Université d'Orléans — L3 Info
Informatique — 2016 - 2017

Travaux Dirigés sur Machine n°8 — Arbre généralisé et XML en C++ – 3h noté

Ce TDM est divisé en trois parties. La première partie consiste à développer un module C++ pour manipuler des arbres généraux et génériques. La deuxième consiste à mettre en place le code pour manipuler des balises XML (classe Tag). Et la dernière consiste à faire le module xml.

Le langage XML est un langage de balises comme le montre les exemple de la Fig. 1.. Pour simplifier le travail de lecture on suppose que :
— chaque balise (ouvrante ou fermante) est seule sur sa ligne,
— chaque donnée est seule sur une unique ligne, et
— les espaces en début et fin de ligne sont superflus.

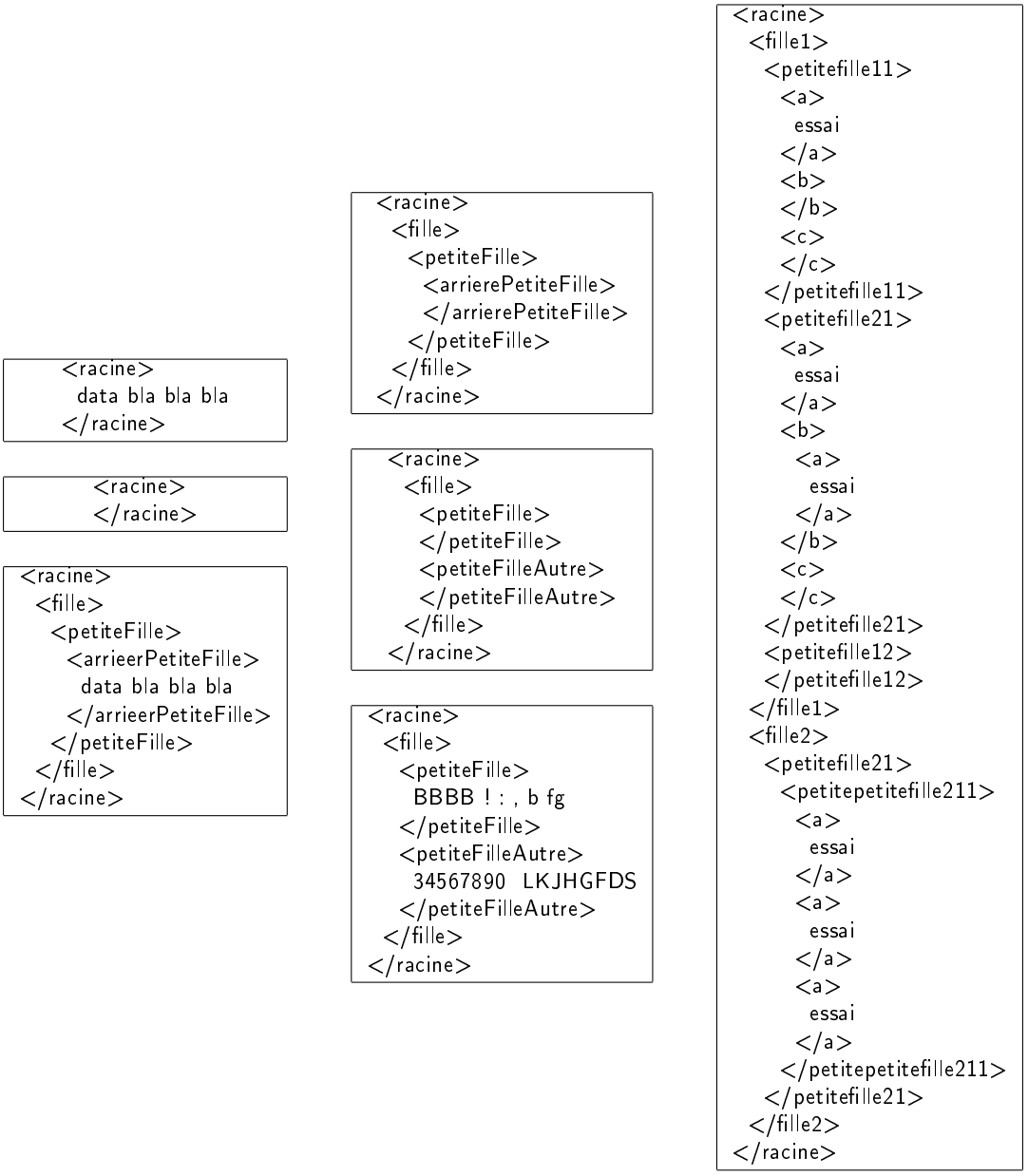

FIGURE 1 – Exemple de code XML simplifié.

Les documents XML ont naturellement une structure d'arbre d'arité quelconque. Dans un premier temps, un module pour les arbres généralisés génériques est à écrire. Ensuite, il s'agit de faire les étiquettes pour les nœuds. Et finalement de faire un module xml avec la lecture et l'affichage d'un document XML.

Exercice 1. Arbres généralisés génériques

On construit une structure de données arbre (tree) dont les nœuds (Node) peuvent contenir n'importe quoi. Cette généricité est à base de `template` (la structure doit donc être entièrement définie dans le fichier tree.hpp).

Un arbre est représenté par des nœuds reliés à leur père, à leur fils (le plus à) gauche et leur frère (immédiatement à) droit comme illustré sur la Fig. 2 . La classe Node sert au chaînage et la classe Tree regroupe la racine et permet d'accéder à tout l'arbre.

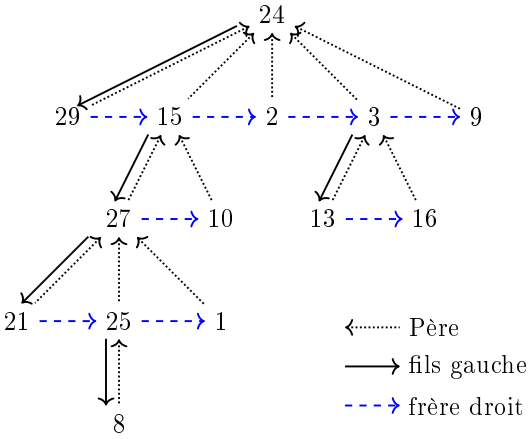

FIGURE 2 – Exemple d'arbre généralisé dont les nœuds contiennent des entiers.

Il faut implémenter les deux classes génériques Node et Tree. Le programme test_tree.cpp vous permet de tester votre implémentation où T est une classe puis un pointeur.

1 Expliquer pourquoi il y a un second paramètre dans le template, à quoi il sert et comment il est utilisé. À quoi correspond le défaut ?

2 À partir de son utilisation dans le code de test, expliquer cet enchevêtrement de *template* et le sens des paramètres. Rédiger la documentation de cette chose.

```
template < class T ,
        void ( * delete_T ) ( T & ) >
template < char const * const open_sons ,
        char const * const sep_brothers ,
        char const * const close_sons >
std :: ostream & Tree < T , delete_T > :: out_put ( std :: ostream & ost
                                                  ) {
```

Exercice 2. Le module tag

Le code est entièrement fourni.

3 Dessiner le diagramme d'héritage des classes du module en précisant bien les champs et méthodes.

Exercice 3. Le module xml

Le module xml propose une classe pour représenter les documents ainsi que des fonctions d'entrée et de sortie.

4 Expliquer comment se passe la création d'un Xml.

La partie compliquée est la lecture. Il n'est pas possible de connaître la nature exacte d'une balise tant que la ligne suivante n'a pas été lue. Il faut raisonner en terme d'automate pour savoir à quel point de la lecture on se trouve, quelle action réaliser et quelles données conserver. Il faut penser au début et donc à l'initialisation du processus.

Attention, c'est assez long et répétitif, il faut vraiment concevoir tous les cas avant de coder quoi que ce soit. Au niveau du code, cela se traduit par un enum et un switch.

5 Dessiner cet automate dans le compte-rendu. Les flèches doivent être étiquetées par la condition pour prendre la transition. Vous devez également représenter l'automate sous forme de tableau à deux dimensions : état × lecture. Chaque case contient alors : les actions à faire et le nom du prochain état.

Diverses primitives et valeurs ont été mises dans le *namespace* anonyme pour simplifier le code. Le programme test_xml.cpp permet de tester l'implantation.