Projet PIM

Qui fait Quoi ?

	nom court	NOM	Prénom			
équipier1	HH	Hamman	Hamza			
équipier2	Fanfan	Puéo–Moulet	Fantin			
équipier3	C	MIGLIASSO	Chloe			

Quoi ?		**Qui ?**				
Modules	**Sous-programmes**	**Spécification**	**Raffinage**	**Codage**	**Test**	**Relecture**
Fonctions de base	id_ad_IP		C / Fanfan			
	table_routage		C / Fanfan			
Routeur simple	trt_ad		C / HH			
Error_routeur_simple	Routage_Exceptions		C			

Raffinages Routeur Simple

Données :

Instantiations + Constantes	Erreurs levées
T_Octet <- mod 2**8 T_Adresse_IP <- 2 ** 32 Octet_IO <- Modular_IO(T_Octet) Adresse_IP_IO <- Modular_IO(T_Adresse_IP) Case <- Enregistrement(T_Adresse_IP, String) LCA_IP <- LCA(T_Adresse_IP, Case) Un_Octet <- 2 ** 8 Poids_Fort <- 2 ** 31 M4 <- -1 M3 <- M4 - 255 M2 <- M3 - 255 * un_octet M1 <- M2 - 255 * un_octet **2 M0 <- M1 - 255 * un_octet **3 M <- [M0, M1, M2, M3, M4] Tab_politique <- [FIFO, LRU, LFU]	Fichier_Inconnu_Error Commande_Inconnu_Error End_Error IP_Adresse_Unknown_Error
Variables + types	**Bibliothèques utilisées**
Cache : Integer -- Taille du cache Politique : Tab_politique -- Politique utilisé Statistiques : Boolean -- Statistiques Table : String -- Nom table de routage Paquet : String -- Nom fichier à router Resultat : String -- Nom fichier résultat Tab_routage : T_SDA Entree : File_Type Sortie : file_Type Nom_Entree : String Nom_Sortie : String	ada.Integer ada.String …

R0 : Traiter la liste des adresses

R1 : **Comment** “Traiter les listes des adresses” ?

Comprendre la ligne de commande Cache, Politique, Statistique, Table, Paquet, Resultat : out

Agir sur la ligne de commande — Statistique : in
Créer la table de routage — Table : in / Tab_Routage: out
Mettre en place l'entrée et la sortie — Paquet, Resultat : in / Entree, Sortie : out, Fichier
Début
Traiter les paquets à router — Entree, Sortie, Tab_Routage: in
Exception
End_Error => null;
Fin
Fermer les fichiers — Entree, Sortie : in

R2 : **Comment** "Comprendre la ligne de commande" ?

Cache <- 10	Cache : out, Integer
Politique <- FIFO	Politique : out, Cache
Statistique <- True	Statistique : out, Boolean
Table <- "table.txt"	Table : out, String
Paquet <- "paquets.txt"	Paquet : out, String
Resultat <- "resultats.txt"	Resultat : out, String
Gérer la ligne de commande	Cache, Paquet, Statistique, Table, Paquet, Resultat : in/out

R2 : **Comment** "Agir sur la ligne de commande" ?

```
Si Statistique Alors
    Put ("Rien a afficher car pas encore de cache")
Sinon
    Rien
FinSi
```

R2 : **Comment** "Créer la table de routage" ?

table_routage (t : in String)

R2 : **Comment** "Mettre en place l'entrée et la sortie"?

```
Nom_Entree <- Unbounded_String (q)
Nom_Sortie <- Unbounded_String (r)
Créer (Sortie, out, String(Nom_Sortie))
Début
    Ouvrir (Entree, in, String(Nom_Entree))
Exception
    Name_Error =>
        Afficher ("Erreur:" & String(Nom_Entree) & " inconnu")
        raise Fichier_Inconnu_Error
Fin
```

R2 : **Comment** “Traiter les paquets à router” ?
    TantQue non Fin_fichier (Entree) **Faire**
        Traiter le paquet à router
    FinTQ

R2 : **Comment** “Fermer les fichiers” ?
    Fermer (Entree)
    Fermer (Sortie)

R3 : **Comment** “Gérer la ligne de commande” ?
    nb_cmd <- 1
    TantQue nb_cmd <= nb_arguments **Faire**
        Traiter ligne de commande
        nb_cmd <- nb_cmd + 1
    FinTQ

R3 : **Comment** “Traiter le paquet à router” ?
    Texte <- **Prendre_nouvelle_ligne** (Entree)
    Ligne <- Entier (Ligne (Entree))
    Trim(Texte, Both)
    Identifier commande ou adresse IP Texte: in String / IP_cmd : out Boolean
    Si IP_cmd **Faire**
        Identifier adresse IP Texte : in / Adresse_IP : out Integer
        Associer adresse IP et Interface Adresse_IP : in / Interface : out, String
        Afficher (Sortie, Adresse_IP & “ ” & Interface)
    Sinon
        Identifier commande
    FinSi

R4 : **Comment** “Traiter ligne de commande” ?
    Selon Argument(nb_cmd) **Dans**
        “-c” => Traiter cas c
        “-p” => Traiter cas p
        “-s” => Statistique <- True
        “-S” => Statistique <- False
        “-t” => Traiter cas t
        “-q” => Traiter cas q
        “-r” => Traiter cas r
        Autres => Traiter erreur
    FinSelon

R4 : **Comment** “Identifier commande ou adresse IP” ?
    IP_cmd <- (Texte(1) in ‘0’ .. ‘9’)

```
R4 : Comment "Identifier adresse IP" ?
        id_ad_IP (Texte: in String)


R4 : Comment "Associer adresse IP et Interface" ?
        Pour i De 1 A 5 Faire
                Debut
                        Enregistrement <- La_Valeur (Sda, Adresse_IP AND M[i])
                        Masque <- Enregistrement.Masque
                        Si non Masque = M[i] Faire
                                Lever Cle_Absente_Error
                        Sinon
                                Rien
                        Interface <- Enregistrement.Interface
                Exception
                        Cle_Absente_Error => Rien;
                Fin
        FinPour


R4 : Comment "Identifier commande" ?
        Selon Texte Dans
                "table" => Afficher_Debug (Sda)
                "cache" => Afficher ("Commande non programmé : affichage Cache")
                "stat" => Afficher ("Commande non programmé : Affichage stat Cache")
                "fin" => Lever End_Error
                Autres => Traiter erreur commande texte
        FinSelon


R5 : Comment "Traiter cas c" ?
        nb_cmd <- nb_cmd + 1
        Début
                c <- Integer (Argument (nb_cmd))
        Exception
                Constraint_Error => Afficher ("Erreur: incompréhension après commande
-c")
        Fin


R5 : Comment "Traiter cas p" ?
        nb_cmd <- nb_cmd + 1
        Début
                p <- Argument(nb_cmd)
        Exception
```

Constraint_Error => **Afficher** (“Erreur : incompréhension après commande -p”)
Fin

R5 : **Comment** “Traiter cas t” ?
nb_cmd <- nb_cmd + 1
Début
t <- **Argument** (nb_cmd)
Exception
Constraint_Error => **Afficher** (“Erreur : incompréhension après commande -t”)
Fin

R5 : **Comment** “Traiter cas q” ?
nb_cmd <- nb_cmd + 1
Début
q <- **Argument** (nb_cmd)
Exception
Constraint_Error => **Afficher** (“Erreur : incompréhension après commande -q”)
Fin

R5 : **Comment** “Traiter cas r” ?
nb_cmd <- nb_cmd + 1
Début
r <- **Argument** (nb_cmd)
Exception
Constraint_Error => **Afficher** (“Erreur : incompréhension après commande -r”)
Fin

R5 : **Comment** “Traiter erreur” ?
Afficher (“Erreur à l’argument” & nb_cmd)
Lever Commande_Inconnu_Error

R5 : **Comment** “Traiter erreur commande texte” ?
Afficher (“Erreur à la ligne : “ & Ligne)
Lever Commande_Inconnu_Error

R0 : Identifier adresse IP

id_ad_IP (Texte: in String)

R1 : **Comment** “Identifier adresse IP” ?
Décomposition du String selon “.”
Créer adresse_IP

R2 : **Comment** “Décomposer le String selon “.” ” ?
Colonne <- 1
Ad_IP <- table 4
Compteur <- 1
Pour Compteur dans Taille(Texte) **Faire**
Si Texte(Compteur) = “.” **Faire**
Colonne <- Colonne + 1
SinonSi Texte(Compteur) De “0” A “9” **Faire**
Ad_IP(Colonne) <- Ad_IP(Colonne) * 10 + Entier(Texte(Compteur))
Sinon
Lever IP_Adresse_Unknown_Error
FinSi
FinPour

R2 : **Comment** “Creer l’addresse_IP” ?
Si ad_IP(1) <= 255 **Faire**
Pour Compteur De 2 A 4 Faire
Si ad_IP(Compteur) > 255 **Faire**
Lever IP_Adresse_Unknown_Error
Sinon
adresse_IP = ad_IP(Compteur) * un_octet
FinSi
Sinon
Lever IP_Adresse_Unknown_Error
FinSi

R0 : Créer la table de routage
table_routage (Table : in String)

R1 : **Comment** “Créer la table de routage” ?
Initialiser la création de la table de routage
TantQue non Fin_fichier (Entree) **Faire**
Comprendre ligne
FinTQ
Fermer (Entree)

R2 : **Comment** “Initialiser la création de la table de routage” ?

Début

Ouvrir (Entree, In_File, Table)

Exception

Name_Error => Fichier_Inconnu_Error

Fin

Initialiser (Tab_Routage)

R2 : **Comment “**Comprendre ligne” ?

Texte <- **Prendre_nouvelle_ligne** (Entree)

Décomposer en trois éléments

Ajouter à Tab_Routage

R3 : **Comment** “Décomposer en trois éléments” ?

```
Tab <- table 3
Colonne <- 1
Compteur_Espace <- False
Pour Compteur dans Taille(Texte) Faire
    Si Texte(Compteur)  = “ ” Et Compteur_Espace Faire
        Rien
    SinonSi Texte(Compteur) = “ ” Et non Compteur_Espace Faire
        Compteur_Espace <- True
        Colonne <- Colonne + 1
    Sinon
        Compteur_Espace <- False
        Tab(Colonne) <- Tab(Colonne) & Texte(Compteur)
    FinSi
FinPour
```

R3 : **Comment “**Ajouter à Tab_Routage”

```
Masque = id_ad_IP(Tab(2))
Destination = id_ad_IP(Tab(1))
Interface = Tab(3)
Enregistrement.masque = masque
Enregistrement.Interface = Interface
Ajouter (Sda, Destination, Enregistrement)
```

Raffinages Routeur avec cache

TODO : Ne reprendre que les actions/expressions complexes qui sont concernées par l’ajout d’un cache sur le Routeur.

Grille d'évaluation des raffinages

	Règle	Evaluation (I/P/A)	Justification/commentaire (optionnel)
Forme	**Respect de la syntaxe** **Ri : Comment "... une action complexe ..." ?** **des actions combinées avec des structures de controle** **Rj : ...**	A	
	Verbe à l'infinitif pour les actions complexes	A	
	Nom ou équivalent pour expressions complexes	A	
	Tous les Ri sont écrits contre la marge et espacés	A	
	Les flots de données sont définis	P	Je crois pas que je l'ai fait partout + pas le temps de tout vérifier
	Une seule structure de contrôle par raffinage	P	Zut, j'en ai mis un peu plus …
	Pas trop d'actions dans un raffinage (moins de 6)	P	Pas sûre
	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	A	
	Pas de structure de contrôle déguisée	A	
	Les raffinages couvrent tous les aspects du sujet.	A	Il manque juste la robustesse de la dernière fonction avec la non vérification du masque, mais ce n'est pas précisé dans le sujet donc ne devrait pas être un problème.
	Qualité des actions complexes	A	

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