said_anis@hotmail.com

COURS ALGORITHMIQUE

CHP 4

Programmation modulaire

Plan du cours

- Introduction
- Sous-programmes : Procédure / fonction
- Variables : locales / globales
- Paramètres : Formels / effectifs
- Passage de variables : Par valeur / référence
- Appel de sous programmes
- Exercices d'application

said_anis@hotmail.com

Introduction

- ***L'analyse modulaire*** *est la décomposition d'une tâche complexe en tâches élémentaires.*
- Améliore la lisibilité du programme
- Permet la décomposition d'un problème en sous problèmes plus faciles à analyser et à résoudre.
- Diminue la taille des programmes
- Diminue les risques d'erreurs

said_anis@hotmail.com

Objectifs

- Décomposer un problème en modules,
- Présenter les solutions sous formes de procédures et de fonctions,
- Ecrire des algorithmes solutions.

Sous-programmes

- Ecrire des sous-programmes permet de découper un programme en plusieurs morceaux.
- Un sous-programme est un ensemble **d'instructions** :
  - Identifiées par un **nom**
  - Décrit un **traitement** simple ou composé
  - Admet *éventuellement* des **paramètres (0 ou plusieurs).**

said_anis@hotmail.com

Sous-programmes : Fonction

- Une fonction est un sous-programme qui retourne un seul résultat.
- Syntaxe :

Fonction NomFonction (paramètre(s) : type) : type de retour

Variables variable(s) locale(s) : type

Début

<Instructions>

NomFonction ← resultat

Fin

said_anis@hotmail.com

Sous-programmes : Procédure

- Une procédure est un sous-programme qui ne retourne aucun résultat.
- Syntaxe :

Fonction NomProcedure (paramètre(s) : type)

Variables variable(s) locale(s) : type

Début

<Instructions>

Fin

said_anis@hotmail.com

Variables locales

- **Déclaration :** à l'intérieur de sous-programme.
- Accessible :
  - seulement dans l'environnement de sous-programme ou elle a été déclarée.
- Inaccessible :
  - Par le programme principal
  - Par les sous-programmes déclarés de même niveau que le sous-programme considéré
- Intérêt :
  - Plus de lisibilité d'algorithme.
  - Minimise les erreurs.

said_anis@hotmail.com

Variables globales

- **Déclaration :**
  - au niveau de l'algorithme qui appelle le sous-programme.
- **Accessible :**
  - par tout dans le programme principal tous ses sous-programmes.
- **Inaccessible :**
  - Lorsque le nom d'une variable locale est identique à une variable globale, la variable globale devient inaccessible dans ce sous-programme.

Paramètres formels

- Un paramètre formel est une variable locale particulière déclarée dans l'en-tête du sous-programme
- Associée à une variable ou valeur (numérique ou définie par le programmeur) du (sous-) programme appelant.
- Les valeurs des paramètres formels peuvent être de type simple (réel, entier,…) ou de type structuré (tableau, type énuméré…)

Paramètres effectifs

- Un paramètre effectif est une expression (variable, valeur, fonction,…) définie dans le programme principal/sous-programme appelant au niveau de l'appel.
- Les paramètres effectifs et les paramètres formels doivent s'accorder de point de vue nombre et ordre.

Passage de variables

- Il y a deux sortes de passage de paramètres :
  - Passage par valeur
  - Passage par référence.

Passage de variables : Par valeur

- Transmission de l'information dans un seul sens :
  - Du **programme/sous-programme appelant** vers **le sous-programme appelé**
  - **Du paramètre effectif au paramètre formel**
- **Expressions acceptées :** tous types d'expressions dont le type correspond au type de paramètre formel
- La valeur qui est transmise sert à initialiser la variable correspondante dans le sous-programme.

Passage de variables : Par valeur

- **Avant l'exécution du sous-programme appelé :**
  - Les valeurs de paramètres effectifs sont affectées aux paramètres formels (réservation d'un nouvel espace mémoire pour le paramètre formel).
- **Pendant l'exécution du sous-programme appelé :**
  - On fait nos calculs au niveau d'espace mémoire réservé au paramètre formel et non pas celui de paramètre effectif.
- **Après le retour d'exécution du sous-programme appelé :**
  - Chaque paramètre effectif garde sa valeur initiale.

Passage de variables : Par référence

- Seulement l'adresse mémoire (la référence) de la variable qui est transmise, et non pas sa valeur.
- Transmission de l'information dans les deux sens :
  - Du **programme/sous-programme appelant** vers **le sous-programme appelé**
  - Du **sous-programme appelé** vers le **programme/sous-programme appelant**
- **Du paramètre effectif au paramètre formel**
- **Expressions acceptées :** uniquement des variables dont le type correspond au type de paramètre formel ou bien un type compatible (jamais une expression, car on a besoin d'une seule adresse mémoire)

Passage de variables : Par référence

- **Avant l'exécution du sous-programme appelé :**
  - On pointe le paramètre formel sur l'adresse (référence) du paramètre effectif (pas de réservation d'un nouvel espace mémoire pour le paramètre formel).
- **Pendant l'exécution du sous-programme appelé :**
  - On fait nos calculs au niveau d'espace mémoire réservé pour le paramètre effectif.
- **Après le retour d'exécution du sous-programme appelé :**
  - Le paramètre effectif a la valeur du résultat obtenu dans le sous-programme appelé.
- **Pour appliquer ce mode de passage de paramètre :**
  - On précède le paramètre formel par **VAR**

Appel de sous programmes

- Il s'agit simplement de donner un nom à un groupe d'instructions. Ensuite, l'appel de ce nom à divers endroits du programme provoque à chaque fois l'exécution de ce groupe d'instructions.

- L'appel se fait par le ***nom de sous-programme*** suivi par la ***liste des paramètres effectifs*** sépares par des virgules, et dans un certain ordre.

Appel de sous programmes

- L'appel de **procédure** a la forme suivante :

Nom_de_la_procédure (Pe1, Pe2,…, Pen)

- L'appel de **fonction** a la forme suivante :

Variable ← Nom_de_la_fonction (Pe1, Pe2,…, Pen)

said_anis@hotmail.com

Appel de sous programmes

Remarques :

- Un sous-programme peut être appelé :
  - Dans le programme principal
  - Dans un autre sous-programme

- On peut aussi appeler une procédure depuis elle-même : c'est la récursivité, que l'on n'étudiera pas dans ce chapitre.

- Pour appeler une procédure P1 depuis une procédure P2, il faut déclarer la procédure P1 avant la procédure P2.

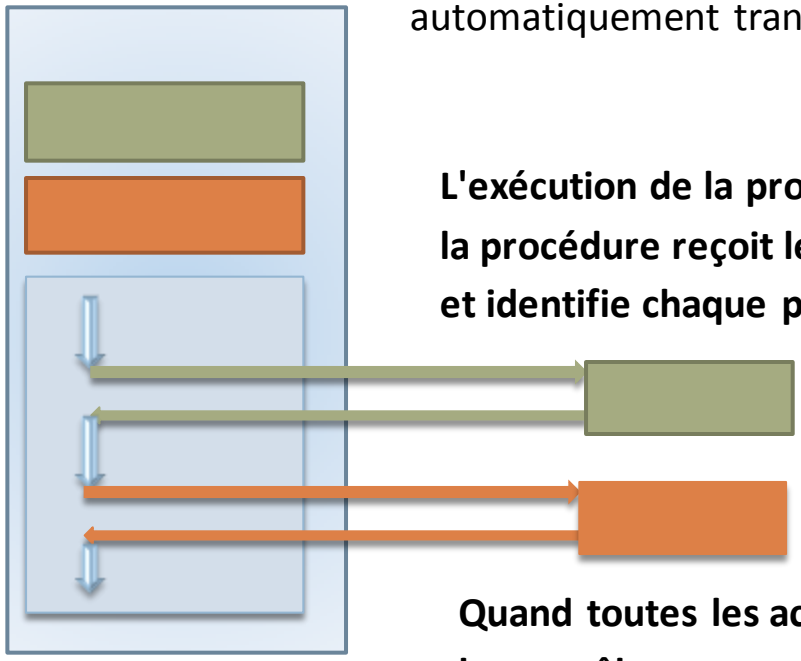
Appel de sous programmes

- Les **paramètres effectifs de l'appel** et les **paramètres formels de l'en-tête** de la déclaration de sous-programme **doivent s'accorder** du point de vue **Ordre**, **nombre** et **type**

- La compilation est très pointilleuse sur les types, mais par contre elle ne détecte pas les inversions de paramètres de même type.

Fonctionnement d'appel d'un sous-programme

Quand on appelle une procédure le contrôle se trouve automatiquement transféré au début de la procédure.



L'exécution de la procédure commence ;
la procédure reçoit les paramètres
et identifie chaque paramètre à une variable dans le même ordre.

Quand toutes les actions de la procédure ont été exécutées, le contrôle retourne à l'action qui suit immédiatement l'action d'appel de la procédure.

Erreurs classiques :

- Mettre un var quand il n'en faut pas : on ne pourra pas passer une expression en paramètre.

- Oublier le var quand il en faut un : la valeur calculée ne pourra pas < sortir > de la procédure.

said_anis@hotmail.com

Surcharge

- La *surcharge* est la possibilité d'utiliser dans le même nom du sous-programme pour déclarer des services différents.
- Exemple de la surcharge du sous-programme sp1 :

```
Procedure Sp1()
Procedure Sp1(e : Entier)
Procedure Sp1(e : chaine de caractère)
Procedure Sp1(e : Booléen)

Fonction   Sp1(e : Booléen) : Entier
Fonction   Sp1(e : Booléen) : Booléen
Fonction   Sp1() : Booléen
```

said_anis@hotmail.com

Exercices d'application

- Ecrire une analyse, un algorithme qui permet de classer en ordre croissant trois variables données strictement positives V1, V2 et V3.

- **Remarque 1:**
  - Le même traitement réalisant la permutation de deux nombres est utilisé trois fois.
  - Isoler ce traitement (faire un module)
  - Appeler ce module autant de fois que nécessaire.

said_anis@hotmail.com

Exercices d'application

- Exercice 2 :

  Ecrire un algorithme d'un programme intitulé som_fact qui permet de calculer et d'afficher la *somme* des factorielles des chiffres d'un nombre n avec (100<= n <=999).

Exemple :

Soit n= 253 Som = 2 ! + 5 ! + 3 !

said_anis@hotmail.com

Exercices d'application

- Exercice 3 :
  - Ecrire une procédure :**ADDITION** permettant d'effectuer l'addition de 2 nombres.
  - Ecrire l'algorithme utilisant la précédente fonction ou procédure.

- Exercice 4 :
  - Ecrire une procédure : DIVISION permettant d'effectuer la division de 2 nombres, on veillera à retourner une indication d'erreur en cas de division par 0.
  - Ecrire l'algorithme utilisant la précédente fonction ou procédure.