**TRAVAUX PRATIQUES : n°4**

**Implémentation du type abstrait Forêt et Arbre**

**MARQUEZ Lucas & ROBIN Florian**

**I – PROBLEMATIQUE :**

Le but de ce TP est de concevoir une implémentation complète pour le type abstrait Forêt et Arbre en s’appuyant sur la spécification fournie. Cette implémentation va permettre de créer des arbres avec des nœuds afin d’en faire une forêt d’arbres. Grâce à la spécification nous pourrons également tester toutes les propriétés en lien avec les arbres.

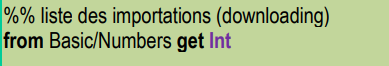
**II – REALISATION :**

Pour chaque implémentation à réaliser il faut procéder en suivant 4 étapes définies de la manière suivante :

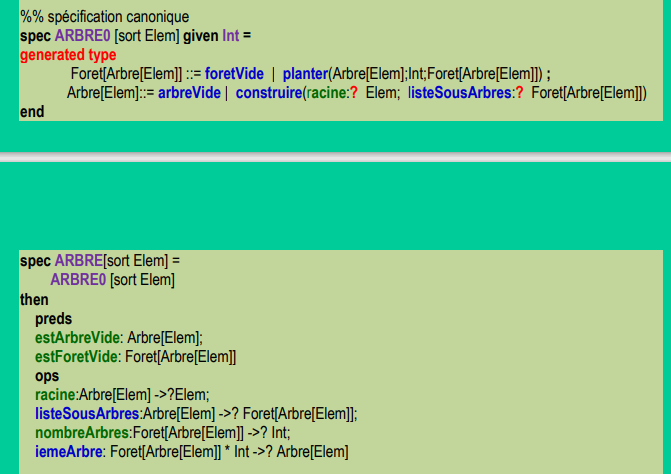
ETAPE 1 : Spécification du type abstrait.

Celle-ci est composée de 3 parties bien distinctes : l’en tête, la signature et la sémantique.

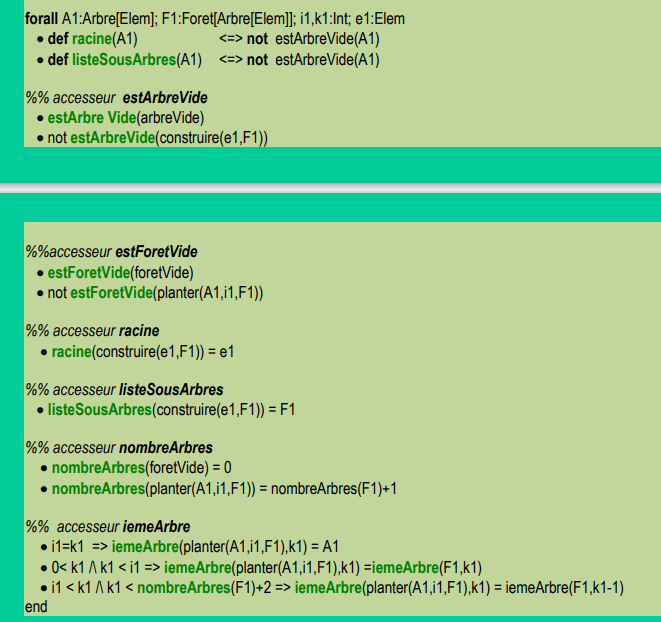
La signature correspond à l’interface de la spécification, c’est-à-dire la partie visible auquel toute personne a accès. Elle permet de déclarer le nom du module de spécification, les types utilisés, les types abstraits à définir ainsi que les opérations du type avec leur signature propre. Ensuite la sémantique est un ensemble de propriétés appelés axiomes qui permettent d’observer, modifier ou de créer l’état d’un objet. Pour finir l’en-tête permet de définir les librairies qui seront utilisés dans la spécification.



**EN-TETE**

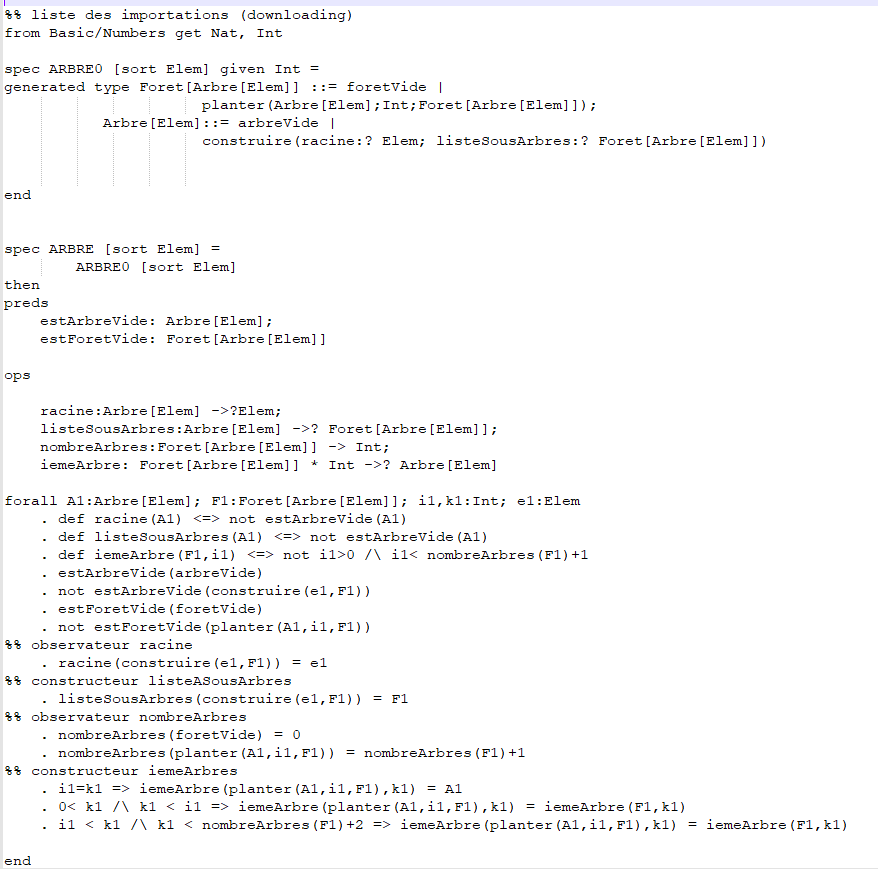


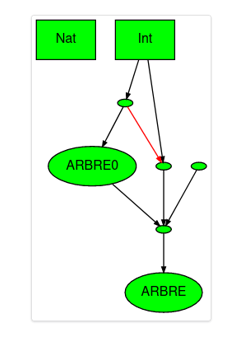
**SIGNATURE**



**SEMANTIQUE**

Il faut ensuite vérifier que la spécification casl est correcte en l’éditant puis en la soumettant a HETS doliator, cet outil permet de valider ou non la spécification.





**On remarque ici que la spécification n’est pas validée et les types Nat et Int ne sont pas correctement définis.**

On peut également définir les mots clefs de la spécification comme ceci :

**Mots clés du CASL** :   
Library : Librairie(s)

From : Depuis

Get : Prendre (« from Basic/Numbers get Rat  » = Importer Rat (rationnel) depuis la librairie définie avant avec Library)

Spec … : Spécification du type abstrait de …

And : Et, permet de définir les types abstraits utilisés

Then : Puis/ensuite

Sort : Type abstrait défini par la spécification

Pred : Prédicat

Ops : (donner la signature des) Opérations (du type)

« : » :  
Forall … : Pour tout …

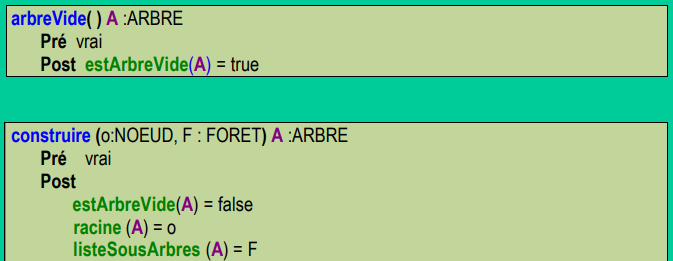
. def … : Axiomes définissant les opérations par leurs propriétés

… ⬄ … : … si et seulement si …

Not … : Non …

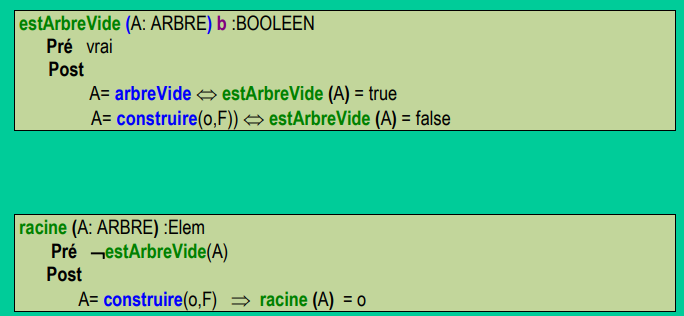
… /\ … : … et … (et logique)

Spec Arbre 0 : la spécification d’Arbre 0 est généré par les constructeurs foretVide, planter, arbreVide, construire et listeSousArbres.

Pour finir nous allons commenter tous les groupes d’axiomes de la spécification 

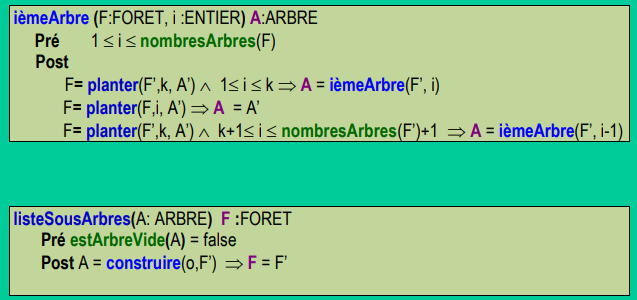
Arbre vide : est un prédicat qui appelle la fonction estArbreVide pour tester si l’arbre est vide ou non.

Construire : teste si l’arbre est non vide et affecte un nœud à la racine de l’arbre A puis affecte une forêt F à la liste de sous arbres.



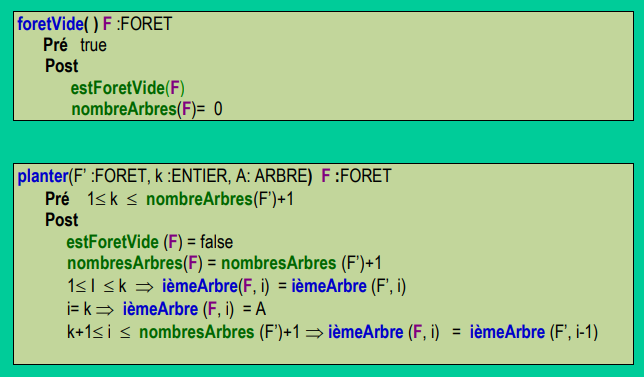
EstArbreVide : un arbre est vide si et seulement si l’accesseur estArbreVide renvoie vraie et si et seulement si l’opération construire n’a pas été appliqué.

Racine : teste si l’arbre est non vide puis applique l’opération construire afin de créer la racine qui est le premier nœud.



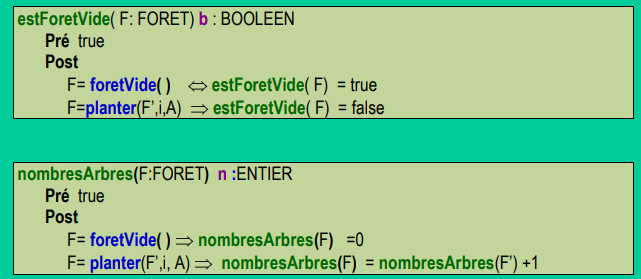
Ieme arbre : si le nombre d’arbres == 1 alors on retourne l’arbre sinon on parcourt tous les arbres de la forêt et on retourne l’arbre qui correspond au rang i.

listeSousArbre : on teste si l’arbre est non vide puis on applique la fonction construire pour créer une nouvelle forêt avec tous les sous arbres qui existent dans cette forêt.



Foret vide : on teste si l’accesseur esForetVide renvoie vraie et si le nombres d’arbres = 0.

Planter : on ajoute un arbre au rang i dans la forêt, on teste si la forêt est non vide, on incrémente le nombre d’arbres. Si le rang vaut 1 on augmente la hauteur de l’arbre et on insère le nouvel arbre. Sinon : on prend l’arbre au rang k-1 et on lui affecte l’arbre de rang k puis on prend l’arbre au rang k et on lui affecte l’arbre au rang k+1. Une fois arrivé au rang i on ajoute l’arbre à la bonne place et on augmente la hauteur de l’arbre.

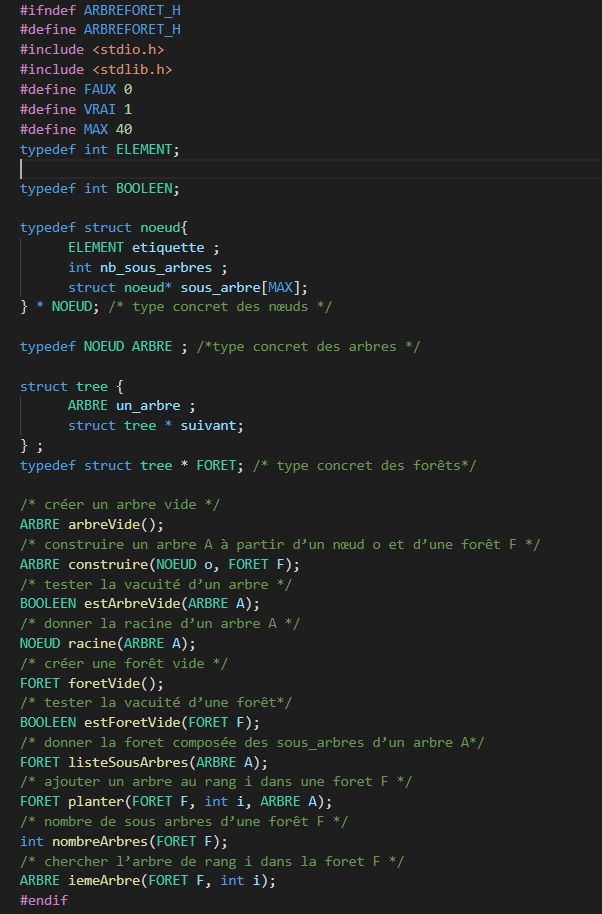


estForetVide : teste si forêt vide si et seulement si l’accesseur estForetVide renvoi vrai, on applique la fonction planter si et seulement si la foret est non vide.

NombresArbres : on teste si la forêt est vide, si elle est vide le nombre d’arbres vaut 0. Puis on applique la fonction planter, celle-ci augmente la hauteur de la forêt en ajoutant un arbre.

ETAPE 3 : Implémentation de la spécification

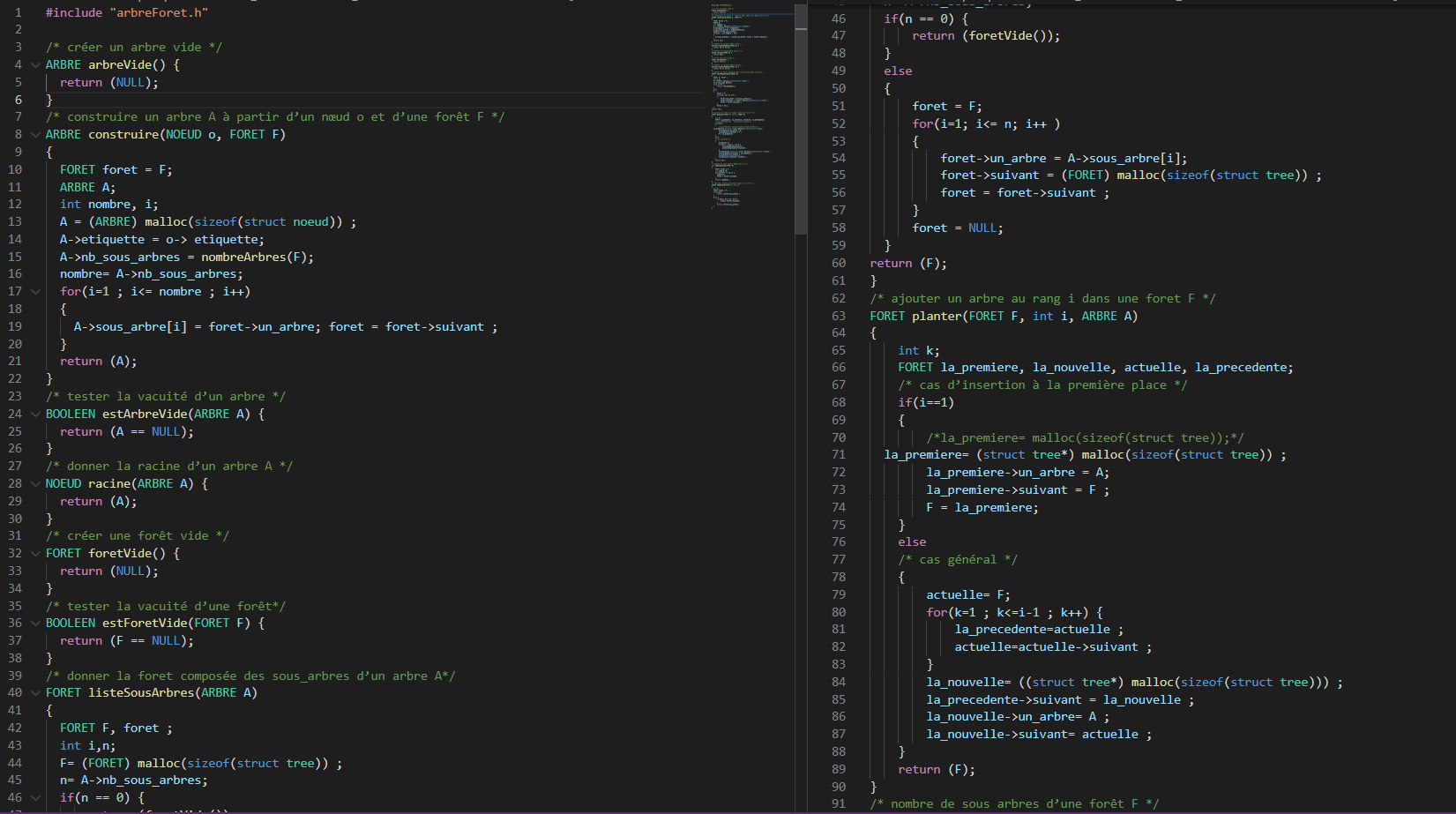
Nous avons donc commencé l’implémentation par le fichier d’interface, le header, dans ce fichier toutes les fonctions seront définies ainsi que le type abstrait arbre et forêt.

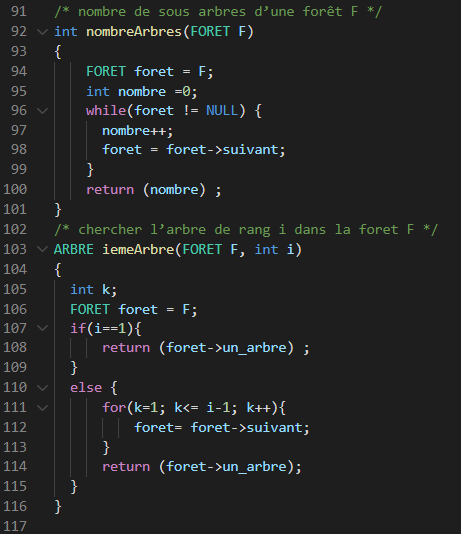


Définition de l’en-tête des fonctions

Définition du type Arbre et Forêt

Ensuite nous avons réalisé le fichier « arbreForet.c » où toutes les opérations sur les fonctions y sont décrites. Ce fichier est non consultable et utilise le header car les en-têtes de fonctions sont déjà déclarés.

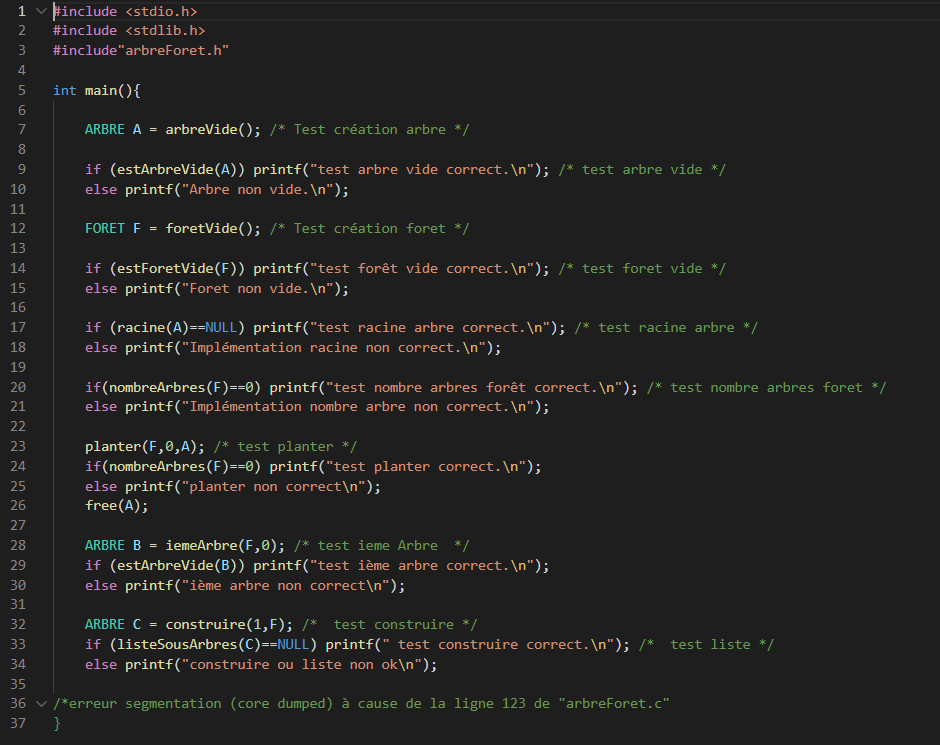




Toutes les opérations sont définies mais comme la spécification n’est pas correcte, l’implémentation de l’opération iemeArbre est également incorrecte, en effet la ligne 108 et 114 renvoient l’erreur « segmentation fault (core dumped) ».

ETAPE 4 : Vérification de l’implémentation

Il suffit de vérifier que toutes les fonctions sont correctement implémentées. Tous les axiomes doivent être satisfaits pour que l’implémentation soit validée. Cette vérification se déroule dans un fichier nommé « preuveArbreForet.c » en cas d’échec le programme s’arrête et renvoie une erreur. Ici au moment d’exécuter la fonction iemeArbre le programme s’arrête à cause d’un mauvais accès mémoire (cf ligne 108 et 114 du screen arbreForet.c ou ligne 123 et 117 si le fichier est ouvert dans un éditeur de code).



**III – CONCLUSION :**

La réalisation de ce tp a été plus compliqué que les autres, la spécification était bien plus complexe qu’auparavant et il semble qu’il y ait des erreurs dans celle-ci ce qui engendre également des erreurs dans l’implémentation. Nous n’avons pas trouvé comment corriger ces erreurs afin que l’implémentation soit correcte mais nous avons tout de même essayé de se rapprocher le plus possible de la solution.