TD2 – éléments de corrigé (algorithmes)

Algorithme 1 : AfficherEntiersBoucleTantQue
Principe : Initialiser une variable i à 7. Tant que i inférieur ou égal à 77, afficher i puis incrémenter i de 1.
Entrée : /
Local : - BORNE_INF : entier constant = 7
- BORNE_SUP : entier constant = 77
- i : entier
Sortie : void
Début

```
Soit i = BORNE_INF
Tant que i ≤ BORNE_SUP Faire
    AfficherEntier(i)
    i = i + 1
Fin Tant Que
```

Fin

Algorithme 1 : AfficherEntiersBouclePour
Principe : Définir une boucle Pour allant de 7 à 77 inclus avec un pas de 1 et afficher i.
Entrée : /
Local : - BORNE_INF : entier constant = 7
- BORNE_SUP : entier constant = 77
- i : entier
Sortie : void
Début

```
Pour i de BORNE_INF à BORNE_SUP avec un pas de +1 Faire
    AfficherEntier(i)
Fin Pour
```

Fin

Algorithme 2 : AfficherTriangleRectangle
Principe : Saisir la hauteur. Parcourir chaque ligne du triangle (1 à hauteur) en maintenant à jour le nombre d'étoiles à afficher sur la ligne courante (commencer à 1 et +2 à chaque ligne).
Entrée : /
Local : - hauteur : entier
- nbEtoiles : entier
- indiceLigne : entier
- indiceEtoile : entier
Sortie : void
Début

```
Soit nbEtoiles = 1
hauteur = SaisirNombreEntier()
Pour indiceLigne de 1 à hauteur avec un pas de +1 Faire
    Pour indiceEtoile de 1 à nbEtoiles avec un pas de +1 Faire
        AfficherCaractère('*')
    Fin Pour
    AfficherCaractère('\n')
    nbEtoiles = nbEtoiles + 2
Fin Pour
```

Fin

Algorithme 3 : AfficherTriangleIsocele
Principe : Saisir la hauteur. Parcourir chaque ligne du triangle (1 à hauteur) en maintenant à jour le nombre d'étoiles à afficher sur la ligne courante (commencer à 1 et +2 à chaque ligne) et le nombre d'espaces avant la première étoile (commencer à hauteur − 1 et -1 à chaque ligne).

```
Entrée :        /
Local :         - hauteur : entier
                - nbEspaces : entier
                - nbEtoiles : entier
                - indiceLigne : entier
                - indiceEspace : entier
                - indiceEtoile : entier
Sortie :        void
Début
    Soit nbEtoiles = 1
    Soit hauteur = SaisirNombreEntier()
    Soit nbEspaces = hauteur - 1
    Pour indiceLigne de 1 à hauteur avec un pas de +1 Faire
        Pour indiceEspace de 1 à nbEspaces avec un pas de +1 Faire
            AfficherCaractère(' ')
        Fin Pour
        Pour indiceEtoile de 1 à nbEtoiles avec un pas de +1 Faire
            AfficherCaractère('*')
        Fin Pour
        AfficherCaractère('\n')
        nbEspaces = nbEspaces - 1
        nbEtoiles = nbEtoiles + 2
    Fin Pour
Fin
```

Algorithme 4 : AfficherTableDeMultiplication
Principe : Saisir la table souhaitée. Parcourir les entiers de 0 à 10 et multiplier la table par cet indice. Afficher l'indice, la table et la multiplication des deux.

```
Entrée :        /
Local :         - table : entier
                - BORNE_INF : entier constant = 0
                - BORNE_SUP : entier constant = 10
                - e : entier
Sortie :        void
Début
    Soit table = SaisirNombreEntier()
    Pour e de BORNE_INF à BORNE_SUP avec un pas de +1 Faire
        AfficherChaîne("{e} * {table} = {e * table}")
    Fin Pour
Fin
```

Algorithme 5 : AfficherPlusPetit10Entiers
Principe : Saisir 10 entiers l'un après l'autre. Initialiser le minimum au premier entier saisi. Pour chacun des 9 suivants, converser le minimum entre le minimum local et l'entier nouvellement saisi. Afficher le minimum stocké.

```
Entrée :        /
Local :         - minium : entier
                - nombre: entier
                - i : entier
Sortie :        void
Début
    Soit minimum = SaisirNombreEntier()
    Pour i de 1 à 9 avec un pas de +1 Faire
        nombre = SaisirNombreEntier()
        Si nombre < minimum Alors
            minimum = nombre
        Fin Si
    Fin Pour
    AfficherEntier(minimum)
Fin
```

Algorithme 6 : AfficherSomme10Entiers
Principe : Saisir 10 entiers. Initialiser une somme à 0 et pour chaque entier l'ajouter à la somme. Afficher la somme.

```
Entrée :    /
Local :     - somme : entier
            - nombre : entier
            - i : entier
Sortie :    void
Début
    Soit somme = 0
    Pour i de 1 à 10 avec un pas de +1 Faire
        nombre = SaisirNombreEntier()
        somme = somme + nombre
    Fin Pour
    AfficherEntier(somme)
Fin
```

Algorithme 7 : DevinerNombreAleatoire
Principe : Saisir la hauteur. Parcourir chaque ligne du triangle (1 à hauteur) en maintenant à jour le nombre d'étoiles à afficher sur la ligne courante (commencer à 1 et +2 à chaque ligne).

```
Entrée :    - BORNE_INF : entier constant = 1
            - BORNE_SUP : entier constant = 100
            - TENTATIVES_MAX : entier constant = 10
Local :     - nombre : entier
            - nombreUtilisateur : entier
            - trouve : booléen
            - tentatives: entier
            - indiceTentative : entier
Sortie :    void
Début
    Soit nombre = GénérerNombreEntierAléatoire(BORNE_INF, BORNE_SUP)
    Soit trouve = faux
    Soit indiceTentative = 0
    Tant Que indiceTentative < TENTATIVES_MAX OU trouve == faux Faire
        nombreUtilisateur = SaisirNombreEntier()
        Si nombreUtilisateur < nombre Alors
            Afficher(plus grand !)
        Sinon Si nombreUtilisateur > nombre Alors
            Afficher(plus petit !)
        Sinon
            tentatives = indiceTentative
            trouve = vrai
        Fin Si
        indiceTentative = indiceTentatives + 1
    Fin Tant Que
    Si trouve == vrai Alors
        AfficherChaîne("Bravo, nombre trouvé en {tentatives} tentatives.")
    Sinon
        AfficherChaîne("Désolé, le nombre à trouver était {nombre}.")
    Fin Si
Fin
```

Algorithme 8 : ChercherNombreArmstrong
Principe : Parcourir les entiers de 100 à 999. Extraire le chiffre des centaines, dizaines et unités. Vérifier si centaines^3 + dizaines^3 + unites^3 est égal à l'indice courant. Si oui, l'afficher.

```
Entrée :        - BORNE_INF : entier constant = 100
                - BORNE_SUP : entier constant = 999
Local :         - centaines : entier
                - dizaines : entier
                - unites : entier
                - armstrong : entier
Sortie :        void
Début
        Soit armstrong = 0
        Pour i de BORNE_INF à BORNE_SUP avec un pas de +1 Faire
                centaines = i / 100 // division entière
                dizaines = (i / 10) % 10 // division entière puis modulo
                unites = (i / 100) % 10 // modulo puis modulo
                armstrong = centaines * centaines * centaines + dizaines * dizaines * dizaines + unites *
                        unites * unites
                Si armstrong == i Alors
                        AfficherEntier(i)
                Fin Si
        Fin Pour
Fin
```

Algorithme 9 : OperationsEntier
Principe : Stocker l'entier saisi, initialiser quitter à faux. Tant que quitter est vrai saisir un entier entre 1 et 4 et appliquer l'opération sur l'entier stocké. quitter = vrai si 4 est saisi.

```
Entrée :        /
Local :         - nombre : entier
                - operation : entier
                - continuer : booléen
Sortie :        void
Début
        Soit continuer = vrai
        Soit operation = -1
        nombre = SaisirNombreEntier()
        Tant Que quitter == faux Faire
                AfficherEntier(nombre)
                operation = SaisirNombreEntier()
                Si operation == 1 Alors
                        nombre = nombre + 1
                Sinon Si operation == 2 Alors
                        nombre = nombre * 2
                Sinon Si operation == 3 Alors
                        nombre = nombre – 4
                Sinon Si operation == 4 Alors
                        quitter = vrai
                Fin Si
        Fin Tant Que
Fin
```

Algorithme 10 : CompterTauxFrequenceA
Principe : Maintenir à jour le nombre de 'a' rencontrés et le nombre de caractères saisis. Tant que l'utilisateur ne saisi pas '*', lire le premier caractère saisi et mettre à jour les deux variables. Quant '*' est rencontré, calculer le taux et la fréquence et les afficher.

```
Entrée :        /
Local :         - nombreA : entier
                - nombreCaracteres : entier
                - quitter : booléen
                - frequenceA : réel
                - c: caractere
Sortie :        void
Début
    Soit c = '\0' // caractère null
    Soit nombreA = 0
    Soit frequenceA = 0.0
    Soit quitter = faux
    Tant Que quitter == faux Faire
        c = SaisirCaractère()
        Si c == '*' Alors
            quitter = vrai
        Sinon Si c == 'a' Alors
            nombreA = nombreA + 1
            nombreCaracteres = nombreCaracteres + 1
        Sinon
            nombreCaracteres = nombreCaracteres + 1
        Fin Si
    Fin Tant Que
    frequenceA = nombreA / nombreCaracteres
    AfficherChaîne("taux de a = {nombreA}/{nombreCaracteres}")
    AfficherChaîne("fréquence de a = {frequenceA}")
Fin
```

Algorithme 11 : AfficherFactorielle
Principe : Parcourir les entier de 2 à l'entier saisi en initialisant factorielle à 1. Multiplier la factorielle par l'entier actuel et mettre le résultat dans la variable factorielle.

```
Entrée :        /
Local :         - nombre : entier
                - factorielle : entier
                - i : entier
Sortie :        void
Début
    Soit nombre = 0
    Soit factorielle = 1
    nombre = SaisirNombreEntier()
    Si nombre < 0 Alors
        AfficherChaîne("erreur nombre négatif !")
    Sinon
        Pour i de 2 à nombre avec un pas de +1 Faire
            factorielle = factorielle * i
        Fin Pour
        AfficherEntier(factorielle)
    Fin Si
Fin
```

Algorithme 12 : AfficherNombresPremiers
Principe : Parcourir les entiers de 2 au nombre saisi moins 1. Pour chaque entier, vérifier si le nombre saisi est divisible par celui-ci, si oui il n'est pas premier. S'il n'est divisible par aucun alors il est premier.
Entrée : /
Local : - nombre : entier
- estPremier : booléen
- i : entier
Sortie : void
Début

```
Soit estPremier = vrai
nombre = SaisirNombreEntier()
Soit i = 2
Tant Que i < nombre ET estPremier == vrai Faire
    Si nombre est divisible par i Alors
        estPremier = faux
    Fin Si
    i = i + 1
Fin Pour
AfficherBooleén(estPremier)
```

Fin