README
Système de gestion de fichiers en ADA

Édouard Lumet <edouard.lumet@etu.enseeiht.fr>

12 février 2018

Sommaire

Première partie

Préambule

Merci de me lire !

Ce document a pour but de décrire les objectifs et l'organisation du projet ainsi que son architecture et les choix réalisés. Le code y sera également expliqué paquetage par paquetage. Dans ce même document seront décrits les problèmes rencontrés ainsi que les solutions apportées. Enfin, un bilan technique puis personnel seront fait afin d'énoncer l'organisation du projet, l'état d'avancement et la durée.

Le projet consiste en la conception d'un système de gestion de fichiers en ADA. Le but premier étant de manipuler un arbre n-aire qui sera la base du SGF, il représente l'arborescence. Ensuite, l'intérêt est de manipuler des paquetages ainsi que de pratiquer la généricité et l'encapsulation.

Deuxième partie

Description des paquetages

Le type arbre

Le type arbre est un type abstrait de données générique privé qui sert de base pour le SGF [1]. Il est nommé *T_arbre*. Du fait de la généricité, il sera nécessaire de l'instancier avec le type souhaité afin de l'utiliser. En effet, les nœuds de l'arbre peuvent contenir n'importe quel type.

Spécifications : *p_arbre.ads*

Les spécifications ne contiennent uniquement les contrats et les signatures des différentes fonctions spécifiques à l'arbre. Ces opérations sont :

- l'initialisation ;
- la création d'un nœud ;
- l'insertion d'un nœud ;
- la suppression d'un nœud ;
- la recherche d'un élément ;
- l'affichage d'un nœud ;
- la vérification si un élément est vide.

Les contrats contiennent toutes les informations nécessaires concernant les fonctions. On y trouve le nom, la sémantique (rôle de la fonction, niveau R0 de raffinage), les paramètres, les pré- et post-conditions ainsi que le type de retour le cas échéant.

Ici, on ne retrouve donc que des opérations basiques étant donné le paquetage qui décrit un type abstrait de données.

Concernant l'implantation de *T_arbre*, c'est un pointeur vers un type *T_nœud* comme nous pouvons le voir ci-dessous :

```
1  TYPE T_arbre IS ACCESS T_noeud ;
2  TYPE T_noeud IS RECORD
3      elt : T_elt ;
4      fils : T_liste ; -- Pointeur sur le premier élément de la liste des fils
5      parent: T_arbre ;
6  END RECORD ;
7  TYPE T_liste IS ACCESS T_cell ; -- Représente la liste des fils d'un noeud
8  TYPE T_cell IS RECORD
9      elt : T_arbre ;
10     suiv: T_liste ;
11 END RECORD ;
```

Modification des spécifications

Suite à des problèmes de visibilité du type *T_arbre*, il a fallu créer des accesseurs et des modificateurs afin de manipuler les objets pointés par ce type privé. Ces fonctions sont Get_Noeud_Fils, Get_Noeud_Elt ou encore Set_Noeud_Elt.

1. Pour rappel, SGF = Système de Gestion de Fichiers

Corps : *p_arbre.adb*

Ici il est question du code à proprement parler des différentes fonctions. Il n'y a rien de particulier à mentionner mise à part l'algorithme de suppression d'un nœud qui peut être intéressant :

```
1  PROCEDURE Supprimer ( Fn : IN OUT T_arbre ) IS
2  BEGIN
3      WHILE Fn /= Null LOOP
4          IF Fn.all.fils.all.suiv = Null THEN
5              IF Fn.all.fils.all.elt = Null THEN -- cas n°1
6                  Fn := Null ;
7              ELSE -- cas n°2
8                  Fn := Fn.all.fils.all.elt ;
9              END IF ;
10         ELSE
11             IF Fn.all.fils.all.elt = Null THEN -- cas n°3
12                 Fn.all.fils := Fn.all.fils.all.suiv ;
13             ELSE -- cas n°4
14                 Supprimer(Fn.all.fils.all.elt) ;
15             END IF ;
16         END IF;
17     END LOOP ;
18 END Supprimer ;
```

Le cas n° 1 décrit le cas le plus simple pour la suppression. Il s'applique lorsque le nœud n'a ni fils ni frère.

Le cas n° 2 s'applique quant à lui lorsque le nœud n'a pas de frère mais a un fils. On descend alors d'un niveau.

Ensuite, le cas n° 3 s'applique lorsque le nœud n'a que des frères. Il s'agit alors d'un simple parcours de liste afin de mettre tous les pointeurs à Null.

Enfin, le cas n° 4 emploie la récursivité sur le fils car c'est le cas où le nœud a à la fois un fils et un frère au moins.

Le SGF (interpréteur)

Le paquetage sgf est un paquetage décrivant toutes les opérations liées au SGF à proprement parler. C'est donc l'interpréteur de commandes qui permet de manipuler le SGF. Il utilise par conséquent le type *T_arbre*. C'est ici que nous instancierons le paquetage *p_arbre* avec le type propre au SGF : *T_ficrep*.

Spécifications : *p_sgf.ads*

A l'instar du type arbre, ces spécifications regroupent les contrats et les signatures des différentes fonctions propres au SGF. Ces opérations sont en fait toutes les commandes supportées par l'interpréteur. On retrouve par exemple la création de fichier, l'affichage du contenu d'un répertoire, le changement de répertoire courant, etc.

Ici, deux choses sont intéressantes à souligner. Premièrement, afin d'utiliser le type *T_arbre*, nous devons faire appel au paquetage correspondant dans le fichier en insérant WITH p_arbre ; tout en haut du fichier ADS. A noter qu'il ne faut pas employer USE. Deuxièmement, intéressons-nous au type *T_ficrep*. Comme nous pouvons le voir

ci-dessous, c'est un simple enregistrement qui sert à renseigner les informations sur les fichiers ou répertoires.

```
-- Type énuméré pour la distinction fichier (f) ou répertoire (r)
TYPE T_enum IS (f,r);

-- Type fichier/répertoire
TYPE T_ficrep IS RECORD
    type: T_enum; -- Permet de distinguer fichier ou répertoire
    nom: Unbounded_String;
    droits: integer;
    taille : integer;
END RECORD;
```

Modification des spécifications

D'importantes modifications ont été apportées sur ces spécifications. En effet, toujours à cause de la visibilité du type *T_arbre*, il a fallu créer des pointeurs sur pointeurs en plus des méthodes d'accès. Apparaît alors le pointeur propre au SGF à l'image du pointeur privé *T_arbre* : *T_sgf*. C'est juste avant la déclaration de ce nouveau pointeur que l'on instancie le paquetage *p_arbre* comme suit :

```
PACKAGE p_arbre_ficrep IS NEW p_arbre(T_ficrep) ;
USE p_arbre_ficrep ;

TYPE T_sgf IS ACCESS T_arbre ;
```

Ensuite, différents types ont été nécessaires soit pour palier le problème de visibilité soit pour répondre aux besoins spécifiques de fonctions qui n'étaient pas prévues au départ (Decouper_Chemin, Extraire_Cmd et Exec_Cmd). Ces fonctions ont pour rôle d'extraire la commande et le chemin de la saisie utilisateur et de reconnaître la commande saisie. Les types ajoutés sont donc des tableaux contenant des chaînes de caractères ou des pointeurs ou encore un type énuméré avec les noms des commandes acceptées. L'utilisation d'un type énuméré permet l'utilisation d'une structure de type case et il permet de gérer facilement las cas non supportés (commande inconnue).

Corps : *p_sgf.adb*

Comme dit plus haut, ici se retrouve le code des différentes fonctions spécifiées. Les plus intéressantes à commenter sont celles qui font l'extraction de commande et de chemin puis la reconnaissance de commande.

Pour commencer, voici l'extraction de commande :

F0 : Extraire la commande de la saisie utilisateur
- F1 : Parcourir le texte jusqu'au premier caractère ' '
- F1 : Placer la commande à l'indice 1 du tableau retourné par la fonction
- F1 : Ajouter l'option de commande dans le tableau et les arguments
  - F2 : Vérifier la présence d'option
  - F2 : Placer l'option à l'indice 2 du tableau le cas échéant
  - F2 : Placer le reste de la chaîne suivant l'option (les arguments) à l'indice 3 du tableau
  - F2 : Remplacer l'option à l'indice 2 par la chaîne vide en l'absence d'option

    - F2 : Placer le reste de la chaîne (les arguments) à l'indice 3
  - F1 : Retourner le tableau représentant la commande après traitement

Pour résumer, en s'assurant toujours de ne jamais dépasser le dernier indice de la chaîne de caractères, on parcourt cette chaîne jusqu'au premier espace (ou jusqu'à la fin) en ajoutant les caractères au fur-et-à-mesure dans une première case du tableau de chaînes qui est retourné. On fait de même pour l'option s'il y en a une et pour le chemin s'il y en a un.

La découpe du chemin quant à elle est plus complexe et s'effectue en deux grandes étapes : la découpe en sous-chaînes en récupérant les noms séparés par des / puis la recherche des pointeurs qui seront retournés dans un tableau de pointeurs.

Voici le raffinage correspondant :

F0 : Découper un chemin absolu ou relatif en sous-chaines et retourner un tableau de pointeurs

- F1 : Vérifier si le chemin est absolu ou relatif (commence par / ou autre)
  - F2 : Placer la chaîne vide en indice 1 du tableau retourné si le chemin est absolu
  - F2 : Ne rien faire sinon
- F1 : Parcourir le reste de la chaîne selon le séparateur '/' jusqu'à la fin de celle-ci
  - F2 : Ajouter les lettres dans une chaîne temporaire jusqu'à rencontrer un '/'
  - F2 : Ajouter le mot fini dans le tableau puis passer à l'index suivant
- F1 : Récupérer les pointeurs pour chaque élément du tableau de sous-chaînes
  - F2 : Récupérer les fils de la racine si le chemin est absolu
  - F2 : Vérifier l'existence des noeuds dans l'arborescence en suivant le chemin saisi
    - F3 : Vérifier pour chaque fils que le nom concorde avec celui saisi et extrait
    - F3 : Récupérer le pointeur si le nom concorde
  - F2 : Récupérer les fils du répertoire (noeud) courant si le chemin est relatif
    - F3 : Vérifier pour chaque fils que le nom concorde avec celui saisi et extrait
    - F3 : Récupérer le pointeur si le nom concorde

Enfin, l'exécution de commandes est finalement très simple. Ayant fait le choix d'un type énuméré *T_cmd*, l'utilisation d'une structure CASE rend la réalisation et la compréhension du code très simples :

```
1  FUNCTION Exec_Cmd ( Fracine : IN T_sgf ; Fcourant : IN T_sgf ; Fsaisie : IN Unbounded_String )
       RETURN T_sgf IS
2      courant : T_sgf ;
3      chemin : T_ptrtabeff ; -- Chemin extrait dont les pointeurs ont été récupérés
4      cmd_complete : T_strtab ; -- Commande extraite
5      cmd : Unbounded_String ; -- Commande seule (sans les options) récupérée sous le forme d'une
       chaîne
6  BEGIN
7      cmd_complete := Extraire_Cmd(Fsaisie) ; -- Extraction de commande
8      cmd := cmd_complete(1) ;
9      chemin := Decouper_Chemin(Fracine, Fcourant, cmd_complete(3)) ; -- Extraction du chemin
10     courant := Fcourant ;
11     CASE T_cmd'value (to_string(cmd)) IS -- Comparaison entre la commande et l'une des valeurs
       du type énuméré
12         WHEN pwd => Rep_Courant(courant) ;
13         WHEN touch => Creer_Fichier(chemin) ;
14         WHEN mkdir => Creer_Repertoire(chemin) ;
```

```
15          WHEN cd => courant := Change_Courant(chemin) ;
16          WHEN ls => Afficher_Contenu(chemin) ;
17          WHEN rm => Supprimer_Ficrep(chemin) ;
18          WHEN OTHERS => Null ;
19      END CASE ;
20      Return courant ;
21  END Exec_Cmd ;
```

Le programme principal (main)

Le programme principal (*main.adb*) est très simple. Il démarre par la création de la racine du SGF. Ensuite, un message de bienvenue est affiché puis une boucle permet l'affichage du prompt afin de saisir les différentes commandes jusqu'à ce que la commande **exit** soit saisie par l'utilisateur.

```
1  PROCEDURE main IS
2
3      -- Variables --
4      Courant : T_sgf ;
5      Racine : T_sgf ;
6      Saisie  : Unbounded_String ;
7
8  BEGIN
9      PUT("Bienvenue") ; NEW_LINE ;
10     PUT("Type␣exit␣to␣quit") ; NEW_LINE ;
11     Creer_Racine(Racine, Courant) ;
12     LOOP
13         NEW_LINE ;
14         PUT("$␣") ;
15         GET_LINE(Saisie) ;
16         Courant := Exec_Cmd(Racine, Courant, Saisie) ;
17     EXIT WHEN to_string(Saisie) = "exit" ;
18     END LOOP ;
19 END main ;
```

Troisième partie

Le projet

Difficultés et solutions

Les difficultés furent nombreuses au cours de ce projet. En effet, mon manque de pratique en programmation s'est fait ressentir, notamment sur des concepts nouveaux tels que les pointeurs, les paquetages, la visibilité ou l'encapsulation et la généricité. Tout autant de concepts qui apportent leur lot de problèmes pour un novice. Ensuite, une autre difficulté tient du raisonnement, de la solution logique, mathématique ou algorithmique à trouver. Ce qui est différent de la méthodologie qui est enseignée dans ce module.

L'encapsulation et la généricité du type *T_arbre* ne permette pas son utilisation hors du paquetage lui-même. Il peut seulement être déclarer lors de la création d'une variable. Pour palier ce problème, j'ai donc fait appel à un pointeur propre au paquetage *p_sgf* qui pointe sur un pointeur de type *T_arbre* et à des accesseurs et modificateurs que l'on appelle méthodes. Cela requiert une vigilance particulière quant aux types manipulés car la plupart sont privés.

Le paquetage générique *p_arbre* et le lien entre ce même paquetage et le paquetage *p_sgf* fut pour moi le plus chronophage. Le temps de comprendre puis d'intégrer la solution afin de l'appliquer a monopolisé la majeure partie de mon projet.

Organisation

Le projet s'est organisé en plusieurs parties :
— rédaction des spécifications (*p_arbre.ads* et *p_sgf.ads*) ;
— rédaction du corps du paquetage *p_arbre.adb* ;
— rédaction du corps du paquetage *p_sgf.adb* ;
— rédaction du programme principal *main.adb*.
— rédaction du manuel utilisateur (HOWTO) et du présent rapport (README)

Cependant à cause des difficultés rencontrées, il a fallu revenir sur la rédaction des spécifications lors de la rédaction des corps de paquetage. Cette dernière a également pris plus de temps au fil des modifications.

Bilan technique et avancement

Résumé des modifications des spécifications

Les modifications réalisées entre le jalon J2 et le jalon J4 se résument ainsi :
— Spécifications *p_arbre.ads* :
  — ajout des méthodes d'accès pour accéder aux nœuds et aux éléments ;
  — remplacement de la fonction Est_Vide par deux fonctions Est_Vide_Arbre et Est_Vide_Liste ;
  — suppression des fonctions de recherche et d'affichage ;
— Spécifications *p_sgf.adb* :
  — ajout des types *T_sgf* (pointeur sur *T_arbre*), *T_strtab* (tableau de chaînes non bornées), *T_cmd* (type énuméré des commandes prises en charges), *T_ptrtabeff* et *T_ptrtab* (tableau de pointeurs), *T_sgflist* (pointeur sur *T_liste*) ;

— ajout des fonctions Extraire_Cmd, Decouper_Chemin et Exec_Cmd afin de gérer les extractions de commande et de chemin ainsi que l'exécution de la commande sans répéter l'opération pour chaque fonction.

Etat d'avancement

Malheureusement le SGF n'est pas fonctionnel. Une erreur sur une méthode d'accès empêche son bon fonctionnement. A cela s'ajoute le manque de développement des fonctions correspondant aux commandes supportées ainsi que la gestion des exceptions et des pré-conditions.

La résolution des problèmes explicités plus haut fut très chronophage. J'estime donc mon avancement à 50%.

Quatrième partie

Bilan personnel

Ce projet fut pour moi très difficile. Comme je l'ai déjà évoqué, mon niveau de compétences en développement et en programmation m'a rapidement fait défaut. Ce fut tout de même intéressant pour appréhender de nouveaux concepts comme les pointeurs, les paquetages, la visibilité ou l'encapsulation et la généricité.

En revanche, le faible nombre d'heures associé au fait d'effectuer ce projet seul a fait que j'étais vite dépassé. Dans ce cas-là il est alors plus difficile d'avoir le recul nécessaire pour progresser et pour comprendre. En effet, encore maintenant il y a des concepts que j'ai du mal à saisir en termes de pratique comme la visibilité ou l'encapsulation.

La tâche la plus chronophage fut la conception et ensuite l'implantation. Je retiens de ce projet que la théorie n'est pas un pré-requis suffisant. La programmation requiert une certaine expérience à l'image du réseau où c'est le troubleshooting qui nous fait progresser, la confrontation aux problèmes. N'ayant jamais rencontré de problèmes liés aux pointeurs et à la visibilité, j'ai dû analyser les causes et les conséquences et tenter d'en tirer les conclusions les plus justes possibles, souvent à tâtons. C'est pour cela que cette phase a pris beaucoup de temps sur le durée globale. Et ce malgré l'entraide en séance de projet. *Je suis parce que nous sommes tous.*