Programmation Impérative.
Projet 2019 - 2020

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 relatives aux individus étaient conservées, elles seraient affichées à la suite des identifiants.

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.

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

1.3 Le registre d'état civil : un dictionnaire

Nous avons caractérisé précédemment chaque personne par un identifiant (ou une clé) unique représenté par un entier.

On souhaite compléter chaque identifiant par l'ensemble des informations d'état civil d'une personne comme le nom, le prénom, la date de naissance, le lieu de naissance, etc. En quelque sorte, il s'agit de définir un registre d'état civil qui est accédé par l'identifiant de la personne (la clé).

Ce registre pouvant contenir un nombre élevé de personnes, il faudra veiller à avoir un accès efficace aux informations d'une personne.

En plus des informations que peut fournir ce registre, sa définition permet de contrôler la cohérence de l'arbre généalogique construit. Par exemple, on ne pourra pas définir un parent plus jeune que son enfant. D'autres contraintes peuvent être définies.

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 (lui compris) d'un individu donné. 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 15 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. Identifier les ancêtres d'un individu sur `n` (donné) générations données pour un nœud donné
11. Vérifier que deux individus `n` et `m` ont un ou plusieurs ancêtres homonymes (mêmes nom et prénom)
12. ...

Remarques

— La liste ci-dessus n'est pas exhaustive, il existe plusieurs autres fonctionnalités qui seraient utiles pour manipuler un arbre généalogique. Vous pouvez compléter cette liste par des fonctions ou services que vous aurez identifiés.
— Les informations présentes dans le registre ne sont pas figées, il est possible d'ajouter d'autre informations.

3 Organisation du projet

Ce projet est décomposé en deux parties. **La deuxième partie ne doit être commencée que quand la première est terminée.**

3.1 Première partie

Dans une première partie, on se limite à l'étude d'un arbre généalogique représentant des personnes liées par des liens de père et mère comme décrit ci-dessus en section 1.2. Un registre d'état civil sera associé à cet arbre.

Il est demandé de réaliser un programme permettant de manipuler des arbres généalogiques associés à un registre d'état civil et offrant les fonctionnalités définies ci-dessus.

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.
— Il faudra éviter les codes redondants.

3.2 Seconde partie : de l'arbre à la forêt

Dans la seconde partie du projet, on souhaite étendre les arbres généalogiques par des informations sur les conjoints. Ainsi, chaque individu (nœud) de l'arbre généalogique pourra être associé à un ensemble de conjoints où chaque conjoint est représenté par son propre arbre généalogique. Par conjoint, on entend des époux/ses ou bien compagnons/gnes. On obtient ainsi plusieurs arbres, soit en d'autres termes une forêt.

Il est demandé de proposer et d'implanter une structure de données permettant de représenter cette extension de l'arbre généalogique décrit en première partie.

Parmi les fonctionnalités qui pourraient être offertes, on pourrait identifier l'ensemble des demi-frères ou demi-sœurs associés à un individu donné.

4 Travail demandé.

L'objectif principal du projet est de :

1. manipuler un arbre généalogique. On définira un module `Arbre_Genealogique` qui devra obligatoirement s'appuyer sur un module **générique** qui spécifie et implante une structure de données de type `Arbre_Binaire` ;
2. tester votre programme ;
3. proposer un menu de commandes qui donne accès aux fonctionnalités énumérées en section 2.
4. proposer une implantation pour la seconde partie
5. implanter la fonctionnalité qui renvoie l'ensemble des demi-frères et demi-sœurs d'un individu donné.

5 Exigences pour la réalisation du projet

1. Le projet est réalisé en binôme au sein du même groupe de TD.
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. SVN sera utilisé pour rendre le travail fourni. Il sera aussi utilisé régulièrement pour montrer que vous avez bien suivi toutes les étapes recommandées pour la réalisation de votre projet.

   Votre dépôt SVN est disponible à l'URL suivante :

   `http://cregut.svn.enseeiht.fr/2019/1sn/pim/projet/eXXX`

   où XXX est à remplacer par le numéro de votre équipe de projet. Les numéros d'équipe seront communiquées via Moodle. Les dépôts ne pourront être créés que quand les équipes seront constituées.
6. 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.

6 Jalons

Voici les **principaux jalons**
— **[J1]** 04/11/2019. Publication du sujet du projet.
— **[J2.1]** 19/11/2019 à 21h00. Remise des spécifications des modules `Arbre_Binaire`, `Registre`, `Arbre_Genealogique` et les autres modules que vous aurez identifiés. Il faudra également remettre une description de l'architecture logicielle définie.
— **[J2.2]**. 19/11/2019. Remise des premiers niveaux de raffinages du programme principal de la première partie.
— **[J2.3]** 21/11/2019 à 21h00 : Remise des programmes de test des modules. Comme nous l'avons vu, ces programmes peuvent être écrits et compilés même si le corps des modules n'est pas encore défini.
— **[J4]** 13/12/2019. Remise du rapport et du manuel utilisateur.
— **[J5]** 20/12/2019. Remise des sources (première et seconde partie), du rapport et du manuel utilisateur. Vous préciserez les changements éventuels apportés depuis la démonstration. Vous pourrez également remettre une mise à jour du rapport qui doit se limiter aux modifications éventuellement réalisées.

Attention, on ne commencera la seconde partie qu'une fois la première terminée.

7 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 **premiers niveaux de raffinages de la partie 1** (fichier `raffinages-partie1.txt`),
— les **programmes de test** (`test_*.adb`),
— une archive [2] 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 [3] et le plan du document
  — l'architecture de l'application en modules
  — la présentation des principaux choix réalisés
  — la présentation des principaux algorithmes et types de données
  — 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)
  — l'organisation du groupe (qui a fait quoi, etc.)
  — un bilan technique donnant un état d'avancement du projet et les perspectives d'amélioration / évolution

2. On utilisera la commande tar pour produire cette archive. Par exemple, pour inclure tous les fichiers Ada et en .txt, on peut faire : `tar cvf sources.tar *.ad[sb] *.txt`
3. La présentation du problème doit être concise car les enseignants connaissent le sujet !

— une bilan *personnel* et *individuel* : 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.

Remarque. En cas de modification (suppression/ajout/modification), dans la version finale du projet, des spécifications de types, de sous-programmes voire de modules, remises lors du jalon **[J2]**, vous devrez décrire et argumenter avec précision les différentes modifications effectuées.

8 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,
— la pertinence de l'architecture logicielle adoptée,
— 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,
— la bonne utilisation de la mémoire dynamique (valgrind).