Programmation

Cursus 1

Level 1

Algorithmie

Module 2 Partie 1

Objectifs

- Aborder la programmation.

Présentation d'un langage universel.

- Organiser un programme.
- Utiliser des données élémentaires.
- Structurer les étapes de résolution d'un problème.

Sommaire

- Les fonctions
- Les procédures
- Les Tableaux

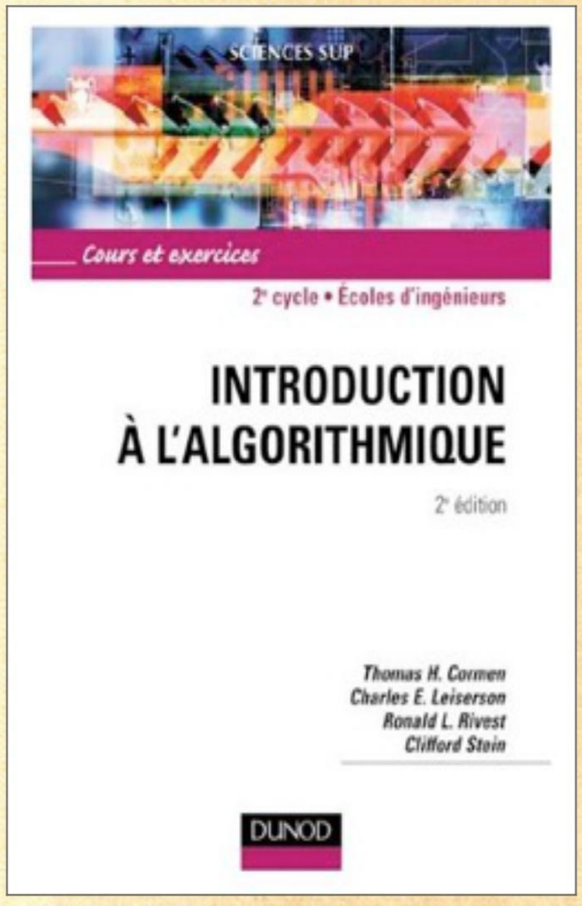
Références Internet

Références Bibliographiques

Introduction à l'algorithmique :

Cours et exercices

de Cormen, Leiserson, Rivest

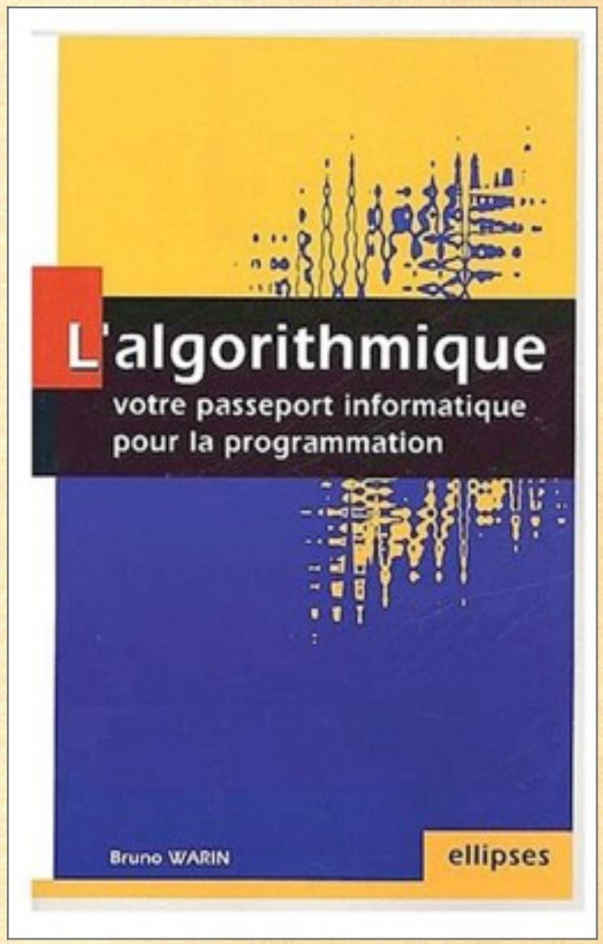


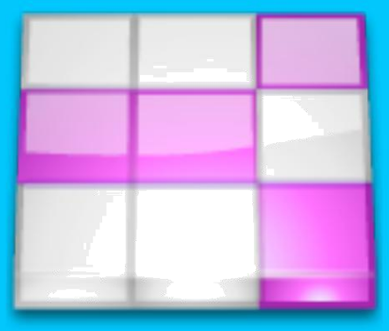
Références Bibliographiques

L'algorithmique :

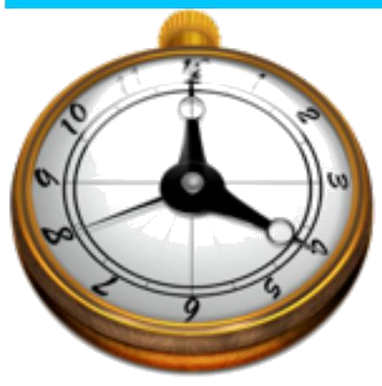
Votre passeport informatique pour la programmation

de Bruno Warin





Liens pédagogiques



Pré-requis:

Historique et modèle de Von Neumann

Nécessaire pour :

Réaliser tout programme.

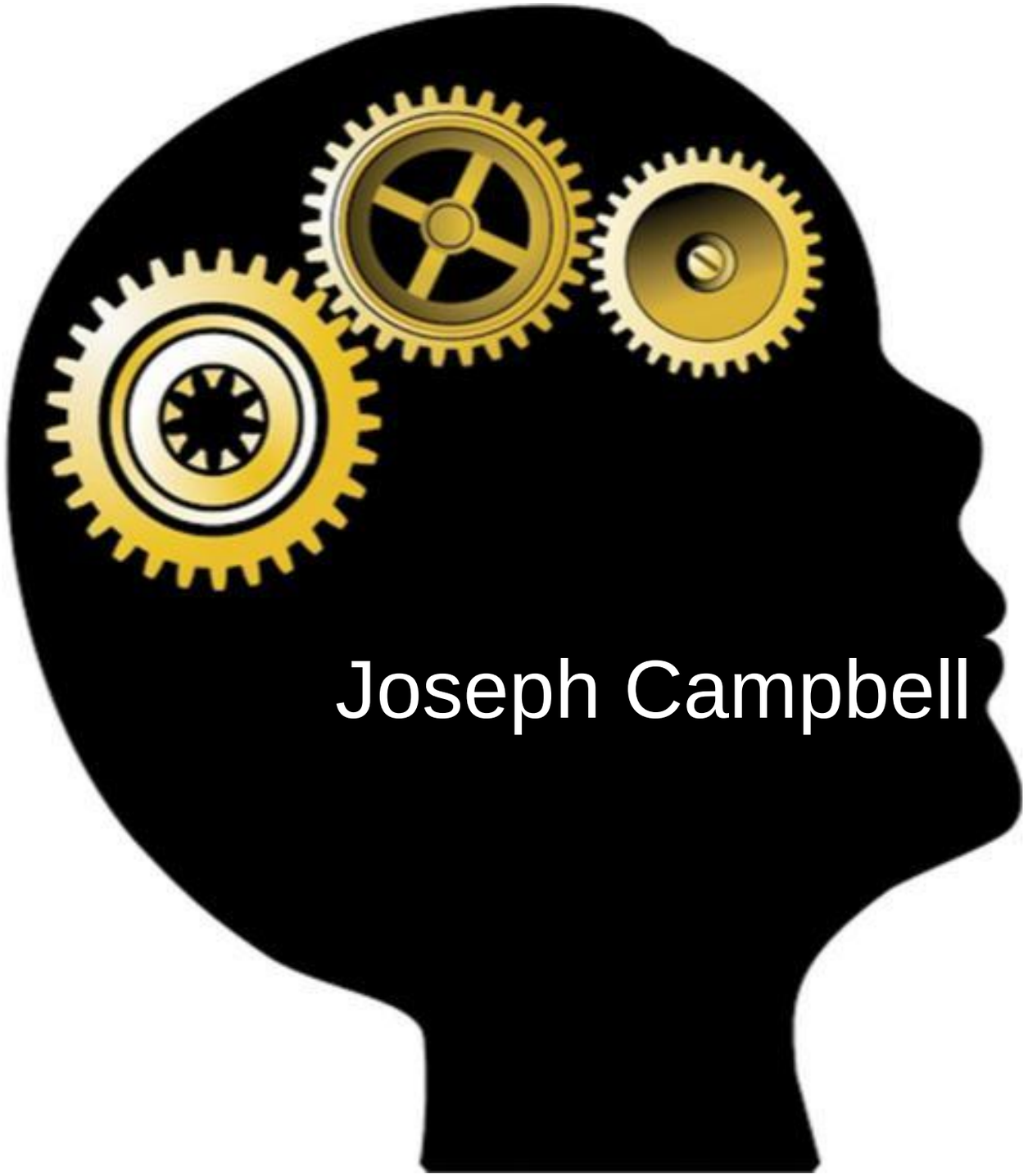
Algorithmie

FONCTIONS ET PROCEDURES

Algorithmie
Structure de données

- Présentation
- Les 5 structures de données élémentaires
- Opérateurs et symboles associés
- Opérations élémentaires
- Les structures de données avancées

Citation



«Les ordinateurs sont comme les dieux de l’Ancien Testament : avec beaucoup de règles, et sans pitié.»

Fonctions et procédures

Tous les algorithmes que nous avons étudiés jusqu'à présent étaient constitués d'un seul bloc (dit principal). Dans certains cas le programmeur peut vouloir décrire et sauvegarder un enchaînement d'actions qui n'existe pas de façon standard dans le compilateur (ou l'interpréteur). Cette sauvegarde lui permettra d'utiliser cet enchaînement autant de fois qu'il en aura besoin dans le programme, et cela au moyen d'un simple appel.

Cet usage est pratique lorsqu'on fait face à un algorithme complexe. Le programmeur décompose donc le problème en mini-blocs ayant chacun un rôle bien précis. Ces mini-blocs sont des sous-programmes (sous-algorithmes) appelés **Procédures** ou **Fonctions**.

Les Procédures

I-1) Définition :

Comme son nom l'indique, une procédure c'est le déroulement classique d'un processus. Elle peut être judiciaire, culinaire, mathématique, informatique, etc. Mais dans tous les cas elle est toujours déclenchée par quelque chose ou bien par quelqu'un. On peut également dire qu'une procédure est une fonction qui ne renvoie pas de résultat.

I-2) Structure :

Une procédure est un mini-programme qu'on déclare en général dans la partie réservée aux variables, ce afin de pouvoir utiliser les variables globales. Étant donné qu'il s'agit d'un bloc à part entière, elle possèdera éventuellement un en-tête, une série de traitements, et une gestion des résultats tout comme l'algorithme qui la contient. En outre, une procédure peut également recevoir des arguments qui lui seront alors passés en paramètres.

La déclaration d'une procédure se fait comme suit :

Les Procédures

PROCEDURE **<nom_de_la_procedure> [** (liste des paramètres : type) **]**

Var : liste des variables

DEBUT {*corps de la procédure***}**

<LISTE DES ACTIONS> {*la liste ne doit pas être vide*}

FINPROCEDURE

Les Procédures

Remarques :

Lorsque la déclaration ci-dessus est faite, il suffit ensuite d'écrire le nom de la procédure dans le bloc principal pour déclencher la liste des actions décrites.

Une procédure peut appeler d'autres sous-programmes définis avant elle.

La liste des paramètres est facultative. Mais quand elle existe, ces paramètres sont déclarés de la même façon qu'on déclare une série de variables de différents types.

Les variables déclarées à l'intérieur de la procédure sont inutilisables à l'extérieur du bloc. Si leur type est prédéfini, alors ce type sera déclaré dans l'en-tête du bloc principal. Idem pour les paramètres de la procédure au cas où il en existe.

Les Procédures

EXEMPLE : voici un algorithme utilisant une procédure qui fait une somme de N nombres.

Algorithme essai_procedure

Const N<-100 :Entier

Procedure Somme

Var

i, s : entier

DEBUT

s<-0

POUR i DE 1 A N FAIRE PAS DE 1

s <-s + i

FINPOUR

Ecrire(‘’La somme des ‘’& N & ‘‘premiers nombres est ‘’& s)

FINPROCEDURE

Les Procédures

```
//Programme principal

DEBUT

    //Instructions
    somme

FIN
```

Les Fonctions

II-1) Définition :

De la même manière qu'une procédure, une fonction est un sous-programme destiné à effectuer un enchaînement de traitements à l'aide d'un simple appel. Cependant, une fonction a pour but principal d'effectuer un calcul puis de renvoyer un résultat.

II-2) Structure :

Une fonction est un mini-programme qu'on déclare dans la partie réservée aux variables, ce afin de pouvoir utiliser les variables globales. Étant donné qu'il s'agit d'un bloc à part entière, elle possèdera éventuellement un en-tête, une série de traitements, et une gestion des résultats tout comme l'algorithme qui la contient. En outre, une fonction peut également recevoir des arguments qui lui seront alors passés en paramètres. Déclaration :

Les Fonctions

FONCTION **<nom_de_la_fonction> [** (liste des paramètres : type) **] : type_fonction**

Var :
liste des variables

DEBUT {*corps de la fonction***}**

<LISTE DES ACTIONS> {*la liste ne doit pas être vide*}

nom_de_la_fonction <- résultat_des_calculs

FINFONCTION

Les Fonctions

Étant donné qu'une fonction a pour but principal de renvoyer une valeur, il est donc nécessaire de préciser le type de la fonction qui est en réalité le type de cette valeur.

Une fonction peut appeler d'autres sous-programmes définis avant elle.

On peut utiliser le nom d'une fonction presque comme une variable globale, puisqu'elle renferme une valeur (affectation, affichage, mais pas lecture)

La liste des paramètres est facultative. Mais quand elle existe, ces paramètres sont déclarés de la même façon qu'on déclare une série de variables de différents types.

Les variables déclarées à l'intérieur de la fonction sont inutilisables à l'extérieur du bloc. Si leur type est prédéfini, alors ce type sera déclaré dans l'en-tête du bloc principal. Idem pour les paramètres de la fonction au cas où il en existe.

À l'intérieur du sous-programme le nom de la fonction ne doit figurer qu'en recevant le résultat final ; sinon la fonction risque de s'appeler indéfiniment et donc de provoquer un bug (voir **Récursivité**).

Les Fonctions

EXEMPLE : voici un algorithme utilisant une fonction qui calcule la somme de N nombres.

```
Algorithme essai_fonction

Const N<-100


Fonction Somme : entier

    Var
        s : Entier
        i : Entier

    DEBUT
        s<-0

        POUR i DE 1 A N FAIRE
            s<-s+i
        FINPOUR
        Somme<-s //renvoi de s dans l'en-tête de la fct somme
    FINFONCTION

//Programme principal
  DEBUT

  Ecrire(''La somme des n premiers nombres entiers est ''& somme)

  FIN
```

Les paramètres

Généralités :

Avant de voir comment un sous-bloc exploitera ses paramètres, voici quelques notions : Une **variable globale** est une variable définie dans l'en-tête du programme principal.
Elle est utilisable dans n'importe quel sous-programme sans nécessité de redéfinition. Toutefois, si dans un sous-bloc il existe une variable qui porte le même nom que la variable globale, alors c'est cette variable locale qui sera considérée à l'intérieur du sous-bloc.

Une **variable locale** est une variable définie à l'intérieur d'un sous-programme. Sa portée (visibilité) est limitée au bloc qui la contient. Il serait donc erroné de l'utiliser dans le bloc principal ou dans un autre sous-bloc appartenant à l'algorithme.

Les paramètres

Considéré comme une variable locale, un **paramètre** est une valeur du bloc principal dont le sous-programme a besoin pour

Exécuter avec des données réelles l'enchaînement d'actions qu'il est chargé d'effectuer. On distingue 2 types de paramètres :

- Les paramètres formels qui sont la définition du nombre et du type de valeurs que devra recevoir le sous-programme pour se mettre en route avec succès. On déclare les paramètres formels pendant la déclaration du sous-programme.
- Les paramètres effectifs qui sont des valeurs réelles (constantes ou variables) reçues par le sous-programme au cours de l'exécution du bloc principal. On les définit indépendamment à chaque appel du sous-programme dans l'algorithme principal.

Les paramètres

Fonctionnement et utilisation des paramètres :

Lorsque la déclaration d'un sous-programme comporte des paramètres formels, ceux-ci, doivent être représentés chacun par son identificateur ainsi que par son type.

Ainsi pendant la construction de l'algorithme principal, il faudra toujours veiller à ce que chaque appel du sous-programme soit suivi d'une liste de paramètres effectifs correspondant (en nombre, rang, et type) à la liste des paramètres formels. Cependant les noms des paramètres de même ordre ne sont pas obligatoirement identiques.

On a vu plus haut qu'un paramètre effectif pouvait être une constante ou une variable. Lorsqu'il s'agit d'une variable, 2 cas de figures se proposent :

Les Paramètres

1) Utiliser la valeur de la variable et à la sortie du sous-programme lui restituer cette valeur malgré les éventuelles modifications subies. On parle de **passage de paramètre par valeur**.

2) Utiliser la variable elle-même et lui attribuer dans le bloc principal les modifications rencontrées dans le sous-programme. On parle de **passage de paramètre par adresse**.

NB : un sous-programme avec paramètres est très utile parce qu'il permet de répéter une série d'opérations complexes pour des valeurs qu'on ne connaît pas à l'avance.

Les Paramètres

A- Passage de paramètres par valeur

Comme on l’a dit, passer un paramètre par valeur revient à n’utiliser que la valeur de la variable au moment où elle est passée en paramètre. À la fin de l’exécution du sous-programme, la variable conservera sa valeur initiale.

Syntaxe :

PROCEDURE **<nom_procédure>** (**param1 :**type1 **;** **param2, param3 :**type2)

FONCTION<nom_Fonction>(**param1 :**type1 **; param2, param3 :**type2) :Type

Les paramètres

EXEMPLE : écrire un algorithme dans lequel une fonction retourne le produit de 2 entiers et une procédure affiche le résultat du produit de 2 entiers.

Algorithme produit

Procedure ProMultiplier(nb1, nb2 : entier)

Var

Res :ENTIER

DEBUT

Res<-nb1*nb2
Ecrire(‘Le résultat de ce produit est ‘&Res)

FINPROCEDURE

Les Paramètres

Fonction FctMultiplier(nb1, nb2 : entier) :entier

Var

Res :entier

DEBUT

Res<-nb1*nb2
FctMultiplier<-Res //Retourner Res

FINFONCTION

Les Paramètres

//Programme principal
VAR

Res **: ENTIER**
nbre1, nbre2 : **ENTIER**

DEBUT

Ecrire(‘’Entrez les deux nombres qu’’il faudra multiplier’’)

Lire(nbre1, nbre2)

ProMultiplier(nbre1, nbre2)

Res<-FctMultiplier(nbre1,nbre2)

Ecrire(‘’Le resultat de la fonction produit :’’&Res)

FIN

Les Paramètres

B- Passage de paramètres par adresse (ou par variable)

Ici, il s'agit non plus d'utiliser simplement la valeur de la variable, mais également son emplacement dans la mémoire (d'où l'expression « par adresse »). En fait, le paramètre formel se substitue au paramètre effectif durant le temps d'exécution du sous-programme. Et à la sortie il lui transmet sa nouvelle valeur.

Un tel passage de paramètre se fait par l'utilisation du mot-clé **var** (uniquement sur le paramètre formel, et jamais sur un paramètre effectif).

Syntaxe : FONCTION **<nom_fonction>** (VAR **param1 :**type1 **; param2 :**type2) : entier

NB : Il est possible de passer des paramètres par valeur et adresse dans la même fct ou procédure

EXEMPLE : écrire un algorithme dans lequel une fonction utilise le résultat d'une procédure
'Produit de 2 entiers' pour en calculer le carré. Calculer ensuite le carré de ce carré.

Algorithme produit_carre

Procedure multiplier(nbre1, nbre2 : entier)

VAR

Produit :entier

DEBUT

produit <-nbre1*nbre2

Ecrire(‘Le résultat de ce produit est ‘& produit)

FINPROCEDURE

Fonction carre(**var** racine : réel) : réel

DEBUT

racine<-racine*racine

carre<-racine

FINFONCTION

VAR

nbre1,nbre2 :entier

produit :réel

DEBUT

Ecrire(‘entrer les deux nombres qu’’il faudra multiplier’)

Lire(nbre1, nbre2)

Multiplier(nbre1, nbre2)

Ecrire(‘Le carré de cette multiplication est ‘& carre(produit))

Ecrire(‘Le carré de ce carré sera alors ‘& carre(produit))

FIN

Les Tableaux

I-1) Les tableaux à une dimension (vecteurs) :

Syntaxe : **Type <nom_du_tableau> = Tableau[1..N] de <TypeElt>**

I-2) Les tableaux à deux dimension :

Syntaxe : **Type <nom_du_tableau> = Tableau[1..N,1..N] de <TypeElt>**

En algo les tableaux démarrent à 1 quand on ne précise pas.

Exemple 1

Algorithme Tableau

```
CONST
        MAX: ENTIER <- 50
VAR
    T1 : Tableau[1..MAX] de CHAINE
    T2 : Tableau[1..5,1..10] de ENTIER

DEBUT
     T1[1]<-'a'
    ECRIRE T1[1]
    T2[2,7]<-12
    ECRIRE T2[2,7]

FIN
```

Les Tableaux

Exemple 2
Programme qui initialise un tableau d'entiers à 1 dim et 2 dim avec des zéros

Algorithme INITTAB

CONST

MAX: ENTIER <- 50

Procedure InitTabT1(T1 : Tableau[1..MAX] de ENTIER)

Var

i :ENTIER

DEBUT

```
POUR i de 1 A MAX FAIRE
    T1[i]<-0
FINPOUR
```

FINPROCEDURE

Les Tableaux

Procedure InitTabT2(T2 : Tableau[1..5,1..10] de ENTIER)

Var

i :ENTIER

j :ENTIER

DEBUT

```
POUR i de 1 A 5 FAIRE
    POUR j de 1 A 10 FAIRE
        T2[i,j]<-0
    FINPOUR
FINPOUR
```

FINPROCEDURE

Les Tableaux

//Programme principal
VAR
T1 : Tableau[1..MAX] de ENTIER
T2 : Tableau[1..5,1..10] de ENTIER

DEBUT

InitTabT1(T1)
InitTabT2(T2)

FIN

Attention :
Une chaine est un tableau de 1 dimension en algo.

s<-''Bonjour''
s[2]<-'*'
s vaut B*njour