Mise en pratique

- Structure alternative (instruction conditionnelle)
  - SI … ALORS … SINON … FSI
  - Structures alternatives imbriquées
  - Sélection choix multiples
- Répétition d'un traitement
  - Boucle POUR
  - Boucle TANT QUE … FAIRE
  - Boucle RÉPÉTER ... TANT QUE
  - Exercice 0
  - Exercice 1
  - Exercice 2
- Les fonctions et les procédures
  - Exercice 0
  - Exercice 1
- La récursivité
  - Exercice 0
  - Exercice 1
- Les tableaux
  - Exercice 0
- Les graphes
  - Exercice : Le renard, le lapin et les carottes

Structure alternative (instruction conditionnelle)

SI … ALORS … SINON … FSI

Saisir une valeur entière et afficher son double si cette donnée est inférieure à un seuil donné.

Algorithme SimpleOuDouble {Cet algorithme saisit une valeur entière et affiche son double si cette donnée est inférieure à un seuil donné}

{déclarations : réservation d'espace mémoire}
constantes seuil: entier ← 10
variables val: entier

début
{préparation du traitement}
afficher("Donnez-moi un entier :") {saisie de la valeur entière}
saisir(val)

si val < seuil **alors** {comparaison avec le seuil}
afficher("Voici son double :", val * 2)
sinon
afficher("Voici la valeur inchangée :", val)
fin si
fin

Structures alternatives imbriquées

Saisir une valeur entière et afficher :

- *“Reçu avec mention Assez bien” si une note est supérieure ou égale à 12,*
- *“Reçu avec mention Passable” si une note est supérieure à 10 et inférieure à 12,*
- *“Insuffisant” dans tous les autres cas.*

```
Algorithme Appreciation {Affiche une appréciation différente en fonction de la note}

{déclarations : réservation d'espace mémoire}
variables      note: entier

début
        {préparation du traitement}
        afficher("Donnez-moi une note :") {saisie de la valeur entière}
        saisir(note)

        si note ≥ 12 alors {comparaison avec un des seuils}
                afficher("Reçu avec mention Assez bien")
        sinon
                si note > 10 alors
                        afficher("Reçu avec mention Passable")
                sinon
                        afficher("Insuffisant")
                fin si
        finsi
fin
```

Sélection choix multiples

Reproduire avec l'instruction SELON l'équivalence de cette structure conditionnelle :

```
variables       abreviation: chaîne de caractères

début
        {préparation du traitement}
        afficher("Civilité (Mme / Mlle / M / Autre) :")
        saisir(abreviation)

        si abreviation = "Mme" alors
                afficher("Madame")
        sinon
                si abreviation = "Mlle" alors
                        afficher("Mademoiselle")
                sinon
                        si abreviation = "M" alors
                                afficher("Monsieur")
                        sinon
                                afficher("Transgenre")
                        fin si
                fin si
        fin si
fin
```

```
Algorithme Civilite {Affiche la civilité complète en fonction de l'abréviation}

{déclarations : réservation d'espace mémoire}
variables       abreviation: chaîne de caractères

début
        {préparation du traitement}
        afficher("Civilité (Mme / Mlle / M / Autre) :")
        saisir(abreviation)

        selon abreviation
                "Mme" : afficher("Madame")
                "Mlle" : afficher("Mademoiselle")
                "M" : afficher("Monsieur")
                "Autre" : afficher("Transgenre")
                autres : afficher("Choix non listé")
fin
```

Répétition d'un traitement

Boucle POUR

En fonction d'un nombre d'itérations saisi, faire la somme des entiers saisis et afficher le résultat de l'opération.

```
Algorithme FaireLeTotal {Fait la somme des entiers saisis sur un nombre d'itérations saisi}

{déclarations : réservation d'espace mémoire}
variables      nbIteration, cpt, val, totalVal: entiers

début
        {préparation du traitement}
        afficher("Combien de valeurs voulez-vous saisir ?")
        saisir(nbIteration)

        totalVal ← 0 {initialisation du total à 0 avant le cumul}
        pour cpt ← 1 à nbIteration faire
                afficher("Saisissez une valeur :")
                saisir(val)
                totalVal ← totalVal + val {cumul}
        fin pour
        afficher("Le total des ", nbIteration, " valeurs saisies est : ", totalVal)
fin
```

Boucle TANT QUE … FAIRE

Reproduire l'algorithme de la boucle POUR avec l'instruction TANT QUE … FAIRE.

Algorithme FaireLeTotal {Fait la somme des entiers saisis sur un nombre d'itérations saisi. *L'expression logique d'arrêt sera la saisie de la valeur "-1".*}

{déclarations : réservation d'espace mémoire}
constantes stop: entier ← -1
variables val, totalVal: entiers

début
totalVal ← 0 {initialisation du total à 0 avant le cumul}
afficher("Saisissez une valeur (-1 termine la saisie) :")
saisir(val)

tant que val différent de stop **faire**
totalVal ← totalVal + val {cumul}
afficher("Saisissez une valeur (-1 termine la saisie) :")
saisir(val) {relance}
fin tant que
afficher("Le total des valeurs saisies est : ", totalVal)
fin

Boucle RÉPÉTER ... TANT QUE

Saisir des données et s'arrêter dès que leur somme dépasse 500.

```
Algorithme SaisirMax500

{déclarations : réservation d'espace mémoire}
constantes    stop: entier ← 500
variables     val, totalVal: entiers

début
        totalVal ← 0 {initialisation du total à 0 avant le cumul}

        répéter
                afficher("Saisissez une valeur :")
                saisir(val)
                totalVal ← totalVal + val {cumul}
        tant que totalVal ≤ stop
fin
```

Exercice 0

Écrire un programme qui échange la valeur de deux variables.
Exemple, si a ← 2 et b ← 5, le programme donnera a ← 5 et b ← 2.

Algorithme InverserVal

{déclarations : réservation d'espace mémoire}
variables a, b, tmp : entiers

début
a ← 2
b ← 5
tmp ← a
a ← b
b ← tmp
fin

ou

Algorithme InverserVal

{déclarations : réservation d'espace mémoire}
variables a, b: entiers

début
a ← 2
b ← 5
a ← a + b
b ← a - b
a ← a - b
fin

Exercice 1

Écrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de coté le cas où le produit est nul).
Attention toutefois : on ne doit pas calculer le produit des deux nombres.

```
Algorithme ConnaitrePositifNegatif

{déclarations : réservation d'espace mémoire}
variables       a, b : entiers

début
        afficher("Saisissez un premier nombre entier :")
        saisir(a)
        afficher("Saisissez un deuxième nombre entier :")
        saisir(b)

        si (a > 0 ET b > 0) OU (a < 0 ET b < 0) alors
                afficher("Le produit des nombres est positif")
        sinon
                afficher("Le produit des nombres est négatif")
        fin si
fin
```

Exercice 2

Écrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne.
En cas de réponse supérieure à 20, on fera apparaître un message : "Plus petit !" , et inversement, "Plus grand !" si le nombre est inférieur à 10.

```
Algorithme PlusPetitPlusGrand

{déclarations : réservation d'espace mémoire}
variables       val : entier

début
        répéter
                afficher("Saisissez une valeur :")
                saisir(val)
                si (val < 10) alors
                        afficher("Plus grand !")
                sinon
                        si (val > 20) alors
                                afficher("Plus petit !")
                        fin si
                fin si
        tant que (val < 10 OU val > 20)
fin
```

Les fonctions et les procédures

Exercice 0

Écrire un algorithme qui échange la valeur de deux variables en s'appuyant sur une fonction ou une procédure.
Exemple, si a ← 2 et b ← 5, le programme donnera a ← 5 et b ← 2.

```
Algorithme InverserVal

{déclarations : réservation d'espace mémoire}
variables       a, b: entiers
procédure inverser(E/S val1, E/S val2 : entiers)

        début
                val1 ← val1 + val2
                val2 ← val1 - val2
                val1 ← val1 - val2
        fin

début
        a ← 2
        b ← 5
        inverser(a,b)
fin
```

Exercice 1

Écrire le sous-algorithme de la fonction “moyenne” qui renvoie la moyenne de deux entiers. Écrire l’algorithme qui contient la déclaration de la fonction moyenne et des instructions qui appellent cette fonction.

```
Algorithme calculMoyenne

{déclarations : réservation d’espace mémoire}
variables       note1, note2 : réels

fonction moyenne(x, y : réels) : réel
        début
                retourner (x  + y) / 2
        fin

début
        afficher(“Saisissez deux notes à la suite :”)
        saisir(note1)
        saisir(note2)
        afficher(“La  moyenne de A et B est : ”, moyenne(note1, note2))
fin
```

La récursivité

Exercice 0

En utilisant une fonction récursive, écrire un algorithme qui détermine le terme U_n de la suite de Fibonacci définit comme suit :
$U_0 = 0$
$U_1 = 1$
$U_n = U_{n-1} + U_{n-2}$, $n >= 2$

```
Algorithme suiteFibonacci

{déclarations : réservation d'espace mémoire}
variables       n : entier
fonction fibonacci(k : entier) : entier
        début
                selon k
                        0 : retourner 0
                        1 : retourner 1
                        autres : retourner fibonacci(k - 1) + fibonacci(k - 2)
        fin

début
        afficher("Saisissez un nombre entier positif :")
        saisir(n)

        afficher("Le terme U_n de la suite de Fibonacci avec n (fourni) égal ", n, " est : ", fibonacci(n))
fin
```

Exercice 1

En utilisant une fonction récursive, écrire un algorithme qui écrit la structure d'un tableau HTML (<table><tr><td></td></tr></table>) en permettant l'écriture de plusieurs lignes et plusieurs cellules si l'utilisateur indique qu'il souhaite poursuivre chaque étape.

```
Algorithme ecrireTableHTML

procédure ecrireTD()
        {déclarations : réservation d'espace mémoire}
        variables       confirm : énuméré {oui, non}
        début
                afficher("Confirmez-vous l'écriture d'une cellule ?")
                saisir(confirm)
                si (confirm = "oui") alors
                        afficher("<td></td>")
                        ecrireTD()
                fin si
        fin

procédure ecrireTR()
        {déclarations : réservation d'espace mémoire}
        variables       confirm : énuméré {oui, non}
        début
                afficher("Confirmez-vous l'écriture d'une ligne ?")
                saisir(confirm)
                si (confirm = "oui") alors
                        afficher("<tr>")
                        ecrireTD()
                        afficher("</tr>")
                        ecrireTR()
                fin si
        fin

début
        afficher("<table>")
        ecrireTR()
        afficher("</table>")
fin
```

Les tableaux

Exercice 0

Écrire un programme qui affiche en ordre croissant les notes d'une promotion de 10 élèves, suivies de la note la plus faible, de la note la plus élevée et de la moyenne.

```
Algorithme notesPromo
    {déclarations : réservation d'espace mémoire}
    constantes   stop: entier ← -1
    variables    somme: réel
                 cptEleves, saisie: naturel
    type         notes : tableau[1 … MAX] de réels
    fonction minimum(t : tableau[1 … MAX] d'entiers; rang, nbElements :
    naturels) : naturel
        variables    i, indice : naturels
        début
            indice ← rang
            pour i ← rang+1 à nbElements faire
                si t[i] < t[indice] alors
                    indice ← i
                fin si
            fin pour
            retourner indice
        fin
    procédure inverser(E/S val1, E/S val2 : entiers)
        variables    tmp : naturel
        début
            tmp ← val1
            val1 ← val2
            val2 ← tmp
        fin
    procédure triParMinimumSuccessif(E/S t : tableau[1 … MAX]
    d'entiers, E nbElements : naturel)
        variables    i, indice : naturels
        début
            pour i ←1 à nbElements-1 faire
                indice ← minimum(t, i, nbElements)
                si i ≠ indice alors
                    inverser(t[i],t[indice])
                fin si
            fin pour
        fin
```

```
        fonction moyenne(total : réel, compteur : naturel) : réel
                début
                        retourner total / compteur
                fin

début
        {préparation du traitement}
        cptEleves ← 0 {initialisation du nombre d'élèves}
        somme ← 0.0 {initialisation du total à 0 avant le cumul}
        répéter
                afficher("Saisissez une note pour l'élève n° :", cptEleves+1)
                saisir(saisie)
                si saisie > stop alors
                        cptEleves ← cptEleves + 1 {incrémentation du nombre d'élèves}
                        notes[cptEleves] ← saisie
                        somme ← somme + notes[cptEleves] {cumul}
                fin si
        tant que saisie > stop

        triParMinimumSuccessif(notes,cptEleves)
        afficher("Notes de la promotion par ordre croissant : ")

        pour i ← 1 à cptEleves faire
                afficher(notes[i])
        fin pour

        afficher("Note la plus faible ", notes[1])
        afficher("Note la plus élevée ", notes[10])
        afficher("Moyenne des notes ", moyenne(somme, cptEleves))
fin
```

Les graphes

Exercice : Le renard, le lapin et les carottes

Un fermier doit passer la rivière dans une barque juste assez grande pour lui et son renard, ou lui et son lapin, ou lui et ses carottes. Les carottes seront mangées s'il les laisse seules avec le lapin, et le lapin sera mangé s'il le laisse seul avec le renard. Comment faire passer tout ce monde sans dégâts ?

1. Le fermier transporte le lapin sur l'autre rive.
   Le renard ne mangeant pas les carottes, tout va bien.
2. Le fermier revient.
3. Le fermier transporte le renard sur l'autre rive et reprend le lapin.
   Le renard risquant de manger le lapin, le fermier évite ainsi ce risque en le ramenant avec lui.
4. Le fermier dépose le lapin et prend les carottes pour les transporter sur l'autre rive.
5. Le fermier dépose les carottes.
   Il ne risque rien puisque le renard ne se nourrit pas de carottes.
6. Le fermier revient seul.
7. Le fermier transporte à nouveau le lapin sur l'autre rive.

Tout le monde a été transporté sans risque que l'un ne mange l'autre.

Cette situation peut être modélisée à l'aide d'un graphe.
Désignons par P le passeur, par C la chèvre, par X le chou et par L le loup. Les sommets du graphe sont des couples précisant qui est sur la rive initiale, qui est sur l'autre rive. Ainsi, le couple (PCX,L) signifie que le passeur est sur la rive initiale avec la chèvre et le chou (qui sont donc sous surveillance), alors que le loup est sur l'autre rive. Une arête relie deux sommets lorsque le passeur peut passer (sic) d'une situation à l'autre. En transportant la chèvre, le passeur passe par exemple du sommet (PCX,L) au sommet (X,PCL). Le graphe ainsi obtenu est biparti : les sommets pour lesquels le passeur est sur la rive initiale ne sont reliés qu'aux sommets pour lesquels le passeur est sur l'autre rive…
Naturellement, on ne considèrera pas les sommets dont l'une des composantes est CX ou LC car ces situations sont interdites.
Il suffit ensuite de trouver un chemin (le plus court par exemple) entre la situation initiale (PCXL,-) et la situation finale souhaitée (-,PCXL). La figure suivante donne un tel chemin :