Méthodologie de la Programmation.
Projet 2020 - 2021

Un arbre généalogique

Dans le cadre de ce projet, nous proposons de réaliser un système de gestion d'arbres généalogiques.

1 Arbres généalogiques

1.1 Définition

Un arbre généalogique permet à un individu donné de répertorier l'ensemble des ses ancêtres. Chaque individu dans l'arbre possède au plus deux ancêtres directs, sa mère et son père [1] et, par transitivité, plusieurs ancêtres indirects. Par exemple,

— `la mère de la mère du père d'un individu`, et

— `le père du père de la mère du père de la mère d'un individu`

sont deux ancêtres indirects d'un individu donné.

On peut désigner les ancêtres d'un individu en indiquant le numéro de **génération** des ancêtres par rapport à cet individu. Le numéro de génération est obtenu en calculant la distance entre l'individu de référence et l'ancêtre considéré. L'individu lui-même est de génération 0, son père et sa mère de génération 1, ses grands-parents de génération 2, etc.

1.2 Type arbre généalogique

Pour ce projet, chaque nœud d'un arbre généalogique comprend l'identifiant unique de l'individu et des informations sur l'individu comme par exemple son nom, son prénom, son sexe, sa date de naissance, etc. Dans la suite du sujet, on n'exploitera pas ces informations. On pourrait envisager une version où on ne conserve que l'année de naissance de l'individu, une autre où on conserve prénom, nom, sexe, date de naissance, lieu de naissance, date de décès, etc.

Nous donnons ci-dessous, un exemple d'arbre généalogique pour un individu ayant pour identifiant `18`. Cet individu est à la racine de l'arbre. Si des informations étaient conservées sur les individus, elles seraient affichées à la suite des identifiants.

```
0   1   2   3    génération
----------------------------
18
    -- père : 2
        -- père : 15
            -- mère : 5
```

1. Pour être aligné avec la législation actuelle, il serait préférable de parler de $\texttt{parent}_1$ et de $\texttt{parent}_2$. Dans la suite, nous utiliserons père et mère.

```
    -- mère : 26
        -- père : 4
-- mère : 8
    -- mère : 33
        -- père : 25
        -- mère : 42
```

Voici quelques éléments relatifs à la structure de l'arbre :

- Un nœud contient l'identifiant d'un individu. D'autres informations pourraient être conservées suivant l'utilisation qui sera faite de l'arbre ;
- La racine de l'arbre est le nœud correspondant à un individu, ici d'identifiant 18.
- Depuis chaque nœud, on peut avoir accès aux parents de l'individu, le père et la mère. Le père de 18 est 2 et sa mère 8.
- L'arbre est partiellement rempli. Par exemple, on ne connaît pas le père de 8 ni la mère de 26.
- Sur cet arbre, on remonte jusqu'à la génération 3 (4, 5, 25 et 42).
- On considère que l'on ne manipule que des arbres, c'est-à-dire que deux noeuds ne peuvent pas avoir un ancêtre commun.

2 Interface de manipulation d'un arbre généalogique

Pour manipuler des arbres généalogiques, le programme réalisé devra offrir des fonctionnalités de définition, de manipulation et d'affichage. On s'intéressera tout particulièrement aux fonctionnalités suivantes :

1. Créer un arbre minimal contenant le seul nœud racine, sans père ni mère.
2. Ajouter un parent (mère ou père) à un noeud donné.
3. Obtenir le nombre d'ancêtres connus d'un individu donné (lui compris). Le nombre d'ancêtres connus de 2 est 5.
4. Obtenir l'ensemble des ancêtres situés à une certaine génération d'un nœud donné. Par exemple, les ancêtres de génération 2 du nœud 18 sont les noeuds 15, 26 et 33. Les ancêtres de génération 1 du nœud 33 sont les nœuds 25 et 42.
5. Afficher l'arbre à partir d'un nœud donné.
6. Supprimer, pour un arbre, un nœud et ses ancêtres. Par exemple, si on veut supprimer le noeud 15 et ses ancêtres, on supprime les nœuds 14 et 5.
7. Obtenir l'ensemble des individus qui n'ont qu'un parent connu.
8. Obtenir l'ensemble des individus dont les deux parents sont connus.
9. Obtenir l'ensemble des individus dont les deux parents sont inconnus.
10. Obtenir la succession d'ancêtres d'une génération donnée pour un nœud donné.
11. Identifier les descendants d'une génération donnée pour un nœud donné
12. Obtenir la succession de descendants d'une génération donnée pour un nœud donné

La liste ci-dessus n'est pas exhaustive, il existe plusieurs autres fonctionnalités qui seraient utile pour manipuler un arbre généalogique.

Remarques.

- Toutes les fonctionnalités précédentes manipulent des arbres binaires ou des chemins dans un arbre binaire. Il est important de traiter la notion d'arbre binaire dans vos programmes.

— Les fonctionnalités précédentes peuvent être construites à partir d'opérations de base sur les arbres binaires.
— Il faudra éviter les codes redondants.

Vous pouvez compléter cette liste par des fonctions ou services que vous aurez identifiés.

3 Travail demandé.

L'objectif principal du projet est de :

1. manipuler un arbre généalogique. On définira un module `Arbre_Genealog` qui devra obligatoirement s'appuyer sur un module **générique** qui spécifie et implante une structure de données de type `Arbre_Bin` ;
2. tester votre programme ;
3. proposer un menu de commandes qui donne accès aux fonctionnalités énumérées en section 2.

4 Exigences pour la réalisation du projet

1. Le projet est individuel.
2. Le langage utilisé est le langage Ada limité aux concepts vus en cours, TD et TP.
3. Le programme devra compiler et fonctionner sur les machines Linux de l'N7.
4. L'ensemble des concepts du cours devront être mis en œuvre, en particulier :
   — Ecriture des spécifications pour tous les programmes et sous-programmes
   — Conception en utilisant la méthode des raffinages
   — Justification des choix des types de données manipulés
   — Conception de modules : encapsulation, généricité, TAD, etc.
   — Définition des tests et le processus de test mis en place
5. Les développements associés au projet comprendront les répertoires suivants.
   — Le répertoire « livrables » ne devra contenir que les livrables explicitement demandés.
   — Le répertoire « src » contiendra les sources de votre application.
   — Le répertoire « doc » sera utilisé pour le rapport.
   — Vous pouvez bien sûr créer d'autres répertoires si vous en avez besoin.

5 Livrables

Les **documents à rendre** sont :
— la **spécification des modules** en Ada (et donc la définition des principaux types dans la partie privée) (extension `.ads`),
— les **programmes de test** (`test_*.adb`),
— une archive contenant le **code source final** de vos programmes (sources.tar),
— un **rapport** (rapport.pdf) contenant au moins :
   — un résumé qui décrit l'objectif et le contenu du rapport (10 lignes maxi),
   — une introduction qui présente le problème traité et le plan du document
   — la **présentation des principaux types de données** et la justification des choix de ces structures

  - la **présentation des principaux algorithmes** définis
  - la **présentation des raffinages ayant conduits aux algorithmes principaux**. Vous vous attacherez à présenter les principales étapes de ces raffinages. Ces raffinages devront montrer comment les algorithmes précédents ont été obtenus
  - **l'architecture de l'application en modules**
  - la présentation des principaux choix réalisés
  - la démarche adoptée pour tester le programme
  - les difficultés rencontrées et les solutions adoptées en justifiant vos choix (en particulier quand vous avez envisagé plusieurs solutions)
  - un bilan technique donnant un état d'avancement du projet et les perspectives d'amélioration / évolution
  - un bilan *personnel* : intérêt, temps passé, temps passé à la conception, temps passé à l'implantation, temps passé à la mise au point, temps passé sur le rapport, enseignements tirés de ce projet, etc.
- Un **manuel utilisateur** (manuel.pdf) qui décrit comment utiliser les programmes développés et qui est illustré avec des copies d'écran.

6 Principaux critères de notation

Voici les principaux critères qui seront pris en compte lors de la correction du projet :
- le respect du cahier des charges,
- la qualité des raffinages,
- la facilité à comprendre la solution proposée,
- la pertinence des modules définis,
- la pertinence des sous-programmes définis,
- la qualité des sous-programmes : ils ne doivent pas être trop longs, ne pas faire trop de choses, ne pas avoir trop de structures de contrôle imbriquées. Dans ces cas, il faut envisager de découper le sous-programme !
- la bonne utilisation de la programmation par contrat (en particulier les préconditions et les invariants de type) et de la programmation défensive,
- la pertinence des tests réalisés sur les sous-programmes
- la pertinence des types utilisateurs définis,
- l'absence de code redondant,
- le choix des identifiants,
- les commentaires issus des raffinages,
- la bonne utilisation des commentaires,
- le respect de la solution algorithmique (les raffinages) dans le programme,
- la présentation du code (le programme doit être facile à lire et à comprendre),
- l'utilisation des structures de contrôle adéquates,
- la validité du programme,
- la robustesse du programme,

7 Date de restitution

Le projet devra être rendu le 28 Janvier 2021 ã 23h00 par **Moodle**.