Règles d'écriture Algorithmique

Plan Qualité  
Version 2.0

Sommaire

[1. Description d'un module algorithmique 3](#_Toc282002406)

[1.1. Exemple complet 3](#_Toc282002407)

[1.2. Description d'un module 4](#_Toc282002408)

[2. Règles de la programmation structurée 5](#_Toc282002409)

[2.1. Structures fondamentales de contrôle 5](#_Toc282002410)

[2.1.1. La séquence 5](#_Toc282002411)

[2.1.2. L'alternative 5](#_Toc282002412)

[2.1.3. Le choix multiple 6](#_Toc282002413)

[2.1.4. Tant que 6](#_Toc282002414)

[2.1.5. Jusqu'à 6](#_Toc282002415)

[2.1.6. Pour 6](#_Toc282002416)

[2.1.7. Boucle infinie 7](#_Toc282002417)

[2.2. Gestion des erreurs 8](#_Toc282002418)

[2.2.1. Gestion sans paramètre 8](#_Toc282002419)

[2.2.2. Gestion avec paramètre d’erreur 8](#_Toc282002420)

[2.2.3. Traitement d'une interruption 9](#_Toc282002421)

[3. Les types de données 10](#_Toc282002422)

[3.1. Le type booléen 10](#_Toc282002423)

[3.2. Le type caractère 10](#_Toc282002424)

[3.3. Le type entier 10](#_Toc282002425)

[3.4. Le type réel 11](#_Toc282002426)

[3.5. Le type énuméré 11](#_Toc282002427)

[3.6. Le type pointeur 11](#_Toc282002428)

[3.7. Le type chaîne 11](#_Toc282002429)

[3.8. Le type tableau 12](#_Toc282002430)

[3.9. Le type composé 12](#_Toc282002431)

[3.10. Le type conteneur 12](#_Toc282002432)

[4. Les affectations 12](#_Toc282002433)

[*5.* Opérateurs sur les variables 13](#_Toc282002434)

[6. Exemple en Version procédurale 14](#_Toc282002435)

[7. Exemple en Version objet 15](#_Toc282002436)

# Description d'un module algorithmique

Un module (procédure, méthode) doit décrire les informations suivantes :

* Nom de l’activité,
* Définition de ces paramètres
* Définition de ces données locales,
* Description de l'utilisation de données globales (données membres de la classe)
* Algorithme des actions.

## Exemple complet

**testeur**(cotesEtalon : tableau d’entiers, tolerances : tableau d’entiers)

**description**

Test 24h/24 de pièces entrant sur le poste, les **cotes** de la pièce testée doivent toute être dans les intervalles de **tolérances** des **cotesEtalon**, sinon la pièce est évacuée. Toute les 100 pièces défectueuses, on informe le superviseur.

Si la machine de test ne réponds pas (panne), le test est arrêté.

Le nombre de cotes à tester reste à définir

**preCondition**

Il n’y a pas de pièce sur le poste de test,

**cotesEtalon** sont des entiers positifs,

**tolerances** sont des entiers strictement positifs.

const

NB\_MAX = 100 ' on avertit le superviseur

var

nbPiècesMauvaises : entier ' compte le nombre de pièces mauvaises

cotes : tableau d’entiers ' côtes de la pièce courante

exception

si pb de communication avec machine de test

signaler la panne au superviseur

si réponse du superviseur <> STOP

reprendre ' on recommence le même traitement

finSi

finSi

si pb de communication avec superviseur

le signaler

finSi

début

initialisation de la machine de test ' peut générer une erreur

nbPiècesMauvaises  = 0

boucle

attendre la présence d'une pièce

demander les cotes à la machine de test ' peut générer une erreur

si cotes hors tolérance des cotesEtalon

sortir la pièce

incrémenter nbPiècesMauvaises

si nbPiècesMauvaises  = NB\_MAX

informer le superviseur

nbPiècesMauvaises = 0

finSi

finSi

finBoucle ' fin du bloc faire

**fin**

## Description d'un module

nom(déclaration des paramètres)

description

commentaire sur ce module

preCondition

condition nécessaire à l'exécution de cet algorithme

type

Declaration des nouveaux types

const

déclaration des constantes

var

déclaration des variables locales

exception

si évènement1

action1

finSi

si événement2

action2

finSi

début

pseudo code de l'algorithme

fin

# Règles de la programmation structurée

* Chaque module (activité) a un seul point d'entrée et un seul point de sortie.
* Tout module peut être remplacé par deux modules en séquence.
* Tout module peut être remplacé par une des **7 structures de contrôle** :
  + **séquence**,
  + itération **(tantQue, jusqu'à, pour),**
  + alternative **(si, si/sinon, suivant)**.
* On peut appliquer les 2 règles précédentes aussi souvent que l'on désire et dans n'importe quel ordre.
* Commencer par décrire le module le plus global.
* La longueur d'un module ne dépasse pas 'une page'. Il est facile à lire et à comprendre.
* On remplace tout pseudo code confus ou complexe par la description dans un autre module de cette activité.
* Les données globales sont limitées et si possible remplacées par des passages de paramètres.

## Structures fondamentales de contrôle

### La séquence

effacer l'écran

afficher le menu

demander le choix

### L'alternative

var

choix : caractère

sortir : booléen

………

début

………

demander le choix

si choix = 'q'

fermer le fichier

sortir = VRAI

sinon

traitement

finSi

……

fin

### Le choix multiple

var

choix : entier

fin : booléen

………

début

………

afficher menu

demander le choix

suivant choix

1 : ajouter

2 : supprimer

3,5,6 : insérer ' énumération

4 : copier

7 à 10 : quitter ' intervalle de validité

autre:

afficher 'caractère erroné'

finsuivant

………

### Tant que

**tantQue** le tapis est libre

demander au robot de prendre une pièce

déposer la pièce sur le tapis

finTantQue

### Jusqu'à

répéter

prendre une pièce

la poser dans la caisse

jusqu'à caisse pleine

### Pour

pour variable de début à fin pas incrément

action 1

action 2

finPour

Exemple :

var

chiffre, résultat : entier

………

début

………

résultat = 0

pour chiffre de 1 à 10

résultat = résultat + chiffre

finPour

afficher résultat

...

Autre possibilité liée à la théorie des ensemble (très utile en environnement objet), L'ensemble peut être un tableau, une liste, un conteneur, etc.L'ordre de traitement des éléments importe peu.

pour chaque élément dans ensembleA

si élément dans ensembleB

traiter l'élément

finSi

finPour

Exemple :

longueur(texte : chaîne, alpha : entier) : entier

var

caractère : caractère ' caractère en traitement ds la chaîne

nbCar : entier ' nombre de caractères de texte

début

pour chaque caractère dans texte

nbCar = nbCar + 1

si caractère est alphanumérique ' si élément dans ensembleB

alpha = alpha + 1

FinSi

finPour

longueur = nbCar ' valeur de retour

fin

### Boucle infinie

boucle

Attendre une seconde

Incrémenter l'heure

Afficher l'heure

finBoucle

## Gestion des erreurs

Les ‘petites’ erreurs doivent être gérées localement en utilisant les règles algorithmiques classiques. Les erreurs graves (utilisateur ou système) mettent en œuvre le principe des exceptions (division par zéro, écriture d’un fichier impossible, etc.).

### Gestion sans paramètre

module1 \* gestion simple avec sortie du module en cas d’erreur

exception

si err1

afficher ‘erreur grave’ \* puis sort du module module1

finSi

début

boucle

tant que condition

action1

action2 \* génère une erreur err1 dans certains cas

action3

finTantQue

action4

action5

finBoucle

fin

### Gestion avec paramètre d’erreur

exception

si err2 (texte : chaîne) ' exception avec un paramètre textuel

afficher texte

continuer ' reprend l’exec sur la ligne suivante

finSi

début

boucle

tantQue condition

action1

action2

si condition2

erreur err2, ‘action3 impossible’ ' provoque une exception

finSi

action3

finTantQue

action4

action5

finBoucle

fin

**reprendre** : reprend l’exécution sur la ligne qui a provoqué l’erreur.

**continuer** : reprend l'exécution sur la ligne qui suit celle qui a provoqué l'erreur.

L'absence d'instruction fait que le programme s'arrête. Si il n’y a pas de structure de gestion des exceptions dans le module courant, la sortie du bloc se fait jusqu’à la rencontre d’une structure exception ou arrête le programme si il n’y en a pas.

### Traitement d'une interruption

L'interruption est considéré comme une exception on écrira donc :

module1(…)

…

exception

si IT1

Pseudo-code de la routine d'interruption

finSi

début

…

fin

Si une interruption IT1 survient, le traitement normal est suspendu et le traitement d'exception s'exécute. Le traitement normal reprend ensuite.

##### Remarque

La condition d'une structure alternative ou itérative est booléenne c'est à dire qu'elle a la valeur Vrai ou Faux. Si l'expression testée est booléenne, il ne faut pas écrire la condition suivante :

si trouvé = Faux ' erreur !!!

mais :

si non trouvé ' bien !!!

La commande **reprendre** permet de relancer l’exécution sur la ligne qui a provoqué l’erreur.

Si la condition2 est vrai, on génère une exception qui arrête le traitement en cours. On exécute le traitement de l'exception et on reprend le traitement sur la ligne qui suit l’exception (action5).

Si il n’y a pas de structure de gestion des exceptions dans le module courant, la sortie du bloc se fait jusqu’à la rencontre d’une structure exception ou arrête le programme si il n’y en a pas.

# Les types de données

Les types simples

* les types discrets : entier, caractère, booléen, énuméré , pointeur.
* les types continus : réel.
* Les types structurés
* chaîne, tableaux, composition, conteneurs (ou collection).

## Le type booléen

Il va permettre de stocker les valeurs VRAI ou FAUX

trouvé : booléen

si trouvé

traitement

trouvé = FAUX

finSi

Opérateurs spécifiques : et, ou, non, xou.

## Le type caractère

Il va permettre de stocker un caractère ASCII (sur 7 bits)

carLu : caractère

suivant carLu

'a' : traitementA

'c' : traitementC

....

autre : traitementDéfaut

finSuivant

## Le type entier

Il va permettre de gérer des compteurs ou des valeurs entières

nbCaisses : entier

nbCaisses = 0

tant que il y a une nouvelle caisse ou nbCaisses = 10

prendre la caisse

incr nbCaisses

finTantQue

## Le type réel

Il va permettre de stocker des valeurs réelles

température : réel

lire température

si température >= 22,5

refroidir

finSi

## Le type énuméré

Il va permettre de stocker des données qui ont un ensemble de valeurs très limitées et en général conceptuelle.

couleurs : [rouge | jaune | bleu | blanc | noir]

étatPoste : [occupé | libre | enAttente | enPanne]

choix : ['A' | 'F' | 'S' | 'Q']

## Le type pointeur

Il va permettre de faire référence à un objet d'un type donné.

ptrVal : pointeur d'entier

ptrID : pointeur de personne

nombre = (ptrVal)

## Le type chaîne

Il va permettre de stocker des textes

texte : chaîne

réponse : chaîne

texte = ‘sésame’

réponse = texte + ‘ tu vas t’ouvrir !‘

## Le type tableau

Permet de stocker plusieurs valeurs de même type (celui du tableau) dans une donnée unique (le tableau), la valeur sera accessible par un itérateur (indice dans le tableau).

notes : tableau de NB\_MAX réels ' pour une classe

résultats : tableau de 300 entiers ' suivi d'un patient

image : tableau de 800 par 600 pixels ' écran SVGA couleur

pixels : entier ' correspondant à une couleur

## Le type composé

Ce sont des données qui sont composées d'un ensemble de plusieurs données. Comme, par exemple, une fiche contenant diverses informations.

type

personne = nom + prénom + age + dateNaissance

dateNaissance = jour + mois + année

nom, prénom = chaîne

age = entier

jour, mois, année = entier

var

point : X + Y

groupe : liste de 200 Personne

## Le type conteneur

Un type «conteneur» est une construction destinée à contenir un ensemble de valeurs. Le seul type conteneur généralement considéré comme un type de base est le tableau.

Le conteneur peut prendre diverses formes : liste, pile, arbre binaire, ensemble à clef, etc..

Ils sont très utilisés en programmation objet. Ce sont des types **abstraits**.

# Les affectations

Elles se font de droite à gauche avec le signe '='.

Exemples :

valeur = 5.21 + 2 ' affectation d'une valeur

moyenne = somme/nbElèves ' affectation de la division

' des contenus des variables

texte = 'Titre du chapitre'

car = 'a'

notes(nbId) = 12.3

présent = VRAI

Affectation de type énuméré :

feu : [rouge | orange | vert | clignotant | éteint]

feu = orange

Affectation de structure composée :

notes.présent = VRAI

notes.valeur = 12.5

notes(i) = notes(j)

personne.nom = 'Dupont'

groupe(23).dateNaissance.jour = 12

Affectation de tableaux :

i, j : entier

tab(i,j) = tab(i-1, j-1) + tab(i-1, j)

tab(i) = 0

# Opérateurs sur les variables

***Arithmétiques :*** +, -, \*, /, modulo, div, etc.

***Logiques :*** et, ou, non, xou, etc.

***Comparaisons :*** <, >, <=, >=, =, <>, etc.

# Exemple en Version procédurale

**Dictionnaire de données (données globales)**

**type**

note = présence + valeur

présence = booléen ' présence de l'élève au contrôle

valeur = réel ' valeur de la note si l'élève est présent

élève = chaîne ' nom de l'élève

**const**

NB\_MAX = 100 ' nombre maximal d'élèves ds une classe

**var**

Notes : liste de NB\_MAX note ' notes de la classe entière

élèves : liste de NB\_MAX élève

nbElèves : entier ' nombre d'élèves dans la classe

moyenne : réel

CalculDeLaMoyenne() : réel

var

somme : réel ' somme partielle de toutes les notes

numId : entier ' index d'un élève

début

saisir les notes

somme = 0

pour numId de 1 à nbElèves

somme = somme + notes(numId)

finPour

moyenne = somme / nbElèves

afficher la moyenne

fin

saisirLesNotes()

var

numId : entier ' index d'un élève par valeur

début

pour numId de 1 à nbElèves

afficher 'Note de ', élève(numId)

lire notes(numId).valeur

finPour

fin

# Exemple en Version objet

MaClasse : public AutreClasse ' ici avec héritage

invariant

………………

public

const

NB\_MAX = 100 ' nombre max d'élèves dans une classe

var

nbElèves : entier ' nombre d'élèves dans la classe

moyenne : réel

méthode

calculDeLaMoyenne()

protégé

var

…………

méthode

saisirLesNotes()

privé

type

note = présence + valeur

présence = booléen ' présence de l'élève au contrôle

valeur = réel ' note si élève présent

élève = chaîne ' nom de l'élève

var

notes : liste de NB\_MAX note ' notes de la classe entière

élèves : liste de NB\_MAX élève

méthode

calculDeLaMoyenne()

var

somme : réel ' somme partielle de toutes les notes

numId : entier ' index d'un élève

préCondition

…………

exception

…………

début

saisir les notes

somme = 0

pour numId de 1 à nbElèves

somme = somme + notes(numId)

finPour

moyenne = somme / nbElèves

fin

saisirLesNotes()

var

numId : entier ' index d'un élève par valeur

début

pour numId de 1 à nbElèves

afficher 'Note de ', élève(numId)

lire notes(numId).valeur

finPour

fin

……………

fin