Raffinage du compressage

R0 : Compresser un fichier texte donné

R1 : Comment "Compresser un fichier texte donné" ?

Lire la commande
Si Argument_Count = 2 et (Argument(1) = '-b' ou Argument(1) = '-bavard') **Alors**
    Nom_du_fichier := Argument(2)
    Faire le compressage de Nom_du_fichier par codage de Huffman — text : **In**
    Créer un fichier Nom_du_fichier.hff — text_compress : **Out**
    Afficher l'arbre de Huffman — arbre_bin : **In**
    Afficher la table de Huffman — table_Huffman : **In**
Sinon
    Nom_du_fichier := Argument(1)
    Faire le compressage de Nom_du_fichier par codage de Huffman — text : **In**
    Créer un fichier Nom_du_fichier.hff — text_compress : **Out**

R2 : Comment "Faire le compressage par codage de Huffman" ?

Initialiser(table_frequence) — table_frequence : **Out**
Compter la fréquence des octet de tableau_frequence — Nom_du_fichier: **In**, table_frequence : **In Out**
Initialiser(arbre_bin) — arbre bin: **Out**
Créer l'arbre de Huffman — list_abr: **In Out**, arbre_bin: **In Out**
Initialiser(table_Huffman) — table_Huffman : **Out**
Parcourir de manière infixe l'arbre arbre_bin — arbre_bin: **In Out**, suite_bits: **In Out**, table_Huffman: **Out**, code: **In Out**, caractere: **In Out**
Compresser text — Nom_du_fichier: **In**, Table_Huffman: **In Out**, caractere: **In Out**

R3 : Comment "Compter la fréquence de chaque octet dans tableau_frequence" ?

Lire le fichier Nom_du_fichier
Intégrer caractere_special dans tableau_frequence — tableau_frequence: **In Out**
TantQue ce n'est pas la fin du fichier **Faire**
    Lire octet
    Convertir octet en binaire Cle — Octet: **In**, Cle: **Out**
    Si Cle_Presente (tableau_frequence, Cle) **Alors**
        Enregistrer (tableau_frequence, Cle, La_Donnee(tableau_frequence, Cle)+1)
    Sinon
        Enregistrer (tableau_frequence , nouveau_arbre, 1)
    Fin Si
Fin Pour

R3 : Comment "Créer l'arbre de Huffman arbre_bin" ?

TantQue Taille(tableau_frequence) >= 2
Min_frequence(Cle_min, liste_abr)
arbre_1 <- La_donnee(liste_abr, Cle_min)
Supprimer (list_abr, Cle_min)
Min_frequence(Cle_min, liste_abr)
arbre_2 <- La_donnee(liste_abr, Cle_min)
Supprimer (list_abr, Cle_min)
Fusionner (arbre_1, arbre_2)
Enregistrer (liste_abr, La_cle(arbre_1), arbre_1)
Fin TQ
arbre_bin <- La_donnee(liste_abr, La_Cle(arbre_1))

R3 : Comment "Parcourir de manière infixe l'arbre arbre_bin" ?

Si arbre_binaire est une feuille **Alors**
Ajouter 1 à suite_bits
Enregistrer (tableau_Huffman, La_Cle(arbre_bin), code)
Ajouter La_Cle(arbre_bin) à caractere

Sinon
Ajouter 0 à suite_bits
Ajouter 0 à code
Parcourir de manière infixe l'arbre arbre_binaire_gauche
Enlever le dernier bit de code
Ajouter 1 à code
Parcourir de manière infixe l'arbre arbre_binaire_droite
Enlever le dernier bit de code
FinSi

R3 : Comment "Compresser text" ?

Traduire les caractères dans l'ordre infixe en doublant le dernier
Traduire le parcours infixe
Traduire text
Traduire le caractère spécial
TantQue longueur(code) < 8 **Faire**
Ajouter 0 à code
Fin TantQue
Convertir code en octet
Ajouter code dans Nom_du_fichier.hff

R4: Comment “Traduire text” ?

```
TantQue ce n’est pas la fin du fichier Faire
        Lire octet
        Convertir octet en binaire Cle
        Ajouter La_donnee(table_Huffman, Cle) à code
        Si longueur(Code) = 8 Alors
                Ajouter code dans Nom_du_fichier.hff
        SinonSi  longueur(Code) > 8 Alors
                TantQue longueur(Code) > 8 Faire
                        Ajouter le dernier bit de code à Code_prochain
                        Supprimer le dernier bit de code
                Fin TantQue
                Ajouter code dans Nom_du_fichier.hff
                Code <- Code_prochain
                Code_prochain <- null
        FinSi
Fin TantQue
```

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 contrôle Rj : ...	+
	Verbes à l'infinitif pour les actions complexes	A
	Noms ou équivalent pour expressions complexes	+
	Tous les Ri sont écrits contre la marge et espacés	A
	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)	A
	Bonne présentation des structures de contrôle	+
Fond (D21-D22)	Le vocabulaire est précis	+
	Le raffinage d'une action décrit complètement cette action	+
	Le raffinage d'une action ne décrit que cette action	+
	Les flots de données sont cohérents	A
	Pas de structure de contrôle déguisée	+
	Qualité des actions complexes	+

Raffinage du decompressage

R0 : Décompresser un fichier coder à partir du codage de Huffman

R1 : Comment " Décompresser un fichier coder à partir du codage de Huffman"?

Nom_fichier < -- Argument (1)
Contrôler que le type du fichier est bien .hff — Nom_fichier : **In**
Lire le fichier compresser — Nom_fichier : **In** Texte_compresse : **Out**
Reconstruire la table du codage de Huffman — Texte_Compresse : **In** Codage_Huffman : **Out**
Faire le décompressage avec la table de codage de Huffman — Codage_Huffman : **In** Texte_Decompresse **Out**

R2 : Comment "Contrôler si le type du fichier est bien .hff " ?

```
Si Longueur (Nom_fichier) <= 4
        Lever Pas_hff_exception
Sinon
        Type_fichier < -- Nom_fichier (Longueur (Nom_fichier) -3, Longueur (Nom_fichier)) ;
        Si Type_fichier /= ". hff"
                Lever Pas_hff_exception
        Fin Si
FinSi
```

R2 : Comment "Reconstruire la table du codage de Huffman " ?

Récupérer les caractères à utiliser pour la décompression — text_compresse : **In** Tableau_caractere **out**
Récupérer le parcours infixe suite_bits de l'arbre de Huffman — text_compresse , Tableau_caractere : **in** Suite_bits **out**
position_feuille <- 0
Reconstruire la table du codage de Huffman à partir du parcours infixe suite_bits — Suite_bits: **In** Codage_Huffman **Out**

R2 : Comment " Faire le décompressage avec la Table de Codage de Huffman " ?

Créer un fichier sans le type .hff
Code < – Null_Unbounded_String
Répéter

Changer d'octet courant lorsqu'il a été lu en entier — Compteur, Octet, Octet_Un_chaine : **In Out**
Append (Code, Octet_chaine (Compteur))
Regarder la correspondance de Code avec les codes de la table de Huffman — Codage_Huffman : **In** Code : **In Out**
Compteur < -- Compteur + 1
Fin Répéter

R3 : Comment " Récupérer les caractères à utiliser pour la décompression " ?

Lire l'octet en cours et l'attribuer à Premier_caractere
Ajouter le premier caractère — Premier_carcatere : In Tableau_caractere : Out
Lire l'octet en cours et l'attribuer à Nouveau_caractere
TantQue Nouveau_caractere /= Premier_caractere
Ajouter le Nouveau caractère — Tableau_caractere : **In** Nouveau_caractere **In Out**
Compteur < -- Compteur + 1
Fin **TantQue**

R3 : Comment " Récupérer le parcours infixe suite_bits de l'arbre de Huffman " ?

Initialiser les paramètres du parcours infixe Octet, Octet_un_chaine, Octet_chaine, Nbr_feuille_totale, Nbr_feuille, Compteur : **Out**
Répéter
    Changer d'octet courant lorsque celui-ci a été lu en entier Octet_Un_chaine, Octet_chaine, Compteur :**In Out** Octet **In**
    Si Octet_chaine (Compteur) = '1'
        Nbr_feuille < -- Nbr_feuille +1
    FinSi
    Append (Suite_bits, Octet_chaine (Compteur))
    Compteur < -- Compteur + 1
    Sortir Quand Nbr_feuille_totale = Nbr_feuille
Fin Répéter

R3 : Comment "Reconstruire la table du codage de Huffman à partir du parcours infixe suite_bits" ?

Initialiser (Codage_Huffman) Codage_Huffman : **Out**
Code < -- Null_Unbounded_String
Position_feuille < -- 1
Pour I allant de 1 à Longueur(Suite_bits) **Faire**
    Traiter le caractère de la suite de bits qui est lu Code, Suite_bits, Position_feuille : **In Out**
Fin Pour

R3 : Comment "Changer d'octet courant lorsque celui-ci a été lu en entier" ?
Si Compteur = 9
    Lire l'Octet en suivant du fichier_compresse
    Octet_Un_chaine < -- Octet_to_binaire (Octet)
    Octet_chaine < -- To_String (Octet_Un_chaine)
    Compteur
FinSI

R3 : Comment " Regarder la correspondance de Code avec les codes de la table de Huffman"
temp < -- Codage_Huffman
Pour i allant de 1 à Taille(Codage_Huffman) **Faire**
    Si Code = La_Cle(Temp)
        Bonne_Donnee < -- La_Donnee_en_cours(temp)
        Si To_String(Octet_to_binaire(La_Donnee_en_cours(temp))) /= "11111111"
            Écrire la Bonne Donnee dans le fichier décompresser
            Code := Null_Unbounded_String
        Fin Si
    Fin Si
    temp <- La_Suivante (temp)
Fin Pour

R4 : Comment "Ajouter le premier caractère" ?

Initialiser (Tableau_caractere) Tableau_caractere : **Out**
Compteur < -- 2
Cle < -- Octet_to_binaire (Premier_caractere)
Enregistrer (Tableau_caractere, Compteur, Cle)

R4 : Comment "Ajouter le nouveau caractère" ?

Cle < -- Octet_to_binaire (Nouveau_caractere)
Enregistrer (Tableau_caractere, Compteur, Cle)
Premier_caractere < -- Nouveau_caractere
Nouveau_caractere < -- T_Octet'Input(S)

R4 : Comment “Initialiser les paramètres du parcours infixe“ ?

```
Lire l’Octet suivant du fichier compresser
Octet_Un_chaine < -- Octet_to_binaire (Octet)
Octet_chaine < -- To_String(Octet)
Nbr_feuille_totale < -- Compteur-1
Nbr_feuille < -- 0
Suite_bits < -- Null_Unbounded_String
Compteur < -- 1
```

R4 : Comment “Traiter le caractère de la suite de bits qui est lu“ ?

```
Si Element (Suite_bits, i) = ‘0’
        Append(Code, To_Unbounded_String (“0“))
Sinon
        Enregistrer( Codage_Huffman, Code, Binaire_to_octet (La_Donnee (Tableau_caractere,
position_feuille)))
        position_feuille < -- position_feuille+1
        Code < -- Unbounded_Slice(Code, Longueur(Code)-1)
  TantQue Element(Code, Length(Code)) = '1'
        Code := Unbounded_Slice(Code, 1, Length(Code)-1)
        Fin TantQue
        Replace_Element(Code, Length(Code), ‘1’)
Fin Si
```

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	A
	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)	+
	Bonne présentation des structures de contrôle	A
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	+

Principaux types de données et structuration en module

Principaux types de données :

- LCA: (Table_frequence, Table_Huffman, Suite_bits, Code)
- Arbre Binaire : (Arbre_bin, Arbre_fils)
- Unbounded_String : Texte_compresser_chaine

Structuration en module :

Module d'un Arbre Binaire :

- Initialiser un Arbre Binaire. L'Arbre Binaire est vide.
procedure Initialiser (Arbre_bin: out T_Arbre)

- Est-ce qu'un Arbre binaire est vide ?
function Est_Vide (Arbre_bin : in T_Arbre) return Boolean;

- Enregistrer une Donnée associée à une Clé dans un arbre binaire.
procedure Enregistrer_noeud (Arbre_bin : in out T_Arbre ; Cle : in T_Cle ; Donnee : in T_Donnee)

```
- Obtenir la donnée associée à une Cle dans l'arbre binaire.
-- Exception : Cle_Absente_Exception si Clé n'est pas utilisée dans l'Sda
function La_Donnee (Arbre_bin : in T_Arbre ; Cle : in T_Cle) return T_Donnee;


- Supprimer tous les éléments d'un Arbre_binaire.
procedure Vider (Arbre_bin : in out T_Arbre)


-- Aller à l'arbre fils de droite d'un arbre binaire.
procedure Droite(Arbre bin : in out T_Arbre)


-- Aller à l'arbre fils de gauche d'un arbre binaire.
procedure Gauche(Arbre bin : in out T_Arbre)


-- Fusionner deux arbres
function Donnee(Arbre bin : in) return T_Donnee


-- Renvoyer la donnée du noeud d'un arbre binaire
function Donnee(Arbre bin : in) return T_Donnee


-- Renvoyer la valeur du noeud d'un arbre binaire
function Valeur(Arbre bin : in) return T_Donnee
```

Module LCA :

Même module que pour le TP10

```
function Octet_to_binaire(Octet : in T_Octet) return String is
        Bit, temp : T_Octet;
        Octet_string : Unbounded_String;

        begin
        temp := Octet
        Bit := temp / 128;
        Octet_string := To_Unbounded_String(Integer(Bit))
        temp := temp * 2;
        for N in 1..7 loop
                Bit := temp / 128;
                Put(Integer(Bit), 1);
                temp := temp * 2;
        end loop;
        New_Line;


  end Octet_to_binaire;
```