Objectif

- Mettre en œuvre les concepts de base des langages de programmation informatique

19 PROGRAMMATION PROCÉDURALE

1 Règles et démarches pour écrire un programme

Parmi les solutions possibles, la programmation procédurale est un paradigme de programmation basé sur le concept d'**appels de procédures**.

Une procédure, ou une **fonction**, contient simplement une série d'étapes à réaliser. Toute procédure peut être appelée à n'importe quelle étape du fil d'exécution du programme, donc aussi à partir d'une étape du fil d'exécution d'une autre procédure.

En pratique, la fonction se distingue de la procédure parce qu'elle **retourne un résultat** ; comme en mathématique lorsqu'on écrit *y = sin(x)*, l'expression *sin()* est une fonction qui reçoit une valeur réelle *x* et qui retourne le sinus de cette valeur, rangé ici dans *y*.

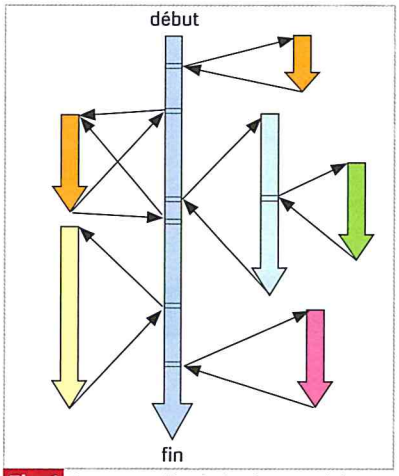


Fig. 1 Concept d'appels de procédures

2 Organiser les étapes à réaliser

Les constructions de base du « *comment faire* » de la programmation s'appuient sur celles du langage parlé.

La série d'étapes à réaliser contenue par une procédure est un **enchaînement** de portions de programme propres ; chaque sortie d'une portion peut être reliée à l'entrée d'une autre portion…

EXEMPLE

Pour faire une pleine cafetière de café.
Je commence par mettre un filtre.
Puis je mets du café moulu dans le filtre.
Puis je remplis le réservoir d'eau.
Et je termine par mettre la cafetière en marche

e **portion de** ogramme est dite pre si elle possède unique point ntrée et un unique nt de sortie.

Une procédure n'est cependant pas toujours linéaire, son cheminement peut être agrémenté de **prises de décision**. Une prise de décision est matérialisée par une **condition** dont l'évaluation ne peut être que vraie ou fausse (expression booléenne).
La mise en place de conditions entraîne la construction de **structures fondamentales** que l'on appelle **alternatives** ou **itérations** et qui sont aussi des portions de programme propres.

EXEMPLE

« *Si la cafetière est débranchée alors je la raccorde au secteur* »
→ alternative → condition : « *la cafetière est débranchée* » Oui/Non ?

« *Tant que le réservoir n'est pas plein, j'ajoute de l'eau* »
→ itération → condition : « *le réservoir est plein* » Oui/Non ?

« *J'attends jusqu'à ce que le café soit entièrement écoulé* »
→ itération → condition : « *le café est entièrement écoulé* » Oui/Non ?

erme **alternative** lique un traitement ditionnel (choix e deux possibilités), erme **itération** luit une action épétition…

3 Prendre en compte l'aspect informationnel

Le « *quoi faire* » est matérialisé par les données manipulées par les programmes. Même si la définition de ces données ne représente par ici l'axe principal d'analyse, elle doit malgré tout être traitée en phase de conception détaillée.

Définir une donnée consiste à lui attribuer un **nom** et un **type** qui caractérisent son codage en mémoire.

EXEMPLE

Données liées à « faire une pleine cafetière de café » :

- → quantité de café moulu (en grammes) = `poidsCaféMoulu`, valeur entière ;
- → contenance du réservoir d'eau (en litres) = `volumeEau`, valeur réelle ;
- → durée d'écoulement du café (en minutes) = `DuréeEcoulement`, valeur entière.

4 Outils de représentation

Pour analyser et décrire ce que doit faire un programme, il faut disposer d'outils décrivant les actions à effectuer et les chemins à parcourir entre ces actions. Les structures de base sont les **outils de description** de ces chemins, et pour les symboliser, graphiquement ou par du texte, il faut des **outils de représentation**.

Ces outils doivent rester des moyens d'analyse et ne pas préjuger du langage de programmation réel qui sera employé en phase de conception…

A L'organigramme

C'est un mode de représentation graphique constitué de symboles tels que des rectangles, losanges ou flèches. Bien qu'il existe un grand nombre de symboles de représentation, les **cinq symboles** présentés ci-contre suffisent dans la pratique. L'organigramme a pour avantage d'être très visuel, mais a pour inconvénient de négliger l'aspect définition des données.

Tous les symboles ont leur **point d'entrée situé en haut**. Le sens de lecture est par défaut du haut vers le bas, sauf indication contraire par retour fléché.

L'étape condition n'est pas une portion de programme propre (elle possède deux sorties), elle n'est donc jamais utilisée seule.

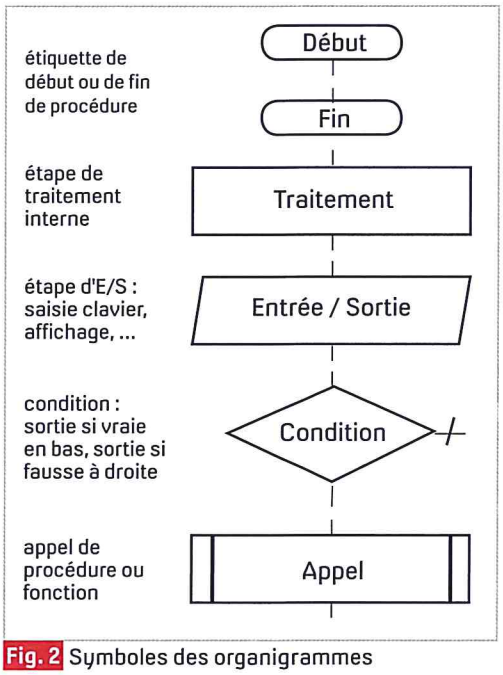


Fig. 2 Symboles des organigrammes

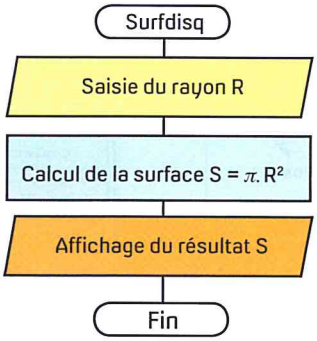
B Le pseudo-langage

La forme des organigrammes et la syntaxe du pseudo-langage qui sont présentées ici sont empruntées à la suite logicielle libre MoniaOrg/MoniaPL.

Ce mode de **représentation textuelle** est parfaitement adapté aux langages évolués car il utilise comme eux un **jeu de mots-clés** pour symboliser les structures fondamentales. Il permet d'autre part de définir les données utilisées, ce qui représente un gros avantage par rapport aux organigrammes.

Le pseudo-langage est facile à mettre en œuvre, mais requiert néanmoins le respect d'une syntaxe rigoureuse d'écriture (inspirée de la syntaxe du langage PASCAL) ; le sens de lecture du pseudo-langage est naturel, de haut en bas et de gauche à droite.

EXEMPLE



Organigramme de 1er niveau

Représentation de l'analyse d'un programme permettant de « Calculer la surface d'un disque connaissant son rayon… »

```
Programme surfdisq ;

Variables          rayon est un réel ;
                   surface est un réel ;

Début
    Afficher("Rayon du disque ? " ) ;
    Saisir( rayon ) ;
    surface <- Pi * Carré( rayon ) ;
    Afficher("Surface = ", surface ) ;
Fin
```

séparateur d'instructions

L'étape de calcul utilise des **opérateurs** : le produit (*) de π et du carré de **`rayon`** est stocké (<-) à l'emplacement mémoire spécifié par la variable **`surface`**…

Le pseudo-langage peut faire appel à des procédures et fonctions considérées comme standards : affichage sur écran, saisie de ce qui est frappé au clavier, fonctions mathématiques, manipulations de fichiers… Dans l'exemple précédent, quatre ressources standards sont utilisées :

`Afficher("Rayon du disque ? " )`	# procédure qui affiche le texte spécifié entre guillemets
`Afficher("Surface = ", surface )`	# affiche le texte entre guillemets puis la valeur de la variable surface
`Saisir( rayon )`	# procédure qui attend une saisie clavier terminée par Entrée et stocke la saisie dans la variable indiquée (ici rayon)
`Pi`	# fonction retournant la valeur de la constante π
`Carré( rayon )`	# fonction retournant la valeur indiquée (ici rayon) élevée au carré

Dans les organigrammes, le symbole « Appel » ne concerne pas les appels aux ressources considérées comme standards ; ce symbole est réservé à la mise en place du découpage fonctionnel décidé par le programmeur. C'est l'illustration du concept de programmation procédurale.

INFORMATIONS

EXEMPLE

Logiciel de supervision 2D des déplacements d'un robot

Le logiciel de supervision utilise une boîte à outils de dessin afin de modéliser des robots en vue de dessus. Le robot WifiBot peut ainsi être simplement représenté par des formes rectangulaires…

La boîte à outils est composée notamment des trois procédures suivantes :

Procédure **`Position( x : entier ; y : entier ) ;`**
positionne le « crayon » de dessin à la position `x,y` sur le repère 2D ; les coordonnées sont transmises lors de l'appel de la procédure, ce sont des paramètres.

Liste des paramètres

Fonction **`PosX : entier ;`**

Fonction **`PosY : entier ;`**

retournent la position x ou y courante du crayon.

`Procédure Couleur( c : COULEUR ) ;`
permet de fixer la couleur `c` pour le tracé et le remplissage.

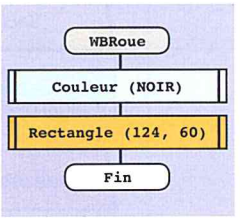
`Procédure Rectangle( l : entier ; h : entier ) ;`
dessine un rectangle de dimensions `l x h` avec son centre à la position du crayon ; le rectangle est rempli avec la couleur courante de dessin.

EXEMPLE

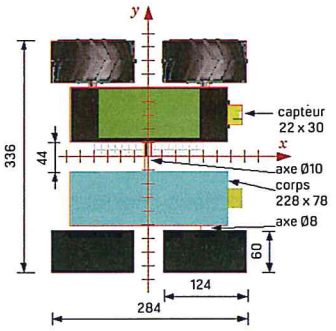
```
Procédure WBCorp ;
Début
    Couleur (BLEU ) ;
    Rectangle ( 228, 78 ) ;
    Position ( PosX + 125, PosY ) ;
    Couleur ( JAUNE ) ;
    Rectangle( 22, 30 ) ;
    Position ( PosX - 125, PosY ) ;
Fin
```

valeurs des paramètres

Modélisation d'un ensemble cirps/capteur



Modélisation d'une roue



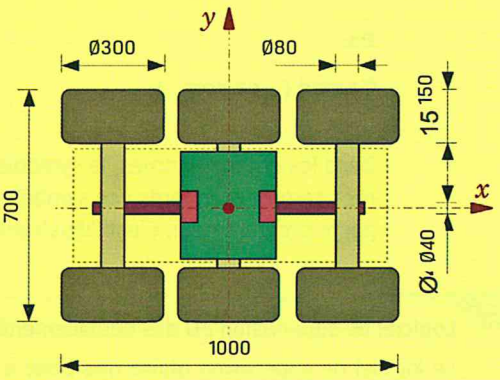

```
Procédure WifiBot ;
Début
    Position ( 0, 0 ) ;          ≠ axe central
    Couleur (GRIS ) ;
    Rectangle ( 10, 44 ) ;
    Position ( 0, -61 ) ;        ≠ corps/capteur droit
    WBCorps ;
    Position ( 80, -138 ) ;      ≠ roue avant droite
    WBRoue ;
    Position ( -PosX, PosY ) ;   ≠ roue arrière droite
    WBRoue ;
    … …
Fin
```

Début de construction du modèle complet

ACTIVITÉ 1 Justifiez les appels à **`Position()`** présents dans la procédure WBCorps.

ACTIVITÉ 2 Le logiciel de supervision 2D est maintenant utilisé pour suivre les évolutions du ROBOVOLC. Ce robot possède trois essieux articulés, ses principales dimensions sont données ci-contre. Proposez une procédure RoboVolc de modélisation de ce robot ; cette procédure doit notamment faire appel à :

- une procédure **RVRoue** de dessin d'une roue (à symboliser par un rectangle) ;
- une procédure **RVEssieu** de dessin d'un essieu (composé d'un logement de moteurs Ø80 et de deux roues).



Dimensions du ROBOVOLC en mm.

C Commentaires

Les commentaires n'ont aucune incidence sur le programme final, ils sont ignorés par les outils utilisés pour fabriquer un programme exécutable à partir du programme source.

Dans tout langage de programmation, les commentaires sont des informations textuelles supplémentaires ajoutées par le développeur. Les commentaires sont destinés à documenter une ligne d'instructions particulière (algorithme complexe), à expliquer le rôle d'une procédure ou fonction, ou encore à justifier certains choix technologiques…

En pseudo-langage, une ligne ou fin de ligne de commentaire est introduite par le caractère #. Un commentaire sur plusieurs lignes ou placé au milieu d'une instruction est borné par des accolades { et }.

Sur les organigrammes, les commentaires sont attachés par une ligne pointillée comme illustré en figure 3.

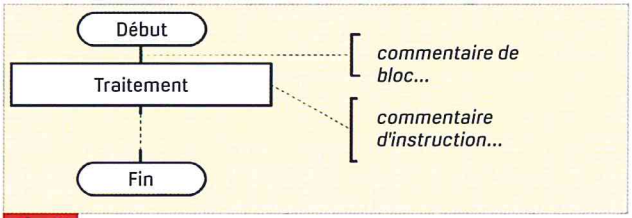


Fig. 3 Commentaires des organigrammes

D Formalisation des structures fondamentales

Les structures fondamentales sont matérialisées par des portions de programme propres construites à partir d'une étape condition. La condition peut être composée de plusieurs conditions combinées par les opérateurs logiques **ET**, **OU**, et **NON**.

EXEMPLE

Conditions composées :

```
( a < 0 ) ET ( b > 3 )
NON ( a = b )
( a > 0 ) OU ( b = 1 )
```

Alternative complète
(portion de programme propre)

```
SI ( Condition) ALORS
    Traitement_1
SINON
    Traitement_2
FINSI
```

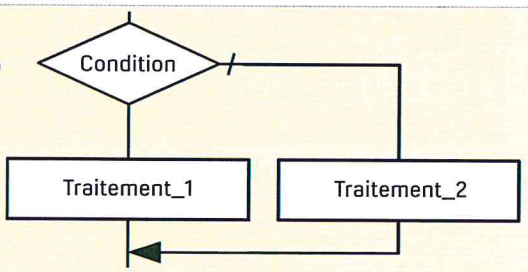


Fig. 4 Alternative complète

Alternative partielle
(portion de programme propre)

```
SI ( Condition) ALORS
    Traitement
FINSI
```

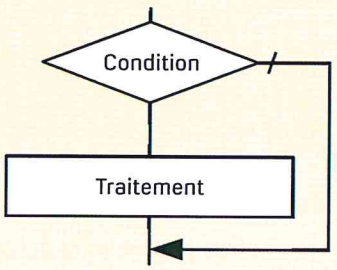


Fig. 5 Alternative partielle

Itération à condition initiale
(portion de programme propre)

```
TANT QUE ( Condition) FAIRE
    Traitement
FINTQ
```

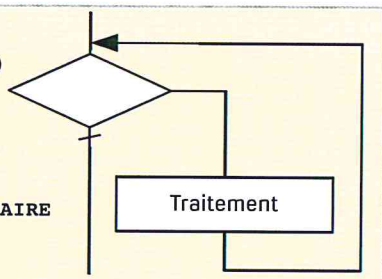


Fig. 6 Itération à condition initiale

Itération à condition finale
(portion de programme propre)

```
REPETER
    Traitement
JUSQU'A CE QUE ( Condition) ;
```

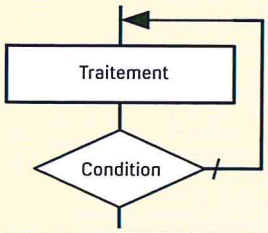


Fig. 7 Itération à condition finale

Dans une itération à condition finale, le traitement est toujours exécuté au moins une fois.

Dans une itération à condition initiale, le traitement peut ne jamais être exécuté si la condition est fausse dès le départ.

tention !
ur les deux types
tération, le traite-
ent (ou un autre
ocessus concurrent)
it agir sur la condition
us peine de rester
oqué dans une
ucle sans fin...

INFORMATIONS

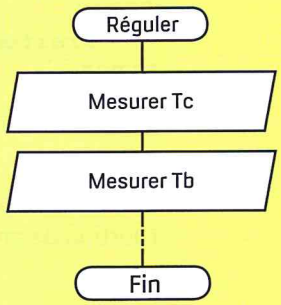
ACTIVITÉ 3 Représenter les organigrammes des algorithmes suivants en les simplifiant au moyen de conditions composées :

```
Programme Algo1 ;
Début
    SI ( c1 ) ALORS
        TraitementA ;
    SINON
        SI ( c2 ) ALORS
            TraitementA ;
        SINON
            TraitementB ;
        FINSI
    FINSI
Fin
```

```
Programme Algo2 ;
Début
    SI ( c1 ) ALORS
        SI ( c2 ) ALORS
            TraitementA ;
        SINON
            TraitementB ;
        FINSI
    SINON
        TraitementB ;
    FINSI
Fin
```

ACTIVITÉ 4 La régulation d'un chauffe-eau solaire à circulation forcée doit respecter les contraintes suivantes : le circulateur est mis en marche si *Tc* est supérieur à *Tb* + *Se*, il est arrêté lorsque *Tc* devient inférieur ou égal à *Tb* + *Sd*. Le régime permanent est donc établi quand *Tc* est compris entre *Tb* + *Sd* et *Tb* + *Se*, ce qui correspond à la condition nécessaire pour que le capteur solaire puisse transmettre de l'énergie au ballon d'eau chaude (avec *Tc* la température du capteur, *Tb* celle du ballon, *Se* le seuil d'enclenchement, et *Sd* celui de déclenchement).

Complétez l'organigramme de la procédure afin de traduire ces contraintes de régulation (seuils sont supposés fixés et connus).



E Cas particulier d'itération

En pseudo-langage, l'écriture **POUR.. FAIRE** sous-entend une évolution de type incrémentation (ajout de 1 à la variable de boucle). Pour mettre en place une boucle avec une évolution différente (ajout supérieur à 1, décrémentation...), il convient d'utiliser la forme traditionnelle de l'itération.

L'itération à condition initiale permet de construire une structure dérivée nommée **boucle**.
Cette construction est particulièrement intéressante lorsque le nombre de fois à exécuter le traitement est connu à l'avance.
La variable de boucle (nommée `i` surla figure 8) est en général de type entier.

Boucle
(portion de programme propre)

VI : valeur initiale
VF : valeur finale

```
i <- VI
TANT QUE ( i < = VF ) FAIRE
    traitement
    i <- i + 1
FINTQ

POUR i <- VI jusqu'à VF FAIRE
   traitement
FINP
```

initialisation
condition de poursuite
évolution
écriture simplifiée

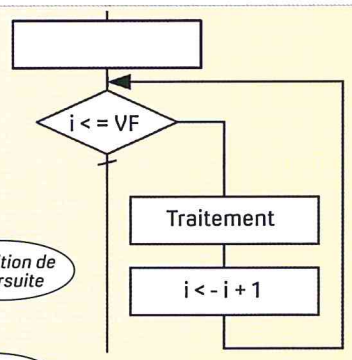


Fig. 8 Boucle

EXEMPLE

Les variables `i`, `j` et `v` sont des entiers...

Bloc d'instructions d'affichage de puissances de 2

```
v <- 1 ;
pour i <- 0 juqu'à 10 faire
    Afficher (v, "  ") ;
    v <- v * 2 ;
finp
```

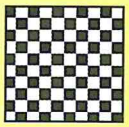
Bloc d'instructions affichant la table de multiplications

```
pour i <- 1 juqu'à 10 faire
  pour j <- 1 juqu'à 10 faire
    v <- i * j ;
    Afficher (v:4) ;
  finp
  Afficher ( CRLF ) ;
finp
```

affichage justifié à droite sur 4 caractères

ACTIVITÉ 5 Le logiciel de supervision 2D des pages précédentes utilise des repères pour marquer les points de trajectoire du robot.
Les points de passage sont matérialisés par des petites cibles de taille 20 x 20 mm.

Les points à atteindre (fin de trajectoire) sont représentés par des damiers de taille 100 x 100 mm.



En utilisant les ressources de la boîte à outils **`Position(),PosX,PosY,Couleur(), Rectangle()`**, proposer une représentation pseudo-langage des procédures de dessin de ces 2 éléments graphiques.

F Cas fortuits de solutions « non structurées »

Dans certains cas, la représentation graphique « naturelle » d'un scénario conduit à une structure qu'il est malheureusement impossible de mettre en œuvre en respectant les structures fondamentales autorisées…

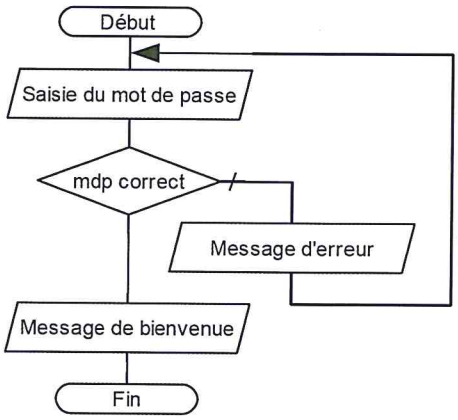
Il est cependant toujours possible, dans un deuxième temps, de faire apparaître des structures autorisées en répétant certaines étapes de traitement ou de test.

EXEMPLE

L'accès au logiciel de pilotage distant du ROBOVOLC passe par un système d'authentification particulièrement têtu : « Le système demande à l'utilisateur de saisir son mot de passe (mdp) ; en cas d'erreur, il en informe l'utilisateur et réitère la demande jusqu'à obtenir enfin le bon sésame ! »

L'organigramme « naturel » ci-contre n'est pas transposable en pseudo-langage.

On obtient en effet une sorte de mélange des deux types d'itérations…



1re solution : avec une itération à condition initiale :

```
Début
  ≠ Saisie du mot de passe
  TANT QUE ( NON mdp correct ) FAIRE
    ≠ Message d'erreur
    ≠ Saisie du mot de passe
  FINTQ
  ≠ Message de bienvenue
Fin
```

2e solution : avec une itération à condition finale :

```
Début
  REPETER
    ≠ Saisie du mot de passe
    SI ( NON mdp correct ) ALORS
      ≠ Message d'erreur
  FINSI
 JUSQU'À CE QUE ( mdp correct) ;
  ≠ Message de bienvenue
Fin
```

ACTIVITÉ 6 Le système d'authentification précédent a été modifié : « Le système demande à l'utilisateur de saisir son mot de passe ; en cas de succès, le système affiche un message de bienvenue ; mais en cas d'erreur, il informe l'utilisateur et réitère la demande un nombre limité de fois (3 essais au maximum). Si tous les essais sont infructueux, le système affiche un message de rejet. » Il y a donc une nouvelle étape « Message de rejet » et une nouvelle condition du style « Nombre max. d'essais atteint » ; proposer un organigramme traduisant naturellement le déroulement ci-dessus, puis, le cas échéant, au moins une solution pseudo-langage structurée de ce système.

5 La gestion des données

A Types de base

Le type caractérise la **taille mémoire d'une donnée** (son encombrement) et le jeu de valeurs qu'elle peut prendre.

En pseudo-langage, il n'y a pas de distinction majuscules/minuscules, les accents sont acceptés mais ignorés, la mise au pluriel est autorisée pour les types de base...

Booléen :	type logique de valeur **VRAI** ou **FAUX**
Octet :	entier non signé sur 8 bits (0 .. 255)
Caractère :	entier non signé sur 8 bits, codage des symboles ASCII
Entier :	entier signé sur 16 bits (-32 768 .. +32 767)
Entier court :	entier signé sur 8 bits (-128 .. +127)
Entier long :	entier signé sur 32 bits (-2 147 483 648 .. +2 147 483 647)
Mot :	entier non signé sur 16 bits (0 .. 65 535)
Mot long :	entier non signé sur 32 bits (0 .. 4 294 967 295)
Réel :	valeur réelle simple précision (codage IEEE sur 32 bits)
Réel double :	valeur réelle double précision (codage IEEE sur 64 bits)

Fig. 9 Types numériques de base pseudolangage

B Identificateurs

Le terme identificateur couvre l'ensemble des **éléments définis par le programmeur** : constante, variable, nom de programme, nom de procédure ou de fonction, paramètre formel...

Règles de construction d'un identificateur en pseudo-langage :

- l'identificateur ne doit pas être un mot réservé du langage ;
- les seuls caractères autorisés sont les lettres, les chiffres et le caractère de soulignement ;
- l'identificateur ne doit pas commencer par un chiffre ;
- la longueur d'un identificateur peut être quelconque.

ACTIVITÉ 7 **Trouvez les identificateurs invalides :**

```
Seconde          2nd          _2nd
Début  Fini                   iNtRo
mot_2            résultat     pi/2
ProduitCalculé   sti-2d       _mon_indice_
longueur         LONG         gagne:-)
```

C Constantes

Les constantes permettent d'améliorer la lisibilité des programmes en favorisant l'écriture de **mnémoniques** adaptés en lieu et place de valeurs. Les constantes sont souvent écrites en lettres majuscules pour les distinguer des variables.

Un identificateur de constante est à considérer comme une image de remplacement de sa valeur, ce n'est donc en aucun cas utilisable en écriture car une déclaration de constante n'entraîne aucune réservation de mémoire...

valeur décimale (base 10) : *[<signe>]<0..9>*
valeur hexadécimale (base 16) : *$<0..9|A..F>*
valeur binaire (base 2) : *%<0|1>*
valeur booléenne : *FAUX|VRAI*
valeur réelle : *[<signe>]<0..9>[.<0..9>][E|e[<signe>]<0..9>]*
caractère : *'<caractère_imprimable>'*
chaîne de caractères : *"<caractère(s)_imprimable(s)>"*

Fig. 10 Règles d'écriture des valeurs constantes

EXEMPLE

En pseudo-langage, le mot réservé **CONSTANTE(S)** marque le début d'une zone de définitions de constantes...

```
CONSTANTES    VALMIN  = -200 ;
              TVA     = 19.6 ;
              ADR     = $03FA ;
              MASQUE  = %11100000 ;
              MESSAGE = "mot de passe ?" ;
              EPSILON = 1e-6 ;
```

D Variables et constantes typées

L'identificateur d'une variable caractérise un emplacement mémoire permettant de stocker une donnée du type indiqué. La déclaration de variable correspond à la **réservation** de cet emplacement mémoire. Syntaxiquement, l'identificateur peut être séparé du type par *est un(e), sont des, ou le caractère :*
Les constantes typées sont assimilables à des **variables initialisées** (soit la déclaration de la variable suivie de l'affectation d'une valeur). Elles se comportent ensuite comme des variables, leur valeur est donc modifiable.

EXEMPLE

```
VARIABLES    indice est un entier ;
             x , y sont des réels doubles ;

CONSTANTES   compteur est un entier long = 0 ;
             erreur est un booléen = FAUX ;
             séparateur est un caractère = ';' ;
```

n pseudo-langage, mot réservé ARIABLE(S) marque début d'une zone de éclarations e variables...

ACTIVITÉ 8 Dans le cadre du développement durable, on souhaite développer un logiciel utilitaire concernant l'amendement des sols. Proposer les définitions de constantes nécessaires en vue de la résolution du problème suivant, sachant que toutes les manipulations doivent être faites dans les unités S.I. (ne sont donc autorisés ici que : m^2, m^3, kg, €, sauf pour la seule entrée du programme : N en hectares) :
« La suie, issue du ramonage des cheminées à bois, est un engrais azoté. Elle peut être utilisée en agriculture à raison de 1/6 de m^3 par are, mélangée avec 2 à 3 fois le même volume de terre. Sachant que la suie coûte 2,50 € l'hectolitre, que le mètre cube de suie pèse 1 205 kg et que les frais de transport s'élèvent à 3,90 € la tonne, plus un forfait de 15 € à chaque rotation du camion (équipé d'une benne de 12 m^3), quelle est la dépense à faire pour amender N hectares de champs ? »
Pour mémoire, un hectare = 100 ares = 100 x 100 m.

E Opérateurs

Tous les langages de programmation savent reconnaître et interpréter dans les expressions les symboles utilisés communément en mathématiques pour effectuer des calculs ou des comparaisons.

Une variable représentant un emplacement mémoire, les opérateurs d'accès permettent de référencer cet emplacement.

L'**opérateur d'affectation** possède deux opérandes : un à droite et un à gauche. Son usage signifie que l'on stocke le résultat de l'expression de droite à l'emplacement mémoire spécifié par la variable de gauche.

L'opérateur **DIV** est la division euclidienne qui ne considère que la partie entière du résultat ; l'opérateur **MOD** donne le reste de cette division entière.

entier N est pair si pression booléenne MOD 2 = 0) vraie...

opérateur arithmétiques	
+	addition
-	soustration/négation
*	multiplication
/	division réelle
DIV	division entière
MOD	modulo
opérateurs relationnels	
=	équivalent à
<>	différent de
<	inférieur à
>	supérieur à
<=	inférieur ou égal à
>=	supérieur ou égal à
opérateurs logiques	
NON	négation logique
ET	ET logique
OU	OU logique
opérateurs d'acès	
<-	affection
[ ]	accès à un élement de tableau

Fig. 11 Principaux opérateurs

EXEMPLE

Un éleveur devant ranger **N** œufs en boîtes de 12 obtiendra **(N DIV 12)** boîtes complètes et une boîte incomplète de **(N MOD 12)** œufs.

Un nombre **N** de secondes est égal à **(N DIV 60)** minutes et **(N MOD 60)** secondes...

INFORMATIONS

ACTIVITÉ 9 À partir du travail effectué lors de l'activité précédente, ajouter les déclarations de variables nécessaires et remplacer les commentaires du programme suivant par les instructions pseudo-langage adéquates (à base d'opérateurs arithmétiques et d'affectations), de manière à obtenir un produit répondant au besoin initial.

On suppose `N réel` initialisé par l'utilisateur avec la surface à amender (en hectares)

```
Début
# calcul du volume de suie pour les N hectares (en m³)
# calcul du poids de suie correspondant (en kg)
# calcul du coût brut de la suie (en €)
# calcul des frais de transport hors forfaits rotation (en €)
# calcul du nombre de rotation du camion
# calcul de l'investissement global (en €)
Fin
```

F Expressions et instructions

Une **expression** est une construction assemblée au moyen d'opérateurs, et dont l'évaluation de proche en proche vaut une valeur du type de ce qui est calculé.

Le mécanisme de **transtypage** permet de forcer l'évaluation d'une expression dans un type différent de celui d'origine.

EXEMPLE

`x est un réel = 1.7 ;` l'expression `x + i` est de type réel, elle vaut `3.7`.
`i est un entier = 2 ;` l'expression transtypée `entier( x + i )` est de type entier, elle vaut `3`.

Les termes d'une expression peuvent être des variables, des constantes ou des appels de fonctions.

EXEMPLE

```
1.5 * sin( x ) / ( x – 1 )
```

La mise en place de parenthèses permet de contrôler les priorités entre opérateurs.

Une condition est une expression simple ou composée évaluable de manière booléenne. Une valeur nulle est considérée comme valant `FAUX`, toutes les autres valeurs valent `VRAI`.

EXEMPLE

`x est un réel = 1.7 ;` l'expression `booléen( x + i )` est de type booléen, elle vaut `VRAI`.
`i est un entier = 0 ;` l'expression `b OU booléen( i )` est de type booléen, elle vaut `FAUX`.
`b est un booléen = FAUX ;` l'expression `b ET booléen( x )` est de type booléen, elle vaut `FAUX`.

Les étapes de traitement des structures fondamentales sont des blocs d'instructions.

On appelle **instruction** toute construction répondant à une des formes suivantes :

- affectation constituée d'une valeur de gauche, de l'opérateur `<-` et d'une expression ;
- appel de procédure ;
- structure fondamentale (portion de programme propre).

ACTIVITÉ 10

- Repérer dans le programme ci-dessous les expressions et leur type.
- Encadrer les instructions dans le programme ci-dessous.
- Représenter le programme sous forme d'organigramme et encadrer les portions de programme propres.
- Expliquer de manière textuelle ce que fait ce programme.

```
Programme mystère ;

Variables  n, somme sont des entiers ;
           produit est un réel ;

Début
    somme <- 0 ;
    produit <- 1 ;
    répéter
        Afficher ( "Valeur entière ? " )
        Saisir ( n ) ;
        si ( n <> 0 ) alors
            somme <- somme + n ;
            produit <- produit * n ;
            Afficher (" somme = ", somme, TAB ) ;
            Afficher ("produit = ", produit, CRLF ) ;
        finsi
    jusqu'à ce que ( n = 0 ) ;
Fin
```

'AB et CRLF sont les constantes le type caractère ui provoquent espectivement ur le terminal de ortie une tabulation orizontale et un assage à la ligne uivante...

SIN

6 Les types dérivés

Les types dérivés sont des types de données définis suivant les besoins de programmation afin de modéliser des éléments composés d'un sous-ensemble ou d'un assemblage de types de base (ou d'autres types dérivés préalablement définis...).

Pour améliorer la lisibilité des programmes, tout type défini (qu'il soit de base ou dérivé) peut être renommé par un **alias** en rapport avec le contexte de développement.

pseudo-langage, mot réservé PE(S) marque début d'une zone déclarations d'alias types...

A Énumération

Une **énumération** est un ensemble fini de valeurs spécifiées sous forme de mnémoniques (par convention en majuscules) ; ces mnémoniques constituent un ensemble ordonné d'entiers positifs dont le premier vaut 0.

INFORMATIONS

EXEMPLE

```
# alias associés à des énumérations

TYPES  JOUR   = {LUN, MAR, MER, JEU, VEN, SAM, DIM } ;
       PODIUM = { BRONZE, ARGENT, OR } ;
       CASE   = { LIBRE, PION_BLANC, PION_NOIR, DAME_BLANCHE, DAME_NOIRE };

       CASES  = CASE ;                    # pour l'orthographe...

# exemples de variables initialisées...

CONSTANTES  jo est un JOUR = LUN ;        # jo est un entier qui vaut 0
            po est un PODIUM = ARGENT ;   # po est un entier qui vaut 1
```

B Tableau

Le nom d'un tableau représente l'adresse de base de la zone mémoire occupée par le tableau, cet identificateur représente donc un pointeur.

Un **tableau** est une suite linéaire d'éléments de même type. La dimension (le nombre d'éléments) est indiquée par une valeur entière entre crochets. Le premier élément d'un tableau a pour indice 0. Les crochets `[ ]` sont en réalité un **opérateur d'accès**. Si t est une variable de type tableau, la construction `t[0]` fait référence au premier élément du tableau et `t[N]` représente l'élément situé à la position `N-1` dans le tableau.

Un tableau peut avoir plusieurs dimensions ; il n'y a pas de limite théorique quant au nombre de dimensions...

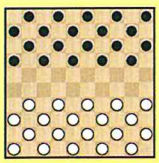
EXEMPLE

```
TYPES
DAMIER = tableau[10,10] de CASES ;      # tableau à 2 dim., 100 cases

CONSTANTES      ChiffreRomain est un tableau[7] de caractères = "MDCLXVI" ;
                ValeurRomain est un tableau[7] d'entiers
                = ( 1000,500,100,50,10,5,1 ) ;
VARIABLES       i est un entier ;
                jeu est un DAMIER ;

# ... ...
POUR i <- 0 JUSQU'A 6 FAIRE
   Afficher( Chiffre Romain[i], " vaut ", Valeur Romain[i], CRLF ) ;
FINP

jeu[0,0] = PION_BLANC ;
```

on peut aussi écrire la dimension sous forme d'un intervalle ***0..6***

ACTIVITÉ 11

En vous aidant des déclarations des exemples précédents (types **DAMIER** et **CASE**), proposer un programme de placement initial des pions sur un plateau de jeu de dames (la case de coordonnées 0,0 est en bas à gauche).



C Chaîne de caractères

En programmation, le texte est manipulé comme une suite linéaire de caractères. Un caractère spécial de valeur nulle, nommé **terminateur de chaîne**, est placé dans la suite pour indiquer la fin de la chaîne.

En pseudo-langage, on peut simplifier l'écriture `tableau[N] de caractères` en `chaine[N]`. Par convention, le type `chaine` sans dimension est équivalent à `chaine[255]`.

EXEMPLE

Quand un tableau de caractères est initialisé avec une constante de type chaîne (entre guillemets), le caractère terminateur de chaîne est automatiquement mis en place.

```
CONSTANTE   ch est une chaine[20] = "ABC" ;

VARIABLE    message est une chaine ;


# ... ...
message <- "Hello world !" ;        # affectation
Afficher( message, CRLF ) ;         # affiche "Hello world !"
message[6] <- caractère(0) ;        # transtypage sur valeur nulle
Afficher( message, ch, CRLF ) ;     # affiche "Hello ABC"
```

réserve une suite de 20 caractères et initialise les 4 premiers avec les valeurs 'A', 'B', 'C' et 0

ACTIVITÉ 12 À l'aide d'une table des codes ASCII, proposer une procédure de mise en majuscules d'une chaîne de caractères.

La procédure reçoit en paramètre l'adresse de la chaîne à convertir ; les caractères peuvent être parcourus par une itération dont la condition de fin est la détection du terminateur de chaîne.

Prototype de la procédure : `Procédure Majuscules( ch : pCaractère );`

dresse du début
n tableau de
actères est
pointeur de type
aractère...

SYNTHÈSE

La programmation procédurale est basée sur le concept d'appels de procédures et fonctions qui contiennent des suites d'étapes à réaliser.

Une fonction se distingue d'une procédure parce qu'elle retourne un résultat.

La programmation procédurale s'appuie sur un ensemble de structures fondamentales pour traduire le « comment faire » en phase d'analyse globale. L'aspect informationnel, c'est-à-dire le « quoi faire », est traité en phase de conception détaillée.

Les structures fondamentales sont des portions de programme propres : un seul point d'entrée et un unique point de sortie. Ces structures fondamentales sont :

- l'**enchaînement** ;
- l'**alternative** (partielle ou complète) ;
- l'**itération** à condition initiale ou à condition finale.

Les prises de décision intervenant dans les alternatives et les itérations sont des conditions évaluées de manière booléenne.

La structure de boucle est une écriture particulière d'itération utilisée lorsque le nombre de répétitions du traitement est connu à l'avance.

Avantages du paradigme « programmation procédurale » liés à la qualité logicielle :

- programmes modulaires et structurés → **lisibilité et maintenabilité** améliorées ;
- réutilisabilité des portions de code → **vérifiabilité** et optimisation augmentées ;
- interdiction d'usage des sauts inconditionnels *(GOTO)* → **intégrité** et robustesse du code renforcées.

Organigramme

L'organigramme est un outil de **représentation graphique** de la structure des programmes.

L'organigramme est indépendant de tout langage réel de programmation.

L'organigramme est surtout utilisé en phase d'analyse préliminaire, il n'est pas adapté à la gestion des données.

Pseudo-langage

Le pseudo-langage est un outil de **représentation textuelle** de la structure des programmes. Il ne bénéficie d'aucune norme dédiée et reste donc relativement souple quant à son écriture.

Le pseudo-langage est indépendant de tout langage réel de programmation, mais il respecte la sémantique des langages évolués tels que le Pascal, le C, ou le PHP…

Le pseudo-langage est adapté à la gestion des données. Il est utilisé en pratique en phase de conception détaillée pour traduire l'organigramme, qui fournit l'aspect fonctionnel, et pour ajouter l'aspect informationnel.

Une **variable** doit être typée, son **identificateur** représente un emplacement mémoire réservé dont la taille est fournie par le **type** (taille mesurée en octets).

Une **expression** est une construction composée de **constantes**, d'**appels de fonctions** et de **variables** assemblés au moyen d'**opérateurs**.

Une **instruction** peut prendre trois formes : soit une **affectation** constituée d'une valeur de gauche, d'un opérateur d'affectation et d'une expression, soit un **appel de procédure**, soit une **portion de programme propre**.

Le **tableau** est un type dérivé défini comme une suite linéaire d'éléments de même type.

La **chaîne de caractères** est un tableau de caractères ; un élément spécial nommé **terminateur de chaîne** (caractère nul) indique la fin de la chaîne.

INFORMATIONS