I.U.T. d'informatique
Semestre 1

2009-2010

Convention d'écriture des programmes en langage algorithmique

Christian PERCEBOIS

TABLE DES MATIERES

1. PRINCIPES GENERAUX 2
1.1. Lisibilité 2
1.2. Sobriété 2
1.3. Uniformité 2
2. IDENTIFICATEURS 2
2.1. Identificateur de type 3
2.2. Identificateur de variable, de programme, de procédure ou de fonction 3
2.3. Identificateur de constante 3
3. LIGNES BLANCHES 3
4. ESPACES 4
5. COUPURES 4
6. COMMENTAIRES 5
6.1. Rôle d'une constante ou d'une variable 5
6.2. En-tête du programme 5
6.3. En-tête d'un sous-programme 5
6.4. Trace d'un affinage 6
7. GLOSSAIRE 6
8. INSTRUCTIONS 6
8.1. Séquence 6
8.2. Sélection 7
8.3. Répétition 7
9. RECOMMANDATIONS 7
9.1. Taille des unités de programme 7
9.2. Constantes 7
9.3. Ordre des paramètres d'un sous-programme 8
9.4. Sélections imbriquées 8
9.5. Retour de fonction 8
9.6. Sorties de boucles 9
9.7. Conditions 9
9.8. Déclenchement d'exception 10
10. EXEMPLE 10

Convention d'écriture des programmes en langage algorithmique

Rédiger des programmes conformément à une convention d'écriture constitue un des critères de qualité du logiciel et s'impose à tout développement, notamment pour les raisons suivantes :

- une convention d'écriture améliore la lisibilité des programmes,
- un programme est rarement maintenu par son auteur,
- le coût de cette maintenance est souvent élevé,
- le texte d'un programme est parfois livré au client.

Dans ce qui suit, nous explicitons la convention d'écriture des programmes adoptée en algorithmique.

1. Principes généraux

1.1. Lisibilité

Il convient de tendre vers un équilibre entre lisibilité et efficacité du code. Une structure esthétique d'un point de vue algorithmique peut être un frein aux performances. Inversement, l'optimisation irraisonnée du code peut produire des textes peu structurés et peu lisibles. Pour autant, privilégiez toujours la lisibilité à la facilité d'écriture…

1.2. Sobriété

Les structures alambiquées constituent une entrave à la lisibilité et à la compréhension. A la paraphrase du code instruction par instruction, préférez des commentaires offrant une lecture additionnelle au code. Par contre, sobre ne veut pas dire laconique : soignez vos commentaires.

1.3. Uniformité

La programmation a aussi ses idiomes, à savoir des constructions utilisées par les programmeurs. Afin de faciliter la communication, il est important de respecter ces idiomes. Une convention d'écriture concourt à cet objectif en définissant un format uniforme d'écriture des programmes. Contribuez au partage de la culture commune des informaticiens parlant le même dialecte.

2. Identificateurs

En règle générale, les identificateurs doivent avoir des noms explicites. La seule exception autorisée à cette règle est d'utiliser des variables muettes *i*, *j*, *k*, ... pour les indices de tableaux et les répétitions dont on connaît à l'avance le nombre d'itérations.

2.1. Identificateur de type

Un identificateur de type s'écrit en minuscules et commence par une majuscule. Si l'identificateur est composé de plusieurs mots, chaque mot commence par une majuscule.

type Date : ... ;
type MatriceCreuse : ... ;

2.2. Identificateur de variable, de programme, de procédure ou de fonction

Un identificateur de variable, de programme, de procédure ou de fonction s'écrit en minuscules. Si l'identificateur est composé de plusieurs mots, chaque mot autre que le premier commence par une majuscule. Un identificateur de procédure est un verbe à l'infinitif (action). Un identificateur de fonction est un nom, un adjectif ou une forme verbale (expression).

moyenne <Réel> ;
dateDuJour <Date> ;
programme dateDuLendemain
procédure chercherMaximum ;
fonction max (**entrée** t <T1>) **retourne** <T2> ;
fonction estVide **retourne** <Booléen> ;

2.3. Identificateur de constante

Un identificateur de constante s'écrit en majuscules. Si l'identificateur est composé de plusieurs mots, chaque mot est séparé du suivant par le symbole souligné.

constante MAX … ;
constante LG_MAX … ;
constante NB_ELEVES … ;

3. Lignes blanches

Les lignes blanches permettent d'isoler des paragraphes du programme et contribuent à clarifier sa structuration.

On introduit une ligne blanche :
- après la clause d'importation (mot-clé **importer**),
- après la déclaration des constantes et des types (mots-clés **constante** et **type**),
- après l'en-tête du programme (mot-clé **programme**),
- après le glossaire du programme (mot-clé **glossaire**).

Une ligne blanche sépare aussi chacun des sous-programmes (mots-clés **procédure** et **fonction**).

Pour identifier aisément les différents paragraphes d'un programme, les mots-clés **importer**, **constante**, **type**, **programme**, **glossaire**, **début**, **fin** et **traite-exception** sont alignés. Le même alignement est appliqué aux mots-clés

procédure (respectivement **fonction**), **glossaire**, **début**, **fin** et **traite-exception** d'une procédure (respectivement d'une fonction).

4. Espaces

Les espaces isolent les unités syntaxiques d'une expression ou d'une instruction.

On introduit un espace :

- avant une parenthèse ouvrante, un point-virgule et une marque de type (délimitée par < et >),
- après une virgule, après tout mot-clé et en début de commentaire (introduit par --),
- avant et après le symbole d'affectation (<-), les opérateurs arithmétiques binaires et les opérateurs relationnels.

```
lire (nbValeurs, valeurs) ;
f <- factorielle (n) * k ;
fonction premier retourne <T> ;
type TabEntiers : tableau [1 à 10] de <Entier> ;
glossaire max <Entier> ; -- maximum de x et de y
```

5. Coupures

Lorsqu'une expression ne contient pas sur une seule ligne, appliquer les principes de mise en page suivants :

- couper après une virgule,
- couper avant un opérateur,
- introduire une indentation de 4 espaces par rapport à la ligne précédente s'il s'agit d'une sous-expression,
- aligner les sous-expressions de même niveau.

```
procédure rechercherOccurrence
        (entrée tab <TabEntiers>,
        entrée n <Entier>, entrée x <Entier>,
        sortie trouvé <Booléen>,
        sortie rang <Entier>) ;

a <- plusGrandEntier (alpha, bêta, gamma)
        - alpha - bêta - gamma - delta - oméga ;

retourner
        (unMonôme
                (coefficient (m1) + coefficient (m2),
                exposant (m1))) ;
```

Si un en-tête de fonction porte sur plusieurs lignes, aligner les mots-clés **fonction** et **retourne**, en introduisant éventuellement une indentation de 4 espaces pour la liste des paramètres :

```
-- calcule le polynôme dérivé P'(x)
fonction polynômeDérivé (entrée p <Polynôme>)
retourne <Polynôme> ;

-- calcule le polynôme P(x) + Q(x)
fonction plus2Polynômes
        (entrée p <Polynôme>, entrée q <Polynôme>)
retourne <Polynôme> ;
```

6. Commentaires

On distingue quatre types de commentaires : le commentaire décrivant le rôle d'une constante ou d'une variable, le commentaire d'en-tête du programme, le commentaire d'en-tête d'un sous-programme et le commentaire de trace d'affinage.

6.1. Rôle d'une constante ou d'une variable

Une déclaration de constante ou de variable est complétée par un court commentaire indiquant son rôle dans le programme ou le sous-programme. Ce commentaire est situé sur la même ligne que la déclaration.

```
constante N <Entier> = 10 ;     -- nombre de lignes
damier <Carré> ;                -- damier de jeu
i <Entier> ;                    -- indice de ligne
```

6.2. En-tête du programme

Un programme est précédé d'un court commentaire indiquant son rôle.

```
-- calcule et affiche la date du lendemain
-- à partir de 3 entiers associés respectivement
-- au jour, au mois et à l'année de la date du jour
programme dateDuLendemain
```

6.3. En-tête d'un sous-programme

L'en-tête d'une procédure ou d'une fonction est précédé d'un cartouche explicitant le plus exactement possible le rôle du sous-programme. Ce cartouche mentionne systématiquement les paramètres formels du sous-programme, ainsi que le résultat produit s'il s'agit d'une fonction. Signaler aussi les exceptions levées par le sous-programme.

```
-- recherche la position de la première occurrence x
-- dans la tranche de tableau tab[1..n]
-- si x existe, trouvé = VRAI et rang
-- désigne la place de l'élément dans le tableau tab,
-- sinon trouvé = FAUX et rang est indéfini
procédure rechercherOccurrence
        (entrée tab <TabEntiers>,
        entrée n <Entier>, entrée x <Entier>,
        sortie trouvé <Booléen>,
        sortie rang <Entier>) ;
```

6.4. Trace d'un affinage

Le corps d'un programme, d'une procédure ou d'une fonction inclut la trace de son affinage en commentaires. Une action de l'algorithme (*comment*) se termine soit au début de la trace suivante (*quoi*) de même niveau d'indentation, soit au prochain niveau d'indentation inférieur au niveau courant, soit à la fin du texte. Instructions et commentaires sont alignés ; ils ne sont pas mélangés sur la même ligne.

```
procédure rechercherOccurrence ...
glossaire
        i <Entier> ;      -- indice du tableau tab
début
        -- se positionner sur le premier élément
        -- du tableau tab
        i <- 1 ;
        -- examiner les éléments du tableau tab
        tantque  i <= n et tab[i] /= x faire
                -- passer à l'élément suivant
                -- du tableau tab
                i <- i + 1 ;
        fin tantque ;
        -- fournir le résultat de la recherche
        si i <= n alors
                trouvé <- VRAI ;
                rang <- i ;
        sinon
                trouvé <- FAUX ;
        fin si ;
fin
```

7. Glossaire

Dans un glossaire de programme, de procédure ou de fonction, il n'est autorisé qu'une seule déclaration de variable par ligne. Une indentation de 4 espaces par rapport au mot-clé **glossaire** est appliquée pour chaque déclaration de variable.

```
glossaire
        i <Entier> ;              -- indice du tableau tab
        terminé <Booléen> ;       -- pour stopper l'examen
```

8. Instructions

Une ligne contient au plus une instruction.

8.1. Séquence

Les instructions d'une séquence sont alignées. Lorsqu'il s'agit de la séquence d'instructions du programme ou du sous-programme, une indentation de 4 espaces par rapport au mot-clé **début** est appliquée. Les récupérateurs d'exception sont également indentés.

```
début
        lire (x) ;
        insérerElément (tab, n, x) ;
traite-exception
        lorsque débordement faire
                écrire (″plus de place dans tab″) ;
        fin lorsque ;
fin
```

8.2. Sélection

Les mots-clés **si**, **sinon** et **fin si** sont alignés. La condition **si** ... **alors**, les mots-clés **sinon** et **fin si** occupent chacun une ligne. Les instructions d'une sélection sont alignées avec une indentation de 4 espaces par rapport aux mots-clés **si** et **sinon**.

```
si condition alors
        instruction1 ;
sinon
        instruction2 ;
fin si ;
```

Pour la sélection réduite, adopter le format suivant :

```
si condition alors
        instruction ;
fin si ;
```

8.3. Répétition

Les mots-clés **tantque** et **fin tantque** sont alignés. La condition **tantque** ... **faire** et les mots-clés **fin tantque** occupent chacun une ligne. Les instructions d'une répétition sont alignées avec une indentation de 4 espaces par rapport au mot-clé **tantque**.

```
tantque condition faire
        instruction ;
fin tantque ;
```

9. Recommandations

9.1. Taille des unités de programme

Le corps d'un programme ou d'un sous-programme ne doit pas, en règle générale, excéder 40 lignes (commentaires compris). Décomposer le traitement en sous-programmes sinon.

9.2. Constantes

Les constantes numériques n'apparaissent pas directement dans le code et sont remplacées par un identificateur de constante. Ainsi, la déclaration :

```
type TabEntiers : tableau [1 à 10] de <Entier> ;
```

doit s'écrire :

```
constante N <Entier> = 10 ;     -- nombre d'éléments
type TabEntiers : tableau [1 à N] de <Entier> ;
```

De même, écrire en utilisant la constante N :

```
-- se positionner sur le premier élément du tableau tab
i <- 1 ;
-- examiner tous les éléments du tableau tab
tantque i <= N faire
        -- traiter l'élément courant du tableau tab
        ...
        -- passer à l'élément suivant du tableau tab
        i <- i + 1 ;
fin tantque ;
```

9.3. Ordre des paramètres d'un sous-programme

Aucun ordre des paramètres d'un sous-programme n'est imposé. Toutefois, et en général :

- les paramètres transmis en mode entrée ou en mode mise à jour précèdent les paramètres en mode sortie,
- les paramètres ayant un lien sémantique sont consécutifs, par exemple un tableau et son nombre d'éléments ou encore un indicateur de résultat et le résultat lui-même,
- pour les types abstraits de données, le premier paramètre est un représentant du type.

Par exemple, préférer :

procédure rechercherOccurrence
(**entrée** tab <TabEntiers>, **entrée** n <Entier>,
entrée x <Entier>,
sortie trouvé <Booléen>, **sortie** rang <Entier>) ;

à :

procédure rechercherOccurrence
(**sortie** trouvé <Booléen>,
entrée tab <TabEntiers>,
sortie rang <Entier>, **entrée** n <Entier>,
entrée x <Entier>) ;

9.4. Sélections imbriquées

Lorsque plusieurs conditions portent sur une même variable de choix et que ces conditions s'excluent mutuellement, préférer des sélections imbriquées à des sélections en séquence.

```
si c = 'A' alors
        ...
sinon
        si c = 'B' alors
                ....
        sinon
                si c = 'C' alors
                        ...
                sinon
                        -- autres valeurs de c
                        ...
                fin si ;
        fin si ;
fin si ;
```

9.5. Retour de fonction

Lorsque c'est possible, n'écrire qu'un seul **retourner** par fonction, en général placé en tant que dernière instruction de la fonction. Sinon, s'assurer que toute branche d'exécution de la fonction se termine par un **retourner**.

Abréger certaines écritures en exprimant directement la valeur retournée par la fonction. Ainsi, transformer la séquence des deux instructions :

```
c <- a + b ;
retourner (c) ;
```

en un seul **retourner** :

```
retourner (a + b) ;
```

De même, l'écriture :

```
si a = b alors
        retourner (VRAI) ;
sinon
        retourner (FAUX) ;
fin si ;
```

doit se simplifier en :

```
retourner (a = b) ;
```

9.6. Sorties de boucles

La ou les variables de contrôle d'une répétition sont systématiquement initialisées juste avant la répétition. En général, l'évolution de ces variables constitue la dernière instruction de la répétition.

```
-- se positionner sur la première ligne
i <- 1 ;
-- parcourir toutes les lignes
tantque i <= n faire
        -- se positionner sur la première colonne
        -- de la ligne
        j <- 1 ;
        -- parcourir toutes les colonnes de la ligne
        tantque j <= n faire
                écrire (matrice[i, j]) ;
                -- passer à la colonne suivante
                j <- j + 1 ;
        fin tantque ;
        -- passer à la ligne suivante
        i <- i + 1 ;
fin tantque ;
```

Ne pas forcer la condition de sortie d'une répétition en affectant intempestivement sa ou ses variables de contrôle. Réécrire la condition de sortie de la répétition, en introduisant si besoin une condition composée.

9.7. Conditions

Eviter d'utiliser les constantes VRAI et FAUX pour exprimer la condition d'une sélection ou d'une répétition.

Par exemple, pour *a* et *b* variables booléennes, remplacer l'écriture :

```
si a = VRAI et b = FAUX alors
        …
```

par :

```
si a et non b alors
        …
```

9.8. Déclenchement d'exception

Il s'agit de détecter une anomalie au plus tôt et de l'intercepter au travers d'une sélection réduite, le traitement standard continuant en séquence. Si l'exception est propagée, la clause **déclenche** l'indique dans l'en-tête du sous-programme.

```
-- retourne la racine carrée de l'entier a
fonction racineCarrée (entrée a <Entier>)
retourne <Entier>
déclenche impossible
glossaire
        racine <Entier> ;   -- racine carrée de a
début
        si a < 0 alors
                déclencher (impossible) ;
        fin si ;
        -- calculer la racine carrée
        ...
        retourner (racine) ;
fin
```

10. Exemple

Le code ci-dessous est un extrait d'un programme de jeu de morpion, rédigé en respectant les règles précédemment énoncées. Tous les sous-programmes n'ont pas été définis pour alléger le texte.

```
constante N <Entier> = 10 ;     -- nombre de lignes du damier
type Carré : tableau [1 à N, 1 à N] de <Caractère> ;

-- retourne VRAI si tous les caractères d'une rangée du tableau
-- damier d'ordre n sont égaux au caractère c et FAUX sinon ;
-- une rangée est soit une ligne i, soit une colonne j, soit la 1ère
-- diagonale, soit la 2ème diagonale
fonction gagnant
        (entrée damier <Carré>, entrée n <Entier>,
        entrée i <Entier>, entrée j <Entier>,
        entrée c <Caractère>)
retourne <Booléen>
début
        retourner
                (ligneGagnante (damier, n, i, c)
                ou colonneGagnante (damier, n, j, c)
                ou premièreDiagonaleGagnante (damier, n, c)
                ou deuxièmeDiagonaleGagnante (damier, n, c)) ;
fin

-- affecte une case du damier d'ordre n par le caractère c,
-- en demandant au joueur (autant de fois que nécessaire)
-- les numéros de ligne i et de colonne j d'une case ;
-- fournit les indices  i et j de la case affectée
procédure choisirCase
        (màj damier <Carré>,
        entrée n <Entier>, entrée c <Caractère>,
        sortie i <Entier>, sortie j <Entier>)
déclenche débordement
```

```
glossaire
        trouvé <Booléen> ;          -- pour stopper le dialogue
début
        trouvé <- FAUX ;
        tantque non trouvé faire
                -- lire les indices ligne et colonne de la case
                lire (i) ;
                lire (j) ;
                -- s'assurer de leur validité
                si i < 1 ou i > n ou j < 1 ou j > n alors
                        déclencher (débordement) ;
                fin si ;
                -- examiner si la case est libre
                si damier[i, j] = ' ' alors
                        damier[i, j] <- c ;
                        trouvé <- VRAI ;
                fin si ;
        fin tantque ;
fin

-- autres définitions de sous-programme
…

-- jouer au morpion contre la machine
-- auteur : Christian Percebois
programme jouerAuMorpion

glossaire
        damier <Carré> ;    -- damier de jeu
        i <Entier> ;        -- indice d'une ligne du damier
        j <Entier> ;        -- indice d'une colonne du damier
        joueur <Entier> ;   -- joueur courant
        nbc <Entier> ;      -- nombre de cases occupées
        gagné <Booléen> ;   -- indique si un des 2 joueurs a gagné
début
        -- préparer le jeu
        …
        -- jouer : alternativement la machine et le joueur
        joueur <- 0 ;
        nbc <- 0 ;
        gagné <- FAUX ;
        tantque nbc < N * N et non gagné faire
                si joueur = 0 alors
                        -- chercher une case libre dans le damier
                        chercherCase (damier, N, 'X', i, j) ;
                        gagné <- gagnant (damier, N, i, j, 'X') ;
                sinon
                        -- choisir une case libre du damier
                        choisirCase (damier, N, 'O', i, j) ;
                        gagné <- gagnant (damier, N, i, j, 'O') ;
                fin si ;
                afficherDamier (damier, N) ;
                -- changer de joueur
                …
        fin tantque ;
        -- examiner si le joueur courant a gagné
        si gagné alors
                écrire (joueur) ;
        fin si ;
traite-exception
        lorsque débordement faire
                …
        fin lorsque ;
fin
```