RAPPORT

Projet
de Programmation Impérative
2019/2020

Younes SAOUDI
1SN-D
e214

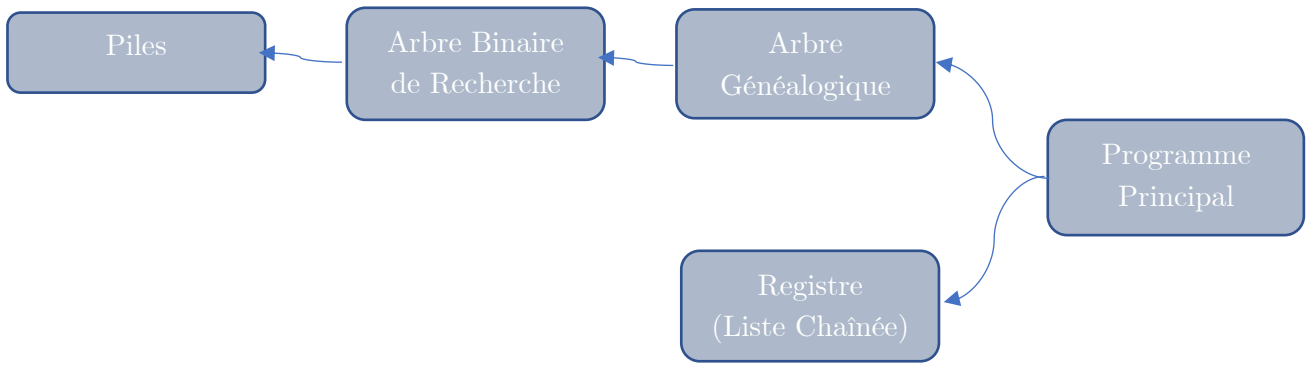
Table des matières

Introduction

Ce rapport a pour but de présenter l'architecture de l'application que j'ai programmée, ses algorithmes principaux et éventuellement les difficultés que j'ai rencontrées pendant la conception et l'implémentation de cette application.
Cette dernière consistait à construire des modules à partir desquels un utilisateur manipulera un arbre généalogique ainsi que son registre d'état civil en effectuant des vérifications, des opérations et des affichages et éventuellement une forêt d'arbres généalogiques.

Architecture

L'application suit l'architecture suivante en modules :



Présentation des principaux choix réalisés, des algorithmes et types de données

D'une part, l'arbre généalogique est un arbre binaire de recherche, ce qui permet une recherche efficace des clés de chaque individu et leur unicité tout en utilisant les piles pour connaître l'intervalle des clés possibles que peut prendre un fils ainsi qu'afficher l'ensemble des feuilles, des individus ayant un seul fils ; etc. D'autre part, le registre est une liste chaînée dont les clés sont toujours ordonnées dans le sens croissant.

1- L'Arbre Binaire de Recherche :

Ce module est constitué de pointeurs vers les nœuds de l'arbre, ces derniers étant constitués d'un enregistrement contenant la clé du nœud (c'est un `Integer`), la donnée de ce nœud qui est un type générique devant être

instancié ainsi que les pointeurs des fils gauches et fils droits.
Puisque le type de la donnée d'un nœud est générique, une constante générique `Zero` doit être instanciée, c'est le zéro du type générique, ou la valeur que retourne les fonctions quand la clé n'existe pas : C'est le `Character 'E'` dans l'arbre généalogique pour signifier Erreur. De même, quand une fonction doit retourner la clé d'un nœud mais que ce nœud n'existe pas, la valeur -34404 est retournée, cela est du *LeetSpeak* signifiant `ERROR` (3 : E / 4 : R / 0 : O).

D'autre part, les fonctions qui cherchent les feuilles d'un arbre, les nœuds ayant un seul fils ou les nœuds en ayant deux retournent toutes des piles contenant leurs clés.

2- Le Registre :

Le registre est une liste chaînée de cellules dont les clés sont dans l'ordre croissant. Chaque cellule contient la clé, le nom complet, la date de naissance, le lieu de naissance, l'âge d'un individu et le pointeur sur la prochaine cellule.
Le nom complet et le lieu de naissance sont des `Unbounded_Strings` tandis que l'âge et la clé sont des `Integers`. La date de naissance est un type `T_Date` : un enregistrement contenant le jour (`Integer`), le mois (`T_Mois`) et l'année (`Integer`).
Le type `T_Mois` est une `Enumération` de `JANVIER, FEVRIER, MARS, AVRIL...`

3- Programmation Défensive :

Les procédures et fonctions programmées sont presque toutes implémentées défensivement. Ceci a été fait pour assurer que l'utilisateur puisse être guidé dans son utilisation de l'application sans que celle-ci s'arrête à chaque fois qu'il effectue une opération erronée. Ainsi, une fonction qui détermine l'intervalle de clés qu'un nœud peut prendre était nécessaire. Cette fonction `NewKeyInterval` retourne une pile qui, suivant le prédécesseur du nœud que l'on veut ajouter, contient les bornes de l'intervalle des valeurs des clés possibles. J'ai choisi les conventions suivantes :

-Un intervalle [min,max] sera représenté dans la pile par {max, min} avec min<max

-Quand une seule valeur est possible la pile est {max,min} avec min=max

-Quand aucune valeur est possible, ceci est représentée par le fait que la pile est de la forme {max,min} avec min>max

-Quand l'intervalle n'est pas minoré, i.e. l'intervalle est de la forme]-INFINI, max], la pile devient par convention {min,0,-181199}. Le fait qu'elle

contient trois valeurs signifie qu'elle a une borne infinie. Si le bottom de la pile est -181199, alors l'intervalle n'est pas minoré.

-Quand l'intervalle n'est pas majoré, i.e. l'intervalle est de la forme [min, +INFINI[, la pile devient par convention {-181199,0,max}. Le fait qu'elle contient trois valeurs signifie qu'une borne est infinie. Si le sommet de la pile est -181199, alors l'intervalle n'est pas majoré.

Quand aucune valeur n'est possible, les procédures d'ajout et de modification demandent à l'utilisateur s'il veut modifier la clé du prédécesseur (un appel récursif à la procédure de modification est ainsi fait), ou de multiplier toutes les clés par 10. Dans ce cas, si l'utilisateur voulait que ce soit un fils gauche, la clé devient `Cle_Parent*10 - 5` sinon elle devient `Cle_Parent*10 + 5`. Ceci assure l'unicité des clés et leur bonne répartition dans l'arbre puisqu'après multiplication par 10, toutes les clés auront comme chiffre d'unité 0 alors que la nouvelle aura un chiffre d'unité 5 et sera inférieure à la clé du parent si c'est un fils gauche et supérieure sinon.

4- L'Arbre Généalogique :

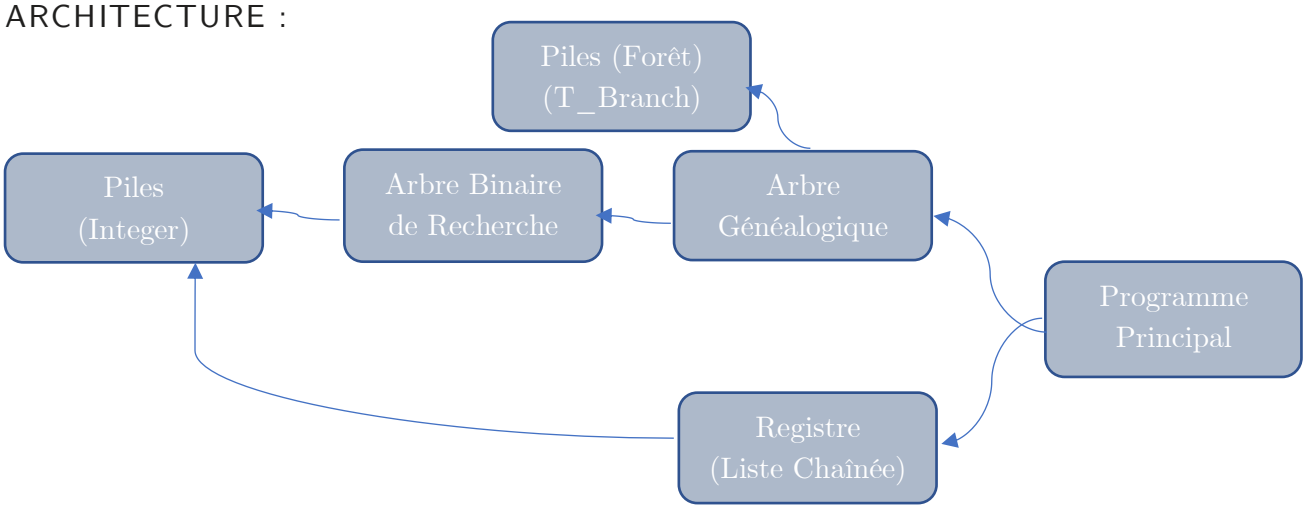
L'arbre généalogique est un module qui instancie le module générique Arbre Binaire de Recherche en un module dont la donnée générique est un `Character` : Soit '`P`' (Père) soit '`M`' (Mère), et utilise toutes les fonctions et procédures de l'arbre binaire de recherche. Les seules procédures qui font plus qu'un simple appel aux sous-programmes de l'ABR sont celles qui affichent les piles retournées par les fonctions de l'ABR (Ensembles de feuilles, d'individus ayant un seul parent connu ; etc.)

Démarches pour tester le programme

D'une part, j'ai testé chaque module avec des fichiers de test `test_*.adb` qui construisent un exemple d'arbre binaire/généalogique/registre, sur lequel plusieurs opérations sont effectuées (modifications, suppression, ajout, recherche) et vérifiées avec des `pragma assert`.
Pour le programme principal, il fallait construire d'abord un arbre assez compliqué après avoir testé les fonctionnalités du menu (le choix d'options, l'accès aux sous-menus, le retour au menu principal, quitter le programme ; etc.) pour le manipuler en utilisant les sous-programmes du registre et de l'arbre généalogique redéfinis comme sous-programmes du programme principal tout en vérifiant le bon fonctionnement avec des affichages ou avec les procédures de vérification.

Seconde Partie

ARCHITECTURE :



Dans la seconde partie, j'ai effectué quelques modifications. Tout d'abord, l'enregistrement du registre T_Registre contient aussi une pile Conjoints de clés Integer de type T_Pile importé du module Arbre_Genealogique. Cette pile Conjoints contient, comme son nom l'indique, les clés des conjoints d'un individu. L'ajout de ses clés n'est guère automatique et devra se faire manuellement avec la fonction Ajouter_Conjoint du module Registre.
Deuxièmement, le module Arbre Généalogique instancie un nouveau type de Pile, Foret.T_Pile, qui contiendra les arbres généalogiques d'une forêt. En utilisant ce type, les nouveaux sous-programmes Initi_Foret, Descendants_Noeud_Foret et Half_Sibling_Foret permettent respectivement de :

- Initiliaser la forêt en l'empiler d'arbre généalogiques initialisés avec des clés saisies par l'utilisateur
- Récupérer tous les descendants d'un individu dans la forêt en les mettant leurs clés dans une pile d'Integers
- Afficher les demi-frères et demi-sœurs d'une clé à partir de la forêt.

En réalité, d'une part, Descendants_Noeud_Foret parcourt chaque arbre de la forêt en cherchant les occurrences d'un individu et empilent la clé de son enfant dans chaque arbre ; d'autre part, la fonction Half_Siblings_Foret utilise la fonction Descendants_Noeud_Foret pour récupérer tous les enfants des parents d'un individu dans deux piles différentes, les concatène dans une seule pile qui est ensuite affichée en supprimant les enfants communs entre les deux parents. L'ensemble de demi-frères et demi-sœurs est ainsi obtenu.
Cette implémentation n'utilise pas du tout la pile des clés des conjoints existant dans le registre.
Le fichier `mainforet.adb` est l'implémentation de la forêt généalogique dans un menu.

Difficultés rencontrées

La difficulté rencontrée était de « protéger » le programme des mauvaises manipulations de l'utilisateur, surtout quand cela s'agissait de l'ajout d'une valeur impossible ou de la modification d'une clé dont la valeur initiale est déjà la seule valeur possible. Obtenir l'intervalle correct des clés possibles était difficile à concevoir et implémenter.
Une autre difficulté était de bien comprendre ce qui était demandé de nous. Tous mes camarades avaient une interprétation différente du sujet et cela s'est empiré quand différents élèves demandaient une explication à différent professeurs. Le sujet ne précise pas exactement ce qu'il faut faire ni ce qui n'est pas nécessaire.
Pour la seconde partie, la difficulté résidait plutôt dans la conception des types puis dans l'accès aux différents arbres de la forêt pour les manipuler.

Bilan Technique

La première partie du projet a été complètement implémentée que ceux soient les modules ou le menu (`main.adb`). Il y a pourtant des choses qu'il faudrait améliorer :

- Le programme principal ne gère pas parfaitement les éventuelles DATA_ERROR dans la sélection des options des menu ou la saisi des clés, noms, dates ; etc. Ceci est parce que je n'ai pas eu le temps d'implémenter une méthode qui gère ces exceptions en gardant les valeurs des arbres et registres : le programme redémarre complètement si l'utilisateur saisit un mauvais type.
- Si on veut ajouter un fils (ABR) à un nœud donné et que :
    - 1- L'intervalle des clés que peut prendre ce fils est nul
    - 2- L'intervalle des clés que peut prendre le nœud auquel on veut ajouter un fils est une seule valeur

  alors, le programme annoncera à l'utilisateur que l'insertion est impossible puis lui demandera s'il veut changer la clé du prédécesseur. Si c'est le cas, le programme essaiera de changer la clé du prédécesseur qui ne prend qu'une seule valeur. L'utilisateur sera ainsi obligé de saisir cette valeur. Le programme réessaiera alors d'insérer la clé mais en vain puisque la clé du prédécesseur n'a pas changé pour ensuite lui redemander s'il veut la modifier. Ceci peut se produira indéfiniment à moins que l'utilisateur décide pendant une itération l'option de multiplier toutes les clés par 10 au lieu de modifier la clé du prédécesseur.

- Puisque les valeurs -181199 et -34404 sont utilisées par le programme intérieurement, ce dernier interdit leur ajout dans l'arbre pour assurer le bon fonctionnement de l'application en demandant à l'utilisateur de saisir autre chose s'il les saisit durant n'importe quelle étape de la manipulation.

La seconde partie a été implémentée dans les modules ainsi que dans le menu principal mais dans un fichier différent : `mainforet.adb`. Les sous-programmes qui nécessitent l'utilisation du registre et de l'arbre/la forêt généalogique en même temps ont été implémentés comme sous-programmes des programmes principaux `main.adb` et `mainforet.adb`.

Bilan Personnel

Temps passé à la conception et l'implémentation : 50.5h
Temps passé à la mise au point : 6.25h
Temps passé sur le rapport : 3.5h
Enseignements tirés de ce projet :

- Manipulation du type Unbounded_Strings.
- Manipulation des Arbres Binaires de Recherches
- Manipulation des piles et des listes chaînées.
- Programmation défensive.