Rapport du projet 3

Manal Hajji , Hassan Khalil

13 Décembre 2019

Résumé

Ce rapport a pour objectif de décrire et de présenter les différentes étapes adoptées pour résoudre un problème de recherche, de traitement et de gestion : la répértorisation des ancêtres d'un individu . Il détaillera alors notre démarche de réalisation du programme proposé comme solution ,de ses sous programmes et ses tests conçus pour assurer le bon fonctionnement des algorithmes. Il mentionnera les diffucultés rencontrées et les solutions adoptées, ainsi que notre organisation du groupe pour mener ce projet. Un bilan technique donnant un état d'avancement du projet et les perspectives d'amélioration sera adressé en dernière partie.

Introduction

Généralement, la généalogie est une science ayant pour objet la recherche de l'origine et l'étude de la composition des familles. Elle consiste à établir les liens entre les individus. Mais la généalogie c'est aussi la recherche des renseignements sur la famille .

On souhaite par suite implanter un programme de traitement afin de réaliser une application qui facilite la recherche et qui permet la répértorisation des ancêtres à un individu.

Table des matières

1 L'architecture de l'application en modules.

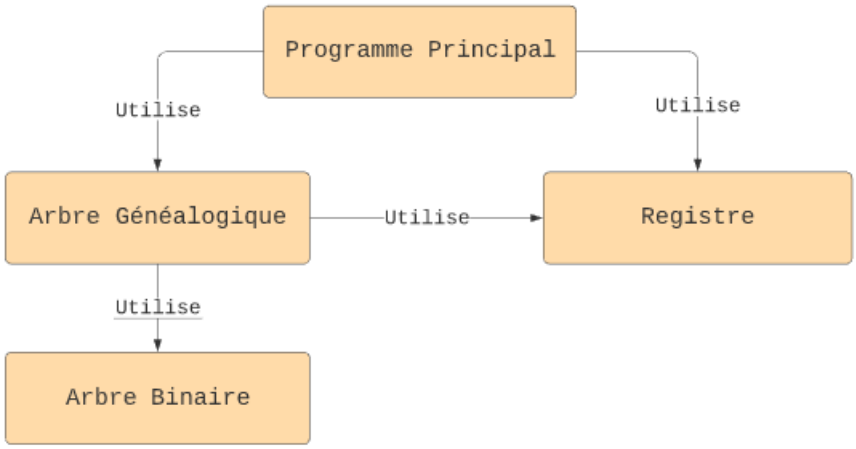

FIGURE 1 – Architecture de l'application en modules.

2 La présentation des principaux choix réalisés.

- La première étape consiste à recueillir, organiser et présenter l'ensemble des informations sur les individus d'une famille afin d'établir des bases de données contenant l'information élémentaire : l'identification précise de chaque individu. Pour cela on a créé un module registre où on enregistre toutes ces données.
- La deuxième étape consiste à spécifier le module arbre généalogique pour établir le lien entre fils et ancêtre. Sauf que cette action requiert un module intermédiaire "arbre binaire" pour faciliter la manipulation de l'arbre généalogique. Une spécification et implantation d'un programme qui manipule l'arbre binaire s'avère cruciale.
- La troisième partie qui concerne l'arbre généalogique fait appel à ce qui est traité dans les deux modules arbre binaire et registre
- la dernière étape est l'étape d'implantation d'un programme principal qui interagit avec l'utilisateur et qui se base sur ce qui est implanté et spécifié dans arbre généalogique et registre.
- Le programme principal proposé assure alors une solution au problème et une application facile à utiliser pour répertorier les ancêtres d'un individu, c'est à dire , traiter les liens entres les individus figurant dans l'arbre et chercher leurs informations .

3 La présentation des principaux algorithmes et types de données.

3.1 Registre

3.1.1 Types de données :

On a modélisé le registre par un arbre binaire de recherche.

— Le type registre (T_registre) est présenté par un pointeur sur une cellule qui est un enregistrement composé d'un identifiant , une donnée de type T_Donnee , un sous arbre gauche et un sous arbre droit de type T_registre.

— Le type T_Donnee est lui aussi un enregistrement composé d'un nom , d'un prénom , du sexe , de la date de naissance de type t_Date , du lieu de naissance et de l'âge d'un individu.

— Le type T_Date est aussi un enregitrement du jour, du mois de type T_Mois et de l'année.

— Le type T_Mois est une énumération des mois de l'année.

```
type T_Mois is (JANVIER, FEVRIER, MARS, AVRIL, MAI, JUIN, JUILLET,
                AOUT, SEPTEMBRE, OCTOBRE, NOVEMBRE, DECEMBRE);
```

```
type T_Date is
   record
       Jour : Integer;
       Mois : T_Mois;
       Annee : Integer;
       -- Invariant
       --     Annee > 0
       --     Jour >= 1
       --     Jour <= Nombre_Jours (Mois, Annee)
   end record;

type T_Donnee is
    record
        nom : unbounded_string ;
        prenom : unbounded_string ;
        sexe : character ;
        date_naissance : T_Date ;
        lieu_naissance : unbounded_string ;
        age : Integer ;
    end record ;

type T_Cellule ;
type T_registre is access T_Cellule ;
type T_Cellule is
    record
        identifiant : integer ;
        donnee : T_Donnee ;
        sous_arbre_gauche : T_registre ;
        sous_arbre_droit : T_registre ;
    end record ;
```

3.1.2 Implantation du registre :

L'algorithme registre.adb implante différentes fonctions et procédures afin de manipuler un registre.

Parmi ces manipulations on cite :

- initialiser_registre : permet de savoir si un registre est vide.
- ajouter : permet d'ajouter un identifiant et une donnée dans un registre.
- modifier_nom, modifier_prenom, modifier_sexe, modifier_date, modifier_lieu, modifier_age : permet de modifier respectivement nom, prénom, sexe, date de naissance, lieu de naissance et l'âge d'un individu.
- id_present : permet de vérifier si un identifiant est présent dans le registre ou non.
- supprimer : permet de supprimer un identifiant d'un registre.
- taille : permet d'obtenir la taille d'un regsitre.
- afficher_info : permet d'afficher l'information correspondant à un identifiant.
- obtenir_nom, obtenir_prenom, obtenir_sexe, obtenir_jour_naissance, obtenir_mois_naissance, obtenir_annee_naissance, obtenir_age, obtenir_lieu_naissance : permet d'obtenir le nom prénom le sexe ... d'un individu.
- creer_date : permet de créer la date à partir du jour , mois et l'annee.
- creer_donnee : permet de créer la donnee à partir du nom , prenom , sexe, date de naissance ,lieu de naissance , age .
- vider_reg : permet de vider le registre
- Est_vide : permet de vérifier si un registre est vide ou non.

3.2 Arbre Binaire

3.2.1 Types de données :

On a présenté le type arbre binaire (T_Arbre_Binaire) par un pointeur sur un noeud. un noeud est un type enregistrement qui contient une clé de type T_Cle générique , un sous arbre droit et un sous arbre gauche de type T_Arbre-Binaire

```
private
    type T_Noeud ;

    type T_Arbre_Binaire is access T_Noeud ;

    type T_Noeud is
        record
            Cle : T_Cle ;
            Sous_Arbre_Droit : T_Arbre_Binaire ;
            Sous_Arbre_Gauche : T_Arbre_Binaire ;
        end record ;
```

```
generic
   type T_Cle is private ;
```

3.2.2 Implantation de l'arbre binaire :

l'algorithme arbre_binaire.adb contient des implantations de différentes fonctions et procédures pour simplifier la manipuation d'un arbre binaire. on en cite :

- creer_arbre_minimal_binaire :permet de initialiser un arbre avec une clé racine.
- Ajouter_fils_Gauche : permet d'ajouter le fils gauche à un noeud sachant sa Clé.
- Ajouter_fils_Droit : permet d'ajouter le fils droit à un noeud en connaissant sa Clé.
- obtenir_nombre_fils : permet d'obtenir le nombre de fils à partir d'un noeud sachant sa Clé.
- hauteur : permet d'obtenir le nombre de niveau d'un arbre .
- obtenir_ensemble_fils_niveau : permet d'obtenir l'ensemble de fils appartenant à une génération d'un noeud sachant sa Clé.
- Supprimer_Sous_Arbre : supprimer un sous arbre d'un arbre à partir d'un noeud sachant sa Clé.
- Est_vide : permet de vérifier si un arbre est vide.
- vider_arbre : permet de Vider un arbre.
- obtenir_ensemble_1_fils : permet d'obtenir l'ensemble des noeuds ayant un seul fils.
- obtenir_ensemble_2_fils : permet d'obtenir l'ensemble des noeuds ayant deux fils.
- obtenir_ensemble_0_fils : permet d'obtenir l'ensemble des noeuds n'ayant aucun fils.
- identifier_fils_n_niveau : permet d'obtenir l'ensemble des fils d'un noeud sachant sa cle sur n générations.
- Taille_Arbre : permet d'obtenir la taille d'un arbre binaire.
- afficher_arbre : afficher les éléments d'un arbre à partir d'une clé.
- Est_Present : permet de savoir si une cle est présente ou pas dans un arbre.

3.3 Arbre généalogique

3.3.1 Types de données :

Le type arbre généalogique (T_Arbre_Genealogique) est un type arbre binaire avec le Sous_Arbre_Gauche attribué au père, le Sous_Arbre_Droit attribué à la mère et le type T_Cle de type entier.

```
procedure afficher_cle (cle : in integer) ;


package Arb_Genealogique is
    new  Arbre_Binaire (Integer,afficher_cle);
use Arb_Genealogique;

type T_Arbre_genealogique is new T_Arbre_Binaire ;
```

3.3.2 Implantation de l'arbre généalogique :

l'algorithme arbre_genealogique.adb contient des implantations de différentes fonctions et procédures pour simplifier la manipuation d'un arbre généalogique. on en cite :

- Creer_Arbre_minimal : permet de créer un arbre dont la racine est Cle .
- Ajouter_pere : permet d'ajouter le père à un noeud sachant son identifiant.
- Ajouter_mere : permet d'ajouter la mère à un noeud sachant son identifiant.
- Obtenir_nombre_ancetre : permet d'obtenir le nombre d'ancêtres à partir d'un fils sachant son identifiant.
- obtenir_ensemble_ancetre_generation : permet d'obtenir l'ensemble d'ancêtres appartenant à une génération d'un fils sachant son identifiant.
- afficher_arbre_genealogique : permet d'afficher un arbre à partir d'un identifiant.
- supprimer_sous_arbre_genealogique : permet de supprimer un Sous-arbre d'un arbre à partir d'un fils sachant son identifiant.
- obtenir_ensemble_1_parent : permet d'obtenir l'ensemble des fils ayant un seul parent.
- obtenir_ensemble_2_parent : permet d'obtenir l'ensemble des fils ayant deux parents.
- obtenir_ensemble_0_parent : permet d'obtenir l'ensemble des fils ayant deux parents.
- identifier_ancetre_n_generation : permet d'obtenir l'ensemble des ancêtres d'un noeud sachant son identifiant sur un nombre de générations n.
- ancetre_homonymes : permet de savoir si deux individus ont au moins deux ancêtres sont homonymes.
- vider : permet de vider un arbre de ses éléments.
- Est_arbre_vide : permet de vérifier si un arbre est vide ou non.

4 La démarche adoptée pour tester le programme

4.1 Tests des Modules

Afin de tester les modules , nous avons réalisé un algorithme de test pour chacun d'entre eux. Il était à chaque fois question de tester les procédures et de vérifier l'efficacité des fonctions avec un ou plusieurs "pragma assert".

```
mhajji@omble:~/pim/tp/e410/livrables$ ./test_registre
nom : nom
prenom : prenom
sexe : M
Date de naissance : ( 10 FEVRIER 1956)
Lieu de naissance : Toulouse
age : 56
```

FIGURE 2 – Test du registre.

```
------------ les fils de la cle C dans le 1er niveau  ---------------
G  F



_____Creer un arbre de caracteres_____

0  1  2  generation
---------------------
 A
    -- pere : B
        -- pere : D
        -- mere : E
    -- mere : C
        -- pere : F
        -- mere : G
```

FIGURE 3 – Test de l'arbre binaire.

4.2 Test du programme pricipal

Pour s'assurer du bon fonctionnement du programme principal nous l'avons nous-mêmes exécuter pour tester toutes les fonctionnalités proposées et pour vérifier que le programme répond bien à ceux que l'utilisateur souhaite faire.

```
------------------- Merci , vous avez rempli toutes les informations ------------------------


     Si vous voulez accéder à votre arbre généalogique , tapez A .
     Si vous voulez accéder au registre, tapez R .
     Si vous voulez quitter, tapez Q .
     Votre choix : r
------------------- Bienvenue dans votre registre ---------------------------------


------------------- Voici les fonctionnalités que vous pouvez utiliser : ---------------------------

    1) Ajouter un individu au registre
    2) Modifier les informations de l'individu
    3) Vérifier si un individu est présent dans le registre
    4) Supprimer un individu du registre
    5) Obtenir le nombre d'individus dans le registre
    6) Afficher les informations d'un individu
    7) Obtenir une information sur un individu
    8) Vider le regisre
    9) Retourner au menu
Votre choix : 8
            Vous avez choisi de vider le registre

Patientez...
     Vidage complété
```

FIGURE 4 – Test du programme principal.

5 Difficultés rencontrées et solutions adoptées

Nous avons rencontrés des difficultés au niveau de conceptualisation de la fonction "ancetre_homonymes" , au niveau de lecture et d'affectation des chaînes de caractères de taille limitée, au niveau de modélisation du registre ,et au niveau de la gestion des exceptions probables du programme principal.

On a essayé d'implanter des fonctions qui retournent un ensemble de type T_Ensemble dans arbre généalogique.Toutefois, Le type reste invisible lors de la compilation.

On a de même essayé plusieurs solutions mais elles n'ont pas résolu le problème. Finalement, on a changé les fonctions en procédures d'affichage.

```
gcc -c -gnata arbre_genealogique.adb
arbre_genealogique.adb:163:32: "obtenir_suivant" is not visible (more references
 follow)
arbre_genealogique.adb:163:32: non-visible declaration at ensembles_chainage.ads
:88, instance at arbre_binaire.ads:17
arbre_genealogique.ads:78:132: "T_Ensemble" is not visible (more references foll
ow)
arbre_genealogique.ads:78:132: non-visible declaration at ensembles_chainage.ads
:9, instance at arbre_binaire.ads:17
gnatmake: "arbre_genealogique.adb" compilation error
mhajji@eowyn:~/pim/tp/e410/livrables$
```

FIGURE 5 – Erreur de compilation de l'arbre généalogique en utilisant des fonctions qui retournent un type T_Ensemble .

Comme solutions pour les autres difficultés, on a utilisé des chaînes de caractères de taille illimitée en important le package "Ada.Strings.Unbounded".

Pour le registre, on a choisi de le modéliser par un arbre binaire de recherche. Les exceptions sont traitées.

6 Partie 2 du projet

— On a choisi le type arbre forêt comme un pointeur qui pointe sur une cellule de type enregitrement d'un identifiant , un arbre généalogique et l'ensemble des conjoints qui sont associés à cet identifiant et arbre forêt suivant.

— Pour l'implantation on est arrivé à implanter la majorité des fonctionnalités initialiser_arbre , present(Abr,id) , vider(Abr) etc ... (Abr : un arbre forêt)

```
package Ensemble is
    New Ensemble_chainage(T_Element => Integer);
use Ensemble;


type T_Noeud;

type T_arbre_foret is new access T_Noeud;

type T_Noeud is
    record
        id : in Integer;
        Abr_gen : T_arbre_genealogique; --Arbre généalogique associé à id.
        conj : T_Ensemble; -- Ensemble des conjoints de id.
        suivant : T_arbre_foret;
    end record;
```

FIGURE 6 – Type de données de arbre forêt.

7 Modifications apportées aux programmes

— Pour le registre , on a décidé de supprimer l'âge des informations parce qu'il n'apporte pas une nouvelle information sur l'individu puisqu'on a déjà sa date de naissance.

— On a aussi constaté que l'introduction de l'âge implique plusieurs complications au niveau du programme principal , il lève plusieurs exceptions : par exemple si l'utilisateur entre un âge qui n'est pas compatible avec sa date de naissance . On a traité cette exception sauf que un autre problème s'impose : si l'individu est mort ou non.
Donc il faut vérifier si il est vivant ou non , si non il faut entrer sa date de naissance et traiter de nouveau les exceptions, ce qui ajoute plusieurs calculs au niveau du programme , qui ne sont pas d'une grande importance dans l'arbre généalogique.

8 Organisation du groupe

Le découpement du travail et sa réalisation étaient comme suit :
- registre.ads, registre.adb et test_registre.adb : réalisés par Khalil Hassan
- arbre_binaire.ads, arbre_binaire.adb et test_arbre_binaire.adb : réalisés par Hajji Manal.
- arbre_genealogique.ads et arbre_genealogique.adb : réalisés par les deux membres du groupe.
- gestion_arbre_genealogique.adb :
  * Le corps fondamental du programme est écrit par Hajji Manal.
  * La gestion des exceptions est traitée par Khalil Hassan.

9 Bilan technique

Les fonctions et procédures implantées répondent aux multiples fonctionnalités que l'utlisateur peut effectuer. Néanmoins, la question de complexité et d'optimisation peut ne pas être totalement couverte dans nos programmes. De surcroît, l'ergonomie peut être améliorée sur l'interface .

10 Bilan personnel

10.1 Hajji Manal

Ce projet m'a permis de développer mes compétences techniques et cognitives au niveau de la gestion et du traitement d'un projet et d'exploiter mes connaissances développées en programmation impérative. Le travail en groupe m'a été beaucoup plus instructif que toute seule car cela nous a permis , mon binôme et moi, de partager nos connaissances et chercher des solutions pour chaque difficulté envisagée.

Temps passé en heures :
- Modules : 4
- Programme de tests : 3
- Programme principal : 2
- rapport : 5

10.2 Khalil Hassan

Ce projet m'a beaucoup intéressé. Il m'a amené à confronter de nouveaux problèmes et chercher des solutions et donc améliorer ma connaissance en ce qui concerne la programmation impérative et la programmation en Ada.

J'étais très investi et motivé dans le développement du programme tout en respectant les contraintes et les différentes étapes , et ceci grâce au travail en groupe et mes connaissances acquises en TPS et TDS.

Temps passé en heures :
- Modules : 4

— Programme de tests : 3
— Programme principal : 2
— manuel utilisateur : 5