette occasion. Le but de cet exercice de programmation est donc de montrer une connaissance approfondie de la programmation orientée objet et un usage adéquat des constructions du langage C++, en particulier de l'usage des « **`template`** » et du mécanisme d'héritage.

Les différences entre l'énoncé de première session et celui-ci sont indiqués **en gras, ainsi.**

Le problème posé consiste à définir un « *dictionnaire* », c'est-à-dire une suite de chaînes de caractères, que nous appellerons ici « mot », triée par ordre alphabétique. Pour des raisons d'efficacité, la structure interne choisie sera celle d'un *trie* : **une structure logique** d'arbre de caractères sans doublons, tel que décrit ci-dessous.

1 Les deux structures de base

Avant d'implanter notre dictionnaire, nous allons définir une classe abstraite correspondante : ses opérations primitives sont « `rech` » à valeur booléenne qui indique si le mot y est présent, « `ins` » qui insère le mot fourni comme paramètre (s'il n'y était pas déjà, sinon c'est sans effet) et « `del` » qui retire le mot du dictionnaire (s'il y était, sinon c'est sans effet). La valeur de ces ceux méthodes est un booléen indiquant si l'opération a effectivement eu lieu. Un dictionnaire stocke le nombre de mots qu'il contient, valeur consultable par une méthode.

Pour simplifier, un « mot » est ici une simple chaîne ASCIIZ : tableau de caractères (**`const char`**`[]`) qui se termine par le caractère nul (code `0x00`).

Par ailleurs, nous allons utiliser un arbre *n*-aire pour implanter notre *trie*. Il nous faut donc définir un **`template`** d'arbre *n*-aire dont les paramètres sont naturellement le type de base de l'information qu'il contient et son degré (la valeur de *n*) ; tous ses nœuds ont exactement *n* sous-arbres fils, certains pouvant être vides. Ses opérations primitives sont « `vide` » à valeur booléenne, « `info` » qui a pour valeur l'information contenue dans sa racine, « `fils(i)` » à valeur (référence vers) arbre et ayant pour paramètre le numéro du fils qu'il renvoie (avec $0 \leq i < n$), « `del(i)` » qui supprime (qui vide) le sous-arbre fils et « `ins(i, A)` » qui remplace le sous-arbre à la position donnée par celui fourni. Toutes ces opérations ne sont possibles que sur un arbre non vide. Les méthodes d'insertion et suppression ne renvoient pas de valeur.

Comme tout conteneur, pour des raisons d'efficacité notamment dans les transmissions de paramètres, les objets de type « arbre » ne contiennent qu'un pointeur vers un objet instance d'une classe « nœud » imbriquée privée ; ce pointeur et les nœuds sont inaccessibles, **mais il est possible de consulter ou de modifier l'information à la racine d'un arbre**. La classe arbre possède donc au moins deux constructeurs : l'un créant un arbre vide (sans racine), l'autre un arbre contenant une seule information dans sa racine.

Enfin, tant le dictionnaire abstrait que l'arbre *n*-aire ne peuvent être copiés, ni par construction, ni par assignation. Par contre, ils peuvent être transférés de ces deux façons.

*Université libre de Bruxelles (ULB) <yves.roggeman@ulb.ac.be>

2 Le trie

Un *trie* **abstrait** est un arbre dont chaque nœud contient **logiquement** un seul caractère, ainsi qu'un booléen indiquant s'il est significatif. Si c'est le cas, le nœud représente la chaîne des caractères rencontrés sur le chemin qui y mène depuis la racine. Cela a deux conséquences logiques : l'information contenue dans la racine est inutilisée (et non significative), mais toute feuille non-racine est nécessairement significative. Pour des raisons d'efficacité, tous les nœuds ont toutefois la même structure ; ce sont les algorithmes des méthodes qui en tiennent compte. Un *trie*, même logiquement vide, contient toujours une racine [1].

Le *trie* hérite naturellement de la classe abstraite de dictionnaire. Nous choisissons ici de l'implanter par un arbre *n*-aire tel que décrit ci-dessus. Il hérite donc aussi de cette classe ; certaines méthodes seront héritées, d'autres redéfinies, d'autres masquées, pour respecter la logique d'un dictionnaire.

Pour simplifier, vous définirez une classe de caractère particulière qui ne peut contenir que les 26 lettres latines capitales (de code `0x41` à `0x5A`). Elle contient un constructeur de conversion et un opérateur de conversion avec le type **`char`** ; la construction lève une exception si la donnée est incorrecte.

Par convention, notre *trie* ne pourra contenir que des chaînes de telles capitales ; l'arbre aura donc comme degré 26. Vous pouvez définir une méthode de conversion d'une capitale en entier de 0 à 25.

3 Réalisation

Il vous est demandé d'écrire en C++ la définition de toutes les classes et méthodes décrites ci-dessus. Vous écrirez une fonction de test de l'arbre *n*-aire pour différentes valeurs de ses paramètres, y compris différents types de base, une autre pour tester le *trie*, vu comme tel et vu comme une instance concrète de dictionnaire. Enfin, vous écrirez une fonction qui imprime le contenu d'un *trie*, une chaîne par ligne, dans l'ordre alphabétique en utilisant les méthodes disponibles.

Par ailleurs, il vous est demandé de créer également une autre classe fille concrète de dictionnaire sous forme interne d'une liste triée par ordre alphabétique croissant de chaînes ASCIIZ. Vous lui appliquerez les mêmes tests qu'à l'autre implantation. Il est interdit de se servir des conteneurs `std::string` **et** `std::list` **ou autres de la bibliothèque.**

L'évaluation portera essentiellement sur la pertinence des choix effectués dans l'écriture : la codification, la présentation et l'optimisation du programme justifiées par la maîtrise des mécanismes mis en œuvre lors de la compilation et l'exécution du code. D'une manière générale, le respect strict des directives, la concision, la précision, la lisibilité (clarté du texte source), l'efficacité (pas d'opérations inutiles ou inadéquates) et le juste choix des syntaxes typiques du langage C++ seront des critères essentiels d'appréciation. De brefs commentaires dans le code source sont souhaités pour éclairer les choix de codification.

Votre travail doit être réalisé pour le lundi 17 août 2020 à 10 heures au plus tard. Vous remettrez tout votre travail empaqueté en un seul fichier compacté (« .zip » ou autre) *via* le site du cours (INFO-F202) sur l'Université Virtuelle (https://uv.ulb.ac.be/). Ceux-ci devront contenir en commentaire vos matricule, nom et prénom. Le jour de l'examen, vous viendrez avec une version imprimée de ces divers fichiers. Une impression du résultat d'une exécution du programme est également demandée.

1. En fait, un *trie* est une forêt ; sa « racine » n'est qu'une tête de liste de ses arbres.