Examen

janvier 2016 - Durée 2h

Documents autorisés : **Fiche Traduction Algo-ADA** et **Mémento ADA** vierges de toute annotation manuscrite

Ne vous laissez pas impressionner par la longueur du sujet, il y a beaucoup d'annexes dont vous connaissez déjà le contenu. Lorsque l'on vous demande de modifier un programme indiquez seulement ce que vous modifiez en vous aidant des numéros de lignes.

1 - Génération aléatoire

[*barême indicatif : 5 pts*]

On donne la spécification d'un paquetage **generation_aleatoire** définissant :
- la procédure `init` qui initialise le générateur aléatoire avec un germe donné.
- la fonction `alea_1_2_3` qui prend deux données entières comprises entre 1 et 100 et qui rend un naturel compris entre 1 et 3 selon la spécification précisée ci après.

```
1  -- paquetage de génération aléatoire - spécification
2  -- fichier  generation_aleatoire.ads
3
4  package generation_aleatoire is
5
6     subtype pourcentage is natural range 0..100;
7
8     -- procédure qui initialise le générateur aléatoire avec le germe a
9     procedure init(a : integer);
10
11    -- fonction qui génère un entier (1, 2 ou 3)
12    -- suivant deux probabilités p1 et p2 exprimées en pourcentage
13    --    la valeur 1 est générée avec une probabilité égale à p1 %
14    --    la valeur 2 est générée avec une probabilité égale à p2 %
15    --    la valeur 3 est générée avec une probabilité égale à 100-p1-p2 %
16    -- Précondition : p1+p2<=100
17    function alea_1_2_3(p1, p2 : pourcentage) return natural;
18
19 end generation_aleatoire;
```

Question 1-1 :

Pour répondre à cette question vous utiliserez le générateur aléatoire générique dont la spécification est rappelée en Annexe 1. Écrivez le corps du paquetage **generation_aleatoire**.

On veut écrire un programme qui génère une séquence aléatoire de caractères (A, D ou G) dans un fichier texte. Le programme est appelé avec 5 arguments :

argument 1 n : la longueur de la séquence

argument 2 p1 : la probabilité p1 exprimée en pourcentage

argument 3 p2 : la probabilité p2 exprimée en pourcentage

argument 4 nom_f : le nom du fichier à créer

argument 5 germe : le germe pour le générateur

Les valeurs p1, p2 sont deux entiers entre 0 et 100 avec p1+p2 $\leq$ 100. Le fichier créé doit contenir une seule ligne de n caractères, chaque caractère étant : soit 'A' avec la probabilité p1 %, soit 'D' avec la probabilité p2 %, soit 'G' avec la probabilité 100-p1-p2 %.

```
1  -- fichier generation_sequence.adb
2  -- programme générant une séquence aléatoire de caractères (A,D ou G)
3  -- et l'écrivant dans un fichier texte
4
5  with ada.text_io, ada.command_line;
6  use  ada.text_io, ada.command_line;
7  with generation_aleatoire;
8  use  generation_aleatoire;
```

```
9
10 procedure generation_sequence is
11    p1, p2 : pourcentage;
12    n : natural;
13    germe : integer ;
14    f : file_type ;
15 begin
16    if argument_count /= 5 then
17       -- nombre d'argument incorrect
18       put_line("syntaxe : " & command_name & " n p1 p2 nom_f germe");
19    else
20       -- convertir les arguments
21       n := natural'value(argument(1));
22       p1 := natural'value(argument(2));
23       p2 := natural'value(argument(3));
24       germe := integer'value(argument(5));
25       if p1 + p2 > 100 then
26          put_line("erreur : p1+p2 doit etre <= 100");
27       else
28          -- creer le fichier sequence sequence de caracteres
29          -- A COMPLETER (Question 1.2)
30       end if;
31    end if;
32 end generation_sequence;
```

Question 1-2 :

Complétez la procédure **generation_sequence** afin de créér le fichier texte contenant la séquence de caractères demandée.

2 - Exploitation de la séquence créée

[*barême indicatif : 5 pts*]

Étant donné un fichier composé d'une suite de caractères `A`, `G` et `D`, on s'intéresse à la longueur des sous-séquences de `A` consécutifs : on souhaite connaître le nombre de sous-séquences de `A` de chaque longueur contenues dans le fichier. On suppose que la longueur maximale est 10. Par exemple, dans le fichier suivant (des espaces non significatifs ont été ajoutés afin de faciliter la lecture) :

```
D A G AAAAAA DG AAAAAA G A D A GG A G AAAAAAA G A D A DG AA G A D A D AAA
```

il y a huit séquences de longueur 1, une de longueur 2, une de longueur 3, deux de longueur 6, une de longueur 7, et aucune de toutes les autres longueurs. On souhaite réaliser un affichage tel que :

```
LG :  NB
 1 :  8
 2 :  1
 3 :  1
 4 :  0
 5 :  0
 6 :  2
 7 :  1
 8 :  0
 9 :  0
 10 :  0
```

On dispose du paquetage **stats** (Cf. spécification en Annexe 2) qui permet d'initialiser un tableau de valeurs, d'incrémenter l'effectif associé à une valeur ou de donner l'effectif associé à une valeur.

On écrit un programme **affichage_stats** qui prend en argument un fichier de caractères, lit le contenu de ce fichier et affiche les statistiques désirées à l'écran.

```
1 -- fichier affichage_stats.adb
2 with ada.text_io, ada.command_line, stats ;
3 use ada.text_io, ada.command_line, stats ;
4
5 procedure affichage_stats is
6    f : file_type ;
7    Av : boolean := false ; -- vrai en cours de séquence de 'A'
8    v : natural := 0 ;
9    c : character ;
10 begin
11    if argument_count /= 1 then
12       put("syntaxe : " & command_name & "fichier_mouvements");
```

```
13     else
14        -- lire fichier de caracteres et construire les statistiques
15        -- A COMPLETER (Question 2.1)
16
17        -- afficher les statistiques
18        -- A COMPLETER (Question 2.2)
19
20     end if ;
21  end affichage_stats ;
```

Question 2-1 :

Compléter la partie de lecture du fichier. Il s'agit de détecter les séquences formées de caractères 'A'. Vous pouvez par exemple utiliser un booléen qui passe à vrai lorsqu'un premier 'A' est détecté puis reste à vrai tant que des 'A' sont lus et qui repasse à faux dès qu'on lit un 'G' ou un 'D'. Au fur et à mesure de la lecture de 'A' il faut mettre à jour le tableau de statistiques.

Question 2-2 :

Compléter la partie affichage des statistiques.

3 - Robots

[*barême indicatif : 5 pts*]

Dans cette partie, nous nous plaçons dans le cadre du mini-projet Robots : on s'intéresse aux longueurs des séquences de `AVANCER` consécutifs dans les trajets des robots. On donne en annexe 4 le paquetage **systeme_robot** et en annexe 5 le programme **test_performance**.

Question 3-1 :

Indiquez les modifications à apporter au paquetage **systeme_robot** pour que la procédure `suivant` ait un paramètre résultat de type `character` valant `'A'`, `'G'` ou `'D'` selon que le robot avance, tourne à gauche ou à droite.

Question 3-2 :

Indiquez les modifications à apporter au programme **test_performance.adb** pour que l'entier affiché dans le fichier résultat pour chaque terrain soit :

- -1 si le robot n'est pas sorti,
- la longueur du plus long trajet rectiligne (séquence de `AVANCER` consécutifs) sinon.

4 - Histogramme

[*barême indicatif : 5 pts*]

Dans cette partie, on souhaite comparer à l'aide d'un histogramme la manière dont deux robots (correspondant à deux automates différents) se déplacent sur un ensemble de terrains donné. Pour cela on dispose des résultats sur les longueurs des séquences `AVANCER` effectuées par les deux robots. Par exemple, le tableau 1 indique les effectifs sur les longueurs de 1 à `NL` = 10 pour les deux robots.

LG	1	2	3	4	5	6	7	8	9	10
Robot A	27	50	22	12	6	3	1	2	1	1
Robot B	40	30	20	15	12	9	7	5	1	2

FIGURE 1 – Effectifs sur les longueurs de 1 à `NL` = 10 pour les deux robots

On souhaite représenter ces données à l'aide d'un histogramme :

1. pour une longueur donnée, on représente la valeur correspondante à chaque robot sous forme d'un rectangle blanc avec un bord noir pour le premier robot et un rectangle vert avec un bord noir pour le second, la hauteur en pixels de chaque rectangle étant proportionnelle à la valeur, le rectangle le plus haut ayant la hauteur du cadre `H_CADRE`.
2. la largeur de chaque rectangle est donnée par `L_RECT`, l'espace entre deux couples de rectangle est égal à `L_RECT`

La figure 2 montre l'histogramme correspondant au tableau 1, avec les différentes dimensions.

Question 4-1 :

Déterminez en fonction de `NL`, `MARGE` et `L_MAX`, la formule pour calculer la valeur maximale de `L_RECT` afin que la largeur totale de la fenêtre graphique `LF` soit inférieure ou à égale à `L_MAX`. Puis, à l'aide de la formule obtenue, calculez la valeur de `L_RECT` pour `NL = 10`, `MARGE = 10` et `L_MAX = 600`.

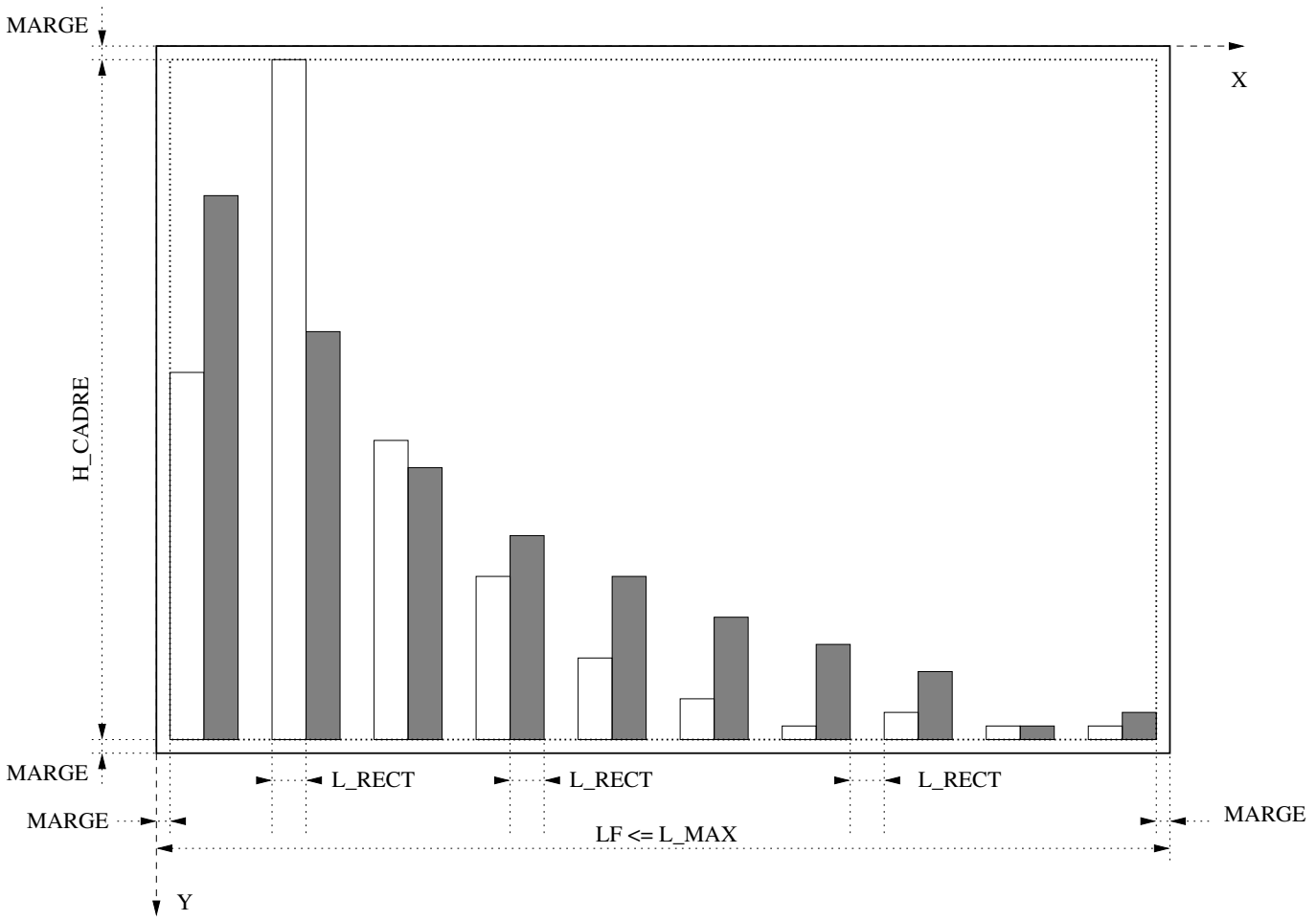

FIGURE 2 – Tracé de l'histogramme correspondant aux données de la figure 1

Le programme de tracé d'un histogramme **trace_histogramme.adb** est donné ci-dessous. Vous trouverez en annexe 3 la spécification du paquetage **graphsimple** permettant de gérer le graphisme.

```
1  -- tracé de l'histogramme
2  -- fichier trace_histogramme.adb
3
4  with graphsimple;
5  use graphsimple;
6
7  procedure trace_histogramme is
8
9      -- statistiques sur les robots
10     type TableauValeurs is array(positive range <>) of natural;
11    StatRobotA : TableauValeurs := ( ... );
12    StatRobotB : TableauValeurs := ( ... );
13    NL : positive := StatRobotA'length;
14
15     -- dessin de histogramme dans une fenetre graphique
16     procedure graphique_histogramme is
17
18         -- largeur de l'espace entre le bord de la fenetre et
19         -- le cadre en pointillé du dessin des classes
20         MARGE : constant natural := 10;
21
22         -- hauteur du cadre en pointille
23         H_CADRE : constant natural := 500;
24
25         -- largeur maximale de la fenetre
26         L_MAX : constant natural := 600;
27
28         -- hauteur de la fenetre graphique
29         HF : natural := H_CADRE+2*MARGE;
```

```
30
31          -- ordonnees min et max pour les cadres en pointillé
32          Y_min : natural := MARGE;
33          Y_max : natural := MARGE+H_CADRE;
34
35          -- largeur d'un rectangle / espace entre deux séries de rectangle
36          L_RECT : natural;
37
38          -- bornes des abscisses pour le dessin des classes
39          XC_min, XC_max : natural;
40
41          -- effectif maxi parmi les effectifs des classes
42          e_max : natural;
43
44          -- largeur de la fenetre graphique
45          LF : natural;
46
47        ---- variables annexes pour le tracé des rectangles ----
48        XR_min, XR_max : natural; -- bornes des abscisses du rectangle
49        YR_min, YR_max : natural; -- bornes des ordonnées du rectangle
50
51      begin
52          -- calcul de L_RECT, LF, XC_min, XC_max
53          ...
54
55          -- calcul de l'effectif maxi e_max
56          e_max := 0;
57          for i in 1..NL loop
58              if e_max < StatRobotA(i) then  e_max := StatRobotA(i);  end if;
59              if e_max < StatRobotB(i) then  e_max := StatRobotB(i);  end if;
60          end loop;
61
62          -- initialise la fenetre graphique
63          Initialiser(LF,HF);
64
65          -- debut du dessin
66          DebutDessin;
67
68          -- tracé du cadre des classes
69          ChangerCouleur(noir);
70          RectanglePointille(XC_min,Y_min,XC_max,Y_max);
71
72          -- tracé des classes
73          -- A COMPLETER (question 4.2) --
74
75          -- fin du dessin et affichage
76          FinDessin;
77
78      end graphique_histogramme;
79
80  begin
81      graphique_histogramme; -- tracer l'histogramme
82      AttendreClic;          -- attendre un clic souris
83      Clore;                 -- et fermer le graphique
84  end trace_histogramme;
```

Question 4-2 :

Complétez le code du programme **trace_histogramme.adb** afin de pouvoir tracer l'histogramme.

5 - Annexes

[*barême indicatif :*]

Annexe 1 : Générateur aléatoire : ada.numerics.discrete_random

```
-- extrait de ada.numerics.discrete_random.ads
generic
    type Result_Subtype is (<>);
   // le type paramètre <> designe n'importe quel type discret
package Ada.Numerics.Discrete_Random is
   type Generator is limited private;
   function Random (Gen : Generator) return Result_Subtype;
   procedure Reset (Gen : in Generator; Initiator : in Integer);
   procedure Reset (Gen       : in Generator);
   [...]
   private
   ... -- not specified by the language
end Ada.Numerics.Discrete_Random;
```

Annexe 2 : Paquetage de gestion d'un tableau d'effectifs : stats

```
-- fichier stats.ads
package stats is
   -- Gestion d'un tableau d'effectifs pour des valeurs entières positives
   -- le tableau d'effectifs (de longueur 10) est local a stats.adb

   -- Mise à 0 de tous les effectifs
   procedure initstats ;

   -- Ajout de 1 à l'effectif de la valeur v
   procedure ajoutervaleur(v : positive) ;

   -- Retourne l'effectif de la valeur v
   function effectif(v : positive) return natural ;
end stats ;
```

Annexe 3 : Paquetage graphsimple, un extrait

```
-- extrait du fichier graphsimple.ads
package graphsimple is

   -- Constantes definissant les couleurs
   type Couleur is (Noir, Blanc, Rouge, Bleu, Vert, Jaune);

   -- Initialise la fenetre graphique
   procedure Initialiser(x, y : Positive);

   -- Clot la fenetre graphique
   procedure Clore;

   -- Trace un rectangle dont les extremites d'une de ses diagonales sont
   -- (x1, y1) et (x2, y2)
   procedure Rectangle(x1, y1, x2, y2 : Natural);

   -- Remplit un rectangle dont les extremites d'une de ses diagonales sont
   -- (x1, y1) et (x2, y2)
   procedure RectanglePlein(x1, y1, x2, y2 : Natural);

   -- Debute une sequence de traces. Le dessin effectue sera affiche lors
   -- de l'appel de FinDessin
   procedure DebutDessin;

   -- Termine une sequence de traces. Le dessin effectue depuis l'appel de
   -- DebutDessin est affiche
   procedure FinDessin;

   -- Suspend l'execution du programme jusqu'a un appui du bouton de la souris
   procedure AttendreClic;

   -- Change la couleur utilisee pour les tracés
   procedure ChangerCouleur(c : Couleur);
end graphsimple;
```

Annexe 4 : Paquetage systeme_robot

```
1  -- paquetage systeme_robot
2  -- extrait du fichier systeme_robot.ads
3
4  with automate_paq, terrain_paq, robot_paq;
5  use automate_paq, terrain_paq, robot_paq;
6  with ada.text_io; use ada.text_io;
7
8  package systeme_robot is
9
10    -- type ConfigCase correspondant au type Entree défini
11    -- dans le paquetage automate_paq
12    subtype ConfigCase is Entree;
13
14    -- type ActionRobot correspondant au type Sortie défini
15    -- dans le paquetage automate_paq
16    subtype ActionRobot is Sortie;
17
18    -- initialiser un SystemeRobot avec un ficher automate fa ou de nom nom_fa
19    -- et un fichier terrain ft ou de nom nom_ft
20    procedure init(fa : in file_type ; ft : in file_type);
21    procedure init(nom_fa : in string; ft : in file_type);
22    procedure init(fa : in file_type ; nom_ft : in string);
23    procedure init(nom_fa : in string; nom_ft : in string);
24
25    -- faire avancer le SystemeRobot d'une case
26    procedure avancer;
27
28    -- faire tourner le SystemeRobot 'a gauche
29    procedure tourner_a_gauche;
30
31    -- faire tourner le SystemeRobot 'a droite
32    procedure tourner_a_droite;
33
34    -- détermine l'état suivant du SystemeRobot suivant son automate
35    procedure suivant;
36
37    -- indique si le SystemeRobot est sorti
38    function est_sorti return boolean;
39
40    -- indique si le SystemeRobot a atteint son nombre max de transitions
41    function max_transitions_atteint return boolean;
42
43    [...]
44
45    -- recupere la config de la case devant le SystemeRobot
46    function config_case_devant return ConfigCase;
47
48    -- ecrire à l'écran le SystemeRobot
49    procedure ecrire;
50
51 end systeme_robot;
```

```
1  -- paquetage systeme_robot
2  -- extrait du fichier systeme_robot.adb
3
4  with ada.text_io, ada.integer_text_io;
5  use ada.text_io, ada.integer_text_io;
6
7  with automate_paq, terrain_paq, robot_paq;
8  use automate_paq, terrain_paq, robot_paq;
9
10 package body systeme_robot is
11
12    r : Robot;       -- robot du systeme
13    a : automate;    -- automate du systeme
14    t : terrain;     -- terrain du systeme
15    nb_t : natural; -- nb de transitions effectués par le robot
16    nb_max_t : natural; -- nb maxi de transitions permises
```

```
17
18    [...]
19
20    -- détermine l'état suivant du SystemeRobot suivant son automate
21    procedure suivant is
22
23       ar : ActionRobot;
24       cc : ConfigCase;
25    begin
26       cc := config_case_devant;
27       faire_transition(a, cc, ar);
28       case ar is
29          when AVANCE => avancer;
30          when DROITE => tourner_a_droite;
31          when GAUCHE => tourner_a_gauche;
32       end case;
33       nb_t := nb_t+1;
34    end suivant;
35
36    [...]
37
38 end systeme_robot;
```

Annexe 5 : Programme test_performances

```
1  -- programme pour évaluer un robot automate dans une suite de terrains
2  -- syntaxe : test_performance fichier_automate fichier_terrains fichier_resultat
3  --  avec : fichier_automate = fichier contenant la description d'un automate
4  --         fichier_terrains = fichier contenant une suite de terrains
5  --         fichier_resultat = fichier dans lequel sont ecrits les statistiques
6  --          du robot. fichier texte avec le format suivant
7  --             nb_de_terrains
8  --             sur chaque ligne, le nombre de transitions utilisé par le robot
9  --               pour sortir du terrain ou -1 si le robot n'est pas sorti
10
11 with ada.text_io, ada.integer_text_io, ada.float_text_io, ada.command_line;
12 with systeme_robot;
13 use ada.text_io, ada.integer_text_io, ada.float_text_io, ada.command_line;
14
15 procedure test_performance is
16
17    procedure process is
18
19       type EtatRobot is (Marche,Bloque,Sorti);
20
21       fT : file_type; -- fichier des terrains
22       fR : file_type; -- fichier des résultats
23       nb_terrains : positive;
24       boucle : boolean := true;
25       er : EtatRobot;
26
27    begin
28       -- ouverture du fichier des terrains
29       open(fT, in_file, argument(2));
30
31       -- ouverture du fichier des resultats
32       create(fR, out_file, argument(3));
33
34       -- lecture du nombre de terrains
35       get(fT, nb_terrains);
36       put(fR, nb_terrains); new_line(fR);
37
```

```
38        for i in 1..nb_terrains loop
39
40           -- initialiser le SystemeRobot avec l'automate et le terrain
41           systeme_robot.init(fA, argument(2));
42
43           -- le SystemeRobot est en état de marche
44           er := Marche;
45
46           -- boucle sur les transitions du robot
47           while er=Marche loop
48
49              -- effectuer une transition
50              systeme_robot.suivant;
51
52              -- verifier si le robot est sorti ou
53              --  si le nb max de transitions est atteint
54              if systeme_robot.est_sorti then
55                 boucle := false;
56                 er := Sorti;
57              elsif systeme_robot.max_transitions_atteint then
58                 boucle := false;
59                 er := Bloque;
60              end if;
61
62           end loop;
63
64           -- ecrire le resultat
65           if er=Sorti then
66              put(fR, systeme_robot.nb_transitions,width=>1);
67           else
68              put(fR, "-1");
69           end if;
70           new_line(fR);
71
72        end loop;
73
74        -- fermeture des fichiers
75        close(fR);
76        close(fT);
77
78     end process;
79
80  begin
81     if argument_count /= 3 then
82        put("syntaxe : " & command_name);
83        put_line(" fichier_automate fichier_terrains fichier_resultat");
84     else
85        process;
86     end if;
87  end test_performance;
```