



1er cycle 2ème année
Durée 2h

Initiation à l'algorithmique

Notes de Travaux Dirigés et Travaux Pratiques autorisées.

Le devoir (4 pages) comporte 4 exercices et 1 problème indépendants.

On attachera beaucoup d'importance à la clarté de la rédaction et aux commentaires.

Les durées sont indicatives.

Exercice 1 (10 mn)

Soit T un tableau d'entiers strictement positifs ~~trié par ordre croissant~~. On désire trouver le plus grand indice i tel que le produit $\prod_{k=1}^i T[k]$ soit strictement inférieur à m , m étant un nombre positif donné.

On considère les définitions :

```
const n=6;
```

```
type t_tab = array[1..n] of integer;
```

Ecrire une fonction indice d'en-tête :

```
function indice (const T:t_tab ; const m:integer) : integer ;
```

retournant un tel indice

Exemple : $T = (2, 5, 3, 4, 5, 2)$

indice(T,11) délivre 2

indice(T,1) délivre 0

indice(T,4000) délivre 6

Exercice 2 (10 mn)

On se place dans le cadre du TP7 - Partie 2 :

Ecrire une fonction booléenne qui contrôle qu'une solution "sol" vérifie une équation "eq" ;
"nbinc" est le nombre d'inconnues :

```
function controle(const sol:t_solution ;
```

```
const eq:t_equation ; const nbinc:integer) boolean ;
```

Exercice 3 (15 mn)

On se place dans le cadre du TP5 - Partie 2:

Nous choisissons dans cet exercice de représenter un polynôme à coefficients réels dans une structure. Soient les définitions suivantes :

```
const n = 50;
type s_poly = record
    degre : integer ; { degre du polynome}
    coefficients : array [0..n] of real;
end;
```

Tous les coefficients sont représentés et rangés selon les degrés croissants.

exemple : le polynôme $5x^7 - 3x^4 + 2x - 1$ sera ainsi stocké :

degre = 7 et coefficients = (-1., 2., 0, 0, -3., 0, 0, 5.)

Ecrire une procédure *transfo* d'en-tête

```
procedure transfo( const poly: s_poly; var c:t_coeff; var d:t_degr);
```

qui, à partir de cette nouvelle représentation, crée les tableaux c et d tels qu'ils ont été vus dans la partie 2 du TP5.

Exercice 4 (25 mn)

Dans cet exercice, on commentera très clairement l'algorithme utilisé.

Soient les définitions suivantes :

```
const n = 4; {nombre de chiffres des nombres}
type t_tab = array [1..n] of integer;
```

Un nombre est un anagramme d'un autre si il est constitué des mêmes chiffres (dans un ordre quelconque).

On suppose que les 2 nombres sont représentés par leurs n chiffres mémorisés dans un tableau.

Ecrire une fonction *anag* d'en-tête

```
function anag(const t1,t2:t_tab): boolean ;
qui retourne vrai si t1 est anagramme de t2, faux sinon.
```

Exemple : $t1 = (1, 2, 6, 1)$ et $t2 = (2, 1, 1, 6)$ $anag(t1, t2) = true$
 $t3 = (2, 1, 2, 6)$ $anag(t1, t3) = false$

PROBLEME (1h)

On s'intéresse à la modélisation d'un jeu de 52 cartes. Une carte est définie par une couleur : trèfle, carreau, coeur ou pique. Il y a 13 cartes de chaque couleur dans un jeu : 2, 3, 4, 5, 6, 7, 8, 9, 10, valet, dame, roi, as.

On définit les constantes suivantes en Pascal :

```
const trefle=1, carreau=2, coeur=3, pique=4;  
const valet=11, dame=12, roi = 13, as = 14;
```

Les cartes de 2 à 10 sont codées par leur numéro.

On en déduit le type `t_carte` :

```
type t_carte = record  
    couleur : integer;  
    hauteur : integer  
end;
```

Ainsi le roi de pique sera représenté par le doublet (couleur:4; hauteur:13), le 9 de carreau par (couleur:2; hauteur:9), etc...

Pour définir un paquet de cartes, on adopte le type suivant :

```
type t_paquet = record  
    nbcartes : integer ; (le nombre de cartes du paquet)  
    cartes : array[1..52] of t_carte (au maximum 52)  
end;
```

Le problème ne consiste pas à modéliser un jeu de cartes connu mais à écrire un certain nombre de procédures ou fonctions répondant à des spécifications précises .

Question 1

Ecrire une procédure d'en-tête *fairePaquet* qui crée un paquet d'une seule carte:

```
procedure fairePaquet(const carte : t_carte ; var paquet : t_paquet);
```

Question 2

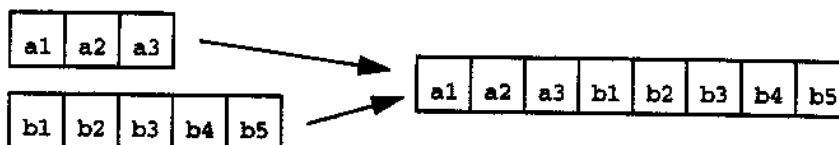
Ecrire la fonction *bienTrie* qui vérifie que dans un paquet de cartes il n'y a jamais 2 cartes consécutives de même hauteur.

```
function bienTrie(const paquet : t_paquet): boolean ;
```

Question 3

On veut fabriquer un paquet de cartes à partir de 2 paquets en mettant le 2ème paquet au bout du 1er paquet. Ecrire la procédure :

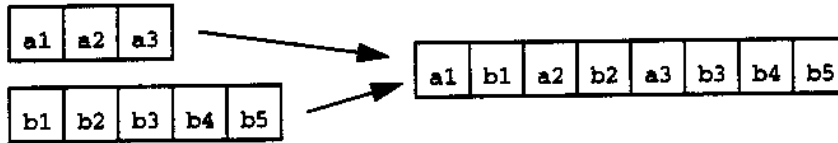
```
procedure mettreAuBout(var nvPaquet : t_paquet ;  
    const paquetA, paquetB : t_paquet);
```



Question 4

On veut mélanger 2 paquets de cartes en prenant alternativement une carte dans chaque paquet. Si un paquet contient plus de cartes que l'autre, les cartes restantes sont mises à la fin du paquet. Ecrire la procédure :

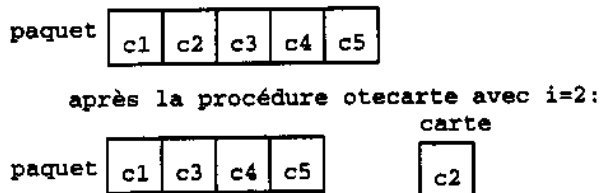
```
procedure fusionne (var nvPaquet : t_paquet ; const paquetA, paquetB : t_paquet);
```



Question 5

Ecrire une procédure qui extrait la ième carte d'un paquet et modifie ce paquet sans changer l'ordre des cartes :

```
procedure oteCarte(var paquet : t_paquet; var carte : t_carte ;  
const i : integer);
```



Question 6

Ecrire une procédure *oteCouleur* qui fabrique un nouveau paquet contenant toutes les cartes du paquet d'origine d'une couleur donnée, par exemple tous les trèfles. Le paquet d'origine est modifié en conséquence :

```
procedure otecouleur (var paquet : t_paquet ; var paquetCouleur: t_paquet;  
const couleur : integer)
```

Question 7

Ecrire une fonction `estRange` qui teste si un paquet de cartes est rangé par couleur : trèfle, carreau coeur et pique. L'ordre d'apparition des couleurs est quelconque, il peut ne pas y avoir de carte d'une couleur donnée. On expliquera par un commentaire la technique utilisée.

```
function estRange (const paquet : t_paquet): boolean ;
```

Ca	Ca	Tr	Co	Co	Co	Co	Co	Pi
----	----	----	----	----	----	----	----	----

 est un paquet rangé

Co	Co	Co	Tr	Tr
----	----	----	----	----

 est un paquet rangé

Co	Co	Pi	Pi	Co	Co	Ca
----	----	----	----	----	----	----

 n'est pas un paquet rangé