université PARIS-SACLAY

Travaux Pratiques de programmation n°5

Cours de programmation modulaire

—Licence MI/IM/MNSI - Info 3—

Révisions

Cette séance vous permet de faire le point sur vos acquis concernant les notions de `structs`, `énumerations`, et `la surcharge d'opérateurs`. Elle doit servir à identifier s'il y a des aspects que vous ne maîtrisez pas encore, et à les travailler afin qu'ils soient bien maîtrisés avant d'aborder les notions de classes et objets la semaine prochaine.

TP Noté Ce TP sera utilisé pour votre note de contrôle continu, il est à rendre sur ecampus d'ici mardi soir.
Nous rappelons que les notes de partiel et examen ont un coefficient beaucoup plus important que la note de contrôle continu. Faire sérieusement ce TP par soi-même afin de bien maîtriser les notions qu'il contient est la meilleure stratégie à long terme.

▸ Exercice 1. (État civil)

Le but de l'application est d'enregistrer des informations d'état civil sur des personnes. Une personne est décrite par :

- son nom
- son genre (masculin ou féminin)
- son conjoint éventuel (pas forcément d'un genre différent)
- ses enfants, s'il en a
- ses parents, si on connaît cette information.

L'information n'est pas toujours complète : on peut ne pas connaître l'un ou l'autre (ou les deux) parents d'une personne. Le statut marital des personnes peut aussi évoluer dans le temps, de même que le nombre d'enfants.

On utilise pour cela les types ci-dessous : un « état civil » contient des informations sur les personnes dans un vecteur. Pour désigner une personne enregistrée dans le système, on peut utiliser l'indice de l'élément du vecteur qui contient les informations sur la personne. De plus, pour représenter une personne (ou une information) inconnue, on utilisera la valeur `-1` (qui ne peut pas être un indice). Par exemple si `Jean` est enregistré à l'indice `3` et `Marie` est enregistré à l'indice `5`, pour indiquer que `Jean` est marié à `Marie`, on mettra `5` comme valeur du champ `indConjoint` pour `Jean` et `3` comme valeur du même champ pour `Marie`. Si `Jean` n'est pas marié, son champ `indConjoint` contient `-1`.

```
12 enum class Genre { Masc, Fem };
13
14 struct Personne {
15   string nom;
16   Genre genre;
17   // indices dans le champ table de l'état civil de son conjoint et de
18   // ses parents ou -1 si l'information est inconnue
19   int indConjoint, indParent1, indParent2;
20   vector<int> enfants;  // indices des enfants (vecteur eventuellement vide)
21 };
22
23 struct EtatCivil {
24   string titre;     // Titre de l'état civil. Exemple: "Ville de Paris"
25   vector<Personne> table;  // description des personnes de cet état civil
26 };
```

Les questions sont à traiter dans le fichier `EtatCivil.cpp` fourni.

1. Surcharger l'opérateur d'affichage pour pouvoir afficher une variable de type `Genre`.
2. Dans le main, déclarez une variable de type `Genre`, donnez-lui une valeur de votre choix, et affichez-là (pour vérifier votre surcharge de l'opérateur d'affichage).
3. Écrire une procédure qui initialise un état civil : on l'intitule du titre passé en argument et on lui associe un tableau vide. Voici l'en-tête :

   ```
   void initialise(EtatCivil &a, string titre)
   ```

4. Tester la fonction d'initialisation. Pour ceci, tester que le titre a bien été enregistré et que la table des personnes est de taille `0`.
5. Surcharger la fonction d'affichage pour une personne. On devra afficher son nom, son genre, son statut (célibataire ou marié), l'indice du conjoint, pour chaque parent on indiquera son indice s'il est enregistré ou «inconnu» sinon et le nombre des enfants.
6. Écrire une fonction

   ```
   int cherche(const EtatCivil &a, string nom);
   ```

   qui recherche le nom d'une personne dans le système et renvoie son indice si elle le trouve, ou -1 si la personne est inconnue.
7. Écrire des tests pour cette fonction cherche. Pour ceci on pourra utiliser la fonction `creeEtatCivildeTest` fournie qui initialise un état civil de tests. Ne pas oublier de faire des tests positifs où l'on trouve bien la personne, mais aussi des tests négatifs où la personne n'existe pas.
8. Écrire une fonction qui affiche les informations pour une personne dont on fournit le nom.

   ```
   void imprimePersonne(const EtatCivil &a, string nom);
   ```

9. Surcharger l'opérateur d'affichage pour un état civil. On devra afficher son titre et les personnes présente dans la table de l'état civil.
10. Écrire une fonction qui enregistre dans le système une nouvelle personne. Les parents sont inconnus, il n'a pas de conjoint et pas d'enfants.

    ```
    int personne(EtatCivil &a, string nom, Genre s);
    ```

    On interdit toute homonymie dans le système (on ne doit pas avoir deux personnes de même nom) et une personne ne doit pas avoir un nom vide. La valeur de retour est l'indice de la nouvelle personne ou une valeur négative si la personne n'a pas pu être enregistrée.
11. Tester cette fonction avec des tests positifs et négatifs.
12. Écrire une fonction qui enregistre le mariage de deux personnes dont on passe les noms en paramètre. La fonction renvoie `true` si le mariage est possible et `false` sinon. On impose que les deux personnes soient enregistrées et ne soient pas déjà mariées :

    ```
    bool mariage(EtatCivil &a, string lun, string lautre);
    ```

13. Tester cette fonction avec des tests positifs et négatifs.
14. Écrire une fonction qui enregistre la naissance d'une personne. Son en-tête est :

    ```
    bool naissance(EtatCivil &a, string qui, Genre s, string p1, string p2);
    ```

    Les paramètres sont le nom de l'enfant, son genre, les noms des parents ; les parents doivent être enregistrés et être conjoints. Si les conditions ne sont pas remplies l'enfant n'est pas enregistré. La fonction renvoie true ou false selon que la filiation a pu être enregistrée ou non.
15. Tester cette fonction.

▸ Exercice 2. (Généalogie)

Cet exercice est la suite de l'exercice précédent. On travaillera dans le même fichier.

16. Écrire une version qui retourne si une personne est un ancêtre au sens large d'une personne. On pourra se servir d'un vecteur auxiliaire (géré en pile) dans lequel on stockera les indices des personnes dont on doit vérifier si elles sont ou non des ancêtres de la personne concernée, en commençant par ses parents. On rappelle que la méthode `pop_back()` permet de supprimer le dernier élément d'un vecteur. En-tête :

    ```
    bool ascendant(EtatCivil &a, string qui, string ancetre);
    ```

    Tester cette fonction.

17. Écrire une fonction récursive de la fonction précédente.

    Tester cette fonction.

18. Coder une procédure permettant d'afficher, pour un individu donné, son arbre généalogique sous la forme :

    ```
    Individu
        Mère
            Grand-mère maternelle
                ...
            Grand-père maternel
                ...
        Père
            Grand-mère paternelle
                ...
            Grand-père paternel
                ...
    ```

    Voici par exemple l'affichage de l'arbre généalogique de l'individu 9 :

    ```
    Individu 9
       Individu 5
          Individu inconnu
          Individu 1
             Individu inconnu
             Individu inconnu
       Individu 10
          Individu 12
             Individu inconnu
             Individu inconnu
          Individu 11
             Individu inconnu
             Individu inconnu
    ```

    Indication : faire fonction récursive contenant 2 appels récursifs. Démarrer l'écriture de la fonction récursive en pensant au cas d'arrêt puis ensuite réfléchissez à l'appel récursif.