Méthodologie de développement

« L'informatique semble encore chercher la recette miracle qui permettra aux gens d'écrire des programmes corrects sans avoir à réfléchir.

Au lieu de cela, nous devons apprendre aux gens comment réfléchir » - Anonyme -

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Décomposition fonctionnelle

- Une décomposition fonctionnelle consiste à découper l'algorithme en modules logiques (sous-algorithme) ayant une fonction précise…

✓ **Effectuer une décomposition fonctionnelle**

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Décomposition fonctionnelle (1)

- Pourquoi effectuer une décomposition fonctionnelle ?
  - Meilleure structuration et lisibilité.
  - Pas de répétition de certains traitements
  - Réalisation de tests unitaires
  - Travail en projet et répartition des tâches
  - Maintenance plus aisée du programme
  - Réutilisation future

Décomposition fonctionnelle (2)

➔ Algorithme = plusieurs modules

- Pour une meilleure structuration, les modules appartenant à la même logique fonctionnelle ➔ même fichier.
- Deux approches :
  - Top-Down (en raffinant)
  - Down-Top (en regroupant)

Procédures & Fonctions

- Un module correspond soit à une procédure soit à une fonction. Il est indépendant de son contexte (autonome), réutilisable dans plusieurs contextes.

✓ **Comparer la mise en œuvre d'une procédure et d'une fonction**

Procédures & Fonctions (1)

- Une procédure ➔ un traitement sur des données transmis en argument.
  Exemple : *tri(liste_étudiants)*

- Une fonction ➔ traitement sur des données transmis en argument et renvoie une valeur utilisée dans l'affectation d'une variable ou l'évaluation d'une expression.
  Exemple : *m <- moyenne(liste_notes)*

Remarque : Le langage C utilise uniquement des fonctions et considère qu'une procédure est une fonction qui ne renvoie aucune valeur.

Procédures & Fonctions (2)

- Les sous-algorithmes sont définis avant l'algorithme principal.

Procédure nom(…)
Variable
    /* variables */
Début
    /* bloc d'instructions */
Fin

Fonction nom(…) : type
Variable
    /* variables */
Début
    /* bloc d'instructions */
    Retourne valeur
Fin

- Préciser au début de la procédure ou fonction, en commentaires, l'objectif de la procédure, les paramètres en entrée et en sortie (interface)

Portée des variables

- Une variable doit être déclarée avant d'être utilisée.

✓ **Préciser la portée d'une variable et sa durée de vie .**

Portée des variables

- Accès à la variable en fonction de l'endroit de sa déclaration :
  - Déclaration dans un sous-algorithme ➔ Portée locale
  - Déclaration dans l'algorithme ➔ Portée globale
- Même nom de variable locale que variable globale
  ➔ Accès à la variable locale
- Durée de vie d'une variable ➔ Durée de l'exécution de l'algorithme ou du sous-algorithme

Paramètres et arguments

- Un sous-algorithme peut avoir des paramètres qui seront utilisés par l'algorithme principal ou un autre sous-algorithme.

✓ **Effectuer le passage des paramètres entre le module appelant et le module appelé**

Paramètres et arguments (1)

- Paramètres déclarés du module ➔ paramètres formels
  Paramètres passés lors de l'appel ➔ paramètres effectifs ou arguments.

- Paramètre formel ➔ variable locale particulière
  Paramètre effectif ➔ contenu d'une variable ou une expression évaluée (uniquement en entrée).

- Lors de la conception d'un sous-algorithme, on raisonne sur les paramètres formels, sachant que les valeurs effectives (réelles) ne sont pas encore connues.

Paramètres et arguments (2)

- Trois types de passage de paramètres :
  - Le passage de paramètre en entrée (passage par valeur)
  - Le passage de paramètre en sortie (par variable)
  - Le passage de paramètre en entrée/sortie (par variable)

- Attention :
  - Correspondance en nombre de paramètres
  - Correspondance en type de paramètres
  - Un paramètre défini en entrée doit correspondre à un argument qui possède une valeur dans l'algorithme appelant au moment de l'appel.
  - Un paramètre défini en sortie doit recevoir une valeur dans le sous-algorithme et doit pouvoir être stocké dans une variable dans l'algorithme appelant.

Paramètres et arguments (3)

- Notation en algorithmique :

Procédure nom(E var_entree : type, S var_sortie : type,
ES var_ent_sortie : type)
Partie déclarations
Partie instructions

Fonction nom(E var_entree : type, S var_sortie : type,
ES var_ent_sortie : type) : type_retour
Partie déclarations
Partie instructions
Retourne valeur

Paramètres et arguments (4)

- Notation en C :

*type_retour Function nom(type var_entree, type *var_sortie ,*
*type *var_entree_sortie)*
Partie déclarations
Partie instructions

Paramètres et arguments (5)

```
Algorithme Permutation
Procédure permuter(ES a,b : Entier)
/* échanger les valeurs de  variables de type entier */
Variable
    tmp : Entier /* pour la permutation */
Début
    tmp <- a
    a <- b
    b <- tmp
Fin
Variable
    i, j : Entier;
Début
    Lire(i)
    Lire(j)
    permuter(i, j) /* valide */
    permuter(i, 5) /* Non valide */
Fin
```