Raffinages.

R0 : Compresser un fichier

R1 : Comment << Compresser un fichier >> ?

Lire Le fichier et le stocker dans un tableau
Créer le tableau d'occurrence
Construire l'arbre de Huffman du fichier
Créer la table de Huffman
Encoder le texte
Creer le fichier compresser avec l'extension .hff

R2 : Comment << Lire Le fichier >> ?

Ouvrir Le ficher
Stocker Les caracters de Fichier dans un Tableau Fichie : out Tableau des octets

R2 : Comment << Créer le tableau d'occurrence >> ?

Créer Module Dic -- Pour stocker les caractères et leur nombre d'occurrence
Initialiser_Dic (Tab_occurence) Tab_occurence : out T_Dic
Enregistre les caractères et leur occurrence Tab_occurence : in out T_Dic

R2 : Comment << Construire l'arbre de Huffman de Fichier >> ?

Créer Le module Arbre -- Pour modeliser l'arbre de Huffman
Créer Le type Tab_Arb -- Pour Stocker les arbres premieres
Créer Un tableau des Racine d'Arbre Racine : out Tab_Arb
Créer le Procedure min_des_Racines Racine : in min : out entier
Fusionner itérativement les deux minimale de Racine Racine : in out
Retourner l'Arbre de Huffman Arb_Huff : out T_Arbre

R2 : comment << Créer la table de Huffman >> ?

Créer le module Tableau_Code -- Pour Stocker les code_de_Huff et les caractere
Remplir Les Code dans Tab_Huff Tab_Huff : out T_Code Arb_Huff : in T_Arbre

R2 : Comment << Activer l'affichage selon l'option choisi b ou s >> ?

Si option = b Alors

```
        Afficher( Arb_Huff )                          Arb_Huff : in T_Arbre
    Sinon
        Rien
    FinSi
```

R2 : Comment << Encoder le texte >> ?

```
    Declarer le ficher_Comp de type tableau des bits        Ficher_Comp :  Tableau
    Remplir ficher_comp                                     Ficher_Comp : in out Tableau
```

R3 : Comment << Creer Module Dic >> ?

```
    TYPE Cellule
    TYPE T_Dic EST POINTEUR SUR Cellule
    TYPE Cellule EST ENREGISTREMENT
            Caractere :   T_octet
            Occurence : entier
            Suivant : T_Dic
    FIN ENREGISTREMENT
    -- Creer les outil pour modeliser ce type
    Implanter les Sous programme de Controle de T_Dic
```

R3 : Comment << Enregistre les caractere et leur occurence >> ?

```
    i <- 1                                                  i : out entier
    Tanque Ficher(i) /= octet_Fin_Ficher Faire
            Si    Charactere_Present_Dic (Tab_occurence , Ficher(i)) Alors
                  Enregistre_Dic (Tab_occurence , Ficher(i) , Occurence_Dic (Tab_occurence) + 1)
            Sinon
                  Enregistre_Dic(Tab_occurence,Ficher(i), 1)
            FinSi
            i <- i + 1
    FinTQ
```

R3 : Comment << Créer Le module Arbre >> ?

```
    TYPE Node
    TYPE T_Arbre EST POINTEUR SUR Node
    TYPE Node EST ENREGISTREMENT
            Caractere :   T_octet
            Occurence :  entier
            Gauche : T_Arbre
            Droite : T_Arbre
```

FIN ENREGISTREMENT
-- Les outils pour controler l'arbre
Implanter les fonction d'arbre de Huffman

R3 : Comment << Creer Le type T_Racine >> ?

```
T_Racine  EST ENREGISTREMENT
        Taille : entier
        Elements : Tableau(1..Taille_Dic(Tab_Occurence))  De T_Arbre
FIN ENREGISTREMENT
```

R3 : Comment << Créer Un tableau des Racine d'Arbre >> ?

```
Racine.Taille <- Taille_Dic ( Tab_Occurence )          Racine : in out T_Racine
Tanque Non  Est_Vide_Dic (Tab_Occurence ) Faire
    Initialise( Racine.Element (i) )
    Enregistre(Racine.Element(i) ,Tab_Occurence^.Caracter ,Tab_Occurence^.Occurence)
    Depiler_Dic ( Tab _Occurence )
FinTQ
```

R3 : Comment << Créer le Procedure min_des_Racines >> ?

```
min <- 0                                                    min : out entier
Pour i DE 1 A Racine.Taille Faire
    Si Racine.Element( i ) < Racine.Element( min ) Alors          min : in rntier
        min <- i                                                  min : out entier
    Sinon
        Rien
    FinSi
FinPour
```

R3 : Comment << Fusionner itérativement les deux minimale de Racine >> ?

```
Tanque Racine.Taille /= 1 Faire
    m <- min_des_Racines(Racine)                                        m : out entier
    K <- Racine.Element(m)                    K : out T_Arbre       m : in entier
    Racine.Element(m) <- Racine.Element(Racine.Taille)
    Racine.Element(Racine.Taille) <- K
    Racine.Taille <- Racine.Taille - 1
    m <- min_des_Racines(Racine)            m : out entier
    Racine.Element(m) <- Fusionner (Racine.Element(m),Racine.Element(Base.Taille +
1))
FinTQ
```

R3 : Comment << Retourner l'Arbre de Huffman >> ?

Arb_Huff <- Racine.Element(1) Arb_Huff : out T_Arbre

Raffinage du deuxieme programmes

R0 : Décompresser les fichiers dont l'extension .hff

R1 : Comment << Décompresser le fichier >> ?

Lire le fichier compresser et le stocker dans un tableau Fichier : out T-Fichier

Determiner les octets dans le debut du fichier qui represente les characters

Determiner la signature de l'arbre

Creer l'arbre de huffman a partir d'une liste de caracteres et une signature d'arbre

Commencer a decoder le fichier a partir du bit qui suit la derniere bit du signature

R2 : Comment Determiner la liste des octets (characteres)
Liste : list_charactere
Charac_preced: T-Octet := Fichier(2)
Charac_Actu : T_Octet := Fichier(3)
Count : Integer :=1
Tantque charac_preced /= Charac_Actu Faire
Liste.taille = liste.taille +1
Count =Count +1
liste.car(liste.taille)=charac_preced
Charac_Actu <- Fichier(Count)
Charac_preced <- Charac_Actu
FinTQ

R2 : Comment determiner la signature de l'arbre

Signature : out T_Signature
Count : Integer := liste.taille +1
Nombre_de_un : Integer := 0
Tantque Nombre_de_un < liste.taille Faire
Transformer l'octet en un tableau de 8 bits bits: out T_Array
Bits <-transformer(Fichier(Count))

```
    Pour i de 1 a 8 faire
      Si Nombre_de_un /= liste.taille  Alors

     Signature.taille <- Signature.taille + 1
     Signature.tab(signature.taille) =Octet(bits(i))
     Nombre_de_un <- Nombre_de_un + bits(i)

    FinPour
    Count <- Count +1
FinTQ
```

R2 : comment<< Creer l'arbre de huffman a partir d'une liste de caracteres et une signature d'arbre >> ?

```
  i_car ← 1
  i_bit ← 0
Procedure  cree_sgn_arb(i_car, Carcs, sign, i_bit, Arb) est

  Si i_bit = 0 Alors
    Arb ← Nouveau Noeud
    Arb.Occurence ← 0
    i_bit ← i_bit + 1
    Appeler cree_sgn_arb(i_car, Carcs, sign, i_bit, Arb.Gauche)
    i_bit ← i_bit + 1
    Appeler cree_sgn_arb(i_car, Carcs, sign, i_bit, Arb.Droite)
  Sinon Si (i_bit = sign.Taille) ou (i_car = Carcs.Taille) Alors
    Arb ← Nouveau Noeud
    Arb.Occurence ← 0
    Arb.Caractere ← Carcs[i_car]
  Sinon Si (sign.Tab(i_bit) = 0) et (sign.Tab(i_bit + 1) = 1) Alors
    Arb ← Nouveau Noeud
    Arb.Occurence ← 0
    Arb.Caractere ← Carcs[i_car]
    i_car ← i_car + 1
  Sinon Si (sign.Tab(i_bit) = 1) et (sign.Tab(i_bit + 1) = 1) Alors
    Arb ← Nouveau Noeud
    Arb.Occurence ← 0
    Arb.Caractere ← Carcs[i_car]
    i_car ← i_car + 1
  Sinon
    Arb ← Nouveau Noeud
    Arb.Occurence ← 0
    i_bit ← i_bit + 1
    Appeler cree_sgn_arb(i_car, Carcs, sign, i_bit, Arb.Gauche)
```

```
    i_bit ← i_bit + 1
    Appeler cree_sgn_arb(i_car, Carcs, sign, i_bit, Arb.Droite)
  Fin Si
```

R2 : comment<<Commencer a decoder le fichier a partir du bit qui suit la derniere bit du signature >> ?

```
j ← liste_caractere.taille + Signature.Taille
  Répéter
    bit ← Ficher(j)  // Lire le bit du fichier
    Pour k allant de 1 à 8 faire
      Si Est_Feuille(curseur) Alors
        Caract ← Caracter(curseur)  // Récupérer le caractère à la feuille
        Si Caract ≠ fin_ficher Alors
          T_Octet'Write(S, Caract)  // Écrire le caractère dans la sortie
        Fin Si
        curseur ← Arb  // Réinitialiser le curseur à la racine de l'arbre
      Sinon Si (bit(k) = 0) et (Caract ≠ fin_ficher) Alors
        Aller_Gauche(curseur)  // Se déplacer vers le sous-arbre gauche
      Sinon Si (bit(k) = 1) et (Caract ≠ fin_ficher) Alors
        Aller_Droite(curseur)  // Se déplacer vers le sous-arbre droit
      Sinon
        null  // Ne rien faire
      Fin Si
    Fin Pour
    j ← j + 1  // Passer au bit suivant

    Si Caract = fin_ficher Alors
      Quitter la boucle  // Arrêter quand la fin du fichier est atteinte
    Fin Si
  Jusqu'à ce que Caract = fin_ficher
```

Évaluation du code

		Consigne : Mettre O (oui) ou N (non) dans la colonne Etudiant suivant que la règle a été respectée ou non. Une justification peut être ajoutée dans la colonne “commentaire”.	
Commentaire	**Etudiant (O/N)**	**Règle**	**Enseignant (O/N)**
	O	Le programme ne doit pas contenir d'erreurs de compilation.	

	O	Le programme doit compiler sans messages d’avertissement.	
Nous n’avons pas de temp pour l’indenter	N	Le code doit être bien indenté.	
	O	Les règles de programmation du cours doivent être respectées : toujours un Sinon pour un Si, pas de sortie au milieu d'une répétition...	
En decompresser	N	Pas de code redondant.	
	O	On doit utiliser les structures de contrôle adaptées (Si/Selon/TantQue/Répéter/Pour)	
	N	Utiliser des constantes nommées plutôt que des constantes littérales.	
	N	Les raffinages doivent être respectés dans le programme.	
	N	Les actions complexes doivent apparaître sous forme de commentaires placés AVANT les instructions correspondantes, avec la même indentation	
	O	Une ligne blanche doit séparer les principales actions complexes	
	O	Le rôle des variables doit être explicité à leur déclaration (commentaire).	