Algorithmique

Chapitre 2: Programmation fonctionnelle

Dr. N'golo KONATE

konatengolo@ufhb.edu.ci

Programmation fonctionnelle

1. Généralité sur la programmation fonctionnelle
2. Procédure
3. Fonctions
4. Variables locales et globales

Généralités sur la programmation fonctionnelle

Activité

Activité: Ecrire un algorithme qui permet d'obtenir le minimum et le maximum d'un ensemble de trois nombres, et suivant l'ordre de saisie de ces nombres, résoudre l'équation du premier degré associée au premier couple de nombres, et rechercher les racines du polynôme du second degré associé à ce triplet.

Solution

```
Algorithme MultiTaches
Var a, b, c, min, max, delta, sol1, sol2 : réel
Début
écrire ("Un peu de maths")
écrire ("Saisie des nombres")
écrire ("Donnez le premier nombre : ")
lire (a)
écrire ("Donnez le deuxième nombre : ")
lire (b)
écrire ("Donnez le troisième nombre : ")
lire (c)
écrire ("Résultats des différents traitements")
écrire ("Recherche du minimum des nombres", a, ",", b, " et ", c)
min ← a
Si b < min
Alors min ← b
FinSi
Si c < min
Alors min ← c
FinSi
écrire ("Le minimum est ", min)
```

Solution

```
écrire ("Recherche du maximum des nombres", a, ",", b, " et ", c)
max ← a
Si b > max
Alors max ← b
FinSi
Si c > max
Alors max ← c
FinSi
écrire ("Le maximum est ", max)
```

```
écrire ("Résolution de l'équation ", a, "x + ", b, " = 0")
Si (a = 0)
Alors Si (b = 0)
 Alors écrire ("L'ensemble des solutions est R")
 Sinon écrire ("L'ensemble des solutions est l'ensemble vide")
 FinSi
Sinon écrire ("La solution est ", (-b/a))
FinSi
```

Solution

```
écrire ("Recherche des racines de l'équation ", a, "x² + ", b, "x + ", c, " = 0")
Si (a = 0)
Alors Si (b = 0)
 Alors Si (c = 0)
 Alors écrire ("Infinité de solutions")
 Sinon écrire ("Pas de solution")
 FinSi
 Sinon écrire ("Une solution unique : ", -c/b)
 FinSi
Sinon delta ← b*b – 4*a*c
Si (delta < 0)
Alors écrire ("Pas de solution")
 Sinon Si (delta = 0)
 Alors écrire ("Une solution unique : ", -b/(2*a))
 Sinon écrire ("Deux solutions distinctes")
 écrire ("Première solution : ", (-b - delta^0.5) / (2*a))
 écrire ("Seconde solution : ", (-b + delta^0.5) / (2*a))
 FinSi
 FinSi
FinSi
Fin
```

Limites des algorithmes classiques

Pour cette activité, on remarque que:

- ✓Le code est long
- ✓Il contient des traitements répétitifs
- ✓Sa maintenance sera complexe

Limites des algorithmes classiques

Pour outre passer ces limites, il faudrait:

Limites des algorithmes classiques

Pour outre passer ces limites, il faudrait:

✓Décomposer le problème en sous taches a résoudre

Limites des algorithmes classiques

Pour outre passer ces limites, il faudrait:

✓Décomposer le problème en sous taches a résoudre

✓Analyser les instructions a écrire dans chaque sous taches

Limites des algorithmes classiques

Solutions

Pour outre passer ces limites, il faudrait:

- ✓Décomposer le problème en sous taches a résoudre
- ✓Analyser les instructions a écrire dans chaque sous taches

Ce processus de décomposition d'un algorithmes en sous algorithmes introduit la notion de programmation modulaires ou fonctionnelle.

Limites des algorithmes classiques

Solutions

Pour outre passer ces limites, il faudrait:

✓Décomposer le problème en sous taches a résoudre

✓Analyser les instructions a écrire dans chaque sous taches

Ce processus de décomposition d'un algorithmes en sous algorithmes introduit la notion de programmation modulaires ou fonctionnelle. Il existe deux types de sous-algorithmes:

Limites des algorithmes classiques

Solutions

Pour outre passer ces limites, il faudrait:

- ✓Décomposer le problème en sous taches a résoudre
- ✓Analyser les instructions a écrire dans chaque sous taches

Ce processus de décomposition d'un algorithmes en sous algorithmes introduit la notion de programmation modulaires ou fonctionnelle. Il existe deux types de sous-algorithmes:

- ✓Les procédures
- ✓Les fonctions

Limites des algorithmes classiques

Solutions

L'algorithme précèdent peut être décomposé en taches suivantes:

- ✓rechercher le minimum de trois nombres
- ✓rechercher le maximum de trois nombres
- ✓résoudre une équation du premier degré
- ✓rechercher les racines d'un polynôme du second degré.

Limites des algorithmes classiques

Solutions

L'algorithme précèdent peut être décomposé en taches suivantes:

- ✓rechercher le minimum de trois nombres
- ✓rechercher le maximum de trois nombres
- ✓résoudre une équation du premier degré
- ✓rechercher les racines d'un polynôme du second degré.

Chaque tâche constitue un sous-algorithme.

Sous algorithmes

```
Algorithme EssaiAlgoModulaire
Var nbre1, nbre2, nbre3, min, max, delta, sol1, sol2 : réel
Début
écrire ("Un peu de maths")
écrire ("Saisie des nombres à traiter")
écrire ("Donnez le premier nombre : ")
lire (nbre1)
écrire ("Donnez le deuxième nombre : ")
lire (nbre2)
écrire ("Donnez le troisième nombre : ")
lire (nbre3)
écrire ("Résultats des différents traitements")
    minimum
    maximum
    premierDegre
    secondDegre
Fin
```

Procédures

Les procédures

Une procédure est un bloc d'instructions nommé et déclaré dans l'entête de l'algorithme et appelé dans son corps à chaque fois que le programmeur en a besoin.

Les procédures

Une procédure est un bloc d'instructions nommé et déclaré dans l'entête de l'algorithme et appelé dans son corps à chaque fois que le programmeur en a besoin.

Une procédure étant un objet algorithmique, il faut la déclare avant de l'utiliser.

On déclare une procédure en mentionnant son en-tête dans la zone de déclarations de l'algorithme principal selon la syntaxe ci-dessous :

Procédure <nom de la procédure> (param_1:type,……,param_n: type)

Les procédures

Une procédure étant un objet algorithmique, il faut la déclare avant de l'utiliser.

Procédure Nom_Procédure (param_1:type,……,param_n: type);

Const (identificateurs) ← (valeurs)

Var < identificateurs des variables locales > : < types des variables locales >

Début

…

Instructions ; Corps de la procédure

…

Fin ;

Les procédures

L'appel d'une procédure

On appelle une procédure par son nom, et cet appel s'effectue dans le corps de l'algorithme appelant selon la syntaxe suivante :

nomProcédure (liste des paramètres réels)

Les procédures

L'appel d'une procédure

On appelle une procédure par son nom, et cet appel s'effectue dans le corps de l'algorithme appelant selon la syntaxe suivante :

nomProcédure (liste des paramètres réels)

L'appel d'une procédure déclenche immédiatement l'exécution des instructions de cette procédure. Ce qui signifie que si la procédure n'a pas été décrite préalablement, son appel ne déclenchera aucun traitement.

Les procédures

Passage de paramètres

Lorsqu'un algorithme appelle une procédure, il lui transmet des paramètres : on dit qu'il y a passage de paramètres (ou transmission de paramètres).

Les procédures

Passage de paramètres

Lorsqu'un algorithme appelle une procédure, il lui transmet des paramètres : on dit qu'il y a passage de paramètres (ou transmission de paramètres).

Il y a deux points de vue pour le passage de paramètres : celui de l'algorithme appelant et celui de la procédure. En effet :

Les procédures

Passage de paramètres

Lorsqu'un algorithme appelle une procédure, il lui transmet des paramètres : on dit qu'il y a passage de paramètres (ou transmission de paramètres).

Il y a deux points de vue pour le passage de paramètres : celui de l'algorithme appelant et celui de la procédure. En effet :

- ✓ Du côté de l'algorithme appelant, les valeurs transmises sont **les paramètres effectifs** ;
- ✓ du côté de la procédure, les variables sont des **paramètres formels**

Les procédures

Passage de paramètres

A l'appel de la procédure:

- ✓La liste des paramètres effectifs est mise en correspondance (ou en synchronisation) avec la liste des paramètres formels
- ✓La valeur stockée dans chaque paramètre effectif est copiée dans le paramètre formel lui correspondant

Les procédures

Passage de paramètres

Le mode de transmission entre ces deux contextes est la transmission par valeur.

Lorsqu'une procédure a terminé sa tâche, le déroulement de l'algorithme appelant reprend à l'instruction qui succède à l'appel de la procédure.

Les résultats obtenus par une procédure sont exploitables par l'algorithme appelant par l'intermédiaire des variables globales.

Resolution de l'activité avec les procédures

```
Algorithme AlgoPrincipal
// déclaration des procédures
Procédure minimum (x : réel, y : réel, z : réel)
        maximum (x : réel, y : réel, z : réel)
        premierDegre (p : réel, q : réel)
        secondDegre (p : réel, q : réel, r : réel)
        rechercherRacines (e : réel, f : réel, g : réel)
// déclaration des variables globales
Var nbre1, nbre2, nbre3 : réel
Début
écrire ("Un peu de maths")
écrire ("Saisie des nombres…")
écrire ("Donnez le premier nombre : ")
lire (nbre1)
écrire ("Donnez le deuxième nombre : ")
lire (nbre2)
écrire ("Donnez le troisième nombre : ")
lire (nbre3)
écrire ("Résultats des différents traitements")
// appel des procédures
minimum (nbre1, nbre2, nbre3)
maximum (nbre1, nbre2, nbre3)
premierDegre (nbre1, nbre2)
rechercherRacines (nbre1, nbre2, nbre3)
Fin
```

Resolution de l'activité avec les procédures

Procédure minimum

```
Procédure minimum (x : réel, y : réel, z : réel)
Var min : réel // déclaration d'une variable locale
Début
écrire ("Recherche du minimum des nombres", x, ",", y, " et ", z)
 min ← x
 Si y < min
 Alors min ← y
 FinSi
 Si z < min
 Alors min ← z
 FinSi
 écrire ("Le minimum est ", min)
Fin
```

Resolution de l'activité avec les procédures

Procédure maximum

```
Procédure maximum (x : réel, y : réel, z : réel)
Var max : réel // déclaration d'une variable locale
Début
 écrire ("Recherche du maximum des nombres", x, ",", y, " et ", z)
 max ← x
 Si y > max
 Alors max ← y
 FinSi
 Si z > max
 Alors max ← z
 FinSi
 écrire ("Le maximum est ", max)
Fi
```

Fonctions

Les fonctions

Une fonction est une procédure particulière qui génère un seul résultat.

Elle possède deux propriétés fondamentales :

- ✓ le résultat généré est directement exploitable dans une instruction ;
- ✓ la fonction est elle-même le résultat généré.

Les fonctions

Une fonction est une procédure particulière qui génère un seul résultat.

Elle possède deux propriétés fondamentales :

- ✓ le résultat généré est directement exploitable dans une instruction ;
- ✓ la fonction est elle-même le résultat généré.

On déclare une fonction en mentionnant son en-tête dans la zone de déclarations de l'algorithme principal selon la syntaxe ci-dessous :

Fonction < nomFonction > (param_1:type,…,param_n:type) : type du résultat

Les fonctions

Description d'une fonction

Fonction <nomActionNommée> (<liste des paramètres formels> : type) : <type du résultat>

Const < identificateurs des constantes > ← < valeurs des constantes >

Var < identificateurs des variables locales > : < types des variables locales >

Début

< instruction(s) ou action(s) >

retourner (r) // r est le résultat de la tâche de la fonction

Fin

Les fonctions

Appel d'une fonction

L'appel d'une fonction s'effectue dans un algorithme principal.

Il peut se faire par:

✓Une instruction simple (affichage, affectation)

✓ Dans une structure de contrôle (conditionnelle, repetitive)

Les fonctions

Appel d'une fonction

L'appel d'une fonction s'effectue dans un algorithme principal.

Il peut se faire par:

✓Une instruction simple (affichage, affectation)

✓ Dans une structure de contrôle (conditionnelle, répétitive)

Le passage des paramètres se réalise comme dans le cas des procédures.

Cependant avec les fonctions, puisque la fonction retourne un seul résultat et qu'elle est elle-même cette valeur de retour, alors on n'a pas besoin de variable globale dédiée au retour du résultat.

Resolution de l'activité avec les procédures

Algorithme **AlgoPrincipal**

// déclaration des fonctions

fonction minimum (x : réel, y : réel, z : réel): réel

maximum (x : réel, y : réel, z : réel): réel

premierDegre (p : réel, q : réel): réel

secondDegre (p : réel, q : réel, r : réel): réel

rechercherRacines (e : réel, f : réel, g : réel): réel

// déclaration des variables globales

Var nbre1, nbre2, nbre3 : réel

Début

écrire ("Donnez le premier nombre : ")

lire (nbre1)

écrire ("Donnez le deuxième nombre : ")

lire (nbre2)

écrire ("Donnez le troisième nombre : ")

lire (nbre3)

écrire ("Résultats des différents traitements")

Ecrire ("le minimum est:", **minimum** (nbre1, nbre2, nbre3))

Ecrire ("le maximum est:", **maximum** (nbre1, nbre2, nbre3))

Ecrire ("le maximum est:", **maximum** (nbre1, nbre2, nbre3))

Ecrire ("le premier dégré est:", **premierDegre** (nbre1, nbre2))

Fin

Resolution de l'activité avec les procédures

Algorithme **AlgoPrincipal**

// déclaration des fonctions

fonction minimum (x : réel, y : réel, z : réel): réel

maximum (x : réel, y : réel, z : réel): réel

premierDegre (p : réel, q : réel): réel

secondDegre (p : réel, q : réel, r : réel): réel

rechercherRacines (e : réel, f : réel, g : réel): réel

// déclaration des variables globales

Var nbre1, nbre2, nbre3 : réel

Début

écrire ("Donnez le premier nombre : ")

lire (nbre1)

écrire ("Donnez le deuxième nombre : ")

lire (nbre2)

écrire ("Donnez le troisième nombre : ")

lire (nbre3)

écrire ("Résultats des différents traitements")

Ecrire ("le minimum est:", **minimum** (nbre1, nbre2, nbre3))

Ecrire ("le maximum est:", **maximum** (nbre1, nbre2, nbre3))

Ecrire ("le maximum est:", **maximum** (nbre1, nbre2, nbre3))

Ecrire ("le premier dégré est:", **premierDegre** (nbre1, nbre2))

Fin

Resolution de l'activité avec les procédures

fonction maximum

```
fonction maximum (x : réel, y : réel, z : réel): réel
 Var max : réel // déclaration d'une variable locale
Début
 écrire ("Recherche du maximum des nombres", x, ",", y, " et ", z)
 max ← x
 Si y > max
 Alors max ← y
 FinSi
 Si z > max
 Alors max ← z
 FinSi
 retourner(max)
Fin
```