PROJET HUFFMAN

Nicolas CATONI
Alice DEVILDER
Groupe MN03

I. COMPRESSION

Déclaration des types :

— Module Memoire —

Type **T_bit** est mod 2
Type **T_octet** est mod 256

— Module Arbre —

Type **T_noeud**
Type **T_arbre** est POINTEUR sur T_noeud - - Type privé
Type **T_noeud** est ENREGISTREMENT
    freq : Entier;
    octet : T_octet;
    branche_d : T_arbre;
    branche_g : T_arbre;
    Fin_fichier : Booléen; - - Ce booléen indique si le noeud est celui du fin de fichier
Fin ENREGISTREMENT

— Module code —

Type **T_tab_branche** est TABLEAU (0..256) de T_arbre - - Type limited private

Type **T_code** est TABLEAU (1..256) de T_bit

Type **T_trad** est ENREGISTREMENT
    longueur_code : entier;
    valeur : T_code;

Fin ENREGISTREMENT

Type **T_tab_code** est TABLEAU (0..256) de T_trad - - Type limited private

Type **T_parcours_infixe** est TABLEAU (1..(nb_unique_octet)* 2) de T_bit - - Type limited private

Type **T_octet_parcours_infixe** est TABLEAU (0..nb_unique_octet) de T_octet - - Type limited private

Raffinage :

R0 : Compresser un fichier avec le codage de Huffman.

R1 : Comment "Compresser un fichier avec le codage de Huffman" ?

Compression ← Vrai

Interpréter la ligne de commande — nom_fichier : **in out** Chaîne de caractère; Compression : **in** Booléen; Bavard : **out** Booléen

Construire un tableau de fréquence. - - Ce tableau de fréquence correspond à la première branche de l'arbre d'où le nom de sa procédure : "Premiere_branche"

Construire l'arbre de Huffman — tab_branche : **in out** T_tab_branche; arbre : **in out** T_arbre

Construire le tableau de codage de l'arbre

Traduire le contenu du fichier initial (à l'aide du tableau de codage) dans le fichier compressé — nom_fichier : **in** Chaîne de caractères ; nom_fichier_hff : **in** Chaîne de caractères ; tab_code : **in** T_tab_code ; parcours_infixe : **in** T_parcours_infixe ; octet_parcours_infixe : **in** T_octet_parcours_infixe

Afficher les informations liées à la compression

R2 : Comment "Interpréter la ligne de commande" ?

Bavard ← Faux

Selon nombre d'arguments **est**

0 => **Lever** Pas_arguments — Pas_arguments : exception

1 => Rien

2 => **Si** le premier argument est “-b” ou le premier argument est “--bavard” **alors**

Bavard ← Vrai

Sinon

Lever Option_inconnue — Option_inconnue : exception

Fin Si

Autres => **Lever** Trop_arguments — Trop_arguments : exception

Fin Selon

R2 : Comment “Construire un tableau de fréquence” ?

Initialiser la première branche de l’arbre — tab_branche : **out** T_tab_branche

Construire la première branche de l’arbre — nom_fichier : **in** Chaîne de caractères; tab_branche : **in out** T_tab_branche

R2 : Comment “Construire l'arbre de Huffman” ?

zero ← T_octet (0) — zero : T_octet

Initialiser neant — neant : T_arbre

Répéter

- - cf R4 : Comment “Initialiser chaque branche de tab_branche(i)” ? pour Initialiser arbre

Initialiser arbre

Chercher les arbres du tableau possédant les deux plus faibles fréquences — tab_branche : **in** T_tab_branche; min1 : **out** T_arbre ; imin1 : **out** entier ; min2 : **out** T_arbre ; imin2 : **out** entier;

freq ← La fréquence du min1 + la fréquence du min2

- - Affecter à l'arbre la nouvelle fréquence et les min1 et min2 en fils_gauche et fils_droit respectivement

Affecter (arbre, zero, freq, min1, min2)

Initialiser l’arbre du tableau tab_branche d’indice imin2

- - Affecter à l’arbre du tableau tab_branche d’indice imin2 un caractère nulle et une fréquence de taille_fichier + 1

Affecter (tab_branche(imin2), zero, taille_fichier + 1, neant, neant)

tab_branche (imin1) ← arbre;

Jusqu’à freq >= taille_fichier

Fin Répéter

R2 : Comment “Construire le tableau de codage de l'arbre” ?

Initialiser le tableau de code — tab_code : **in out** T_tab_code

Construire la table de codage de Huffman suivant un parcours infixe de l'arbre — arbre : **in** T_arbre ;
tab_code : **out** T_tab_code ;
code_huffman : **in out** T_code ;
code_huffman_longueur : **in out** entier
parcours_infixe : **out** T_parcours_infixe ;
octet_parcours_infixe : **out** T_octet_parcours_infixe ;
indice_parcours_infixe : **in out** entier ;
indice_octet_parcours_infixe : **in out** entier ;

R2 : Comment “Traduire le contenu du fichier initial (à l'aide du tableau de codage) dans le fichier compressé” ?

c ← 0 — c : entier
Créer le fichier compressé “nom_fichier_hff”
Ecrire les octets du parcours infixe dont le premier correspond à l’indice du caractère de fin de fichier
tmp_octet : T_octet
- - Ecrire l’octet de délimitation entre les octets du parcours infixe et le parcours infixe
T_octet'Write(S_out, tmp_octet)
c := c + 1
Ecrire le parcours infixe
Ouvrir le fichier “nom_fichier”
Ecrire les codes de Huffman
Fermer le fichier “nom_fichier”
Ecrire le code de huffman de l’octet de fin de fichier
Fermer le fichier compressé “nom_fichier_hff”

R2 : Comment “Afficher les informations liées à la compression” ?

Si Bavard **alors**
Ecrire ("Arbre de Huffman : ")
- - Procédure Afficher_arbreH
Afficher l’arbre de Huffman
Ecrire ("Tableau d'encodage des caractères : ")
- - Procédure Afficher_tableH

Afficher la table de Huffman

Fin Si

R3 : Comment “Ecrire les octets du parcours infixe dont le premier correspond à l’indice du caractère de fin de fichier” ?

Pour i de 0 au dernier indice de octet_parcours_infixe **Faire**

- - La variable tmp_octet permet d’enregistrer les octets du parcours infixe pour pouvoir les écrire dans le fichier compressé

tmp_octet ← T_octet (octet_parcours_infixe(i))

- - Ecrire tmp dans le fichier compressé

T_octet'Write(S_out, tmp_octet)

c ← c + 1

Fin Pour

R3 : Comment “Ecrire le parcours infixe” ?

len ← 0 len : entier

Pour i de 1 au dernier indice de parcours_infixe **Faire**

- - La variable tmp permet d’enregistrer les bits en octet pour pouvoir écrire le parcours infixe dans le fichier compressé

- - Remplir un buffer jusqu’à ce qu’il soit plein (octet)

tmp(len + 1) ← T_bit (parcours_infixe(i))

Si len = 7 **alors**

- - Ecrire tmp dans le fichier compressé

T_octet'Write(S_out, To_octet(tmp))

c ← c + 1

Fin Si

len ← (len + 1) mod 8

Fin Pour

R3 : Comment “Ecrire les codes de Huffman” ?

Tant que ce n’est pas la fin du fichier **Faire**

octet ← T_octet'Input(S_in)

code_huffman ← tab_code(Integer'Val(octet) + 1).valeur

code_huffman_longueur ← tab_code(Integer'Val(octet) + 1).longueur_code

Pour j de 1 à code_huffman_longueur **Faire**

- - La variable tmp permet d’écrire les codes dans le fichier compressé

```
                - - Remplir un buffer jusqu'à ce qu'il soit plein (octet)
                tmp(len + 1) ← code_huffman(j)
                Si len = 7 alors
                        - - Ecrire tmp dans le fichier compressé
                        T_octet'Write(S_out, To_octet(tmp))
                        c := c + 1
                Fin Si
                len ← (len + 1) mod 8;
        Fin Pour
Fin Tant que
```

R3 : Comment "Ecrire le code de huffman de l'octet de fin de fichier" ?

```
bits_Fin_fichier ← tab_code(0).valeur                    bits_Fin_fichier : T_code
Pour i de 1 à la longueur du code de fin fichier Faire
        - - La variable tmp permet d'écrire les bits de code fin de fichier sous forme d'octet afin de pouvoir écrire le code dans le fichier compressé
        - - Remplir un buffer jusqu'à ce qu'il soit plein (octet)
        tmp(len + 1) ← bits_fin_fichier(i)
        Si len = 7 alors
                - - Ecrire tmp dans le fichier compressé
                T_octet'Write(S_out, To_octet(tmp))
                c ← c + 1
        Fin Si
        len ← (len + 1) mod 8
Fin Pour
T_octet'Write(S_out, To_octet(tmp));
```

R3 : Comment "Initialiser le tableau de code" ?

```
Pour i de 1 à 256 Faire
-- Mettre chaque éléments du tableau à 0
        tab_code(i).longueur_code  ← 0
Fin Pour
```

R3 : Comment "Construire la table de codage de Huffman suivant un parcours infixe de l'arbre" ?

- - Code du caractère

code_huffman_n ← code_huffman code_huffman_n : T_code

- - Longueur du code du caractère

code_huffman_longueur_n ← code_huffman_longueur

code_huffman_longueur_n : entier

- - La coparcours_infixee de l'arbre

cp_arbre ← arbre cp_arbre : T_arbre

- - La variable indice_parcours_infixe correspond à l'indice du parcours infixe

indice_parcours_infixe ← indice_parcours_infixe + 1

- - Pour savoir si l'arbre est terminal, nous avons créé une fonction "Terminal (arbre : T_arbre)" qui renvoie un booléen (dans le module Arbre)

Si l'arbre est Terminal **alors**

- - La variable indice_octet_parcours_infixe correspond à l'indice du octet du parcours infixe

indice_octet_parcours_infixe ← indice_octet_parcours_infixe + 1;

- - Pour savoir si l'arbre contient le caractère de fin de fichier, nous avons créé une fonction "Est_Fin_fichier (arbre : T_arbre)" qui renvoie le booléen Fin_fichier de l'arbre (dans le module Arbre)

Si l'arbre contient le caractère de fin de fichier **alors**

La valeur de tab_code(0) ← code_huffman_n

La longueur du code de tab_code(0) ← code_huffman_longueur

octet_parcours_infixe(0) ← T_octet(indice_octet_parcours_infixe)

indice_octet_parcours_infixe ← indice_octet_parcours_infixe - 1

Sinon

- - La valeur de tab_code d'indice correspondant à l'octet de l'arbre plus 1 prend code_huffman_n

tab_code (Integer'Val(Le_octet(arbre)) + 1).valeur ← code_huffman_n

- - La longueur du code de tab_code d'indice correspondant à l'octet de l'arbre plus 1 prend code_huffman_longueur_n

tab_code (Integer'Val(Le_octet(arbre)) + 1).longueur_code ← code_huffman_longueur_n

octet_parcours_infixe (indice_octet_parcours_infixe) ← T_octet(Le_octet(arbre));

Fin SI

Sinon

code_huffman_longueur_n ← code_huffman_longueur_n + 1

parcours_infixe (indice_parcours_infixe) ← T_bit (0)

code_huffman_n (code_huffman_longueur_n) ← 0

- - Appelle récursif de "Tableau_code" sur la branche gauche de l'arbre et les code_huffman_n et code_huffman_longueur_n

Tableau_code (Branche_g (cp_arbre), tab_code, code_huffman_n, code_huffman_longueur_n, parcours_infixe, octet_parcours_infixe, indice_parcours_infixe, indice_octet_parcours_infixe)

code_huffman_n (code_huffman_longueur_n) ← 1

parcours_infixe(indice_parcours_infixe) ← T_bit (1)

- - Appelle récursif de “Tableau_code” sur la branche gauche de l’arbre et les code_huffman_n et code_huffman_longueur_n

Tableau_code(Branche_d (cp_arbre), tab_code, code_huffman_n, code_huffman_longueur_n, parcours_infixe, octet_parcours_infixe, indice_parcours_infixe, indice_octet_parcours_infixe)

Fin SI

R3 : Comment “Initialiser la première branche de l’arbre” ?

Initialiser neant neant : T_arbre

Pour i de 0 à 256 **Faire**

- - Tab_branche est un tableau de T_arbre donc il s’agit d’initialiser chaque arbre de ce tableau

Initialiser chaque branche de tab_branche(i)

Fin Pour

Affecter à la première case de tab_branche le caractère de fin de fichier et sa fréquence

- - Le booléen Fin_fichierdu noeud de tab_branche(0) devient Vrai

Faire devenir tab_branche(0) la fin de fichier

R3 : Comment “Construire la première branche de l’arbre” ?

Lire le fichier nom_fichier

Tant que ce n’est pas la fin du fichier **Faire**

- - Incrémenter de 1 la fréquence associée à un octet présent dans le fichier (à l’aide de la procédure Affecter, cf R4 : Comment “Affecter …”)

Affecter (tab_branche(Integer'Val(octet)), T_octet(octet), La_freq(tab_branche(Integer'Val(octet))) + 1, neant, neant);

Fin Tant que

Fermer le fichier

- - Considérer les octets non présents dans le fichier

Pour i de 1 à 256 **Faire**

-- Affecter à chaque arbre dont la fréquence est nulle une fréquence égale à taille_fichier, celle-ci étant une variable générique

Si La fréquence de tab_branche(i) est nulle **alors**

Affecter(tab_branche(i), T_octet(Le_octet(tab_branche(i))), taille_fichier + 1, neant, neant);

Fin Si

Fin Pour

R3 : Comment “Chercher les arbres du tableau possédant les deux plus faibles fréquences” ?

-- Initialiser les arbres correspondant aux minima de fréquence

Initialiser neant neant : T_arbre

Initialiser min1

Initialiser min2

-- Remplir les arbres min1 et min2

Affecter (min1, T_octet(0), taille_fichier + 1, neant, neant);

Affecter (min2, T_octet(0), taille_fichier + 1, neant, neant);

Pour i de 1 à 256 **Faire**

-- Si la fréquence du ième arbre du tableau est inférieure ou égale à celle de l'arbre min1, alors on affecte à min2 le 2e minimum soit min1 et à min1 le plus petit des minima soit le ième arbre du tableau

Si la fréquence de tab_branche(i) <= la fréquence de min1 **alors**

min2 ← min1

imin2 ← imin1

min1 ← tab_branche(i)

imin1 ← i

-- Si la fréquence du ième arbre du tableau est inférieure ou égale à celle de l'arbre min2 et supérieure à celle de l'arbre min1, alors on affecte à min2 le ième arbre du tableau

Sinon Si la fréquence de tab_branche(i) <= la fréquence de min2 **alors**

min2 ← tab_branche(i)

imin2 ← i

Fin Si

Fin Pour

R4 : Comment “Initialiser chaque branche de tab_branche(i)” ?

- - Nous devons écrire une procédure pour initialiser les arbres car le type T_arbre est privé

arbre ← Null arbre : **out** T_arbre

R4 : Comment “Affecter à la première case de tab_branche le caractère de fin de fichier et sa fréquence” ?

- - Comme T_arbre est privé nous devons créer une procédure pour modifier les variables de ce type

Si arbre est vide **alors**

arbre ← New T_noeud

Fin Si

- - On affecte l’octet correspondant au caractère ‘~’ de la table ASCII

arbre.all.octet ← T_octet(126)

arbre.all.freq ← 0

arbre.all.branche_g ← neant

arbre.all.branche_d ← neant

arbre.all.Fin_fichier← Faux

Exception :

Pas_arguments => **Si** Compression **alors**
Ecrire ("Usage : ./compresser nom_fichier")
Sinon
Ecrire ("Usage : ./decompresser nom_fichier")
Fin Si
Option_inconnue => **Ecrire** ("L'option est inconnue")
Trop_arguments => **Écrire** ("Il y a trop d'arguments. Il faut en mettre au plus 2.")
NAME_ERROR => **Écrire** ("Ce fichier est inexistant")

II. DÉCOMPRESSION

Déclaration des types :

— Module Memoire —

Type **T_bit** est mod 2
Type **T_octet** est mod 256

— Module Arbre —

Type **T_noeud**
Type **T_arbre** est POINTEUR sur T_noeud - - Type privé
Type **T_noeud** est ENREGISTREMENT
freq : Entier;
octet : T_octet;
branche_d : T_arbre;
branche_g : T_arbre;
Fin_fichier : Booléen; - - Ce booléen indique si le noeud est celui du fin de fichier
Fin ENREGISTREMENT

— Module Decode —

Type **T_parcours_infixe** est TABLEAU (1..longueur_parcours_infixe + 1) de T_bit
Type **T_tab_feuilles** est TABLEAU (1..longueur_tab_feuilles) de T_octet

Raffinages :

R0 : Décompresser un fichier encodé avec le codage de Huffman.

R1 : Comment “Décompresser un fichier encodé avec le codage de Huffman” ?

Interpréter la ligne de commande

Obtenir le nombre de caractères uniques, la longueur du parcours infixe et l’octet de fin de fichier du fichier encodé

nom_fichier : **in** Chaîne de caractères
nb_unique_octet : **out** entier ;
longueur_parcours_infixe : **out** entier ;
octet_Fin_fichier : **out** T_octet ;

Obtenir le tableau de chaque caractère présent dans le fichier et le parcours infixe

nom_fichier : **in** Chaîne de caractères;
parcours_infixe : **out** T_parcours_infixe ;
tab_feuilles : **out** T_tab_feuilles ;

Reconstruire l’arbre de Huffman

parcours_infixe : **in** T_parcours_infixe;
tab_feuilles : **in** T_tab_feuilles ;
octet_ff : **in** T_octet ;
arbre : **out** T_arbre;
indice : **in out** entier ;
indice_feuille : **in out** entier ;
recherche_Fin_fichier : **in out** booléen:

Décoder le fichier compressé

nom_fichier : **in** Chaîne de caractère;
nom_fichier_dec : **in** Chaîne de caractère;
arbre : **in** T_arbre;

Afficher les informations liées à la décompression

R2 : Comment “Interpréter la ligne de commande” ?

```
Bavard ← Faux
Selon nombre d'arguments est
    0  =>  Lever Pas_arguments                    Pas_arguments : exception
    1  =>  Rien
    2  =>  Si le premier argument est “-b” ou le premier argument est “--bavard” alors
               Bavard ← Vrai
           Sinon
               Lever Option_inconnue             Option_inconnue : exception
           Fin Si
    Autres  =>  Lever Trop_arguments             Trop_arguments : exception
Fin Selon
```

R2 : Comment “Obtenir le nombre de caractères uniques, la longueur du parcours infixe et l’octet de fin de fichier du fichier encodé” ?

```
nb_unique_octet ← 0
longueur_parcours_infixe ← 0

Ouvrir le fichier “nom_fichier”
- - Affecter à octet_Fin_fichier la position du caractère de fin de fichier
octet_Fin_fichier ← T_octet'Input(S)
octet_Fin_fichier ← T_octet(integer'val(octet_Fin_fichier))
octet ← T_octet'Input(S)
Obtenir les caractères uniques du fichier encodé          - - ce qui correspond aux feuilles
Obtenir la longueur du parcours infixe
Fermer le fichier “nom_fichier”
```

R3 : Comment “Obtenir les caractères uniques du fichier encodé” ?

```
Répéter
    precedent_octet ←  octet
    - - Affecter à octet les octets lus dans le fichier
    octet ← T_octet'Input(S)
    nb_unique_octet ← nb_unique_octet + 1
Jusqu’à ce qu’on arrive à la fin de fichier ou precedent_octet = octet
```

Fin Répéter

R2 : Comment “Obtenir le tableau de chaque caractère présent dans le fichier et le parcours infixe” ?

Ouvrir le fichier “nom_fichier”

Affecter à tab_feuilles les octets présents dans le fichier “nom_fichier”

Obtenir le parcours infixe écrit dans le fichier “nom_fichier”

Obtenir les bits restants dans le fichier

Fermer le fichier “nom_fichier”

R2 : Comment “Reconstruire l’arbre de Huffman” ?

Initialiser neant neant : T_arbre

Initialiser arbre

- - Parcourir le tableau parcours_infixe afin de trouver le caractère de fin de fichier

Si l’entier correspondant à parcours_infixe(indice) = 1 **alors**

SI indice_feuille = Integer'Val(octet_ff) et recherche_Fin_fichier **alors**

Devient_ff(arbre)

recherche_Fin_fichier ← Faux

Sinon

Affecter (arbre, tab_feuilles(indice_feuille), 0, neant, neant)

indice_feuillle ← feuille + 1

Fin Si

Sinon

indice ← indice + 1

branche_gauche ← Branche_g (arbre) branche_gauche : T_arbre

- - Appelle récursif de ReconstruireH avec branche_gauche

ReconstruireH (parcours_infixe, tab_feuilles, octet_ff, branche_gauche, indice, indice_feuille, recherche_Fin_fichier)

indice ← i + 1

branche_droite ← Branche_d (arbre) branche_droite : T_arbre

- - Appelle récursif de ReconstruireH avec branche_gauche

ReconstruireH (parcours_infixe, tab_feuilles, octet_ff, branche_droite, indice, indice_feuille, recherche_Fin_fichier)

Affecter (arbre, T_octet(0), 0, branche_gauche, branche_droite)

Fin Si

R2 : Comment “Décoder le fichier compressé” ?

noeud_courant ← arbre

Créer le fichier “nom_fichier_dec” qui est le fichier décompressé de “nom_fichier”

Ouvrir le fichier “nom_fichier”

Ignorer les octets de 1 à longueur_tab_feuilles + longueur_parcours_infixe / 8 + 2

Ecrire l’octet encodé par le codage de Huffman

Ecrire les octets restants

Fermer le fichier “nom_fichier”

Fermer le fichier “nom_fichier_dec”

R2 : Comment “Afficher les informations liées à la décompression” ?

Si Bavard **alors**

Ecrire ("Arbre de Huffman : ")

- - Procédure Afficher_arbreH

Afficher l’arbre de Huffman

Fin Si

R3 : Comment “Obtenir la longueur du parcours infixe” ?

nb_un ← 0 nb_un : entier

i ← 0 i : entier

Répéter

- - Affecter à octet les bits lus dans le fichier

bits ← To_bits(T_octet'Input(S))

Répéter

- - Compter le nombre de un dans le parcours infixe

Si bits(i) = 1 **alors**

nb_un ← nb_un + 1

Fin Si

longueur_parcours_infixe ← longueur_parcours_infixe + 1

i ← i + 1

Jusqu’à i = 9 ou nb_un = nb_unique_octet

Fin Répéter

i ← 1

Jusqu’à ce qu’on arrive à la fin du fichier ou nb_un = nb_unique_octet

Fin Répéter

R3 : Comment “Ignorer les octets de 1 à longueur_tab_feuilles + longueur_parcours_infixe / 8 + 2” ?

Pour i de 1..longueur_tab_feuilles + longueur_parcours_infixe / 8 + 2 **Faire**

- - Affecter à octet les octets lus dans le fichier

```
        octet ← T_octet'Input(S_in)
    Fin Pour
```

R3 : Comment “Ecrire l’octet encodé par le codage de Huffman” ?

```
    - - Affecter à bits les bits lus dans le fichier
    bits ← To_bits(T_octet'Input(S_in))
    - - Parcourir l’arbre suivant les bits lus dans le fichier
    Pour i de longueur_parcours_infixe mod 8 + 1..8 Faire
        - - Si le noeud courant est une feuille, écrire l’octet
        Si Terminal (noeud_courant) alors
            octet ← Le_octet (noeud_courant)
            - - Ecrire l’octet dans le fichier décompressé
            T_octet'Write(S_out, octet);
            noeud_courant ← arbre
        Fin Si
        - - Continuer le parcours de l’arbre
        Si bits(i) = 0 alors
            noeud_courant ← Branche_g(noeud_courant)
        Sinon
            noeud_courant ← Branche_d(noeud_courant)
        Fin Si
    Fin Pour
```

R3 : Comment “Ecrire les octets restants” ?

```
    Répéter
        bits ←To_bits(T_octet'Input(S_in))
        Pour i de 1..8 Faire
            Si Terminal (noeud_courant) alors
                Si Est_Fin_fichier (noeud_courant) alors
                    fin_fichier ← Vrai
                Sinon Si ce n’est pas la fin de fichier alors
                    octet ← Le_octet (noeud_courant)
                    - - Ecrire l’octet dans le fichier décompressé
                    T_octet'Write(S_out, octet)
                Fin Si
                noeud_courant ←  arbre
            Fin Si
```

```
                Si bits(i) = 0 alors
                        noeud_courant ← Branche_g (noeud_courant)
                Sinon
                        noeud_courant ← Branche_d (noeud_courant)
                Fin Si
        Fin Pour
    Jusqu'à ce qu'on arrive à la fin du fichier ou fin_fichier
    Fin Répéter
```

R3 : Comment "Affecter à tab_feuilles les octets présents dans le fichier "nom_fichier"" ?

```
    - - Affecter à octet les octets lus dans le fichier
    octet ← T_octet'Input(S)
    Pour i de 1..longueur_tab_feuilles Faire
            - - Affecter à octet les octets lus dans le fichier
            octet ← T_octet'Input(S)
            tab_feuilles(i) ← octet
    Fin Pour
```

R3 : Comment "Obtenir le parcours infixe écrit dans le fichier "nom_fichier"" ?

```
    - - Affecter à octet les octets lus dans le fichier
    octet ← T_octet'Input(S)
    Pour i de 0..(longueur_parcours_infixe / 8) - 1 Faire
            bits ← To_bits(T_octet'Input(S))
            Pour j de 1..8 Faire
                    parcours_infixe(i * 8 + j) ← bits(j)
            Fin Pour
    Fin Pour
```

R3 : Comment "Obtenir les bits restants dans le fichier" ?

```
    - - Affecter à octet les octets lus dans le fichier
    bits ← To_bits(T_octet'Input(S))
    Pour i ide longueur_parcours_infixe - longueur_parcours_infixe mod 8 + 1 à longueur_parcours_infixe Faire
            parcours_infixe(i) ← bits(i + 1 - longueur_parcours_infixe + longueur_parcours_infixe mod 8 - 1)
    Fin Pour
    parcours_infixe(longueur_parcours_infixe + 1) ← 1
```

Exception :

Pas_arguments => **Si** Compression **alors**

Ecrire ("Usage : ./compresser nom_fichier")

Sinon

Ecrire ("Usage : ./decompresser nom_fichier")

Fin Si

Option_inconnue => **Ecrire** ("L'option est inconnue")

Trop_arguments => **Écrire** ("Il y a trop d'arguments. Il faut en mettre au plus 2.")

NAME_ERROR => **Écrire** ("Ce fichier est inexistant")

Evaluation par les étudiants

		Evaluation (I/P/A/+)
Forme (D-21)	Respect de la syntaxe Ri : Comment "... une action complexe ..." ? des actions combinées avec des structures de controle Rj : ...	+
	Verbes à l'infinitif pour les actions complexes	+
	Noms ou équivalent pour expressions complexes	+
	Tous les Ri sont écrits contre la marge et espacés	+
	Les flots de données sont définis	A
	Une seule décision ou répétition par raffinage	A
	Pas trop d'actions dans un raffinage (moins de 5 ou 6)	P
	Bonne présentation des structures de contrôle	+
Fond (D21-D22)	Le vocabulaire est précis	+
	Le raffinage d'une action décrit complètement cette action	A
	Le raffinage d'une action ne décrit que cette action	A
	Les flots de données sont cohérents	+
	Pas de structure de contrôle déguisée	A
	Qualité des actions complexes	A