Ce document reprend une liste de petit problème algorithmique en vue de manipuler les différents concepts.

## Affichage d’une somme :

Ecrire un algorithme qui demande à un utilisateur de saisir deux nombres entiers et affiche la somme des nombres.

Variables :

a : entier

b : entier

Début

Saisir a

Saisir b

Afficher a+b

Fin

## Vérification de nombre :

Ecrire un algorithme qui demande à un utilisateur de saisir deux nombres entiers et affiche « identique » si les deux nombres ont la même valeur ou « différent » sinon.

Variables :

A : entier

B : entier

Début

Saisir a

Saisir b

Si a == b alors

Afficher « identique »

Sinon

Afficher « différent »

FinSi

Fin

## Chaîne la plus longue :

Ecrire un algorithme qui demande à un utilisateur de saisir deux chaînes de caractères et affiche la plus longue des deux.

Variables :

a : chaîne de caractères

b : Chaîne de caractères

Début

Saisir a

Saisir b

Si longueur(a) > longueur(b) alors

Afficher a

Sinon

Afficher b

Finsi

Fin

## Multiple de 7 et de 4:

Ecrire un algorithme qui demande à un utilisateur de saisir un nombre entier et affiche « correct » si le nombre est à la fois multiple de 7 et de 4 ou « incorrect » sinon.

Variable :

n : entier

Début

Saisir n

Si n % 7 == 0 et n % 4 == 0 alors

Afficher « correct »

Sinon

Afficher « incorrect »

Finsi

Fin

## Nombre de ‘e’ :

Ecrire un algorithme qui demande à un utilisateur de saisir une chaîne de caractères et qui affiche le nombre de fois que la lettre ‘e’ apparait dans la chaîne.

Variable :

C : chaîne de caractères

Compteur : entier

Début

Saisir c

Compteur 🡨 0

Pour e dans c

Si e == “e” alors

Compteur 🡨 compteur +1

Fsi

FinPour

Afficher compteur

Fin

## Fonction puissance :

Ecrire une fonction qui prend en paramètre deux nombres entier a et b et retourne a^b.

Fonction puissance(a : entier, b : entier) : entier

Variables locales :

Produit : entier

Début

Produit 🡨 1

Pour i de 0 à b

Produit 🡨 produit \* a

Finpour

Retourner(produit)

Fin

## Fonction Nombre premier :

Ecrire une fonction qui prend en paramètre un nombre entier et renvoie VRAI si le nombre est un nombre premier et FAUX sinon.

Fonction premier(n : entier) : booléen

Variables locales

Div : entier

Début

Div 🡨 2

Tant que div <= racine\_carree(n)

Si n % div = 0 alors

Retourner(FAUX)

Fsi

Div 🡨 div + 1

Fintant

Retourner(VRAI)

Fin

## Fonction Moyenne :

Ecrire une fonction qui prend en paramètre une liste de nombres entiers et renvoie la moyenne des nombres de la liste.

Fonction moyenne(l : entier[]) : réel

Variables locales :

Somme : entier

Début :

Somme 🡨 0

Pour e dans l

Somme 🡨 somme + e

Finpour

Retourner (somme / len(l))

Fin

## Fonction Heure exacte :

Ecrire une fonction qui prend en paramètre un nombre entier (< 86 400) symbolisant un nombre de secondes. Cette fonction ne retourne aucune valeur mais affiche l’heure au format hh:mm:ss

Fonction heure(s : entier) : rien

Variable locales

H,m,s, reste : entier

Début

H = s // 3600

Reste = s % 3600

m = reste // 60

s = reste % 60

Afficher h + ‘ :’ + m + ‘ :’ + s

Fin

## Fonction Factorielle itérative :

Ecrire une fonction qui prend en paramètre un nombre entier positif et qui donne la valeur factorielle de ce nombre. Utiliser une boucle dans cette fonction.

## Fonction Factorielle récursive :

Ecrire une fonction qui prend en paramètre un nombre entier positif et qui donne la valeur factorielle de ce nombre. Utiliser une fonction récursive.

## Fonction Tri3 :

Ecrire une fonction qui prend en paramètre 3 nombres entier et qui affiche ces trois nombres triés dans l’ordre croissant.

## Fonction Multiplication de liste :

Ecrire une fonction qui prend en paramètre 1 liste d’entier l1 et un nombre entier n. La fonction retourne une liste qui contient tous les produits des éléments de l1 par n.

## Fonction palindrome itérative :

Ecrire une fonction qui prend en paramètre une chaîne de caractère et renvoie Vrai si la chaine est un palindrome et Faux sinon. Cette fonction s’appuie sur une boucle.

## Fonction palindrome récursive :

Ecrire une fonction qui prend en paramètre une chaîne de caractère et renvoie Vrai si la chaine est un palindrome et Faux sinon. Cette fonction s’appuie sur une fonction récursive.

## Fonction chemin le plus simple :

Ecrire une fonction qui prend en paramètre un tableau de caractère qui contient des directions [‘N’, ‘E’, ‘S’, ‘E’…] et ne renvoie rien. Cette suite de directions nous permet donc de déterminer un chemin. La fonction va simplifier ce chemin (par exemple un N et un S s’annulent) et affiche le chemin le plus court.

## Fonction cryptage de base :

Ecrire une fonction qui prend en paramètre une chaîne de caractère et renvoie la chaine initiale en ayant modifié les lettres :

* a par e
* e par o
* o par i
* i par u
* u par y
* y par a

## Clé d’un RIB :

Ecrire une fonction qui prend en paramètre :

* un entier pour le code banque
* un entier pour le code caisse
* un entier pour le code compte

Cette fonction va retourner une valeur entière qui contient la clé du RIB donné en paramètre. La clé rib se calcule de la manière suivante :

Clé = 97 – (( 89 \* code banque + 15 \* code caisse + 3 \* code compte) modulo 97)

## Validation d’un RIB :

Ecrire une fonction qui prend en paramètre :

* un entier pour le code banque
* un entier pour le code caisse
* un entier pour le code compte
* un entier pour la clé du RIB

Cette fonction va retourner VRAI si le RIB est valide, FAUX sinon. Vous pouvez vous appuyer sur la fonction juste au-dessus.

## Fonction hauteur sur une carte :

Ecrire une fonction qui prend en paramètre un tableau d’entiers à 2 dimensions qui représente une carte. Cette carte représente les altitudes des différents points. La fonction retourne l’altitude maximum de la carte. Autrement dit, la fonction doit parcourir toutes les cases du tableau à deux dimensions et renvoyer la valeur maximum.

## Développer un jeu de morpion :

Nous allons écrire l’algorithme d’un jeu de morpion en décomposant les différents éléments du jeu.

Nous pouvons commencer à nous poser quelques questions :

* Commençons par imaginer avec quelle structure de données nous pourrions représenter la grille du jeu.
* Quelle est la situation de départ du jeu et quelle est la situation finale.
* Définir ce qu’il se passe à chaque tour du jeu.
* Quelles sont les informations que l’utilisateur doit fournir ?

Voyons comment décomposer le jeu :

* Ecrire une fonction qui vérifie si le jeu est terminé. Quelles sont les paramètres de cette fonction, de quoi a-t ’on besoin pour dire que le jeu est terminé ?
* Ecrire une fonction qui modifie la grille en fonction des informations de l’utilisateur.
* Ecrire le programme principal qui va utiliser les deux fonctions ci-dessus pour finaliser le jeu.