Algorithmes et Programmation Impérative 1

Examen de mai 2007

durée 2h - documents non autorisés

Exercice 1 : *Compréhension du langage* (30MN)

Q 1 . On suppose dans cette question déclarées les procédures, fonctions et variables suivantes :

```
procedure p(const x : CARDINAL; out y : BOOLEAN);
function f(const x : CARDINAL) : CARDINAL;

var
  x,y : BOOLEAN;
  z,t : CARDINAL;
```

Dans ce contexte, indiquez quelles sont les instructions non valides parmi celles ci-dessous ? (problème de type, mode de passage des paramètres, appels de fonctions ou de procédures, ...)

1. `p(z,x);`
2. `p(1,true);`
3. `p(2*z,x);`
4. `p(z,x and y);`
5. `a := p(z,x);`
6. `f(z) := t;`
7. `z := f(t);`
8. `z := f(2*z);`
9. `if f(z) then ...`
10. `f(z);`
11. `p(f(z),x);`
12. `z := f(p(z,x));`

Q 2 . On suppose dans cette question déclarées les procédures, fonctions et variables suivantes :

```
procedure p(const x : INTEGER; var y : INTEGER);
var
  z : INTEGER;
begin
   z := x-y;
   y := z+2*y;
end {p};

var  x,y,z : INTEGER;
```

En supposant les variables initialisées par `{x=1,y=2,z=3}`, indiquez la valeur de ces variables après chacune des instructions suivantes :

1. `p(x,y);`
2. `p(y,z);`
3. `p(x+y,z);`

Exercice 2 : *Compréhension d'un algorithme* (20MN)

Cet exercice porte sur l'algorithme décrit ci-dessous dont il s'agira d'établir le but.

Données : un tableau $t[1..N]$, un indice $k > 1$
CU : la tranche $t[a..k-1]$ est triée
But : ? ? ?
Var. locales : a, b, m, aux, i

```
1  a := 1
2  b := k − 1
3  tant que a ≤ b faire
4    m := ⌊(a+b)/2⌋
5    si t[k] < t[m] alors
6      b := m − 1
7    sinon
8      a := m + 1
9    fin si
10 fin tant que
11 aux := t[k]
12 pour i variant de k − 1 à a en décroissant faire
13   t[i + 1] := t[i]
14 fin pour
15 t[a] := aux
```

Q 1 . Montrez les valeurs des variables a, b et m durant l'exécution des lignes 1 à 10 de cet algorithme lorsque $N = 10$, $k = 7$ et

$$t = \boxed{1}\boxed{2}\boxed{6}\boxed{8}\boxed{9}\boxed{10}\boxed{2}\boxed{4}\boxed{7}\boxed{13}.$$

Vous présenterez ces valeurs sous forme d'un tableau à trois lignes (une ligne par variable). La première colonne de ce tableau indique la valeur des variables a et b avant l'entrée dans le `tant que`, les colonnes suivantes donnent les valeurs des trois variables a, b et m à la fin de chaque étape de l'itération `tant que`.
Q 2 . Avec les mêmes données, quel est l'état du tableau t à la fin de cet algorithme ?
Q 3 . De manière générale, que fait l'algorithme ?

Exercice 3 : *Fichiers* (20mn)

Lors d'une compétition on a établi un fichier donnant le classement toutes catégories confondues des coureurs. Pour ce fichier le type des articles est défini par

```
type COUREUR = record
                  nom       : STRING ;
                  prenom    : STRING ;
                  categorie : CATEGORIE ;
                  temps     : STRING ;   // au format  'hh:mm:ss'
               end {record} ;
```

le type `CATEGORIE` étant défini préalablement par

```
type CATEGORIE = (CF,CM,JF,JM,SF,SM,VF,VM) ;
```

où `C` signifie cadet, `J` junior, `S` sénior, `V` vétérans, `F` féminin et `M` masculin.

Ce fichier du type `file of COUREUR` est stocké dans le répertoire courant avec le nom `scratch.data`

On veut à partir de ce fichier créer des fichiers de résultats par catégorie par exemple `veteransM.data` pour les vétérans masculins.

On a alors défini un nouveau type où n'apparait plus la catégorie

```
type COUREURBIS = record
                     nom    : STRING ;
                     prenom : STRING ;
                     temps  : STRING ;  // au format  'hh:mm:ss'
                  end {record} ;
```

Q 1 . Écrivez la fonction `enCoureurbis` à un paramètre `c` de type `COUREUR` qui transforme le `COUREUR c` en un `COUREURBIS`.
Q 2 . Écrivez maintenant la procédure `creerFichier(cat,nom)`, `cat` de type `CATEGORIE` et `nom` de type `STRING`, qui fabrique le fichier de résultats nommé `nom` pour la catégorie `cat` ; par exemple après l'instruction `creerFichier(VM,'veteransM.data')` on aura dans le répertoire courant un fichier `veteransM.data` contenant les résultats pour les vétérans masculins.

Exercice 4 : Sudoku (50mn)

Le jeu du Sudoku est classiquement constitué d'une grille carrée de taille 9×9 dont certaines cases sont remplies de nombres compris entre 1 et 9, les autres étant vides. Le but du jeu consiste à remplir les cases vides avec les nombres de 1 à 9 de telle sorte que

- chaque ligne de la grille contienne tous les nombres de 1 à 9 ;
- ainsi que chaque colonne ;
- et de même pour chacun des neuf carrés 3×3.

La figure 1 montre un problème de Sudoku, accompagné de sa solution.
Q 1 . Quelle structure de données proposez-vous pour représenter une grille de Sudoku ? Définissez le type `SUDOKU`.
Q 2 . L'objectif de cette question est de réaliser une fonction qui permet de tester la validité d'une grille de Sudoku 9×9 complètement remplie dont voici l'entête :

```
function grilleValide(grille : SUDOKU) : BOOLEAN;
```

On rappelle qu'une grille est correcte si

1. chacune des neuf lignes contient tous les entiers de 1 à 9 ;
2. chacune des neuf colonnes contient tous les entiers de 1 à 9 ;
3. chacun des neuf sous-carrés 3×3 contient tous les entiers de 1 à 9.

		1		2		7		
	5						9	
			4					
	8				5			
	9							
				6				2
		2						
		6						5
					9		8	3

3	6	1	9	2	8	7	5	4
4	5	8	6	3	7	2	9	1
7	2	9	4	5	1	8	3	6
2	8	4	1	9	5	3	6	7
6	9	3	7	4	2	5	1	8
5	1	7	8	6	3	9	4	2
8	3	2	5	1	6	4	7	9
9	7	6	3	8	4	1	2	5
1	4	5	2	7	9	6	8	3

FIG. 1 – Une grille 9×9 et sa solution

Q 2.1. Réalisez la fonction `ligneValide(`g`,`i`)` qui vaut **`true`** si la ligne i de la grille g contient tous les entiers de 1 à 9.

Q 2.2. On désigne chacun des 9 sous-carrés 3×3 d'une grille par un couple (k, l) de deux entiers compris entre 1 et 3 comme il est indiqué sur la figure 2. Ainsi, le carré situé dans le coin supérieur gauche est désigné par le couple $(1, 1)$, celui du coin inférieur droit par $(3, 3)$.

$(1,1)$	$(1,2)$	$(1,3)$
$(2,1)$	$(2,2)$	$(2,3)$
$(3,1)$	$(3,2)$	$(3,3)$

FIG. 2 – Coordonnées des carrés 3×3 dans une grille de SUDOKU

Réalisez la fonction `sousCarreValide(`g`,`k`,`l`)` qui détermine si le sous-carré de coordonnées (k, l) de la grille g contient tous les entiers de 1 à 9.

Q 2.3. En supposant la fonction `colonneValide` réalisée (de façon analogue à `ligneValide`), programmez la fonction `grilleValide`.

Q 3 . Il existe des grilles de SUDOKU 16×16 qui doivent alors être remplies avec les entiers de 1 à 16. Plus généralement, il existe des grilles de SUDOKU de taille $n^2 \times n^2$ pour tout entier naturel non nul n.

Déclarez un type de données permettant la représentation d'une grille complètement remplie de SUDOKU de taille $n^2 \times n^2$, où n est un entier pouvant prendre n'importe quelle valeur comprise entre 1 et une constante `MAX` fixée.

Solutions

Exercice 1

Q 1 .

Solution

1. valide
2. non valide : deuxième paramètre effectif constant alors que le formel est en sortie
3. valide
4. non valide : second paramètre effectif expression
5. non valide : appel procédure $\neq$ expression
6. non valide : membre gauche de l'affectation $\neq$ expression
7. valide
8. valide
9. non valide : expression non booleenne dans la condition
10. non valide : appel de fct $\neq$ instruction
11. valide
12. non valide : appel procédure $\neq$ expression

Barême : 3pts enlever 1/2 pt par erreur. Ne pas compter négatif si nb erreur > 6

Q 2 .

Solution

compter bon les deux interprétations possibles de la question posée

1. les trois instructions sont exécutées dans le même contexte (en parallèle)
2. les trois instructions sont exécutées séquentiellement

premier cas
1. `{x=1, y=3, z=3 }`
2. `{x=1, y=2, z=5 }`
3. `{x=1, y=2, z=6 }`

second cas
1. `{x=1, y=3, z=3 }`
2. `{x=1, y=3, z=6 }`
3. `{x=1, y=3, z=10 }`

Barême : 2pts

Exercice 2

Q 1 .

Solution

a	1	1	2	3
b	6	2	2	2
m		3	1	2

Q 2 .

Solution

$t =$ | 1 | 2 | 2 | 6 | 8 | 9 | 10 | 4 | 7 | 13 |.

Q 3 .

Solution L'algorithme insère l'élément $t[k]$ dans la tranche $t[1..k]$ de sorte que cette tranche soit triée. La recherche de la position d'insertion se fait par dichotomie.

Exercice 3

Q 1 .

Solution

```
function enCoureurBis(c : COUREUR) : COUREURBIS;
var
  result : COUREURBIS;
begin
  result.nom := c.nom;
  result.prenom := c.prenom;
  result.temps := c.temps;
  enCoureurBis := result;
end {enCoureurBis};
```

Q 2 .

Solution

```
procedure creerFichier(const cat : CATEGORIE; const nom : STRING);
var
  f : FILE of COUREUR;
  g : FILE of COUREURBIS;
  c : COUREUR;
begin
  assign(f,'scratch.data');
  reset(f);
  assign(g,nom);
  rewrite(g);
  while not eof(f) do begin
    read(f,c);
    if c.categorie = cat then
      write(g,enCoureurBis(c));
  end {while};
  close(g);
  close(f);
end {creerFichier};
```

Exercice 4

Q 1 .

Solution Un tableau 9×9 d'entiers (compris entre 1 et 9) convient parfaitement.

```
type
  INDICE = 1..9;
  INDICE_ETENDU = 1..10;
  SUDOKU = array[INDICE,INDICE] of CARDINAL;
```

Q 2 .1

Solution

```
type
   VALEURS_AUTORISEES   = 1..9;
   T_BOOLEEN = array[VALEURS_AUTORISEES] of BOOLEAN;

   // init(tab) initialise toutes les cases du tableau tab
   // à la valeur false
   procedure init(out tab : T_BOOLEEN);
   var
      i : VALEURS;
   begin
      for i := low(VALEURS) to high(VALEURS) do
         tab[i] := false;
   end {init};

   function ligneValide(g : SUDOKU; i : INDICE) : BOOLEAN;
   var
      j : INDICE_ETENDU;
      valide : BOOLEAN;
```

```
   dejaVu : T_BOOLEEN;
begin
   j := low(INDICE);
   valide := true;
   init(dejaVu);
   while valide and (j<=high(INDICE)) do begin
      if (g[i,j]<low(VALEURS_AUTORISEES)) or
         (g[i,j]>high(VALEURS_AUTORISEES)) or
         dejaVu[g[i,j]] then
         valide := false
      else
         dejaVu[g[i,j]] := true;
      inc(j);
   end {while};
   ligneValide := valide;
end {ligneValide};
```

Q 2 .2
Solution

```
type
   INDICE_SOUS_CARRE = 1..3;
   INDICE_SOUS_CARRE_ETENDU = 1..4;

   function sousCarreValide(g : SUDOKU; k,l : INDICE_SOUS_CARRE) : BOOLEAN;
   var
      i,j : INDICE;
      valide : BOOLEAN;
      dejaVu : T_BOOLEEN;
      r : RACINE_TAILLE;
   begin
      valide := true;
      init(dejaVu);
      i := 1 + 3*(k-1);
      j := 1 + 3*(l-1);
      while valide and (i mod 3 <> 0) do begin
         if (g[i,j]<low(VALEURS_AUTORISEES)) or
            (g[i,j]>high(VALEURS_AUTORISEES)) or
            dejaVu[g[i,j]] then
            valide := false
         else
            dejaVu[g[i,j]] := true;
         if (j mod 3 = 0) then begin
            inc(i);
            j := 1 + 3*(l-1);
         end else
            inc(j);
      end {while};
      sousCarreValide := valide;
   end {sousCarreValide};
```

Q 2 .3
Solution

```
   function grilleValide(grille : SUDOKU) : BOOLEAN;
   var
      i,j : INDICE_ETENDU;
      k,l : INDICE_SOUS_CARRE_ETENDU;
      valide : BOOLEAN;
   begin
      valide := true;
      // vérification des lignes
      i := low(INDICE);
      while valide and (i<=high(INDICE)) do begin
```

```
      valide := ligneValide(grille,i);
      inc(i);
   end {while};
   // vérification des colonnes
   j := low(INDICE);
   while valide and (j<=high(INDICE)) do begin
      valide := colonneValide(grille,j);
      inc(j);
   end {while};
   // vérification des sous-carrés
   k := low(INDICE_SOUS_CARRE);
   l := low(INDICE_SOUS_CARRE);
   while valide and (k<=3) do begin
      valide := sousCarreValide(grille,k,l);
      if l=3 then begin
         inc(k);
         l := low(INDICE_SOUS_CARRE);
      end else
         inc(l);
   end {while};
   grilleValide := valide;
end {grilleValide};
```

Q 3 .

Solution

On peut choisir de représenter les sudokus de taille bornée par une constante MAX à l'aide d'un enregistrement à deux champs :

- un champ de type INDICE donnant la taille du sudoku représenté ;
- un champ de type tableau à deux dimensions donnnant le contenu de la grille.

```
const
  MAX = ... ;
type
  INDICE  = 1..MAX*MAX;
  INDICE_ETENDU  = 1..MAX*MAX+1;
  SUDOKU  = record
    taille  : INDICE;
    contenu : array[INDICE,INDICE] of CARDINAL;
  end {SUDOKU};
```