Hugo CASTELL - Arthur LEVI
1A FISA SN

Rapport de projet

Ce rapport présente le projet de développement d'une application de gestion d'arbres généalogiques en Ada. L'objectif est de modéliser une structure d'arbre binaire pour représenter les relations familiales, en permettant l'ajout, la suppression, et la recherche de personnes. Le rapport décrit l'architecture modulaire de l'application, les choix techniques, les algorithmes clés, et la démarche de test. Il aborde également nos difficultés rencontrées, les solutions que nous avons adoptées.

Introduction

Le projet consiste à créer une application en Ada pour gérer des arbres généalogiques. L'arbre binaire est utilisé pour représenter les relations parentales, où chaque nœud contient une personne et ses deux parents (gauche pour le père, droite pour la mère).

Table des matières :

Architecture de l'application en modules

L'application est divisée en plusieurs modules :

1. **Binary_Tree** : Implémente une structure d'arbre binaire générique.
2. **List** : Fournit une liste générique pour stocker des éléments (utilisée pour les générations et les recherches).
3. **Genealogic_Tree** : Utilise Binary_Tree pour modéliser un arbre généalogique, avec des fonctions spécifiques comme l'ajout de parents, la suppression de branches, et la recherche de personnes.

En parallèle, nous avons mis en place des fichiers de tests et de menu de commandes :

4. **Menu** : Interface utilisateur en ligne de commande pour interagir avec l'arbre généalogique.
5. **Tests** : Modules de test pour valider les fonctionnalités de chaque composant.

Présentation des algorithmes et types de données

Note : L'ensemble des raffinages est présent dans le fichier Raffinage annexe.

Types de Données

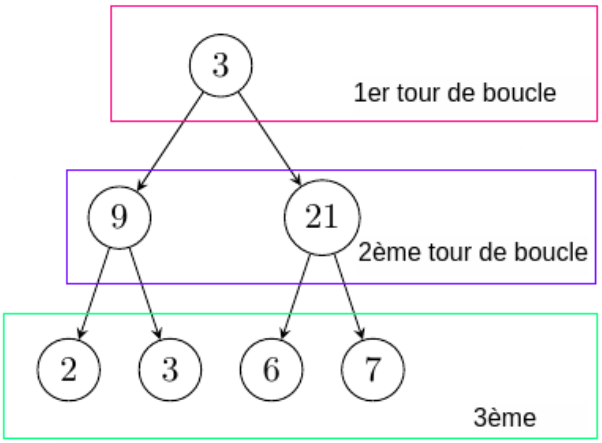
T_Person : Représente une personne avec un identifiant unique.

T_Genealogic_Tree : Arbre binaire où chaque nœud contient une personne et ses deux parents.

Algorithmes Clés

- Parcourir un arbre pour tester chaque personne

  Nous avons décidé de partir sur un parcours par génération :
  - On parcourt une première génération
  - Pour chaque noeud présent, on teste la condition s'il y en a une (par exemple si c'est le bon id…)
  - Si elle n'est pas remplie, alors on regarde s'il a des ancêtres et on les ajoute à une autre liste
  - On passe à la prochaine génération et ainsi de suite



  C'est un algorithme qui est présent dans beaucoup de méthodes : lister les ancêtres, lister une génération, chercher une personne…

- Suppression d'une branche et affichage de l'arbre en récursif

  Ces deux situations requièrent peuvent être abordées différemment car la suppression doit désallouer toutes les feuilles avant de supprimer les branches et

l'affichage de l'arbre doit être fait sur la profondeur pour avoir un rendu cohérent à l'écran. Nous avons ainsi choisi la récursivité.

Suppression :

```
-- Deletes the binary tree and each of its ancestors
procedure Delete(Tree: in out T_Tree) is
begin
  if IsEmpty(Tree) then
    return;
  end if;

  -- Recursive deleting
  if IsBranchEmpty (Tree, False) = False then
    Delete(Tree.Right);
    Tree.Right := null;
  end if;
  if IsBranchEmpty (Tree, True) = False then
    Delete(Tree.Left);
    Tree.Left := null;
  end if;

  -- Delete if leaf confirmed
  if IsLeaf(Tree) then
    Free(Tree);
    Tree := null;
  end if;
end Delete;
```

Affichage :

```
procedure Print(Tree: in T_Genealogic_Tree; Generation : in Natural) is
begin
  if IsEmpty(Tree) then
    return;
  end if;

  for i in 1..Generation loop
    Put("    ");
  end loop;
  Put("-- ");
  PrintPerson(GetData(Tree));
  Put_Line(" ");

  -- Recursive print
  if (IsBranchEmpty (Tree, False) = False) then
    Print(GetRight(Tree), Generation + 1);
  end if;
  if (IsBranchEmpty (Tree, True) = False) then
    Print(GetLeft(Tree), Generation + 1);
  end if;

end Print;
```

Démarche de test

Au fur et à mesure que nous avions progr
tests au préalable pour chacun des modules. Ainsi, chaque méthode devait valider les tests avant que nous passions à la suivante pour nous permettre au mieux de corriger toutes les erreurs.

Evidemment les méthodes déjà validées ont pu être utilisées pour valider les suivantes.

1. **Tests_Binary_Tree** : Vérifie les fonctionnalités de base de l'arbre binaire.
2. **Tests_Person** : Valide la création et la gestion des personnes.
3. **Tests_List** : Teste les opérations sur les listes.
4. **Tests_Genealogic_Tree** : Valide les fonctionnalités spécifiques à l'arbre généalogique.

Difficultés Rencontrées et Solutions Adoptées

La majeure difficulté que nous avons rencontré à été la transition entre le raffinage sur papier et l'Ada car nous n'avions pas anticipé les spécifications du langage et nous avons eu beaucoup de mal à déboguer certaines erreurs.
Pour pallier cette difficulté, nous nous sommes tournés vers d'autres camarades de classe qui nous ont aidés à trouver et à comprendre la source de ces erreurs.

La compréhension du contexte et la réflexion sur le fonctionnement et l'agencement du code aura été la partie la plus simple.

Bilans

Bilan technique de projet

L'application est fonctionnelle et répond aux spécifications demandées sans avoir de fonctionnalités supplémentaires et les tests unitaires couvrent la majorité des cas d'utilisation.

Au niveau des perspectives d'amélioration, nous pouvons imaginer améliorer la partie T_Person qui pourrait être fournie d'informations (nom, prénom, date de naissance etc…).

Bilan du binôme

Au niveau de la répartition des tâches, nous avons essayé de rester équitables sur la charge de travail, même si elle a été plus chargée sur Hugo, prenant en compte ses compétences de BUT Info.

Sinon la répartition des tâches était axée sur une stratégie où une personne couvre les cas d'utilisation en rédigeant les tests tandis que l'autre travaillait sur les méthodes correspondantes (raffinage ou code) avec des discussions à l'oral pour débattre du fonctionnement.

Conception : 30%
Implémentation : 25%
Tests : 30%
Rapport : 15%

Nous étions persuadés de passer moins de temps à implémenter les solutions mais comme dit précédemment, le passage à Ada nous a mis légèrement à l'épreuve.

Bilan personnel : Hugo

Je tire un bilan neutre de ce projet car je n'ai pas eu l'impression de pouvoir aller jusqu'au bout de ce que je voulais faire car je n'arrive pas à assimiler le langage Ada faute d'avoir passé beaucoup trop de temps à manipuler des langages objets comme C# ou Java et je confondais souvent les concepts en développant.

C'est pour cette raison que j'étais très à l'aise sur la partie algorithmique et beaucoup moins sur la partie Ada.

Néanmoins je termine du projet avec un sentiment mitigé car j'aurais voulu ajouter plus de fonctionnalités que celles prévues et dans le même temps je n'arrive pas à comprendre réellement comment Ada fonctionne car je me suis débattu longuement avec la désallocation non vérifiée sur la suppression qui ne fonctionne pas comme une désallocation en C.

Bilan personnel : Arthur

Ce projet m'a vraiment permis de progresser en programmation. J'ai particulièrement apprécié la partie raffinage et algorithmie, où je me suis senti à l'aise. Ces étapes m'ont aidé à structurer les solutions de façon claire avant de passer au code.

Par contre, j'ai eu plus de mal avec la partie développement en Ada, notamment à cause de la gestion des types de données qui m'a donné du fil à retordre. Malgré tout, ça m'a permis de mieux comprendre les spécificités de ce langage, qui est assez différent de ceux que j'avais déjà utilisés.

Un autre point positif, c'est que j'ai découvert et appris à utiliser Git pendant ce projet. Je suis sûr que ça me sera utile à l'avenir, surtout dans les projets collaboratifs où Git est incontournable.

Finalement, ce projet a été formateur, même si j'ai rencontré quelques difficultés techniques. Il m'a permis de renforcer mes compétences en conception et de m'adapter à de nouveaux outils et langages.