Les sous-programmes

Quoi ? Pourquoi ? Comment faire ? Comment utiliser ? Bonnes pratiques ? Remarques ?

Les sous-programmes, c'est quoi ?

Un sous-programme : est un bloc d'instructions nommé et paramétré réalisant une certaine tâche et qui peut être soit une fonction, soit une procédure.

Une fonction : est un sous-programme qui admet zéro, un ou plusieurs paramètres et *renvoie* toujours un seul résultat.

Une procédure : est un sous-programme qui admet zéro, un ou plusieurs paramètres et *ne renvoie* aucun résultat.

Pourquoi et quand on utilise les sous-programmes ?

Le but d'utiliser les sous-programmes c'est de réutiliser un algorithme existant sans avoir le réécrire et de structurer l'algorithme pour le rendre plus compréhensible.

La structuration consiste à décomposer un programme complexe en une série de sous-programmes plus simples, lesquels peuvent à leur tour être décomposés eux-mêmes en fragments plus petits, et ainsi de suite.

Comment créer des sous-programmes ?

Chercher les blocs d'instructions qui se répètent dans votre algorithme et les encapsuler dans des fonctions ou des procédures

La déclaration d'une fonction (Implémentation):

fonction nom_fonction(<liste des paramètres formels>) :<type du résultat>

Var <déclaration des variables locales>

Début

<traitements>

nom_fonction ← résultat (ou bien) Retourner(résultat)

Fin nom_fonction

- ✓ <**liste des paramètre formels**> : p_1 : type_1, p_2 :type_2,……,p_n :type_n

  Remarque : Le résultat retourné à la fin de la fonction doit être de même type ou de type compatible que celui déclaré dans l'entête de la fonction (Signature).

La déclaration d'une procédure :

procedure nom_procédure(<**liste des paramètres formels**>)

Var <déclaration des variables locales >

Début

<traitement>

Fin nom_procédure

- ✓ <**liste des paramètre formels**> : p_1 : type_1, p_2 :type_2,……,p_n :type_n

Comment utiliser les sous-programmes ?

Appel à une fonction : variable←**nom_fonction(<liste des paramètres effectifs>)**

Appel à une procédure **nom_procédure**(<**listes des paramètres *effectifs***>)

<**liste des paramètres effectifs**> : peut-être un identifiant d'une variable, une expression, une constante ou un appel à une fonction

Deux types de paramètres effectifs :

Les paramètres d'Entrée : Identifiant _variable, Constante, Une expression, Un appel de fonction

Les paramètres d'E/S (VAR) : Toujours des identifiants de variables globales

Relation entre paramètres formels et paramètres effectifs :

- Doivent avoir le même nombre
- Correspondance dans l'ordre d'apparition
- Compatibilité de type entre les paramètres formels et effectifs

Modes de passage de paramètres :

- ***Passage par valeur :*** Le paramètre effectif peut être un identifiant d'une variable, une constante, une expression ou un appel de fonction
- ***Passage par adresse (en utilisant VAR) :*** Le paramètre effectif doit être obligatoirement un identifiant d'une variable

Types de variables :

- Une variable ***locale*** est une variable qui ne peut être utilisée que dans un sous-programme ou le bloc où elle est définie.
- Une variable ***globale*** est une variable déclarée dans le programme principal (appelant) et peut être utilisée dans tout le programme.

Quelles sont les bonnes pratiques lors de l'utilisation des sous-programmes ?

1. Factoriser le code qui se répète sous la forme de sous-programmes
2. Choisir des noms explicites (significatifs) à tous les identificateurs de l'algorithme (variables, constantes, nom d'algorithme, nom des sous-programmes)
3. Le nom d'une fonction qui retourne un résultat booléen commence par « est_ »
4. Utiliser les indentations
5. Commentez bien votre algorithme

Remarque

Une fonction en informatique se distingue principalement de la fonction mathématique par le fait qu'en plus de calculer un résultat à partir de paramètres, la fonction informatique peut avoir des « effets de bord » : par exemple afficher un message à l'écran, jouer un son, ou bien piloter une imprimante.