Structures de données et algorithmes II

Devoir de TP libre
2021 - 2022

Cadre du problème : gestion d'un arbre généalogique

On se propose de gérer un arbre généalogique d'individus dans *un cas parfaitement idéal* : on suppose que chaque individu a toujours exactement deux parents (père et mère). On ne considère pas les familles recomposées : il n'y a pas de beau-père, belle-mère, demi-frères ou demi-sœurs. Chaque couple père-mère se partage la même liste d'enfants. Cependant un parent peut être inconnu.

On se propose d'utiliser **une table d'individus** modélisée à l'aide d'un tableau contigu. Chaque individu a un identifiant *id* qui correspond aussi à la position dans le tableau. L'adresse (ou indice) dans le tableau vaut *id* décrémenté de un. Les identifiants commencent donc à 1 (=> position 0 dans le tableau).

La généalogie se résume à la structure suivante :

```
typedef struct s_genealogie {
	Individu	*tab;			// tableau alloué dynamiquement
	Nat		id_cur;			// identifiant actuel
	Nat		taille_max_tab;		// taille max du tableau
} Genealogie;
```

La structure "Individu" est une structure comportant le nom de l'individu, la date de naissance et de décès, ainsi que les identifiants de ses deux parents, l'identifiant de son frère / sœur cadet (immédiatement né après lui) et de son fils / fille ainé s'il existe des enfants. L'identifiant 0 est utilisé comme Ω.

Voici la structure Individu :

```
#define LG_MAX 64

typedef struct s_date { unsigned short jour, mois, annee; } date;
typedef unsigned int ident;

typedef struct s_individu {

	char nom[LG_MAX];
	date  naissance, deces;
	ident  pere, mere, cadet, faine;

} Individu;
```

Lorsque le champ « nom » vaut 0 l'individu est invalide. Lorsque l'individu est en vie, sa date de décès vaut {0,0,0}. Lorsqu'un parent est inconnu (père ou mère), l'identifiant correspondant vaut 0. De même lorsqu'il n'y a pas de fils ou frère cadet, resp. fille ou sœur, les valeurs de **faine** et/ou **cadet** sont à 0.

La figure ci-après illustre un exemple d'arbre généalogique. Sur ce schéma les parents de Julien sont Aline et Pierre et les parents de Cloé sont Alex et Sophie. Plus précisément, pour Sophie, le champ **père** vaudra l'identifiant de Henri, **mère** vaudra l'identifiant de Jeanne, **fainé** vaudra l'identifiant de Cloé et **cadet** vaudra l'identifiant de Clémentine. Pour Alex, **père** vaudra Pierre, **mère** vaudra Aline, **fainé** vaudra Cloé et **cadet** vaudra 0, pour Julien **père** et **mère** vaudront aussi Pierre et Aline, mais **fainé** vaudra 0 et **cadet** vaudra Alex.

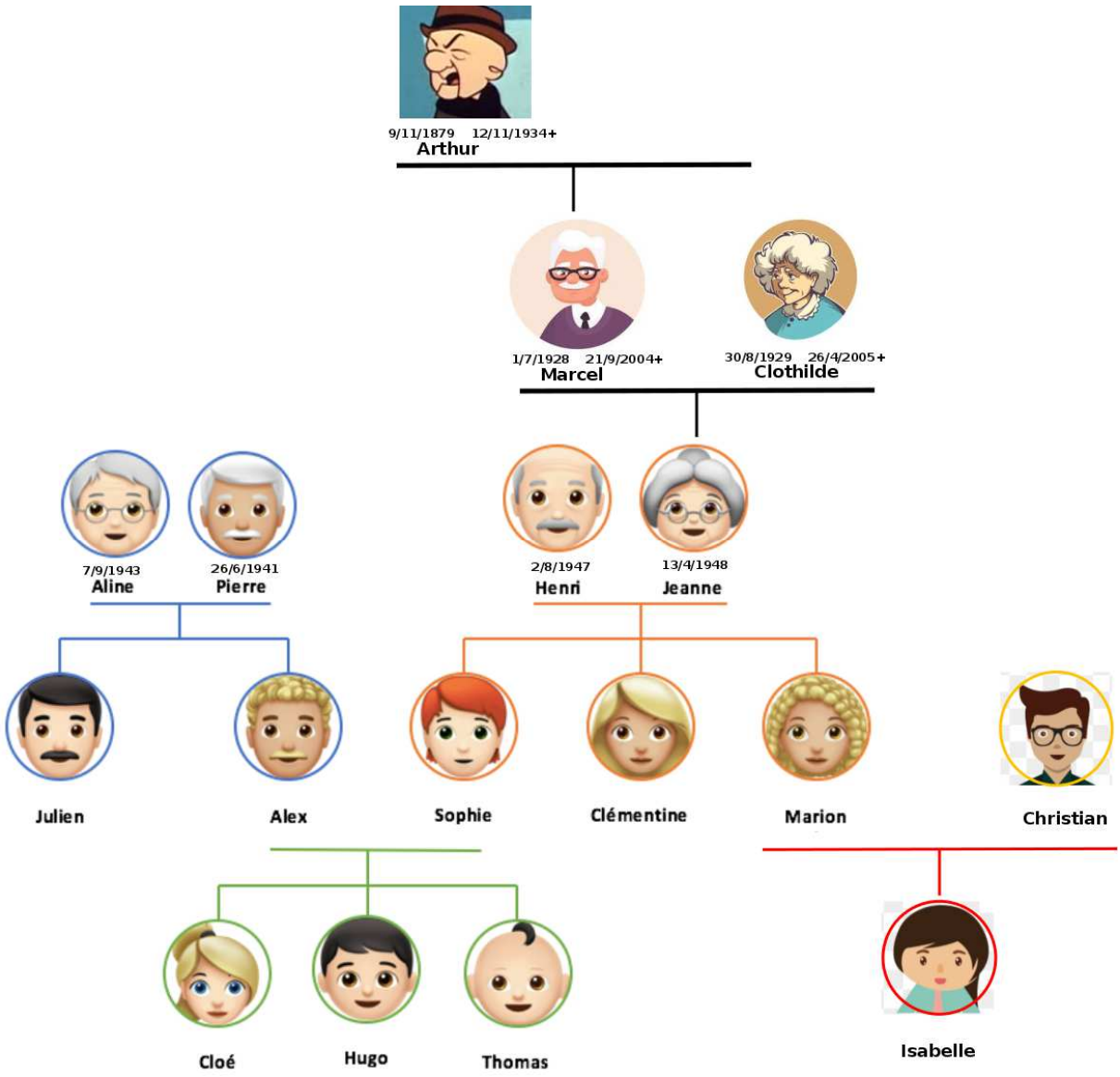

FIGURE 1 – Un exemple d'arbre généalogique.

1. Construction de l'arbre généalogique

1.1 Ecrire le constructeur de base de **Généalogie**, ainsi que son destructeur (libération de la mémoire). On utilisera les profils suivants :

```
void genealogieInit(Genealogie *g);
void genealogieFree(Genealogie *g);
```

1.2 Ecrire une fonction qui permet d'ajouter un individu.

```
ident adj(Genealogie *g, char *s, ident p, ident m, date n, date d);
```

Le code main suivant doit permettre de construire l'arbre généalogique de l'exemple :

```
Genealogie g;
genealogieInit(&g);
ident ia = adj(&g, "Arthur", 0, 0, { 9, 11, 1879 }, { 12, 11, 1934 });
ident im = adj(&g, "Marcel", ia, 0, { 1, 7, 1928 }, { 21, 9, 2004 });
ident ic = adj(&g, "Clothilde", 0, 0, { 30, 8, 1929 }, { 26, 4, 2005 });
ident ije = adj(&g, "Jeanne", im, ic, { 13,4, 1948 }, { 0,0,0 });
ident ihe = adj(&g, "Henri", 0,0,  { 2,8, 1947 }, { 0,0,0 });
ident ial = adj(&g, "Aline", 0,0, { 7,9, 1943 }, { 0,0,0 });
ident ipi = adj(&g, "Pierre", 0, 0,  { 26,6, 1941 }, { 0,0,0 });
ident iju = adj(&g, "Julien", ipi, ial,  { 13,8, 1965 }, { 0,0,0 });
ident ialex = adj(&g, "Alex", ipi, ial, { 18,4, 1969 }, { 0,0,0 });
ident iso = adj(&g, "Sophie", ihe, ije, { 9,11, 1972 }, { 0,0,0 });
ident icl = adj(&g, "Clementine", ihe, ije, { 12,10, 1973 }, { 0,0,0 });
ident ima = adj(&g, "Marion", ihe, ije, { 5,5, 1976 }, { 0,0,0 });
ident ich = adj(&g, "Christian", 0, 0, { 13,2, 1971 }, { 0,0,0 });
ident itho = adj(&g, "Thomas", ialex, iso, { 18,10, 2012 }, { 0,0,0 });
ident icloe = adj(&g, "Cloe", ialex, iso, { 21,6, 2002 }, { 0,0,0 });
ident ihu = adj(&g, "Hugo", ialex, iso, { 12,5, 2005 }, { 0,0,0 });
ident isa = adj(&g, "Isabelle", ich, ima, { 28,4, 2003 }, { 0,0,0 });
```

Remarque : lorsque l'individu est ajouté à la généalogie, il est important de mettre à jour automatiquement les champs **faine** et **cadet** en fonction des enfants déjà connus du père et de la mère (ce sont exactement les mêmes) ! La liste des fils/filles doit par ailleurs être ordonnée du plus âgé (en premier) au plus jeune (en dernier). Il faut donc comparer les dates de naissance. Pour cela, on écrira une fonction *compDate(date, date)*.

1.3 Ecrire une fonction qui permet l'accès à un individu dont on donne l'identifiant :

```
Individu *get(Genealogie *g, ident x);
```

2. Liens de parenté

2.1 Ecrire une fonction qui vérifie si deux individus sont frères ou sœurs :

```
bool freres_soeurs(Genealogie *g, ident x, ident y);
```

Dans l'exemple, si x est Sophie et y est Marion alors la fonction renvoie vrai. Mais si y est Julien alors la fonction renvoie faux.

2.2 Ecrire une fonction permettant de tester si deux individus sont cousins (ils ont au moins un de leur grand parent en commun).

```
bool cousins(Genealogie *g, ident x, ident y);
```

Dans l'exemple, si x est Sophie et y est Marion alors la fonction renvoie faux. Si x est Hugo et y est Isabelle alors la fonction renvoie vrai.

3. Affichage

3.1 Ecrire une fonction permettant d'afficher les noms de TOUS les frères et sœurs d'un individu.

```
void affiche_freres_soeurs(Genealogie *g, ident x);
```

3.2 Ecrire une fonction permettant d'afficher les noms de TOUS les enfants d'un individu.

```
void affiche_enfants(Genealogie *g, ident x);
```

3.3 Ecrire une fonction permettant d'afficher TOUS les cousins / cousines d'un individu

```
void affiche_cousins(Genealogie *g, ident x);
```

3.4 Ecrire une fonction permettant d'afficher TOUS les oncles / tantes d'un individu

```
void affiche_oncles(Genealogie *g, ident x);
```

4. Parcours de l'arbre généalogique

4.1 Ecrire une fonction permettant de tester si un individu x est un ancêtre d'un individu y.

```
bool ancetre(Genealogie *g, ident x, ident y);
```

Dans l'exemple si x est Arthur et y est Thomas alors la fonction renvoie vrai. Par contre, Aline n'est pas un ancêtre d'Isabelle. Attention il faut remonter toutes les générations possibles et connues.

4.2 Ecrire une fonction permettant de tester si deux individus ont un ancêtre commun.

```
bool ancetreCommun(Genealogie *g, ident x, ident y);
```

Dans l'exemple si x est Cloe et y est Isabelle alors la fonction renvoie vrai. En revanche, Julien et Isabelle n'ont pas d'ancêtre commun.

4.3 Ecrire une fonction permettant de récupérer l'ancêtre connu le plus ancien d'un individu.

```
ident plus_ancien(Genealogie *g, ident x);
```

Dans l'exemple si x est Cloe la fonction renvoie l'identifiant d'Arthur. Pour Julien c'est Pierre.

4.4 Ecrire une fonction permettant d'afficher toute la parenté d'un individu : parents avec leurs frères et sœurs (oncles et tantes) , grands-parents avec leurs frères et sœurs (grand-oncle/grand-tante), les arrières grands-parents avec leurs frères et sœurs, etc.

```
ident affiche_parente(Genealogie *g, ident x);
```

L'affichage précisera la génération : -1 parents + oncles/tantes, -2 grands-parents + grand-oncles/tantes, -3 arrière-grands-parents, etc. Par exemple si x est Thomas, on affichera :

```
- 1:
 Julien
 Alex
 Sophie
```

```
 Clementine
 Marion
- 2:
 Pierre
 Aline
 Henri
 Jeanne
- 3:
 Marcel
 Clothilde
- 4:
 Arthur
```

4.5 Ecrire une fonction permettant d'afficher toute la descendance d'un individu : enfants, petits-enfants, arrière-petits-enfants, etc. On indiquera la génération.

```
ident affiche_descendance(Genealogie *g, ident x);
```

Par exemple si x est Clothilde, on affichera :

```
- 1:
 Jeanne
- 2:
 Sophie
 Clementine
 Marion
- 3:
 Cloe
 Hugo
 Thomas
 Isabelle
```

5. Fusion d'arbres généalogiques

Ecrire une fonction permettant de fusionner deux arbres généalogiques A1 et A2. Un individu de A1 est considéré comme identique à un individu de A2 si le nom est identique ET la date de naissance. La fusion élimine les doublons : chaque individu n'apparaît qu'une seule fois dans la généalogie fusionnée. Tous les liens de parenté doivent être conservés.

```
void genealogieFusion(Genealogie *gres, Genealogie *a1, Genealogie *a2);
```

On suppose qu'il n'y a pas d'incohérence entre les deux généalogies: si x=y, x dans A1 et y dans A2 alors nécessairement x.pere=0 ou y.pere=0 ou x.pere = y.pere (idem pour la mère).

Remarques

Le TP libre est à déposer – au plus tard -pour le **dimanche 1 mai 2022, 23h59**. Le fichier archive contiendra :

- Les codes sources commentés du programme.
- Un document word/rtf/pdf illustrant et documentant la réalisation (max. 10 pages).

Le projet peut être réalisé en binôme.

La note est fonction de la quantité et qualité ***du travail personnel*** réalisé.