Paradigme de programmation Impératif

- La programmation impérative est caractérisée par,

la programmation avec un état et des commandes qui modifient l'état :

- Programmation impérative Procédurale

=

Programmation Impérative + sous-programmes

Mots-clès :

assignements, programmation structurée; GOTO, commandes, Flow de contrôle, état, variables, instructions, structures de contrôles

Paradigme impératif

- Abstraction des "computers" basée sur la machine de Turing et la machine de Von Neumann avec ses registres et mémoire.

- Les L. P impératifs fournissent un ensemble de commandes permettant de structurer le code et de manipuler la mémoire.

- chaque L.P définit une vue particulière du Matériel: Machine Pascal, Machine C, Machine JAVA
  =>La machine virtuelle est implémentée par le compilateur dans le langage supporté par le Matériel et le système d'exploitation

Paradigme impératif (principe)

- Un nom peut être assigné à une valeur à un point du programme, et plus tard attribué à nouveau à une autre valeur;

- Un état est constitué par la collection des noms et leurs valeurs associées, et l'emplacement du contrôle dans le programme;

  - l'état est donc un modèle logique qui est l'association entre l'emplacement mémoire et les valeurs;

  - Un programme en exécution génère une séquence d'états (figure).

  - Un changement d'état est déterminé par des opérations d'assignements et une séquence de commandes:

S0--- O0-→ S1 --- O1-→ S2 Sn-1 --- On-1-→ Sn:

Variables et assignements

Terminologie:
En *Architecture*, on parle de : configuration binaire, emplacement mémoire, adresse mémoire;

En *Langages de Programmation*: valeur, variable, nom;

Puisque les variables sont habituellement liées à un nom et à une valeur, le mot variable est souvent employé pour signifier le nom d'une valeur.

Définitions :
1. La plupart des descriptions des LP impératifs sont liées au matériel et aux considérations d'implémentation où :
   - un nom est lié à une adresse,
   - une variable à une cellule mémoire,
   - et une valeur à une configuration binaire.

➔un nom est associé à deux liaisons: une liaison à un emplacement et une liaison à une valeur.

2. L'emplacement s'appelle la l-valeur et la valeur s'appelle la r-valeur.

Nécessité ?: vient du fait de la mise en oeuvre de l'affectation.

Exemple, X: = X+2

Le X de gauche (Left) exprime l'emplacement,
alors que le X de droite (Right) dénote la valeur.

Variables et assignements

- Une variable peut être liée à un emplacement physique à divers moment:
  - compilation (rarement),
  - chargement (allocation statique),
  - exécution (allocation dynamique);

- Du point de vue implémentation, la déclaration d'une variable est utilisée pour déterminer la quantité d'espace mémoire exigée par le programme ;

Exemple : PASCAl*: var name : type;*
C: *Type name*;

- Une variable et une valeur sont liées par un assignement:

Exemple : V:= E:
« assigner le nom V a la valeur de l'expression E jusqu'à ce que le nom V est réassigne à une autre valeur ».

PASCAL: V:=E
C: V= E;
Scheme (setq V E)

- L'assignement n'est pas une définition de constante:

Exemple

```
X:=3;
X:=X+1;
```

Commandes non structurées

- Contient la commande GOTO: la plus puissante, et la plus critiquée.

- Difficulté de lecture et de compréhension du programme : structure linéaire, non hiérarchique

command ::=

identifier := expression (affectation)

| command; command (composition séquentielle)

| label : command (possibilité d'identifier une commande)

| GOTO label (GOTO InConditionnel)

| IF boo_exp THEN GOTO label (GOTO Conditionnel)

Programmation structurée

- Inventée pour décrire un modèle de programmation impliquant :
  - les structures hiérarchiques de programme où chaque commande a un point d'entrée et un point de sortie.

- fournit les structures de contrôle qui facilitent le *raisonnement* sur des programmes impératifs.

- Exemple un ensemble minimal de commandes structurées.

command::= SKIP

| identifier:= expression

| IF guarded_command [[]guarded_command]+ FI

| DO guarded_command [[]guarded_command]+ OD

| command ;command

guarded_command ::=guard --> command
guard ::= boolean expression

où :

Skip: le type le plus simple des commandes sans effet;

IF guard --> command FI = if guard then command

DO guard --> command OD = while guard do command

Structures de contrôle

Trois structures de contrôle : composition, sélection et l'itération

Composition :

- La séquence la plus utilisée est la composition séquentielle de deux ou plusieurs commandes (notée S0; S1).
- Disponible dans chaque L. P impératif;

Sélection :

- permet la spécification d'une séquence de commandes par cas;
- la sélection d'une séquence particulière est basée sur la valeur d'une expression.
- "If" et "case" sont les commandes les plus utilisées, et les plus représentatives.

```
--Ada
if condition then
commands
{ elsif condition then
commands }
[ else
commands]
endif
```

```
case expression is
            when choice1   => commands
            when choice2 => commands
                  .....
              [when others => commands]
end case;
```

Structures de contrôle (suite)

Itération:
une commande itérative a un corps qui est exécuté de façon répétitive et une expression qui détermine quand l'exécution s'arrête. Les plus utilisées:

while-do while condition do
body

repeat-until repeat
body
until condition

for-do for index := lowerBound, upperBound, step do
body

Expression Séquentielle : évaluation court-circuit

Exemple :

```
I :=0 ;
While (i < length) and (list[i] <> valuer)) do i :=i+1 ;
```

- terminaison de la boucle : index dépasse la borne ou la valeur est trouvée ;

- évaluation : si le premier paramètre est faux, le second n'est pas nécessaire à évaluer, puisque l'expression est fausse ;

- ML/ADA : endalso/andthen et orelse :

E andalso F=	Faux si E est faux
	Vrai si E et F sont vrai
E orelse F =	Vrai si E est vraie
	Faux si e et F sont faux

Sous Programme, procédures et fonctions

- Une procédure est une abstraction d'une séquence de commandes.
- L'appel de procédure est une référence a l'abstraction.
  syntaxe

En-tête et appel :
Procedure definition: name(parameter list) { body }
Procedure invocation: name(argument list)

Parameter list: t0 $name_1$, ..., tn-1 $name_{n-1}$
argument list: $expression_1$, ..., $expression_{n-1}$

Parameter	**Argument**
Pascal	
entrée: name : type	expression
entrée/sortie: var name : type	name
Ada	
entrée name : in type	expression
sortie name : out type	name
entré/sortie name : in out type	name
C	
entrée: type name	expression
entrée/sortie: type *name	&name

Difficultés de raisonner en L.P impératifs

- Certains axiomes sont valables seulement si le langage ne comporte pas de synonomie et d'effets de bord;

- Travaux de la théorie des Langages tentent d'expliquer les L. P en termes de constructions bien définies;

- cette complexité de raisonner a été une motivation forte pour fournir des solutions comme les paradigmes fonctionnels et logiques.

Difficultés de raisonner en impératifs(suite)

- Le langage machine inclut les instructions qui permettent à n'importe quelle instruction d'être choisie comme prochaine instruction.

➔Fournir des constructions pour donner à des langages de programmation à un niveau élevé une partie de cette flexibilité : sauts, échapement et exceptions.

Echapement (Escape) :
est une commande qui termine l'exécution d'une construction textuellement englobante .

Return Exp,
est utilisée en C pour quitter un appel de fonction et pour renvoyer la valeur calculée par la fonction.

Exit en ADA/ *Break* en C,
permet de transférer le contrôle du programme de l'endroit ou la commande exit se trouve vers la première commande après la boucle la plus interne qui suit le exit

Continue en C,
transfère la commande au début de la boucle englobante.

Difficultés de raisonner en impératifs(suite)

Exception
c'est un événement "anormal" qui survient au cours de l'exécution

Effet de bord
utilisé pour permettre la communication entre unités de programme:
Exemple:

```
integer f(x:integer)
        {
         y := y+1;
        f := y + x
        }
```

- Si a un point du programme, on sait que y = z = 0, alors f(z) = 1, i.e f(z) retourne 1
Mais 1 + f(z) = f(z) + f(z) est faux

Synonomie (aliasing)

- Deux noms sont des alias s'ils dénotent (partagent) le même objet de données pendant un lancement d'unité.

- La synonomie est un autre dispositif des L. P impératifs qui rend des programmes plus durs à comprendre.

Synonomie (suite)

Exemple 1:

```
procedure confuse (var m, n : Integer );
        begin
           n := 1; n := m + n
        end;
    ...
    i := 5;
    confuse(i,i)
```

m et n sont liés à la même variable i, initialisé à 5;
après l'appel la valeur de i est 2 et non pas 6.

Exemple 2: les pointeurs sont intrinsèquement des générateurs de synonomie.

```
var p, q : ^T;
...
new(p);
q := p ;
```

exemple 3 : Le pointeur p est laissé pointé à une valeur non existante :

```
type pointer = ^Integer
                    var p : Pointer;

                    procedure Dangling(p);
                    var q : Pointer;
                    begin;
                       new(q);  q^ := 23;  p := q;  dispose(q)
                    end;
                    begin
                       new(p); Dangling(p)
                    end;
```

Conclusion: Le problème des alias surgit dès qu'un langage supporte des variables et l'affectation.