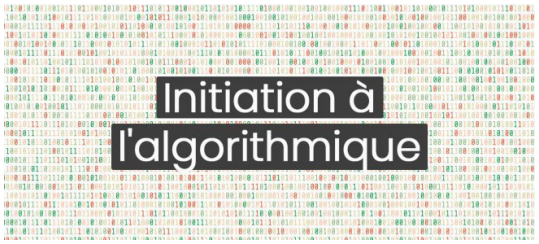
Chapitre 5 : Les sous-algorithmes (fonctions et procédures)

ALGORITHMIQUE (ALSD)



Préparé par Dr. BOUCHEBBAH Fatah

UNIVERSITÉ DE BÉJAIA

FACULTÉ DES SCIENCES EXACTES

DÉPARTEMENT D'INFORMATIQUE

Octobre 2021
Version 3

Table des matières

Objectifs

A l'issue de ce chapitre, vous serez capable de :

- Décrire un sous-algorithme.
- Décrire la différence entre une procédure et une fonction.
- Utiliser efficacement des fonctions et/ou procédures dans des algorithmes.

Introduction

Soit à écrire un algorithme qui calcule le nombre de combinaisons de n objets :

$$C_n^k = \frac{n!}{p!(n-p)!}$$

On peut l'écrire ainsi :

```
ALGORITHME combinaison ;

VAR n, p, r1, r2, r3, i : Entier ;

DÉBUT
 ÉCRIRE ('Valeurs de n et p ? ') ;
 LIRE (n, p) ;
 r1 <- 1 ;
 POUR i ALLANT DE 2 À n FAIRE
  r1 <- r1 * i ;
 FIN POUR ;
 r2 <- 1 ;
 POUR i ALLANT DE 2 À p FAIRE
  r2 <- r2 * i ;
 FIN POUR ;
 r3 <- 1 ;
 POUR i ALLANT DE 2 À n-p FAIRE
  r3 <- r3 * i ;
 FIN POUR ;
 ÉCRIRE ('Résultat = ', r1 DIV (r2 * r3)) ;
FIN.
```

Nous remarquons dans l'algorithme "combinaison" ci-haut que le traitement de calcul de la factorielle d'un nombre est répété 3 fois. L'idée est d'écrire un *"sous-algorithme"* spécialisé dans le calcul de la factorielle d'un nombre puis de l'appeler chaque fois que c'est nécessaire dans *"l'algorithme principal"* (l'algorithme calculant C_n^k).

Le sous-algorithme doit être *paramétrable* pour qu'il *puisse calculer la factorielle de n'importe quel nombre* qui lui sera passer en paramètre.

Les sous-algorithmes

Un sous-algorithme est une suite d'instructions qui est écrite une seule fois, mais peut être l'utilisée autant de fois que c'est nécessaire. Ceci, en effectuant des appels au sous-algorithme avec des données différentes.

Lors de l'écriture des algorithme, nous pouvons observer que certains groupes d'instructions se rapportent à des traitements précis et différents. De ce fait, il est préférable alors de représenter chacun d'eux dans un sous-algorithme. Nous percevons alors un algorithme comme une entité composée d'un ensemble de sous-algorithmes (*ou modules*). La structuration d'un algorithme par modules est la base de la programmation structurée et modulaire.

Donc écrire un algorithme qui résout un problème revient toujours à écrire des sous-algorithmes qui résolvent des sous parties du problème initial. En algorithmique, il existe deux types de sous-algorithmes : les *fonctions* et les *procédures*.

1. Avantages d'utilisation des sous-algorithmes

La structuration d'un algorithmes en modules offre plusieurs avantages, essentiellement :

- L'amélioration de la lisibilité de l'algorithme.
- L'optimisation du nombre d'instructions dans l'algorithme.
- La facilité de la mise au point et de la correction des erreurs dans l'algorithme.
- La décomposition du problème complet et complexe en un ensemble de sous-problèmes faciles à résoudre.

2. La portée d'une variable.

La portée d'une variable est l'ensemble des sous-algorithmes où cette variable est connue. Lorsqu'une variable est connue pour un sous-algorithme, la variable peut être utilisée dans les instructions du sous-algorithme. Une variable peut être *globale* ou *locale*.

2.1. Variable globale

C'est une variable qui est définie au niveau de l'algorithme principal (*i.e.* dans la partie déclaration (ou VAR) de l'algorithme qui résout le problème initial). La portée d'une variable globale est totale. Explicitement, tout sous-algorithme de l'algorithme principal peut utiliser cette variable.

2.2. Variable locale

C'est une variable qui est définie au sein d'un sous-algorithme. La portée d'une variable locale est restreinte au sous-algorithme dans lequel elle est déclarée. Autrement dit, une variable locale ne peut être utilisée que dans des instructions du sous-algorithme qui la contient.

Dans un sous-algorithme, lorsque l'identificateur d'une variable locale est identique à celui d'une variable globale, la variable globale est localement masquée dans celui-ci. Dans ce sous-algorithme la variable globale devient alors inaccessible.

3. Structure générale d'un sous-algorithme

Un sous-algorithme possède une structure analogue à celle d'un algorithme. Cependant, chaque sous-algorithme doit être déclaré au préalablement avant de pouvoir l'utiliser (l'appeler) dans le corps de l'algorithme principal. La déclaration d'un sous-algorithme se fait dans la partie déclaration de l'algorithme principale. Plus exactement, juste après la déclaration des variables et avant le mot réservé DÉBUT qui marque le début du corps de l'algorithme principale.

```
1 ALGORITHME identificateurAlgorithme ;
2 CONSTANTE (ou CONST)
3  Liste des constantes
4 VARIABLE (ou VAR)
5  Liste des variables
6 FONCTION ou PROCEDURE
7  Liste des sous-algorithmes
8 DÉBUT
9  liste des instructions
10 FIN.
```

Un sous-algorithme peut prendre la forme d'une fonction ou une procédure. De plus, il est obligatoirement caractérisé par un identifiant unique qui obéit aux mêmes règles que les identifiants des constantes et des variables. Par ailleurs, lorsqu'un sous-algorithme (une fonction ou une procédure) a été explicité, son nom devient une nouvelle instruction, qui peut être utilisée dans l'algorithme principal ou dans d'autres sous-algorithmes.

L'utilisation d'un sous-algorithme se fait par des instructions particulières que nous nommons *appels au sous-algorithme*. De ce fait, l'algorithme (ou le sous-algorithme) qui fait appel à un sous-algorithme est appelé algorithme (ou sous-algorithme) *appelant*.

Les fonctions

Une fonction est un sous-algorithme qui admet plusieurs paramètres et ne retourne qu'un seul résultat de type simple (Entier, Caractère, Réel, etc.) à l'algorithme (au sous-algorithme) appelant. Par conséquent, il est obligatoire de préciser, dès le début le type de la fonction qui est en même temps le type du résultat à retourner.

1. Syntaxe de déclaration d'une fonction

La syntaxe générale d'une fonction est donnée comme suit :

```
FONCTION nom_fonction (paramètre(s) formel(s)) : type_de_la_valeur_retournée;
CONSTANTE (ou CONST)
 Liste des constantes
VARIABLE (ou VAR)
 Liste des variables
DÉBUT
 Liste des instructions de la fonction
 nom_fonction <- valeur ;
FIN ;
```

Ou en langage Pascal :

```
FUNCTION nom_fonction (paramétre(s) formel(s)) : type_de_la_valeur_retournée;
CONSTANTE (ou CONST)
 Liste des constantes
VARIABLE (ou VAR)
 Liste des variables
BEGIN
 Liste des instructions de la fonction
 nom_fonction := valeur ;
END ;
```

Complément : Gardez en tête !

Les instructions d'une fonction doivent contenir une instruction du style *nom_fonction <- valeur*. C'est l'instruction qui retourne le résultat obtenu en exécutant la fonction à l'algorithme (sous-algorithme) appelant. Où, *valeur* est le résultat de la fonction ayant le même type que le type attendu lors de la déclaration de la fonction.

2. Appel à une fonction

Un appel à une fonction se fait en utilisant le nom de la fonction suivi des paramètres effectifs entre parenthèses. Puisqu'une fonction retourne toujours un résultat, donc il faut prévoir une variable dans l'algorithme (sous-algorithme) appelant pour recevoir le résultat de l'exécution de la fonction.

De ce fait, dans l'algorithme (sous-algorithme) appelant, nous devons avoir une instruction de la sorte :

```
Identificateur_variable <- Nom_fonction (paramètre(s) effectif(s));
```

Exemple : Un algorithme utilisant une fonction

Écrire une fonction qui recherche le maximum de deux valeurs. Puis retourne le résultat à l'algorithme principal qui la contient.

```
ALGORITHME rechmax ;
VAR x,y,grand : Entier ;
FONCTION Max(a,b : Entier) : Entier ;
 DÉBUT
  SI (a>=b) ALORS max <- a ;  SINON max <- b ; FIN ;
DÉBUT
 LIRE(x,y) ;
 grand <- Max(x,y) ;
 Écrire('La plus grande valeur entre',x,'et',y,'est :',grand) ;
FIN.
```

Fondamental : Explication du déroulement

Lors de l'exécution de la fonction Max, le paramètre formel *a* reçoit une copie de la valeur de la variable *x*, et le paramètre formel *b* reçoit une copie de la valeur de la variable *y*. Le résultat est affecté à l'identifiant de la fonction qui sera utilisé dans le l'algorithme principal.

Exemple : Amélioration de l'algorithme combinaison

Refaire l'algorithme *combinaison* en utilisant une fonction qui calcule la valeur factorielle d'un nombre.

```
ALGORITHME combinaison ;
VAR n, p : Entier ;
FONCTION Fact (x : Entier) : Entier ;
 VAR i, r : Entier ;
 DÉBUT
   r <- 1 ;
   POUR i ALLANT DE 2 À x FAIRE
    r <- r * i ;
   FIN POUR ;
   Fact <- r ;
 FIN ;
DÉBUT
 ÉCRIRE ('Valeurs de n et p ? ') ;
 LIRE (n, p) ;
 ÉCRIRE ('Résultat = ', Fact (n) DIV ( Fact (p) * Fact (n-p))) ;
FIN.
```

Les procédures

IV

Contrairement à une fonction qui ne retourne qu'un seul résultat à la fois, une procédure est un sous-algorithme qui ne retourne aucun résultat ou retourne plusieurs résultats qui ne sont pas forcément du même type.

Une procédure peut admettre des paramètres avec de différents passages.

1. Types de passages de paramètres à une procédure

Il existe trois types d'association (que nous nommons passages de paramètres) entre les paramètres formels d'une procédure et les variables de l'algorithme appelant passées en paramètres effectifs :

- En entrée, préfixé par *Entrée* (ou *E*).
- En sortie, préfixé par *Sortie* (ou *S*).
- En entrée/sortie, préfixé par *Entrée/Sortie* (ou *E/S*).

1.1. Le passage de paramètres en entrée

Les instructions du sous-algorithme ne peuvent pas modifier l'entité (variable ou constante) de l'algorithme appelant. En fait, c'est la valeur de l'entité de l'algorithme appelant qui est copiée dans le paramètre (à part cette copie il n'y a pas de relation entre le paramètre et l'entité de l'algorithme appelant).

1.2. Le passage de paramètres en sortie

Les instructions du sous-algorithme affectent obligatoirement une valeur à ce paramètre. Et cette valeur sera affectée à l'entité associée de l'algorithme appelant. Il y a donc une liaison forte entre le paramètre et l'entité de l'algorithme appelant. C'est pour cela que ne pouvons pas utiliser de constantes pour ce type de paramètre.

La valeur que pouvait posséder la variable associée de l'algorithme appelant n'est pas utilisée par le sous-algorithme. Par exemple, le sous-algorithme LIRE (), qui permet de mettre dans des variables des valeurs saisies par l'utilisateur, admet *n* paramètres en sortie.

1.3. Le passage de paramètres en entrée/sortie

C'est un passage de paramètres qui combine les deux types de passages précédents. C'est le passage à utiliser lorsque le sous-algorithme doit modifier la valeur de la variable de l'algorithme appelant. De ce fait, nous ne pouvons pas faire passer une constante par un passage en entrée/sortie.

Complément : Passage de paramètres en langage Pascal

Contrairement au langage algorithmique dans lequel nous spécifions chaque passage de paramètre par un mot réservé (*i.e.* en entrée (*E*), en sortie (*S*), et en entrée/sortie (*E/S*)), en langage Pascal nous n'avons qu'un seul mot réservé, à savoir : *VAR*. En effet, *VAR* est utilisé pour désigner un passage de paramètre en sortie ou en entrée/sortie. Cependant, il n'existe pas de mot réservé pour désigner un passage de paramètre en entrée. Par conséquent, les paramètres passés en entrée à une procédure en Pascal ne sont précédés par aucun mot réservé.

2. Syntaxe de déclaration d'une procédure

Nous déclarons une procédure de la façon suivante :

```
1 PROCÉDURE nom_procédure (E paramètre(s) en entrée; S paramètre(s) en sortie; E/S
  paramètre(s) en entrée/sortie);
2 CONSTANTE (ou CONST)
3  Liste des constantes locales
4 VARIABLE (ou VAR)
5  liste des variables locales
6 DÉBUT
7  Liste des instructions de la procédure
8 FIN ;
```

Ou en langage Pascal :

```
1 PROCEDURE nom_procédure (paramètre(s) en entrée; VAR paramètre(s) en sortie; VAR
  paramètre(s) en entrée/sortie);
2 CONSTANTE (ou CONST)
3  Liste des constantes locales
4 VARIABLE (ou VAR)
5  liste des variables locales
6 BEGIN
7  Liste des instructions de la procédure
8 END ;
```

3. Appel à une procédure

Nous appelons une procédure en indiquant son nom suivi des paramètres effectifs entre parenthèses.

```
1 Nom_procédure (paramètre(s) effectif(s)) ;
```

Exemple : Le maximum de deux nombres en utilisant une procédure

Réécrire l'algorithme rechmax en utilisant une procédure à la place de la fonction Max.

```
1 ALGORITHME rechmaxPro ;
2 VAR x,y,grand : Entier ;
3 PROCÉDURE Max(E a,b : Entier ; S M : Entier);
4  DÉBUT
5   SI (a>=b) ALORS
6    M <- a ;
7   SINON
8    M <- b ;
9  FIN ;
10 DÉBUT
11  LIRE(x) ;
12  LIRE(y) ;
13  Max (x,y,grand) ;
14  Écrire('La plus grande valeur entre',x,'et',y,'est :',grand) ;
15 FIN.
```

Complément : Gardez en tête !

Lors de l'exécution de la procédure Max, la variable *x* et le paramètre *a* sont associés par un passage de paramètre en entrée. De même pour la variable *y* et le paramètre *b*. Ce qui veut dire que la valeur de *x* est copiée dans *a* et la valeur de *y* est copiée dans *b*. Par contre la variable *grand* est associée en paramètre de sortie avec le paramètre *M*. Donc c'est la valeur de *M* qui est copiée dans *grand* à la fin de l'exécution de Max.

Exemple : Permutation de deux valeurs en utilisant une procédure

Écrire un algorithme qui permet de permuter deux valeurs entières. Et ceci en faisant appel à une procédure.

```
1 ALGORITHME permutation ;
2 VAR a,b : Entier ;
3 PROCÉDURE echanger (E/S x, y : Entier) ;
4  VAR temp : Entier ;
5  DÉBUT
6   temp <- x ;
7   x <- y ;
8   y <- temp ;
9  FIN ;
10 DÉBUT
11  Écrire('Entrez deux entiers :') ;
12  LIRE(a,b) ;
13  echanger(a,b) ;
14  Écrire('a=',a,' et b = ',b) ;
15 FIN.
```

Rappel : Déroulement de l'exécution

Lors de l'exécution de la procédure echanger, la variable *a* et le paramètre *x* sont associés par un passage de paramètre en entrée/sortie (même chose avec la variable *b* et le paramètre *x*). Toute modification sur *x* est effectuée sur *a* (de même pour *y* et *b*).