Algorithmes, Programmation Impérative 1

EXAMEN septembre 2006

Durée : 2 heures — Calculatrices et Documents interdits

Exercice 1 : *Connaissance et compréhension du langage* PASCAL.[DURÉE : 20 MINUTES]**[4 pt]** On suppose, dans cette question, déclarées les procédures, fonctions et variables suivantes :

```
procedure p(const x : BOOLEAN; var S:INTEGER);
function  f(const s :BOOLEAN):INTEGER;

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

Q.1 [2 pt] Dans ce contexte, quelles sont les instructions invalides parmi celles ci-dessous ? Précisez pour quelles raisons !

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

Q.2 [2 pt] On suppose, dans cette question, déclarées la procédure et les variables suivantes :

```
procedure q(const x : INTEGER ;
            out   y : INTEGER ;
            var   z : INTEGER );
var t : INTEGER;
begin
    t:= x-z;
    z:=3*z;
    y:= z+t;
end {q};
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 ? (Dans chacune des sous-questions, on supposera que préalablement on a *{x:=1; y:=2; z:=3}*.)

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

Exercice 2 : *Tableau d'éléments ordonnés.*[DURÉE : 40 MINUTES]**[6 pt]**

```
const N       = ... // un entier naturel
type  ELEMENT = ... // un type pour lequel <= est défini
      INDICE  = 1..N
      TABLEAU = array[INDICE] of ELEMENT;
```

On donne d'autre part le code de la fonction `ind_max`

```
// retourne le plus grand indice i
// tel que  la tranche
// t[a..i] soit triée
function ind_max(const a : INDICE; const t : TABLEAU):INDICE;
var i : INDICE;
begin
   i:=succ(a);
```

```
   while (i<=high(t)) and (t[pred(i)]<=t[i] ) do inc(i);
   ind_max:=pred(i);
end {ind_max};
```

Q.1 [1 pt] En utilisant la fonction `ind_max`, codez en PASCAL un prédicat `est_trie` qui permet de tester si un tableau `t` est trié.

Q.2 [2 pt] Codez en PASCAL la fonction dont les spécifications sont les suivantes :

```
// retourne le nombre n tel
// il existe a0=0<a1<a2<...<an
// tel que pour tout i < n la tranche t[ai + 1..ai+1] soit triée
// et pour tout i tel que 0 < i < n on ait t[ai]>t[ai + 1]
function nombre_tranches_triees(const t: TABLEAU):CARDINAL;
```

Par exemple si le tableau t est :

i	1	2	3	4	5	6	7
t[i]	4	3	7	1	2	5	3

alors le nombre de tranches triées est 3. ces tranches sont | 4 | 3 | 7 |, | 1 | 2 | 5 |, et | 3 | On suppose implantées la procédure `fusion` vue en cours, et la fonction `ind_max`. On rappelle la spécification de la procédure `fusion`

```
fusion ( t , a , b , c )
Données : un tableau t, trois indices a, b, c
CU      : les tranches t [a .. b] et
          t [b+1 .. c] sont triées
But     : trier la tranche t[a .. c]
```

On donne le code de la procédure `b` :

```
procedure b(var t : TABLEAU);
var a,b,m:INDICE;
begin
  a:=low(t);
  m:=ind_max(a,t);
  while a<>low(t) and m<>high(t) do
  begin
    b:=ind_max(succ(m),t);
    fusion(t,a,m,b);
    a:=succ(b);
    if a >= high(t) then
    begin
      a:=low(t);
    end;
    m:=ind_max(a,t);
  end {while};
end {b};
```

Q.3 [2 pt] On considère le tableau `t`

i	1	2	3	4	5	6	7
t[i]	4	3	7	1	2	5	3

Détaillez l'exécution de l'instruction `b(t)`. On donnera à chaque étape du `tant que`,la valeur de `a`, la valeur de `m` et la valeur de `t`

Q.4 [1 pt] D'une manière générale, à quoi sert la procédure `b` ? Justifiez.

Exercice 3 : *Le jeu de l'Awalé*[DURÉE : 1 HEURE]**[10 pt]** Les variations étant innombrables[1], nous n'en détaillerons qu'une qui est relativement courante, appelée Abapa, utilisée dans les tournois et reconnue par la fédération internationale (World Oware Federation).

1. Comme les échecs, seulement deux joueurs peuvent s'affronter, mais contrairement aux échecs, les joueurs doivent jouer rapidement et le public peut (mais pas en tournoi) faire du bruit, chanter, discuter les coups.

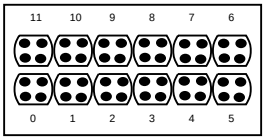

FIG. 1 – configuration initiale

2. Au départ, on répartit quarante-huit graines dans les douze trous à raison de quatre graines par trou.
3. Les joueurs sont l'un en face de l'autre, avec une rangée devant chaque joueur. Cette rangée sera son camp. On choisit un sens de rotation qui vaudra pour toute la partie (sens inverse des aiguilles d'une montre pour la règle décrite). On choisit également un joueur qui commencera la partie.
4. Un tour se joue de la façon suivante : le premier joueur prend toutes les graines d'un des trous de son camp puis il les égraine dans toutes les cases qui suivent sur sa rangée puis sur celle de son adversaire suivant le sens de rotation (une graine dans chaque trou après celui où il a récuperé les graines).

   Si sa dernière graine tombe dans le camp adverse et qu'il y a maintenant deux ou trois graines, le joueur récupère ces deux ou trois graines et les met de côté. Ensuite il regarde la case précédente : si elle est dans le camp adverse et contient deux ou trois graines, il récupère ces graines, et ainsi de suite jusqu'à ce qu'il arrive à son camp ou jusqu'à ce qu'il y ait un nombre de graines différent de deux ou trois.

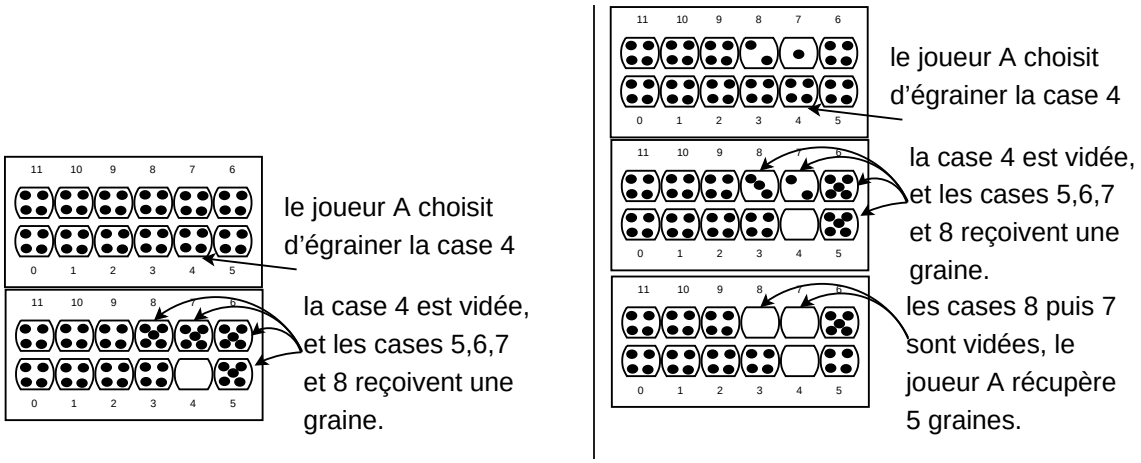

FIG. 2 – deux exemples pour illustrer la règle 4

5. Le but du jeu est d'avoir récupéré le plus de graines à la fin de la partie.
6. On ne saute pas de case lorsqu'on égraine sauf lorsqu'on a plus de douze graines, c'est-à-dire qu'on fait un tour complet : on ne remplit pas la case où l'on vient de prendre les graines.
7. Il faut nourrir l'adversaire, c'est-à-dire que, quand celui-ci n'a plus de graines, il faut absolument jouer un coup qui lui permette de rejouer ensuite. Si ce n'est pas possible, la partie s'arrête et le joueur qui allait jouer capture les graines restantes.
8. Si un coup devait prendre toutes les graines adverses, alors le coup peut être joué, mais aucune capture n'est faite : il ne faut pas affamer l'adversaire.
9. La partie s'arrête quand un des joueurs a capturé au moins 25 graines, soit plus de la moitié.

Ces règles sont plus simples qu'il n'y paraît. Elles sont rapidement assimilées par les débutants qui réalisent rapidement de très beaux coups, ce qui en fait un jeu très agréable pour tous et dans toutes les situations.

Le but des questions qui suivent est d'implanter en langage PASCAL une partie des éléments nécessaires pour réaliser une version informatique du jeu. Pour représenter le plateau, on va utiliser un tableau d'entiers naturels dont les cases seront numérotées par des indices allant de 0 à 11 comme sur les figures.

[1] les règles proviennent de `fr.wikipedia.org/wiki/Awélé`

Toute case dont l'indice est compris au sens large entre 0 et 5 appartient au joueur A. Toutes les autres cases appartiennent au joueur B

Q.1 [0.5 pt] Déclarez un type `INDICE` qui permet de représenter les entiers compris entre 0 et 11.

Q.2 [0.5 pt] Déclarez le type `PLATEAU`.

Q.3 [0.5 pt] Déclarez un type `JOUEUR` qui soit compatible avec la fonction `adversaire` suivante :

```
function adversaire(const j:JOUEUR):JOUEUR;
begin
  case j of
  A : adversaire := B;
  B : adversaire := A;
  end {case};
end{adversaire};
```

Par la suite, on supposera que les déclarations suivantes ont été faites :

```
type SCORE = array [JOUEUR] of CARDINAL;
type ETAT_DU_JEU = record
  p : PLATEAU;
  s : SCORE;
end {record};
```

Q.4 [0.5 pt] Réaliser une fonction `case_suivante`. Remarque la case suivante de 11 doit être égale à 0. Par la suite, on supposera écrite la fonction `case_precedente`.

Q.5 [0.5 pt] Un joueur est victorieux lorsqu'il a récupéré au moins 25 graines. Codez en PASCAL une fonction qui permet de tester si un joueur est victorieux.

Q.6 [0.5 pt] Codez en PASCAL la fonction `camp` dont la spécification est la suivante :

```
// exemple de comportement attendu
// camp(1) vaut A
// camp(11) vaut B
function camp(const i: INDICE):JOUEUR;
```

qui permet de déterminer le joueur qui possède la case dont le numéro est i.

Lorsque la partie commence toutes les cases contiennent 4 graines, et le score de chacun des joueurs est nul.

Q.7 [1 pt] Codez en PASCAL la procédure `initialiser_jeu` dont l'entête est :

```
procedure initialiser_jeu(out e : ETAT_DU_JEU);
```

Dans un premier temps on ne tient pas compte de la règle 6.

Q.8 [1.5 pt] Codez en PASCAL la procédure `egrainer` dont l'entête est :

```
procedure egrainer(var   p : PLATEAU;
                   const i : INDICE;
                   out   f : INDICE);
```

où `p` désigne le plateau à modifier suivant le début de la règle 4 (on ne récupere pas les graines cela sera fait dans la question 10), `i` désigne l'indice de la case à égrainer, et `f` désignera après l'appel le numéro de la case où on a fini d'égrainer.

Q.9 [1 pt] Expliquez comment modifier la procédure précédente afin de prendre en compte la règle 6 ?

Q.10 [1 pt] Codez en PASCAL la procédure `recuperer` sans se préoccuper de la règle 8. L'entête de cette procédure est :

```
procedure recuperer(var   e : ETAT_DU_JEU;
                    const j : JOUEUR;
                    const f : INDICE);
```

où `e` désigne l'état du jeu qui va être modifié en respectant la seconde partie de la règle 4 et la règle 8 sachant que `j` est le joueur qui a joué le coup, et que `f` désigne le numéro de la case où le joueur a fini d'égrainer.

Q.11 [1 pt] Codez en PASCAL une fonction `est_affame`, dont le résultat est un booléen, et qui permet de tester si le joueur passé en paramètre est affamé.

Q.12 [1.5 pt] Codez en PASCAL la procédure `jouer` qui permet d'égrainer la case i, puis de récuperer les graines, en respectant la règle 8