Algorithmie et Programmation

Chapitre 2 : fonctions et procédures

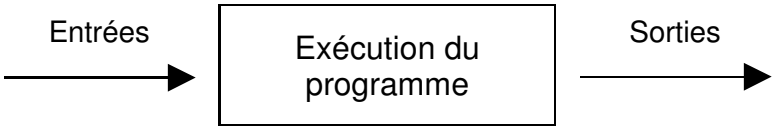
Imaginez un joueur de foot qui s'occupe autant de marquer les buts que de défendre sa cage. Même Zidane aurait du mal face à une équipe de 11 joueurs chacun étant affecté à une tâche spécifique. Dans cette équipe, les joueurs doivent être habilles à s'échanger le ballon, mais pour un résultat vraisemblablement meilleur que l'unique joueur. Vous voyez d'ici le score.
Eh bien, en programmation, c'est la même chose.

Ce chapitre va se focaliser sur la structuration d'un programme en blocs logiques qui interagissent entre eux par un assemblage logique permettant de fournir une solution à un problème. Chaque bloc est un sous-programme qui est capable de remplir une mission/tâche spécifique.

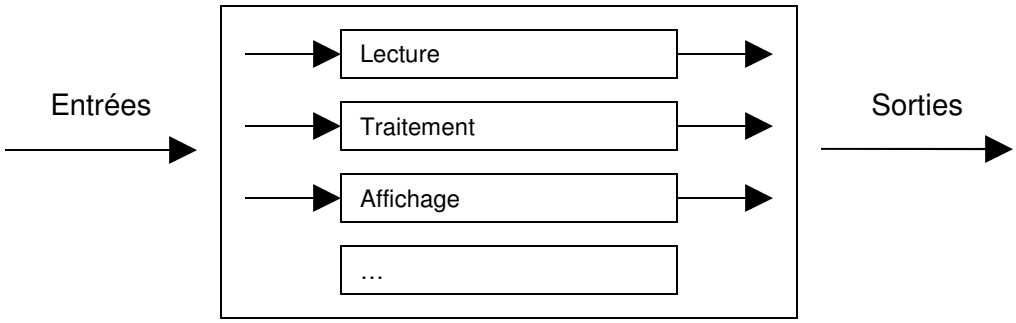
Il s'agit de maîtriser la programmation dite procédurale.

1. Programmation procédurale.

Nous savons qu'un programme est une suite d'instructions qui transforme des données d'entrée en produisant un résultat en sortie. Nous avons schématisé cela par la « boîte noire » :



Dans un programme, nous utilisons déjà des instructions « avancés » pour la gestion de la saisie et de l'affichage, ou pour le transtypage. Ces instructions sont des sous-programmes. En les sollicitant, une mission spécifique leur est déléguée. Ainsi, en zoomant dans le schéma :



Un sous-programme regroupe une séquence d'instructions qui doit être capable de remplir la mission qui lui est confiée. Chaque sous-programme manipule des données en entrée pour fournir le résultat attendu.

Nous avons déjà identifié dans un programme la séquence de trois grandes phases : la gestion des entrées, les traitements, et la gestion de l'affichage. Leur assemblage logique permet de résoudre le problème dans sa globalité.

Réaliser un programme par assemblage de sous-programmes est directement lié à la méthode d'analyse descendante (*top-down-development*) dite de *raffinement progressif*. La phrase suivante illustre parfaitement la démarche qui orientera toute la suite de ce cours :

> ***…diviser chacune des difficultés que j'examinerais, en autant de parcelles qu'il se pourrait et qu'il serait requis pour les mieux résoudre…***
>
> *[Descartes - Discours de la méthode, seconde partie, 1637.]*

Ainsi, pour résoudre un problème automatisable, c'est-à-dire pouvant être résolu à l'aide d'un algorithme, il convient de suivre une démarche de ce type :

```
Définir le problème à résoudre :
        Expliciter les données
                Préciser leur nature
                Leur domaine de variation
                Leurs propriétés
        Expliciter les résultats
                Préciser leur structure
                Leurs relations avec les données
Fin Définir

Décomposer le problème en sous-problèmes

Pour chaque sous-problème identifié Faire
        Si solution évidente Alors
                Ecrire le morceau de programme
        Sinon
                Appliquer cette méthode au sous-problème
        FinDuSi
FinDuPour
```

Il apparaît donc plusieurs niveaux de décomposition du problème (niveaux d'abstraction descendants). Ces niveaux permettent d'avoir une description de plus en plus détaillée du problème et donc de se rapprocher par raffinements successifs d'une description prête à la traduction en instructions du langage.

Un programme peut donc d'abord être résolu globalement, les détails étant reportés à des entités sous-ordonnées qui peuvent elles aussi être subdivisées en sous-entités et ainsi de suite.

2. Sous-programme - Fonction.

Lorsque vous utilisez la touche factorielle de votre calculette, vous ne vous souciez pas de savoir quel est son fonctionnement. La seule chose que vous avez besoin c'est son mode d'emploi, « saisir un nombre et presser la touche ».

De même en programmation, comment utiliser une telle fonction factorielle. Voyons le syntaxe d'appel :

```
public static void main(String[] args) {
    int x = 2;
    int factDeX;
    factDeX = factorielle(x);
}
```

Dans un scénario d'appel d'une fonction il y a deux « acteurs », le bloc appelant et la fonction appelée. L'appelant délègue une tâche à une fonction dite appelée.

Lorsqu'on sollicite/appel une fonction existante, il faut connaître son mode d'emploi. Ici, pour réaliser un calcul de la factorielle, encore faut-il fournir à la fonction une valeur. Et bien sûr cette fonction donne un résultat. Il y a donc une donnée en entrée et une autre en sortie. Comme toute donnée stockée en mémoire, celles-ci sont typées.

Pour connaître précisément le mode d'emploi d'une fonction, il faut se référer à sa déclaration. La fonction **factorielle** est un sous-programme dont les modalités d'utilisation et de fonctionnement sont décrits comme suit :

```
static int factorielle(int n){
    int fact; //pour stocker la factorielle calculée
    //suite d'instructions calculant la factorielle de n
    //...
    return fact;
}
```

La première ligne résume à elle seule le mode d'emploi, elle signifie : c'est une fonction qui s'appelle **factorielle** à qui il faut fournir un valeur de type **int** et qui donne un résultat sous la forme d'une valeur de type **int.** Cette ligne porte le nom de **signature de la fonction.**

Ce sous-programme reçoit une donnée dans la variable mémoire **n** et l'utilise pour calculer sa factorielle et la stocker dans la variable **fact**.

Avant la fin de l'exécution du bloc, l'instruction importante `return` se charge de transférer le contenu de la variable **fact** vers le bloc appelant.

Dans cet exemple, la fonction requiert une valeur. On dit que la fonction est paramétrée, c'est-à-dire qu'elle reçoit un ou plusieurs paramètres. Un paramètre joue alors le rôle d'une variable en entrée dans la fonction.

Lorsque le bloc appelant sollicite une fonction, il faut fournir une valeur à chaque paramètre.

Résumons :

- pour créer une fonction, il faut :
    - Définir un identificateur de fonction : la première lettre sera toujours une minuscule.
    - Définir les paramètres en entrée : un nom et un type pour chaque paramètre
    - Définir le type de la fonction : c'est le type de la variable qui contient le résultat à donner en retour.
    - Définir la variable qui doit contenir le résultat (son type est celui de la fonction)
    - Définir si besoin les variables intermédiaires utiles au traitement
    - Définir les instructions de traitement
    - Définir l'instruction **return** comme dernière instruction

- Pour appeler une fonction, il faut :
    - Consulter le mode d'emploi de la fonction
    - Utiliser le nom de la fonction suivie des valeurs à transmettre
    - Contrôler la récupération du résultat en le stockant dans une variable du même type par exemple.

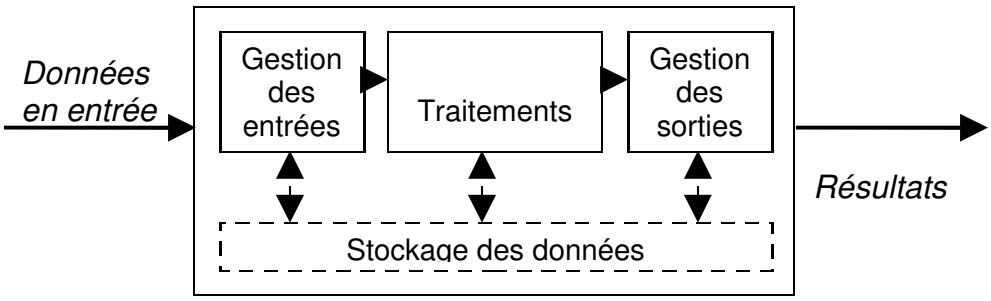
Remarques :

- Lorsque le mode d'emploi indique plusieurs paramètres, lors de l'appel il faut respecter le nombre et l'ordre des paramètres.
- Le type de la variable qui récupère la valeur résultat de la fonction doit être du même type que celui mentionné dans le mode d'emploi comme type de la fonction. C'est aussi celui de la variable associée au **return**.
- Lorsqu'une fonction en doit rien retourner en sortie on utilise le mot clé **void** comme type de retour.

Nous avons vu qu'une fonction utilise des données en entrées sous la forme de paramètres. Ces paramètres agissent comme des variables spécifiques à l'espace mémoire de la fonction. Ainsi, la fonction utilise une copie des données mentionnées à l'appel et non l'original.

3. Visibilité et échange de données.

Tout programme, et donc sous-programme manipule et transforme des données. Soit le schéma de la « boîte noir » intégrant l'aspect procédural :



Chaque procédure possède une zone mémoire privée pour y stocker des valeurs. Ainsi, si la procédure principale réceptionne les données en entrée dans des variables qui lui sont privées, comment la procédure de traitement va-t-elle pouvoir travailler sur ces variables qui lui sont inaccessibles ?

Il faut bien donc que le programme principale s'occupe de la gestion des entrées et sorties de chaque procédure/fonction. C'est le cas dans le code suivant:

```
public static void main(String[] args) {
      //Stock la sortie de la fonction lireEntier dans la var. nb
      int nb = lireEntier("Entrez un nombre: ");
      //Fournit nb comme en entrée à la fonction factorielle, et
      //stock la sortie dans la variable factDeX
      int factDeX = factorielle(nb);
      //Fournit les deux sorties stockées précédement comme entrée à
      //la fonction afficherFact
      afficherFact(nb, factDeX);
  }

  static int factorielle(int n){
      int fact = 1;
      for (int nombre=n; nombre>1; nombre--){
          fact = fact * nombre;
      }
      return fact;
  }

  static int lireEntier(String message){
      Scanner read = new Scanner(System.in);
      System.out.print(message);
      return read.nextInt();
  }

  static void afficherFact(int nombre, int fact){
      System.out.println(nombre + "! = " + fact);
  }
```