ALGORITHMIQUE : Suite des fondamentaux 1/7 - La problématique des variables

Portée des variables: globales et locales

variable globale:

- *Exploitation:* Dans tout le programme
- *durée de vie:* toute l'exécution du programme

variable locale:

- *Exploitation:* que dans la fonction ou le bloc où elle est définie.
- *durée de vie:* détruite automatiquement à la fin du bloc ou de la fonction

La variable globale

Avantages:

- Accessible depuis toutes les fonctions/modules d'un programme
- Idéal pour des **constantes**
- Utile si de multiples fonctions ont besoin de la même donnée

Inconvénients:

- Si on en a trop, on peut surcharger la mémoire
- Peu sécurisé car toute fonction peut en modifier sa valeur
- En cas de refactoring, on doit modifier partout où elle est exploitée

Les variables locales

Avantages:

- N'occupe la mémoire que lors de l'exécution de son bloc
- On peut donner le même nom dans différentes fonctions
- Garantie d'une valeur intacte pendant l'exécution de la tâche

Inconvénients:

- Débuggage plus complexe
- Risque de redondance de données
- Portée limitée

Mode de passage des paramètres d'une fonction

Il en existe deux:

- le passage par valeur *(par défaut)*
- **le passage par référence**

Le passage par référence

Toute affectation du paramètre se traduit automatiquement par **la modification de la variable sur laquelle elle pointe**

```
Procédure firstLast(msg en Caractère par Valeur,
prems en Caractère par Référence, dern en Caractère par Référence)
   prems ← left(msg, 1)
   dern ← right(msg, 1)
Fin Procédure

debut ← "B"
fin ← "J"
firstLast("Coucou", debut, fin)

# debut: "C"
# fin: "u"
```

Explications de l'exemple

`prems` ne devient qu'un pseudonyme pour la variable `debut`.
Tout ce qui arrive à `prems` se produit en réalité sur `debut`

`dern` ne devient alors qu'un pseudonyme pour la variable `fin`.
Tout ce qui arrive à `dern` se produit en réalité sur `fin`

Avantages / Inconvénients

- Paramètre utilisé tant en lecture (en entrée) qu'en écriture (en sortie)
- Variable modifié sans besoin de retour
- Moins de place en mémoire (pas de variable locale créée)

Inconvénients

- Moins sécurisé contre les bugs
- Potentiels effets non désirés
- En cas de refactoring, on doit modifier partout où elle est exploitée