

Chapitre 5 : Procédures et fonctions

Sous programme : Ensemble d'instructions qui seront exécutées par simple appel dans un programme principal → peut être appelé autant de fois qu'on veut

Le nom d'un sous programme doit refléter le traitement réalisé

Sous programme ne peut être appelé indépendamment d'un programme

Pourquoi l'utilise-t-on ? → programmer de manière modulaire (trop d'instructions dans le programme donc on découpe)

→ Lorsqu'un même traitement doit être fait plusieurs fois

Sous programme permet : Division du travail

→ Réduire la complexité de l'application

→ Faciliter la maintenance

2 sous programmes :

Fonctions (return valeur) et procédures (pas de return)

Procédure (peut se placer avant ou après le programme principal) → doit être placée dans la **classe Program** :

```
static void nom_procedure(type+si_paramètre)
{
    instructions...
}
```

Appeler une procédure dans le programme :
nom_procedure(si_paramètre) ;

Attention : Les variables du programme principal ne sont pas connues de la procédures
→ Variables de la procédure pas connues dans le programme principal

Fonctions :

```
static type_sortie nom_fonction(type+si_argument)
{
    // declaration variable retournée
    instructions
    return variable
}
```

Appel fonctions :

variable = nom_fonction(si_argument)

Procédure qui modifie une valeur :

```
static void nom_procedure(type+si_parametre, out type+parametre)
```

out → adresse de la variable est fournie à la procédure donc elle peut changer le contenu

Appeler une procédure dans le programme :

nom_procedure(si_parametre, out parametre)

Attention : la variable dans le out est passée par référence tandis que sans out, les variables passent par valeur.

Ref → variable a besoin d'être initialisée avant d'appeler la procédure car celle-ci attend une valeur

static void nom_procedure(type+si_parametre, ref type+parametre)

return = sortie immédiate de la fonction