Cours Programmation Impérative 1

Les différents paradigmes de la programmation

Paradigme ~ modèle A expliquer

Nous distinguons deux paradigmes :

Programmation déclarative, Programmation fonctionnelle

Programmation déclarative : Relation

- Le programme est une description du problème (ex : base de connaissance).
- L'utilisateur interroge le programme.
- On est, d'une manière générale, indépendant de la machine.
- Les langages utilisés, d'une façon générale, permettent de décrire le problèmes sous forme de faits et de relations (exemple PROLOG). Ils sont, souvent, basés sur des formalismes logiques.

Exemple : arbre généalogique :

--Décrire le genre de chaque donnée avec les propriétés homme et femme

Programme :

Homme(Karl)
Homme(Jean)
Homme(Stephan)
Homme(Sami)
Homme(Albert)
Homme(Robert)

Femme(Pam)
Femme(Jeanne)
Femme(Sarah)
Femme(Samia)

Utilisation : expliquer comment ça déroule sur l'exemple

?Homme(Jean)
oui

?Homme Pam)
non
?Femme(X)
Pam
Jeanne
Sarah
Samia

--Décrire les relations parentales entre les différentes données : parent(a,b) qui signifie a est parent de b :

Programme :

Homme(Karl)
Homme(Jean)
Homme(Stephan)
Homme(Sami)
Homme(Albert)
Homme(Robert)

Femme(Pam)
Femme(Jeanne)
Femme(Sarah)
Femme(Samia)

Parent(Karl, Samia)
Parent(Pal, Samia)
Parent(Pam, Sarah)
Parent(Jean, Sarah)
Parent(Jean, Stephan)
Parent(Jeanne, Stephan)
Parent(Sami, Robert)
Parent(Samia, Robert)
Parent(Samia, Albert)
Parent(Stephan, Albert)

-- Décrire les relations père, mère, grand_père et grand_mère :
-- On utilise des représentants (arguments). Les relations sont conditionnelles.

Pere(X,Y) ← (Homme(X), Parent(X,Y)

A← B signifie A est vérifié si B est vérifié
-- A,B signifie A et B « et logique »

Mere(X,Y) ←(Femme(X), Parent(X,Y)

Grand_pere(X,Y) ←(Pere(X,Z), Parent(Z,Y)
-- X est grand parend de Y s'il existe un Z tel que X est père de Z et ce même Z est parent de Y

Grand_mere(X,Y) ←(Mere(X,Z), Parent(Z,Y)

Utilisation : expliquer comment ça déroule sur l'exemple

? Grand_pere(Karl,Albert)
oui

Propriétés :

- Programmation qui s'appuie sur des concepts logiques qui sont assez expressives
- Le programmeur n'a pas à gérer la mémoire
- Intéressant pour le prototypage et pour mieux comprendre le problème
- Difficile à maintenir
- Problèmes d'efficacité

Programmation fonctionnelle : Fonction

- Le programme décrit un modèle de solution au problème.
- Il est considéré comme une fonction mathématique la sortie est une fonction de l'entrée.

Nous distinguons deux paradigmes :

Fonctionnelle pure et Impérative ou Procédurale

Fonctionnelle pure

- Le programme décrit un modèle de solution abstrait indépendant de la machine.
- Les structures de contrôles utilisées sont essentiellement basées sur les appels de fonctions.
- Ces fonctions manipulent les valeurs introduites en entrée.
- Les langages connus : Lisp, Scheme, ML, Caml

Exemple :

Ecrire un programme qui calcule la puissance a^N

Let rec Puissance = fun(a,N) →
if N=0 then 1 else a*Puissance(a, N-1) ; ;

Utilisation :

Puissance (2,3) ; ; retourne 8

-- a et N représentent les valeurs en entrée. La fonction décrit le modèle de comportement abstrait permettant de calculer la puissance.

Propriétés

- On s'appuie sur des concepts mathématiques c alcul
- Le programmeur n'a pas à gérer la mémoire, facile à utiliser
- Intéressant pour le prototypage

- Difficile à maintenir
- Problèmes d'efficacité

Programmation Impérative ou procédurale : Affectation

- ❖ Le programme est une description du comportement de la machine.
- ❖ Le programme est une procédure. (différence avec une fonction à expliquer)
- ❖ Le programme est composé d'une partie déclaration (résérvation mémoire) et une partie code
- ❖ le code est un ensemble d'instructions ordonnées dans le temps selon trois schémas de base : Séquence , sélection et répétition.
- ❖ C'est l'implémentation d'un modèle de solution.
- ❖ Langages : C, Pascal, Ada.

Programmation en C

Un programme C minimum, d'une manière générale, est une fonction « main » qui représente le programme principale.

Exemple1 : Programme C

```
/*le programme suivant calcule la somme de deux valeurs introduites en entrée et donne */
/*la somme en sortie */

#include <stdio.h>              /* inclure une interface librairie*/
main()          /* programme principale*/
                                /* description des données*/
        int Z, X, Y; /* 3 cases mémoires sont réservées de type entier */
        à expliquer

        {                       /* Début des instructions*/

                /* ce programme lit 2 valeurs et les mémorise dans X et Y ; calcule la somme
                dans dans  Z */
```

```
        printf(‘‘taper un entier positif ou nul‘‘\n) ;
        scanf(’’%d’’, &X) ;
        printf(‘‘taper un entier positif ou nul ‘‘\n) ;
        scanf(’’%d’’, &Y) ;
        Z =X+Y ;
        printf(’’%d’’, Z) ;
}
```

Propriétés

- Le programme est composé de deux parties :
- Zone de données qui doit être réservée et structurée par le programmeur
- Zone code ou instructions qui manipule la zone donnée afin d'atteindre les résultats souhaités
- Programmation qui peut être efficace
- Intéressante pour le développement
- Langages basés sur la déclaration de variables (réservation en mémoire) et sur des instructions élémentaires assez proches de la machine : affectation
- Langages fortement typés et compilés

Instructions et structures de contrôles de base

affectation **syntaxe** X =e
X est une variable
e est une expression de même type que X

sémantique informelle
évaluer e si pas d'erreur mémoriser sa valeur dans X

Exemple : int X ;

X =0 ; X =X+1 ;

Séquences

syntaxe **/* les a_i sont des instructions*/**
:
a_{i-1} ;
a_i ;
a_{i+1} ;

sémantique /* a_i s'exécute d'une manière inconditionnelle après a_{i-1}

Exemple échanger les valeurs de deux variables de type entier

/* = en commentaire signifie égalité contrairement à = dans le programme qui signifie affecteation*/

```
int X,Y;
X =5 ; Y =6 ;
                /* X=5 et Y = 6*/
X =X+Y ;        /* X = 11 et Y =6*/
Y =X-Y ;        /* X = 11 et Y = 5*/
X =X-Y ;        /* X= 6 et Y = 5*/
```

Sélection ou choix
syntaxe **if (c1) a1; [else a2 ;]**
tous ce qui est entre [ ] est facultatif
c1 est une expression booléenne et les **ai** sont des instructions

sémantique

calculer **c1** si vrai (>0) alors exécuter **a1** et ignorer le reste en passant après le **end if ;**
sinon (=0) alors ignorer **a1** et exécuter **a2** et passer après **;**
etc..

Exemple

if (A>0) B =A ; else B =-A ;

Répétition

syntaxe **while (c) a ;**

c est une expression booléénne
a est une instruction

sémantique

1. on évalue **c**
2. si elle est vraie on exécute a et on revient à 1 sinon on passe après **;**

Exemple
Ecrire un programme qui trie dans l'ordre croissant trois variables. Le trie s'effectue en permutant plusieurs fois ces variables.

```
{ int X ,Y,Z;
X =2 ;
Y =3 ;
Z =1 ;

while (X>Y || Y>Z)
        {if X>Y                 /* échanger X et Y*/
                {       X =X+Y ;
                        Y =X-Y ;
                        X =X-Y ;}
        if (Y>Z)
                {       Y =Z+Y ;        /* échanger Y et Z*/
                        Z =Y- Z ;
                        Y=Y-Z ; }
 ;} ;
}
```

Représentation, en C, des types simples et des structures de contrôles avec des exemples :

Les types

type :

- ensemble de valeurs
- ensemble d'opérations autorisées

types prédéfinis :

- **int**

{..-3, -2, -1, 0, 1, 2,….}
+, -, *, /
<, <=, >, >=, !=
: :

- **char**

Les caractères ASCII {..'a', 'b',…}
un ensemble ordonné

<, <=, >, >=, !=
: :

{NOMBRE POSITIF, 0)
&&, ||…

- **float**

-1.0, ……-0.99999

Toute variable doit être déclarée dans une unité de programme(sous-programme, bloc, etc…):
(identificateur en C : minuscule est différent du majuscule)
type_existant <identificateur> ;

int X;
float Y;
char C;

Une variable est une réservation mémoire qui est statique par rapport à l'unité de programme qui contient sa déclaration. Cet emplacement mémoire est accessible par l'identificateur de la variable et sa taille est déterminée en fonction de son type.

On peut les initialiser à la déclaration :

```
int X = 0;
float Y  =1.0;
char C  = 'c' ;
```