FOIN Nicolas
JAMES Christopher
1SN/CM1/Groupe C/e208

RAPPORT DU PROJET

« UN ARBRE GENEALOGIQUE »

PROGRAMMATION IMPERATIVE
2019 - 2020

RESUME

Ce rapport a pour objectif de détailler le déroulement du travail réalisé par notre binôme pour ce projet d'arbre généalogique et de résumé tous nos choix de conception réalisés afin de mener à bien ce travail.

Ce rapport contient:

- Une introduction qui présente le sujet traité et un plan
- Une architecture des modules utilisés qui détaille les liens entre eux
- Le détail des choix que l'on a réalisé en binôme pour optimiser notre projet
- Quelques algorithmes importants et types utilisés pour nos programmes
- Comment nous avons procédé pour tester nos modules et notre programme principal
- Les difficultés rencontrées tout au long de notre projet sur nos modules, nos tests et sur notre conception globale du projet
- L'organisation des tâches dans notre groupe
- Un bilan technique
- Un bilan personnel où l'on détaille notre intérêt pour ce projet, le temps passé sur chaque étape et l'expérience acquise.

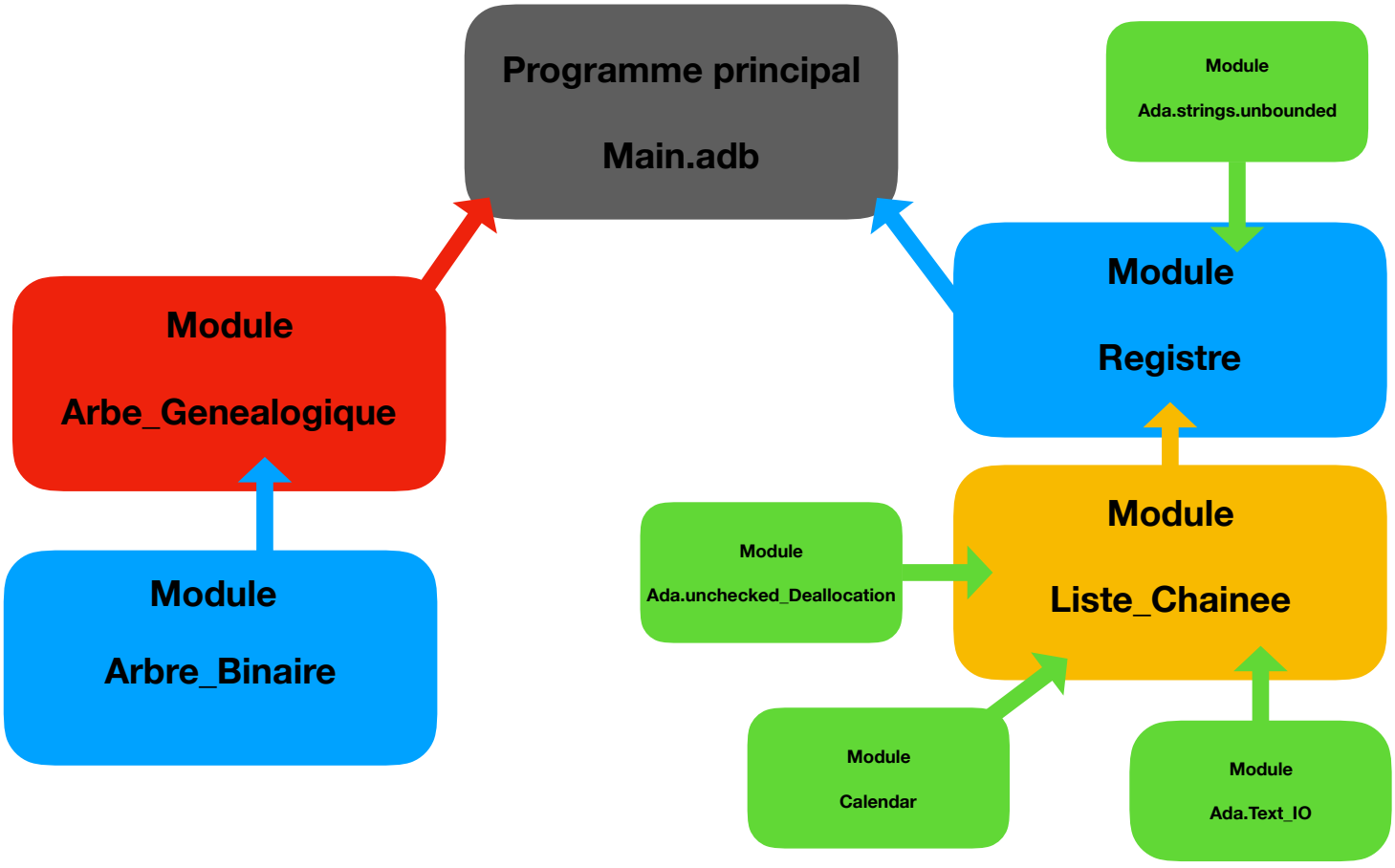
INTRODUCTION

PROBLEME TRAITE

Dans ce projet, nous avons réalisé une structure d'arbre généalogique permettant de répertorier l'ensemble des ancêtres d'une personne à l'aide d'une structure de type arbre binaire et d'un registre contenant l'ensemble des données de chaque personne introduite dans l'arbre.

PLAN DU RAPPORT

I - Architecture de l'application en modules



II - Choix réalisés

Nous avons décidé de réaliser une structure de type **table de hachage** pour notre registre car cela nous semblait le plus pratique et le plus efficace pour stocker et accéder à tous les identifiants dans le registre. On a ainsi eu besoin d'une structure de **listes chainées** pour organiser notre table de hachage.

Ce choix a été plutôt confortable pour nous car grâce à la table de hachage on ne s'est plus du tout préoccupé du problème du classement des identifiants que l'on peut rencontrer dans un arbre binaire de recherche par exemple. L'identifiant d'une personne ajoutée dans le registre est alors simplement le dernier identifiant créé et incrémenté de 1 (lors de l'utilisation du main programme). C'est pour cela qu'on a utilisé un **arbre binaire** simple.

On a également dissocié les modules arbre généalogique et registre pour des commodités de programmation. Dans le programme main, on utilise un arbre généalogique pour la structure d'arbre et un registre pour stocker les données insérées dans l'arbre.

III - Présentation des principaux algorithmes et types de données

A - Principaux algorithmes

a - Dans **liste_chainee**:

Fonction **Tete_liste**:

Fonction **Queue_liste**:

Procédure **Concat**:

b - Dans **Registre**:

Fonction **Est_Present_Registre**: Permet de vérifier si une personne est déjà présente dans un arbre lorsque l'utilisateur veut ajouter une nouvelle personne.

Fonction **Homonymes**: Permet de trouver toutes les personnes ayant les mêmes noms et prénoms.

Procédure **Ajouter_Registre**: Permet d'ajouter une personne dans un registre.

Procédure **Supprimer_Registre**: Permet de supprimer une personne dans un registre.

Fonction **Taille_Registre**: Permet de renvoyer la taille d'un registre (utile lorsque l'on souhaite créer les identifiants).

c - Dans **Arbre_Binaire**:

Procédure **Ajouter_AB**: Permet d'ajouter une donnée dans un arbre après un parent.

Fonction **Est_Present_AB**: Permet de vérifier si une donnée est présente dans un arbre. Utile pour tous les tests dans ce module.

Procédure **Supprimer_AB**: Permet de supprimer une donnée et tous les sous arbres liées au noeud supprimé.

d - Dans **Arbre_Genealogique**:

Procédure **Initialiser_Arbre_Genealogique**: Permet d'initialiser un arbre, utilisée à chaque début de programme.

Procédure **Ajouter_Arbre_Genealogique**: permet d'ajouter une donnée dans un arbre à partir d'un noeud.

B - Les types de données

a - Dans **liste_chainee**:

Type privé **T_Liste** (access T_Cellule) qui défini les listes chainées.

Type generic privé **T_Element** qui défini le type d'éléments à introduire dans les listes.

T_Cellule qui est un enregistrement d'une valeur prise dans T_Element et une T_Liste (pour pouvoir utiliser les pointeurs).

b - Dans **Registre**:

T_Sexe qui est une énumération de Homme ou femme.

T_Donnee qui est un enregistrement d'un identifiant en entier, d'un prénom en string, d'un nom en string, d'une date de naissance en Time et d'un T_Sexe. Ce sont les données que l'on utilise lorsque l'on ajoute une personne dans le registre.

T_Tableau qui est un tableau de capacité finie. C'est le tableau de listes des données (qui sont, elles, de capacité infini).

T_Registre très privé qui contient un tableau de type T_Tableau et le nombre d'individus (ce nombre permet de créer les nouveaux identifiants lorsque l'on modifie le registre).

c - Dans **Arbre_Binaire**:

T_AB très privé qui défini l'arbre binaire (access T_Noeud).

T_Element un type generic privé qui désigne les éléments que contiendra l'arbre.

T_Noeud qui est un enregistrement d'une donnée, et de deux sous arbres (pour pouvoir utiliser les pointeurs).

d - Dans **Arbre_Généalogique:**

Le type **T_Arbre_Genealogique** très privé qui est un type T_AB dont les données sont des entiers.

T_Parents privé, qui est un enregistrement d'un père et d'une mère représentés par des entiers.

IV - Démarche adoptée pour tester les programmes

Pour chaque algorithme dans nos spécifications de module on a défini plusieurs pre et post conditions.

Après avoir compilé sans erreurs les spécifications et implantations des modules, pour chaque module nous avons réalisé un programme de tests contenant des appels de nos procédures/fonctions et des « pragma assert » qui renvoient un message d'erreur si une certaine condition n'est pas vérifiée. Cela nous a permit de vérifier que nos procédures/fonctions faisaient bien ce que l'on souhaitait grâce à l'affichage des arbres ou des registres.

Pour tester le programme principal (nommé main), nous l'avons d'abord compilé puis exécuté plusieurs fois en essayant plusieurs fonctionnalités de nos modules.

V - Difficultés rencontrées rencontrées et solutions adoptées

Nous avons rencontré des difficultés lors de:

- **Le choix du type d'arbre** que l'on utiliserait. L'arbre binaire de recherche ne nous semblait pas pertinent car nous n'avions pas de réelle idée pour hiérarchiser les identifiants et les noeuds associés dans l'arbre. C'est pour cela que l'on a choisi de réaliser une table de hachage et de donner pour valeur à chaque nouvel identifiant, la valeur du dernier identifiant introduit.
- **Le type T_Arbre_Genealogique** nous a aussi posé des problèmes car nous voulions au départ faire un enregistrement d'un arbre binaire et d'un registre (pour avoir à la fois la structure d'un arbre associé à un registre) pour ensuite n'utiliser que ce module dans le programme principal. Mais cela était contraignant car avec un grand nombre de types, l'implantation du module Arbre_Genealogique en devenait compliquée. Nous avons finalement trouvé cela plus simple de dissocier le module Arbre_Genealogique et le Registre pour faciliter leurs implantations et ne les utiliser ensemble que lors du programme principal.
- Quelques soucis liés à **l'allocation dynamique de la mémoire**, que l'on a identifié avec Walkyrie.
- Quelques problèmes liés à nos **tests** et aux **pre/post conditions** qui nous ont ralentis dans l'avancement du projet.

VI - Organisation du groupe

Christopher

Spécification/implantation du module liste_chainee
Spécification/implantation/tests du module Arbre_Binaire
Rapport
Main
Manuel utilisateur
Raffinages

Nicolas

Spécification/implantation/tests du module Registre
Spécification/implantation/tests du module Arbre_Genealogique
Spécification/implantation du module Arbre_Binaire
Raffinages

VII - Bilan technique

Lors de la date de rendu de notre projet, nous avions implanté et testé tous les modules prévus (liste_chainees, Registre, Arbre_Binaire et Arbre_Genealogique) et le programme principal utilisé par l'utilisateur (main) qui a également été testé par nous-mêmes.

Par manque de temps, nous n'avons pas pu réaliser la partie 2 sur la création d'une forêt.

Si l'on aurait eu la possibilité de le faire, nous aurions remarqué dans le cas d'une forêt qu'un noeud pouvait contenir un grand nombre d'individus (les conjoints de la personne associée au noeud).
Pour ce faire, on aurait rajouter dans le type enregistrement T_Donnee de registre une liste de tous les conjoints. En faisant cela, lorsque l'on souhaiterait rajouter un conjoint à un noeud, on aurait créé une procédure qui cherche le noeud et y ajoute tous les conjoints voulu dans la liste associée.

VIII - Bilans personnels

Christopher

Intérêt: Projet intéressant car il m'a permit d'apprendre à utiliser plusieurs modules et à les relier entre eux dans le but de proposer un programme complexe. C'était également le premier projet de programmation en binôme dans lequel on a pu se répartir le travail et discuter lorsque l'on rencontrer des problèmes.

Temps total passé: 23 - 25h

Temps de conception: 4h

Temps d'implantation: 8 - 9h

Temps de mise au point: 7h

Temps de rapport: 4h

Enseignement tiré de ce projet: Implantation et utilisation de plusieurs modules liés, beaucoup de types (génériques, privés, très privés, enregistrements, énumérations), utilisation des pointeurs et une grande partie des programmes sont écrits de manière récursive, ce que l'on avait pas l'habitude de faire auparavant.

Nicolas

Intérêt: Ce projet a suscité mon intérêt car il met en place de manière pratique et concrète des concepts complexes tel que les modules ou les pointeurs.
De plus nous avons pu expérimenter le travail de groupe à travers ce projet en nous répartissant le travail et en communiquant sur les modifications apportées.

Temps total: 24h

Temps de conception: 4h

Temps d'implantation: 8h

Temps de mise au point: 5h

Temps de rapport: 0h

Enseignement tiré de ce projet: Il m'a aussi permit de mieux comprendre les types et comment les utiliser au sein d'un projet complexe, ainsi que des notions plus délicates tels que les types privés et très privés. En me confrontant à des problèmes d'allocation dynamique, j'ai pu comprendre plus en profondeur l'objet pointeur.