ANNÉE SCOLAIRE 2023/2024

COURS D'ALGORITHMIQUE

Pape Abdoulaye **BARRO**
Docteur en Informatique et Télécommunications
Spécialiste en Télémétrie & Systèmes Intelligents

SOUS ALGORITHMES

SOUS ALGORITHMIQUE
DÉFINITION

Lorsque l'Algorithme à écrire devient de plus en plus important (volumineux), des difficultés d'aperçu global sur son fonctionnement se posent. Il devient très difficile de coder et de devoir traquer les erreurs en même temps.

- Il est donc utile de découper le problème en de sous problème;
- de chercher à résoudre les sous problèmes (sous-algorithmes);
- puis de faire un regroupement de ces sous-algorithmes pour reconstituer une solution au problème initial.

Un sous-algorithme est une partie d'un algorithme. Il est d'habitude déclaré dans la partie entête et est réutiliser dans le corps de l'algorithme.

- Un sous-algorithme est un algorithme. Il possède donc les même caractéristiques d'un algorithme.

ALGORITHMIQUE
SOUS ALGORITHMES

- Un sous-algorithme peut utiliser les variables déclarés dans l'algorithme. Dans ce cas, ces **variables** sont dites **globales**. Il peut également utiliser ses propres variables. Dans ce cas, les variables sont dites **locales**. Ces dernières ne pourront alors être utilisable qu'à l'intérieur du sous-algorithme et nulle part ailleurs (**notion de visibilité**). Ce qui signifie que leur allocation en mémoire sera libérer à la fin de l'exécution du sous-algorithme.
- Un sous-algorithme peut être utilisable plusieurs fois avec éventuellement des paramètres différents.
- **Un sous-algorithme peut se présenter sous forme de fonction ou de procédure:**
  - Une **fonction** est un sous-algorithme qui, à partir de donnée(s), calcul et rend à l'algorithme un et un seul résultat;
  - alors qu'en général, une **procédure** affiche le(s) résultat(s) demandé(s).

ALGORITHMIQUE
SOUS ALGORITHMES

- **Syntaxe d'une fonction**

Fonction Nom_Fonction (Nom_Paramètre:Type_paramètre;…): **type_Fonction**
Variable
    Nom_variable : Type_variable ; // Variables locales
    …
Début
    …
    Instructions ;
    …
    Nom_Fonction←resultat ; // Corps de la fonction
Fin

Un appel de fonction est une expression d'affectation de manière à ce que le résultat soit récupéré dans une variable globale de même type:

Nom_variable_globale ← Nom_Fonction (<paramètres>)

ALGORITHMIQUE
SOUS ALGORITHMES

- **Exemple de fonction**

```
Algorithme Calcul_des_n_premiers_nombres_entiers
Variable
        I, Som, N : entier  {variables globales}

        Fonction Somme: entier
        Variable
                S : entier {variable locale}
        Debut {Début de la fonction}
                S ←0
                Pour I ← 1 à N Faire
                        S ←S + I
                FinPour
                Somme ← S
        Fin {Fin de la Fonction}

Debut {Début de l'algorithme}
        Ecrire ('Donner une valeur ')
        Lire(N)
        Som ← Somme() {appel de la fonction}
        Ecrire ('La somme des ', N, 'premiers nombres est', Som)
Fin {Fin de l'algorithme}
```

ALGORITHMIQUE
SOUS ALGORITHMES

- **Syntaxe d'une procédure**

```
Procedure Nom_Procedure (Nom_Paramètre:Type_praramètre;…)
Variable
    Nom_variable : Type_variable ;      // Variables locales
    …
Début
    …
    Instructions ;                      // Corps de la fonction …
Fin
```

L'appel d'une procédure peut être effectué en spécifiant, au moment souhaité, son nom et éventuellement ses paramètres; cela déclenche l'exécution des instructions de la procédure:

Nom_Procedure (<paramètres>)

ALGORITHMIQUE
SOUS ALGORITHMES

- **Exemple de procédure**

```
Algorithme Calcul_des_n_premiers_nombres_entiers
Variable
        I, Som, N : entier {variable globale}
        Procedure Somme
        Debut {Début de la procédure}
                Som ←0 {variable locale}
                Pour I ← 1 a N Faire
                        Som ←Som + I
                FinPour
                Ecrire ('La somme des ', N, 'premiers nombres est', Som)
        Fin {Fin de la Fonction}
Debut {Début de l'algorithme}
        Ecrire ('Donner une valeur ')
        Lire(N)
        Somme() {appel de la procédure}
Fin {Fin de l'algorithme}
```

ALGORITHMIQUE
SOUS ALGORITHMES

- **Mode de passages de paramètres**

On distingue deux types de passage de paramètres: par valeur et par variable (dite aussi par référence ou encore par adresse).

- Passage par valeur

Le mode de passage par valeur qui est le mode par défaut, consiste à copié la valeur des paramètres effectifs dans les variables locales issues des paramètres formels de ma fonction ou de la procédure appelée.

- **Dans ce mode, nous travaillons pas directement avec la variable, mais avec une copie. Ce qui veut dire que le contenu des paramètres effectifs n'est pas modifié. À la fin de l'exécution du sous-algorithme, la variable conservera sa valeur initial.**
- **Syntaxe:**
  - **Procedure nom_procédure (param1:type1; param2, param3:type2)**
  - **Fonction nom_fonction (param1:type1; param2:type2):Type_fonction**

- **Rappel:**
  - **Paramètre formel**

    Un paramètre d'entrée d'un sous algorithme utilisé à l'intérieur de la fonction appelée.
  - **Paramètre effectif**

    Un paramètre d'appel d'un sous algorithme utilisé à l'extérieur de la fonction appelée.

ALGORITHMIQUE
SOUS ALGORITHMES

- **Exemple de mode de passages de paramètres** par valeur

```
Algorithme valeur_absolue_d-un_nombre_entier
Variable
        val: entier
        Procedure Abs(nombre: entier)
        Debut {Début de la procédure}
                Si (nombre<0) alors
                        nombre ← - nombre
                FinSi
                Ecrire (nombre)
        Fin {Fin de la Fonction}
Debut {Début de l'algorithme}
        Ecrire ('Saisir une valeur');
        Lire (val)
        Abs (val)
        Ecrire (val)
Fin {Fin de l'algorithme}
```

  - Ici, **val** reprend sa valeur initiale. Il a juste servi de données pour **Abs**.

ALGORITHMIQUE
SOUS ALGORITHMES

- Passage par adresse

Dans le mode de passage par variable, il s'agit pas simplement d'utiliser la valeur de la variable, mais également son emplacement mémoire.

- **Le paramètre formel se substitue au paramètre effectif tout au long de l'exécution du sous-algorithme et à la sortie, il lui transmet sa nouvelle valeur.**
- **Un tel passage se fait par l'utilisation du mot-clé Var.**
- **Syntaxe:**
  - **Procedure nom_procédure (Var param1:type1 ; param2, param3:type2)**
  - **Fonction nom_fonction (Var param1:type1; param2:type2):Type_fonction**

ALGORITHMIQUE
SOUS ALGORITHMES

- **Exemple de mode de passages de paramètres** par adresse

```
Algorithme valeur_absolue_d-un_nombre_entier
Variable
        val: entier
        Procedure Abs(Var nombre: entier)
        Debut {Début de la procédure}
                Si (nombre<0) alors
                        nombre ← - nombre
                FinSi
                Ecrire (nombres)
        Fin {Fin de la Fonction}
Debut {Début de l'algorithme}
        Ecrire ('Saisir une valeur')
        Lire (val)
        Abs (val)
        Ecrire (val)
Fin {Fin de l'algorithme}
```

  - Ici, val prend une nouvelle valeur.

ALGORITHMIQUE
CAS PRATIQUES N°7

Exercice 31:

- Ecrire une procédure permettant de calculer la somme des n premiers nombres entiers, n étant saisie au clavier.
- Ecrire une procédure permettant de chercher la valeur absolue d'un nombre entier saisi au clavier.
- Ecrire une procédure permettant de faire la permutation de deux entiers a et b.

Exercice 32 :

- Ecrire un programme utilisant une fonction qui reçoit en argument 2 nombres flottants et un caractère (opération), et qui fournit le résultat du calcul demandé.
- Proposer le même programme mais cette fois ci, la fonction ne disposera plus que de 2 arguments en nombres flottants. L'opération est précisée, cette fois, à l'aide d'une variable globale.

Exercice 33 :

- Ecrire un programme faisant appel à une fonction qui ne renvoie aucune valeur et qui détermine la valeur maximale et la valeur minimale d'un tableau d'entiers.

Exercice 34 :

- Écrire un programme qui permet de saisir deux entiers n et p ($1 \leq p \leq n$) et de calculer puis afficher le nombre de combinaison de p éléments parmi n : CNP
  sachant que: $C_n^p = n! / (p! * (n-p)!)$

Exercice 35 :

- Un nombre est dit perParfait s'il est parfait (égale à la somme de ces diviseurs) et son successeur est premier. Exemple : 6 est perparfait car il est parfait (6=3+2+1) et son successeur 7 est premier.
- Déterminer tous les nombres Perparfaits <=100

Affaires à suivre