Centre de Réadaptation de Mulhouse

Algorithme

Procédures et fonctions

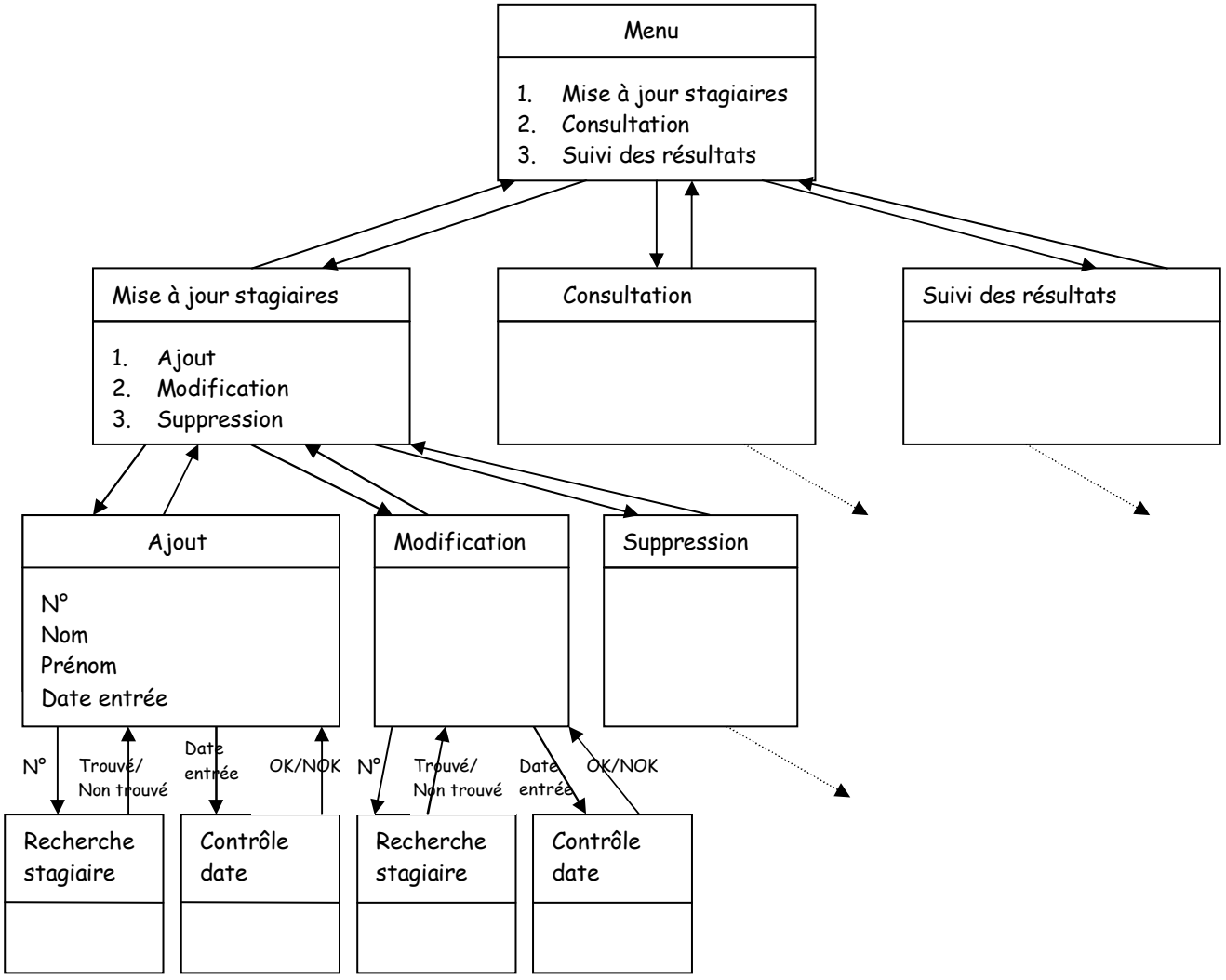
1 Cadre général

L'intégralité du traitement des fonctions réalisées par un logiciel est très rarement implémentée dans une seule unité de traitement et ceci pour 2 raisons majeures:

1. L'algorithme serait trop compliqué donc peut lisible donc difficile à mettre au point et à maintenir
2. Certaines parties de code sont réutilisées plusieurs fois dans le logiciel ou dans d'autres logiciels

Pour ces deux raisons, le traitement sera découpé en plusieurs composants (ou unités de traitements) intégrés selon une structure hiérarchique (un composant est exécuté à l'intérieur d'un autre composant)

Exemple



Principe général:
Un composant appelle un autre composant. A la rencontre de l'appel, l'exécution est déroutée vers le composant appelé qui est exécuté, puis le retour se fait vers le composant appelant, à l'instruction suivant l'appel.
Un composant constitue une **procédure** ou une **fonction**:

- Une **procédure** exécute un traitement correspondant à une tâche précise et ne retourne pas de résultat.
  Exemple: La procédure d'ajout d'un nouveau stagiaire lit les données saisies et écrit ces données dans un fichier
- Une **fonction** est une procédure qui détermine une valeur et la retourne au programme appelant.
  Exemple: La fonction de contrôle de date vérifie la validité d'une date et retourne une valeur booléenne dépendant du résultat du contrôle

Une procédure ou une fonction utilise ou non des **arguments** (paramètres): la procédure "ajout d'un stagiaire" n'a pas de paramètre, la fonction "contrôle d'une date" utilise un paramètre représentant la date à contrôler.

Les arguments peuvent être modifiés ou non par la procédure ou la fonction

Pour une fonction et une procédure, on distinguera bien:

- La **déclaration** qui représente la procédure ou la fonction avec ses instructions
- L'**appel** qui correspond à son utilisation à l'intérieur d'un autre composant

2 Procédure

2.1 Déclaration

PROCEDURE *nom_procédure* (**VAL** *type nom_arg1*, …, **VAR** *type nom_argn*,…)
    Déclarations des variables locales

    Actions
FIN PROCEDURE

- Les arguments sont de 2 types:
  VAL: ils contiennent une valeur qui sera utilisée dans la procédure
  VAR: ils représentent une variable qui pourra être lue et modifiée dans la procédure
- Les variables que l'on déclare localement dans la procédure ne sont connues que de cette procédure

2.2 Appel

nom_procédure (*arg1,…, argn)*

2.3 Exemples

Procédure avec arguments de type valeur

```
PROCEDURE Ranger(VAL chaîne lieu)
   ECRIRE "Ranger ce livre dans", lieu
FIN PROCEDURE
```

Cette procédure est utilisée ainsi:

```
Si type_livre="scol" alors
   Ranger("BOITESCOL")
Sinon
   Ranger("BOITEDIV")
Finsi
```

Procédure avec arguments de type variable

```
PROCEDURE Classifier(VAL entier type_livre, VAR chaîne catégorie, VAR chaîne lieu)
   SELON type_livre
     Quand = 1
        Catégorie="Scolaire > 95"
        Lieu="BIBLIO"
     Quand = 2
        Catégorie="Scolaire < 95"
        Lieu="BOITESCOL"
     Quand = 3
        Catégorie="Poche-Roman"
        Lieu="BIBLIO"
     Quand = 4
        Catégorie="Poche-Divers"
        Lieu="BOITEDIV"
     Quand = 5
        Catégorie="Non Poche"
        Lieu="BIBLIO"
   FINSELON
FIN PROCEDURE
```

Cette procédure est utilisée ainsi:
ENTIER code_type
CHAINE libellé_type
CHAINE endroit
ECRIRE "Quel type ?"
LIRE code_type
Classifier(code_type, libellé_type, endroit)
ECRIRE "Le livre est de type ", libellé_type, "et sera rangé dans ", endroit

3 Fonction

Les explications données pour les procédures s'appliquent aux fonctions.
Une fonction a la spécificité de retourner une valeur, donc représente non seulement un traitement mais également une variable avec un type déclaré

3.1 Déclaration

type **FONCTION** *nom_fonction* (**VAL** *type nom_arg1*, …, **VAR** *type nom_argn*,…)
   Déclarations des variables locales

   Actions
   RETOURNE valeur
FIN FONCTION

3.2 Appel

variable ← nom_fonction(arg1, … argn)

SI *nom_fonction(arg1, … argn) = … alors …*
   ECRIRE *nom_fonction(arg1, … argn)*
FINSI

3.3 Exemples

Exemple 1
ENTIER FONCTION valeur_absolue (**VAL ENTIER** nombre)
   SI (nombre < 0) **ALORS**
      RETOURNE (nombre * (-1))
   SINON
      RETOURNE nombre
   FINSI
FIN FONCTION

Cette fonction est utilisée ainsi:
i <-- valeur_absolue(i)
limite <-- valeur_absolue(-120)

Exemple 2

```
BOOLEEN FONCTION Classifier(VAL ENTIER type_livre, VAR CHAINE catégorie,
VAR CHAINE lieu)
   RETOURNE vrai
   SELON type_livre
      QUAND 1
         Catégorie="Scolaire > 95"
         Lieu="BIBLIO"
      QUAND 2
         Catégorie="Scolaire < 95"
         Lieu="BOITESCOL"
      QUAND 3
         Catégorie="Poche-Roman"
         Lieu="BIBLIO"
      QUAND 4
         Catégorie="Poche-Divers"
         Lieu="BOITEDIV"
      QUAND 5
         Catégorie="Non Poche"
         Lieu="BIBLIO"
      AUTREMENT
         RETOURNE faux
   FINSELON
FIN FONCTION
```

Cette fonction est utilisée ainsi:

```
ENTIER code_type
CHAINE libellé_type
CHAINE endroit
ECRIRE "Quel type ?"
LIRE code_type
SI Classifier(code_type, libellé_type, endroit) = vrai ALORS
   ECRIRE "Le livre est de type ", libellé_type, "et sera rangé dans ", endroit
SINON
   ECRIRE " Le type de livre ", code_type, " n'est pas reconnu"
FINSI
```

4 Quelques règles

- La procédure ou la fonction aura un nom significatif
- Le nombre de paramètres ne doit pas être trop grand (en général inférieur à 5) car cela nuit à la lisibilité du programme
- Une procédure ou une fonction doit être la plus générale possible de manière à pouvoir être réutilisée dans d'autres circonstances
- Si le but d'une procédure est de calculer une valeur simple, il est préférable d'en faire une fonction.

5 Travaux pratiques

5.1 Algorithme

Certains de ces énoncés ayant déjà fait l'objet d'une étude dans un dossier précédent, vous reprendrez les programmes existants en les adaptant pour les transformer en procédures ou fonctions.

Enoncé 1:
Ecrire et utiliser une procédure inversant le contenu de 2 variables a et b passées en arguments

Enoncé 2:
Ecrire et utiliser une procédure calculant le périmètre et l'aire d'un triangle en connaissant les 3 côtés
Périmètre = p = a + b + c et aire = $((p/2-a)(p/2-b)(p/2-c))^{1/2}$

Enoncé 3:
Ecrire et utiliser une fonction calculant la moyenne de 2 nombres

Enoncé 4:
Ecrire et utiliser une fonction déterminant si une année passée en paramètre est bissextile ou non

Enoncé 5:
Ecrire et utiliser une fonction retournant le jour de la semaine d'une date donnée

Enoncé 6:
Ecrire et utiliser une procédure qui reçoit une chaîne de caractères et qui retourne cette même chaîne inversée (Le dernier caractère devient le premier, l'avant dernier le deuxième…). Un seul argument sera utilisé.

5.2 Codage

Après avoir formulé clairement et vérifier les algorithmes, vous procéderez au codage de tous les énoncés en utilisant les éléments de langage suivants:

Déclaration d'une procédure: (**passage de paramètre par valeur**)

void *nom_procedure* (**[type]** *arg1, …*) **{**
    instructions ;
}

Déclaration d'une procédure: (**passage de paramètre par variable**)

void *nom_procedure* **(ref [type]** *arg1, …*) **{**
    instructions ;
}

Exemples :

```
// passage de paramètres par valeur à une procédure
using System;
public class param2{
 public static void Main(){
 int age=20;
 changeInt(age);
 Console.Out.WriteLine("Paramètre effectif age="+age);
 }

 private static void changeInt(int a){
 a=30;
 Console.Out.WriteLine("Paramètre formel a="+a);
 }
}


// passage de paramètres par référence à une procédure
using System;
public class param2{
 public static void Main(){
int age=20;
changeInt(ref age);
 Console.Out.WriteLine("Paramètre effectif age="+age);
 }
```

```
        private static void changeInt(ref int a){
        a=30;
        Console.Out.WriteLine("Paramètre formel a="+a);
        }
    }
```

Déclaration d'une fonction: (passage de paramètre par valeur ou par référence)

[type_retour] *nom_fonction* (**[ref][type]** arg1, …) **{**
instructions ;
return type_retour ;
}

Exemple:

```
public float somme(float nb1, float nb2) {
   float s ;
   s=nb1+nb2 ;
   return s ;          //valeur de retour
}
```

Appel d'une fonction:

Variable = nom_fonction(liste arguments)

Exemple:

```
float    résu,nombre1,nombre2;
  ……..
  résu= somme(nombre1,nombre2) ;
```