Types de données

Module Arbre

```
type T_Arbre est pointeur sur Noeud;

type Noeud est enregistrement
        Cle : Entier
        Valeur : T_Octet
        Gauche : T_Arbre
        Droite : T_Arbre
fin enregistrement
```

Module Liste Chaînée

```
type T_Liste_Chainee est pointeur T_Cellue

type T_Cellule est enregistrement
        Valeur : T_Octet
fin enregistrement
```

Module Flux Binaire

```
type T_Bit est modulo 2

type T_Octet est modulo 256

module Liste_Octet est nouveau Liste_Chainee (T_Valeur => T_Octet)

type T_Flux_Binaire est enregistrement
        Nombre_Bits_Inutilises : Entier
        Liste : Liste_Octet.T_Liste_Chainee
fin enregistrement
```

Module Types Huffman

```
Symbole_Fin : constant T_Octet := T_Octet'Last;
```

```
Fichier_Inexistant_Exception : Exception;
Arbre_Huffman_Vide : Exception;
Pas_Compression_Huffman : Exception;

module Arbre_Integer_Octet est nouveau Arbre (T_Cle => Entier, T_Valeur => T_Octet)

module Liste_Chainee_Octet est nouveau Liste_Chainee (T_Valeur => T_Octet)

sous-type Intervalle_Ascii est T_Octet entre T_Octet'First .. T_Octet'Last

type T_Frequences est tableau de (Intervalle_Ascii) de Entier

type T_Liste est tableau de (1..257) de  Arbre_Integer_Octet.T_Arbre

type T_Tableau est enregistrement
        Tableau : T_Liste
        Taille : Entier
fin enregistrement

type T_Codage est tableau (Intervalle_Ascii) de T_FluxBinaire;

type T_Flux_Fichier est enregistrement
        Flux_Interne : Stream_Access
        Octet : T_Octet
        Indice_Bit : Entier
fin enregistrement
```

Raffinage de Compresser

R0 : Compresser un fichier txt avec l'arbre d'Huffman

Exemples :

./compresser exemple.txt => exemple.txt.hff

./compresser -b exemple.txt => afficher les fréquences des caractères
afficher l'arbre d'Huffman
afficher le code des caractères
exemple.txt.hff

./compresser –bavard exemple.txt => afficher les fréquences des caractères
afficher l'arbre d'Huffman
afficher le code des caractères
exemple.txt.hff

./compresser exmple.txt => "Fichier à compresser inexistant."

./compresser -a exemple.txt => "Option entrée non reconnue."

./compresser => "Vous n'avez pas à entré d'arguments."

./compresser a b c => "Vous avez entré trop d'arguments"

R1 : Comment "Compresser un fichier txt avec l'arbre d'Huffman" ?

Déterminer Nom_Fichier avec les arguments de la ligne de commande
Affichage : out Booléen,
Nom_Fichier : out Chaîne_caractère

Déterminer les fréquences des caractères du texte
Nom_Fichier : in,
Tableau_frequence : out T_Frequences

Créer arbre
Tableau_Frequence : in ;
List_Char : out T_Arbre

Déterminer les codes des caractères
List_Char : in ;
Codage : out T_Codage

Encoder l'arbre de Huffman
List_Char : in,
Arbre_Encode : out T_FluxBinaire

Encoder le fichier grâce à l'arbre
Nom_Fichier : in Chaine, Codage : in
Arbre_Encode : in

Afficher les différentes étapes
Affichage : in, Tableau_frequence : in,
List_Char : in, Codage : in

R2 : Comment "Déterminer Nom_Fichier avec les arguments de la ligne de commande" ?

```
Affichage <- False
SI Argument_Count > 2 ALORS
    LEVER Trop_Arguments_Exception
SINON SI Argument_Count = 1 ALORS
    Nom_Fichier <- Argument(1)
SINON SI Argument_Count = 0 ALORS
```

```
        LEVER Peu_Arguements_Exception
    SINON
        Lire le double argument      Affichage  : out Booléen,
                                     Nom_Fichier : out Chaîne_caractère
    FIN SI
```

R2 : Comment “Déterminer les fréquences des caractères du texte” ?

```
    Initialiser les cases de Tableau_Frequence à 0      Tableau_Frequence : out
    Ouvrir le fichier                                   Nom_Ficher : in ;
                                                        Fichier : out,
                                                        Flux_Fichier : out
    TANT QUE non Fin_Du_Fichier (Nom_Fichier) FAIRE
        Lire octet                          Nom_Texte : in ; Octet : out T_Octet
        Mettre à jour la fréquence de Octet     Octet : in ;
                                                Tableau_Frequence : in out
    FIN TANT QUE
```

R2 : Comment “Créer arbre” ?

```
    Initialiser un tableau                  List_Char : out T_Tableau
    Créer la liste de noeud                 Tableau_frequence : in ;
                                            List_Char : in out
    Trier la liste de noeud de manière décroissante     List_Char : in out
    TANT QUE List_Char.Taille > 1 FAIRE
        Créer noeud intermédiaire                       List_Char : in out
        Trier la liste de noeud de manière décroissante List_Char : in out
    FIN TANT QUE
```

R2 : Comment “Déterminer les codes des caractères” ?

```
    Initialiser                             Code_vide : out T_Flux_Binaire
    Avoir le code de l’arbre                List_Char .Tablea(1) : in T_Arbre,
                                            Code_vide : in ;
                                            Codage : out T_Codage
```

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

```
    Initialiser                             Liste_Position : out Liste_Chainee
    Initialiser                             Code_Structure_Arbre : out T_Flux_Binaire
    Initialiser                             Fichier_Encode : out T_Flux_Binaire
    Déterminer la liste des positions des caractères dans l’arbre
                                            List_Char.Tableau(1) : in ;
                                            Liste_Position : in out T_Liste ;
                                            Code_Structure_Arbre : in out
    Encoder la liste des positions          Liste_Position : in, Arbre_Encode : out
```

R2 : Comment “Encoder le fichier grâce à l’arbre” ?

Ouvrir le fichier — Nom_Ficher : in ;
Fichier : out,
Flux_Fichier : out

Nom_Compresse <- Nom_Fichier + ".hff"
Créer le fichier Nom_Compresse — Nom_Fichier : in Chaine
Ouvrir le fichier Nom_Fichier
Ecrire (Nom_Compresse, Arbre_Encode)
Encoder les octets de Nom_Fichier dans Nom_Compresse
Nom_Fichier, Nom_Compresse : in

R2 : Comment "Afficher les différentes étapes" ?
Afficher les fréquences de chaque caractères — Tableau_frequences : in
Afficher l'arbre d'Huffman — List_Char : in
Afficher les codes de chaques caractères — Codage : in

R3 : Comment "Lire le double argument" ?

```
SI Argument(1) = "-b" OU Argument(1) = "--bavard" ALORS
    Affichage <- True
    Nom_Ficher <- Argument(2)
SINON
    LEVER Option_Inconnue_Exception
FIN SI
```

R3 : Comment "Initialiser les cases de Tableau_Frequence à 0" ?

```
POUR I DE 1 À 256 FAIRE
    Tableau_Frequence(I) <- 0
FIN POUR
```

R3 : Comment "Ouvrir le fichier" ?

```
Ouvrir (Fichier, In_File, Nom_Fichier)
Flux_Fichier <- Stream(Fichier)
Exception
    Quand Nom_Fichier non trouver => LEVER Fichier_Inexistant_Exception
```

R3 : Comment "Mettre à jour la fréquence de Octet" ?

```
Tableau_Frequence(Entier(Octet) +1) <- Tableau_Frequence(Entier(Octet) +1) + 1
```

R3 : Comment "Initialiser un tableau" ?
Entrée : Tableau : out T_Tableau

```
Tableau.Taille <- 0
```

R3 : Comment "Créer la liste de nœuds" ?

```
POUR i = 1..256 FAIRE
    Ajouter les caractères de fréquences non nulles — Tableau_Frequence : in,
                                                       i : in T_Octet ;
```

```
                                                        List_Char : in out
    FIN POUR
    List_Char.Taille <- List_char.Taille + 1
    Créer un nouveau noeud      0 : in Entier,
                                T_Octet'Last : in T_Octet,
                                null : in T_Arbre,
                                null : in T_Arbre ;
                                List_char.Tableau(List_char.Taille) : out T_Arbre

R3 : Comment "Créer un noeud intermédiaire" ?
    Plus_Faible_1 <- List_char.Tableau[List_char.Taille]
    Plus_Faible_2 <- List_char.Tableau[List_char.Taille -1]
    Créer un nouveau noeud      Plus_Faible_1.Taille+Plus_Faible_2.Taille : in Entier,
                                0 : in Entier,
                                Plus_Faible_1 : in T_Arbre,
                                Plus_Faible_2 : in T_Arbre ;
                                Nouv_Noeud : out T_Arbre
    List_Char.Tableau(Liste.Taille) <- null
    List_char.Taille <- List_char.Taille - 1
    List_char.Tableau[List_char.Taille] <- Nouv_Noeud

R3 :  Comment "Trier la liste de noeud de manière décroissante" ?
    POUR Indice = 1..(Tab.Taille-1) FAIRE
        Déterminer le maximum   Tab : in,
                                Indice : in ;
                                Indice_Max : out Entier
        Déplacer le maximum     Indice : in,
                                Indice_Max : in ;
                                Tab : in out
    FIN POUR

R3 : Comment "Avoir le code de l'arbre" ?
Entrées :   Arbre : in T_Arbre,
            Code : in T_FluxBinaire,
            Codage : out T_Codage

    SI Arbre.Gauche = Rien ALORS
        Codage(Arbre.Valeur) <- Code
    SINON
        Créer code          0 : in Entier,
                            Code : in ;
                            Code_Gauche : out T_Flux_Binaire
        Créer code          1 : in Entier,
                            Code : in ;
                            Code_Droite : out T_Flux_Binaire

        Avoir Code Arbre    Arbre.Gauche : in, Code_Gauche : in ; Codage : out
        Avoir Code Arbre    Arbre.Droite : in, Code_Droite : in ; Codage : out
```

```
        Vider                   Code_Gauche : in out
        Vider                   Code_Droite : in out
    FIN SI
```

R3 : Comment “Déterminer la liste des positions des caractères dans l'arbre” ?

```
    SI Arbre = Rien ALORS
        Rien
    SINON SI Arbre.Gauche = Rien ALORS
        Ajouter_Bit             1 : in Entier,
                                Code_Structure_Arbre : in out ;
        Ajouter Arbre.Valeur à Liste_Position       Arbre.Valeur : in T_Octet,
                                                    Liste_Position : in out
    SINON
        Ajouter_Bit             0 : in Entier ;
                                Code_Structure : in out
        Déterminer la liste des positions des caractères dans l'arbre
                                    Arbre.Gauche : in ;
                                    Liste_Position : in out
                                    Code_Structure_Arbre : in out
        Déterminer la liste des positions des caractères dans l'arbre
                                    Arbre.Droit : in ;
                                    Liste_Position : in out,
                                    Code_Structure_Arbre : in out
    FIN SI
```

R3 : Comment “Encoder la liste des positions” ?

```
    Ajouter à Arbre_Encode la position de ‘\$’   Liste_Position : in ;
                                                 Fichier_Encode : in out
    Ajouter tous les octets de la position       Liste_Position : in ;
                                                 Fichier_Encode : in out
    Ajouter_Flux                                 Code_Structure_Arbre : in ;
                                                 Fichier_Encode : in out
```

R3 : Comment “Encoder les octets de Nom_Fichier dans Nom_Compresse”

```
    TANT QUE  non Fin_Du_Fichier (Nom_Fichier) FAIRE
        Lire un octet                   Nom_Fichier : in ; Octet : out T_Octet
        Ecrire (Nom_Compresse, Codage(Entier(Octet) +  1))
    FIN TANT QUE
```

R4 : Comment “Ajouter les caractères de fréquences non nulles” ?

```
    SI Tableau_frequence(i) /= 0 ALORS
        List_char.Taille <- List_char.Taille + 1
        Créer un nouveau noeud      Tableau_frequence(i) : in Entier,
                                    I : in T_Octet,
                                    null : in T_Arbre,
```

null : in T_Arbre ;
List_char.Tableau(List_char.Taille) : out T_Arbre

FIN SI

R4/5 : Comment “Créer un nouveau noeud” ?
Entrées : Frequence : in Entier,
Caractere : in T_Octet,
Noeud_Gauche: in T_Arbre,
Noeud_Droite : in T_Arbre,
Arbre : out T_Arbre

Initialiser Arbre : out
Enregistrer Arbre : in out ;
Frequence : in,
Caractere : in,
Noeud_Gauche: in,
Noeud_Droite : in

R4 : Comment “Déterminer le maximum” ?
Indice_Max <- Indice
POUR Indice2 = Indice+1..Tab.Taille FAIRE
Remplacer l’indice du maximum Indice2 : in
Tab : in ;
Indice_Max : in out
FIN POUR

R4 : Comment “Déplacer le maximum” ?
Memoire <- Tab.Tableau(Indice)
Tab.Tableau(Indice) <- Tab.Tableau(Indice_Max)
Tab.Tableau(Indice_Max) <- Memoire

R4 : Comment "Créer code” ?
Entrées :
Entier_code : in Entier
Code : in T_Flux_Binaire
Codage : out T_Flux_Binaire

Initialiser Codage : out
Ajouter_Flux Code : in ;
Codage : in out
Ajouter_Bit Entier : in ;
Codage : in out

R5 : Comment “Remplacer l’indice du maximum” ?
SI Tab.Tableau(Indice2).Cle >= Tab.Tableau(Indice_Max).Cle ALORS

```
            Indice_Max <- Indice2
    FIN SI
```

Tableau Evaluation

	Règle	**Evaluation (I/P/A/+)**
Forme	**Respect de la syntaxe** **Ri : Comment "... une action complexe ..." ? des actions combinées avec des structures de controle** **Rj : ...**	+
	Verbe à l'infinitif pour les actions complexes	+
	Nom ou équivalent pour expressions complexes	+
	Tous les Ri sont écrits contre la marge et espacés	+
	Les flots de données sont définis	+
	Une seule décision ou répétition par raffinage	+
	Pas trop d'actions dans un raffinage (moins de 5 ou 6)	A
	Bonne présentation des structures de contrôle	A
Fond	**Le vocabulaire est précis**	A
	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

Raffinage de Décompresser

R0 : Décompresser un fichier .hff compressé avec un arbre d'Huffman

Exemples :

./decompresser exemple.txt.hff => exemple.txt.

./decompresser -b exemple.txt.hff => afficher l'arbre d'Huffman exemple.txt

./decompresser –bavard exemple.txt.hff => afficher l'arbre d'Huffman exemple.txt

./decompresser exemple.txt => "Le fichier à décompresser n'a pas été compressé par Huffman"

./decompresser exmple.txt.hff => "Fichier à décompresser inexistant."

./decompresser -a exemple.txt => "Option entrée non reconnue."

./decompresser => "Vous n'avez pas entré d'arguments."

./decompresser a b c => "Vous avez entré trop d'arguments."

R1 : Comment "Décompresser" ?

Déterminer Nom_Fichier avec les arguments de la ligne de commande
Affichage : out Booléen,
Nom_Fichier : out Chaîne_caractère

SI Fin_Nom_Fichier = ".hff" ALORS

Ouvrir le fichier Nom_Fichier — Nom_Fichier : in ; Flux_Fichier : out T_Flux_Fichier

Recréer l'arbre de Huffman — Flux_Fichier : in ; Arbre_Huffman : out T_Arbre

Reconstruire le fichier original — Flux_Fichier : in, Nom_Fichier : in, Arbre_Huffman : in

SINON

LEVER Pas_Compression_Huffman

FIN SI

R2 : Comment "Déterminer Nom_Fichier avec les arguments de la ligne de commande" ?

Affichage <- False

SI Argument_Count > 2 ALORS

LEVER Trop_Arguments_Exception

SINON SI Argument_Count = 1 ALORS

Nom_Fichier <- Argument(1)

SINON SI Argument_Count = 0 ALORS

LEVER Peu_Arguements_Exception

SINON

Lire le double argument — Affichage : out Booléen, Nom_Fichier : out Chaîne_caractère

FIN SI

R2 : Comment “Ouvrir le fichier Nom_Fichier” ?

Ouvrir (Fichier, In_File, Nom_Fichier)

Flux_Fichier_Brut <- Stream(Fichier)

Initialiser le Flux du Fichier Flux_Fichier_Brut : in Stream_Access ;
Flux_Fichier : out

Exception

Quand Nom_Fichier non trouver => LEVER Fichier_Inexistant_Exception

R2 : Comment “Recréer l’arbre Huffman” ?

Décoder la liste des positions Flux_Fichier : in ;
Liste_Position : out T_Liste_Chainee

Décoder la structure de l’arbre Flux_Fichier : in, Liste_Position : in,
Arbre_Huffman : out T_Arbre

R2 : Comment “Reconstruire le fichier original” ?

Nom_Decompresse <- Nom_Fichier - “.hff”

Créer et ouvrir le fichier Nom_Decompresse Nom_Decompresse : in Chaine

Décoder les octets de Nom_Fichier dans Nom_Decompresse Arbre_Huffman : in
Flux_Fichier : in

R3 : Comment “Lire le double argument” ?

SI Argument(1) = “-b” OU Argument(1) = “--bavard” ALORS

Affichage <- True

Nom_Ficher <- Argument(2)

SINON

LEVER Option_Inconnue_Exception

FIN SI

R3 : Comment “Initialiser le Flux du Fichier” ?

Flux.Fichier.Flux_Interne <- Flux_Fichier_Brut

Flux_Fichier.Octet <- 0

Flux_Fichier.Indice_Bit <- 8

R3 : Comment “Décoder la liste des positions” ?

Initialiser Liste_Positions : out

Lire et ajouter la position du symbole de fin Flux_Fichier : in ;
Position_Fin : out T_Octet

Ajouter (Liste_Position , Position_Fin)

Lire un octet Flux_Fichier: in, Octet : out T_Octet

RÉPÉTER

Ajouter (Liste_Position, Octet)

Dernier_Octet <- Octet

```
        Lire un octet                       Flux_Fichier: in, Octet : out T_Octet
    JUSQU'À Octet = Dernier_Octet
```

R3 : Comment “Décoder la structure de l’arbre” ?

```
    Initaliser      Arbre_Huffman : out
    Lire un bit     Flux_Fichier : in, Bit : out Entier
    Position <- 0
    SI Bit = 1 ALORS
        Déterminer la clé de la valeur      Position : in, Position_Symbole_Fin : in ;
                                            Pas_Symbole_Fin : out Entier
        Enregistrer             Arbre_Huffman : in out ;
                                Pas_Symbole_Fin  : in,
                                La_Valeur(Liste_Positions, Positions) : in,
                                null : in,
                                null : in
        Position <- Position + 1
    SINON
        Enregistrer             Arbre_Huffman : in out ;
                                0 : in,
                                0 : in,
                                null : in,
                                null : in

        Décoder la structure de l’arbre     Arbre_Huffman .Gauche : out
        Décoder la structure de l’arbre     Arbre_Huffman .Droite : out
    FIN SI
```

R3 : Comment “Décoder les octets de Nom_Fichier dans Nom_Decompresse” ?

```
    Est_Fin_Du_Fichier <- False
    TANT QUE  non Est_Fin_Du_Fichier FAIRE
        Déterminer le prochain caractère    Nom_Fichier : in, Arbre : in,
                                            Octet : out T_Octet,
                                            Est_Fin_Du_Fichier out Booléen
        SI non Est_Symbole_Fin ALORS
            Ecrire (Nom_Decompresse, Octet)
        FIN SI
    FIN TANT QUE
```

R4 : Comment “Déterminer la clé de la valeur” ?

```
    SI Position = Position_Symbole_Fin ALORS
        Pas_Symbole_Fin <- 0
    SINON
        Pas_Symbole_Fin <- 1
```

FIN SI

R4 : Comment “Déterminer le prochain caractère” ?
TANT QUE Arbre.Gauche /= Rien FAIRE
Lire un bit Nom_Fichier : in, Bit : out Entier
Déterminer quelle branche parcourir Bit : in ; Arbre : out
FIN TANT QUE
Octet <- Arbre.Valeur
Est_Symbole_Fin <- Arbre.Cle = 0

R5 : Comment “Déterminer quelle branche parcourir” ?
SI Bit = 0 ALORS
Arbre <- Arbre.Gauche
SINON
Arbre <- Arbre.Droite
FIN SI

Tableau Evaluation

	Règle	**Evaluation (I/P/A/+)**
Forme	**Respect de la syntaxe** **Ri : Comment "... une action complexe ..." ? des actions combinées avec des structures de controle** **Rj : ...**	+
	Verbe à l'infinitif pour les actions complexes	+
	Nom ou équivalent pour expressions complexes	+
	Tous les Ri sont écrits contre la marge et espacés	+
	Les flots de données sont définis	+
	Une seule décision ou répétition par raffinage	+
	Pas trop d'actions dans un raffinage (moins de 5 ou 6)	A
	Bonne présentation des structures de contrôle	A
Fond	**Le vocabulaire est précis**	A
	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