Cahier de TD n° 2
corrigé

THEME 1 : Tableaux : définition de tableaux, taille maximum, taille utile, affichage, saisie.

Vrai / faux :

répondez aux questions suivantes par vrai ou faux

- Un tableau a toujours une taille maximum : VRAI
- Un tableau a toujours une taille utile : VRAI et FAUX : il doit toujours en avoir une mais cette taille utile n'est pas gérée par le langage, mais par le programmeur.
- Un tableau se manipule comme une variable 'classique' : FAUX
- Lors de la définition d'un tableau, on doit mettre une valeur numérique entre les crochets : VRAI
- On peut utiliser une expression entière comme indice pour manipuler les variables contenues dans un tableau : VRAI
- Les indices des variables d'un tableau de taille maximum *N* vont de 1 à *N :* FAUX
- L'ordinateur vérifie systématiquement que les indices sont valides lors des manipulations des variables d'un tableau : FAUX
- Un indice peut être un entier, un réel ou un caractère : FAUX
- Un tableau ne peut contenir que des variables entières ou caractères : FAUX
- On peut initialiser les variables d'un tableau lors de sa définition : VRAI

Définition et syntaxe :

Parmi les définitions de tableaux suivantes, indiquez celles qui sont incorrectes et pourquoi. Si c'est possible, indiquez la définition qu'il aurait fallu écrire.

- `entier tablo={1,2,3,4,5}; : incorrecte, pas de []`
- `réel t[3]; : correct`
- `tableau x[124]; incorrect, le nom du type des variables n'est pas reel ou entier ou caractere`
- `réel tab_reel[]; : incorrect, pas de valeur entre les []`

- `entier taille; : correct`
  `taille ← 5; : correct`
  `caractère tabc[taille]={'a', 'b', 'c', 'd', 'e'}; : incorrect, la taille maximale doit etre une constante`

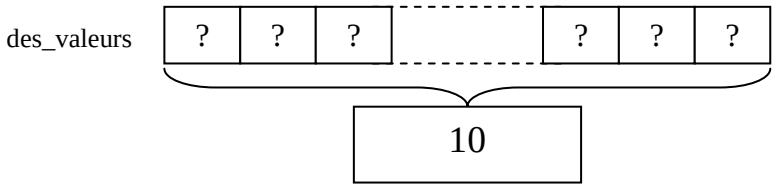
- `entier vals[4]={1,2,3,4}; : correct`
- `entier vals2[18000]; : correct`

saisie / affichage de tableaux

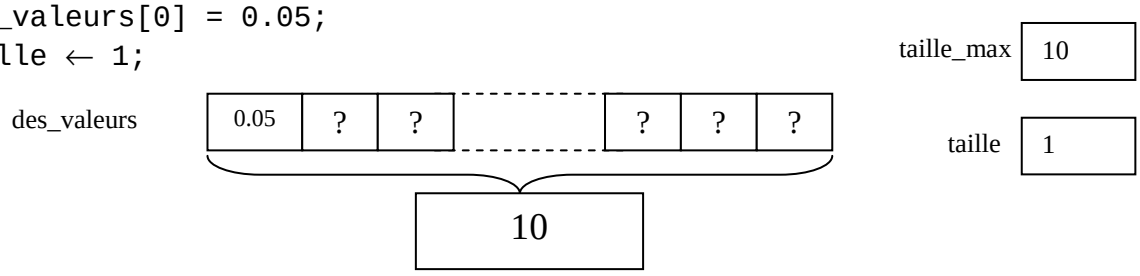
rappel : lorsque l'on indique que l'on saisit ou que l'on affiche un tableau, il faut comprendre : on saisit des valeurs à ranger dans les variables d'un tableau, on affiche les valeurs des variables contenues dans un tableau. De plus, on utilise quasiment systématiquement des boucles lorsque l'on traite un tableau.

Que fait le programme suivant ? Remplissez les schémas au fur et à mesure, et complétez les instructions sur fond gris à la fin du programme.

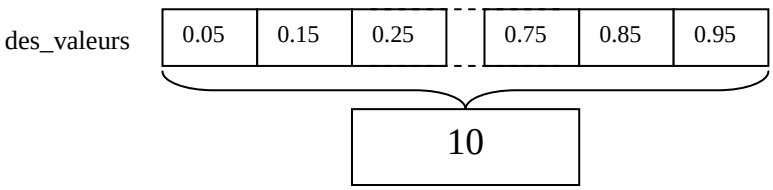
```
programme que_fais_je
reel des_valeurs[10];
entier taille_max, taille;
entier compt;
```




```
taille_max ← 10;
des_valeurs[0] = 0.05;
taille ← 1;
```




```
pour compt de 1 à taille_max-1 faire
{
      des_valeurs[compt] ← des_valeurs[compt-1] + 0.1;
}
```




```
// affichage de toutes les valeurs du tableau

pour compt de 0 à taille faire
```

```
{
     afficher("la variable ", compt ," du tableau vaut :
",des_valeurs[compt], "\n");
}
```

- Ecrire un programme qui fait la saisie de valeurs de type caractère et qui les range dans un tableau contenant au maximum 50 caractères. L'utilisateur peut arrêter la saisie en saisissant le caractère '&'. Avant de commencer le programme, écrivez bien quelles sont les conditions auxquelles on arrête (ou on continue) la boucle dans laquelle se fait la saisie.

```
programme saisie_caracs

caractere t_c[50];
entier tai_ut;
caractere c_saisi;

tai_ut ← 0;
faire
{
     afficher("entrez un caractere :");
     saisir(c_saisi);
     t_c[tai_ut] ← c_saisi;
     tai_ut ← tai_ut+1;
}
tant que ((tai_ut < 50) ET (c_saisi ≠'&'));

tai_ut ← tai_ut-1;   // car on a stocké le caractere '&' avant de
                     // tester sa présence pour la fin de saisie
```

- Ecrire un programme qui fait la saisie de valeurs de type réel, qui les range dans un tableau et qui effectue les calculs suivants :

Reprendre l'exemple précédent pour la saisie, avec la possibilité d'un questionnement de type oui/non fait à l'utilisateur pour continuer ou arrêter la saisie de valeurs. On aura donc le tableau des valeurs, nommé par exemple tab_v, de taille maximale 100, et dont la taille utile est stockée dans la variable t_ut;

```
reel tab_v[100];
entier t_ut, cpt;
reel som_abs, harmo, geom;
```

a) s'il y a plus de 3 valeurs : calcul de la somme des valeurs absolues

```
si (t_ut > 3) alors
{
     som_abs ← 0;
```

```
pour cpt de 0 à t_ut-1 faire
     {
          si (tab_v[cpt] >0) alors
          {
               som_abs ← som_abs + tab_v[cpt];
          }
          sinon
          {
               som_abs ← som_abs – tab_v[cpt];
          }
     }
}
afficher("somme des valeurs absolues :",som_abs);
```

b) s'il y a exactement 2 valeurs : calcul de leur moyenne harmonique $\overline{h}$:

$$\frac{2}{\overline{h}} = \frac{1}{x_1} + \frac{1}{x_2}$$

où x_1 et x_2 sont les deux valeurs du tableau.

```
si (t_ut = 2) alors
{
     harmo ← 2.0 * (tab_v[0]*tab_v[1])/ (tab_v[0]+tab_v[1]);
}
afficher("la moyenne harmonique de :", tab_v[0], " et ", tab_v[1],
"=", harmo);
```

c) s'il y a exactement 4 valeurs : calcul de leur moyenne géométrique $\overline{g}$:

$$\overline{g} = \sqrt[4]{x_1.x_2.x_3.x_4}$$

où x1,…x4 sont les quatre valeurs du tableau.

Pour le calcul de la racine quatrième : calculer la racine quatrième revient à élever le nombre à la puissance ¼, soit 0,25. On utilisera une instruction de calcul : *puissance(x,y)* qui calcule le nombre *x* (réel) élevé à la puissance *y* (nombre réel), et qui donne un résultat réel.

```
Si (t_ut = 4) alors
{
     geom=puissance(tab_v[0]    *    tab_v[1]    *    tab_v[2]    *
tab_v[3],1.0/4.0);
}
afficher("la moyenne géometrique de:");

pour cpt de 0 à t_ut-1 faire
{
     afficher(tab_v[cpt] ," ,");
}
afficher(" vaut :",geom);
```

THEME 2 : tableaux : utilisation et algorithmes classiques

Recherche de la deuxième plus grande valeur d'un tableau

On considère un tableau (peu importe le type des variables qu'il contient, c'est à vous de le choisir) comportant au maximum *N* variables et dont *M* variables sont utilisées. En vous inspirant de l'algorithme de recherche de la valeur maximum, écrivez un programme qui recherche la deuxième plus grande valeur du tableau (celle qui se rapproche le plus du maximum sans l'atteindre).

Il faut a) rechercher le maximum, l'échanger avec la valeur située dans la variable d'indice 0 du tableau, puis recommencer une recherche du maximum à partir de l'indice 1. Ce deuxième maximum est alors la valeur demandée. On considère que les valeurs ne se trouvent qu'en un exemplaire dans le tableau.

Suppression de doublons

Soit un tableau contenant des variables entières. Ecrivez un programme à qui l'on fournit une valeur entière et qui supprime tous les exemplaires sauf un de cette valeur si elle se trouve en plusieurs exemplaires dans le tableau.

```
entier tab[50]←{1,2,3,4,6,4,3,2,9,5,8,7,4,5,2};
entier util;  // taille utile du tableau
entier cpt, cpt2;  // compteurs pour parcourir tableau
entier nb_renc;  // nb de fois où l'on rencontre la valeur
entier val_saisi;  // valeur saisie pour élimination des doublons

util ← 15;

pour cpt de 0 a util-1 faire
{
  afficher(tab[cpt]);
}
afficher("\n");

afficher("entrez la valeur a traiter:");
saisir(val_saisi);

nb_renc ← 0; // pas encore rencontree

pour cpt de 0 à util faire
{
  // on teste si la valeur voulue est dans la
  // variable d'indice cpt du tableau

  si (val_saisi = tab[cpt]) // si oui
  {
```

```
        nb_renc ← nb_renc+1; // on indique qu'on la rencontre

        si (nb_renc > 1) // il s'agit donc d'un doublon
        {
                pour cpt2 de cpt à util-2 faire
                {
                   tab[cpt2]←tab[cpt2+1];
                }

                util←util-1; // on a supprimé une valeur du tableau
        }
    }
  }

  afficher("la valeur ", val_saisi, " a ete rencontree ",nb_renc
," fois\n");

 // affichage du tableau débarrassé de ses doublons

  pour cpt de 0 à util-1 faire
  {
    afficher(tab[cpt]);
  }
```

A partir du programme précédent, écrire un programme qui supprime tous les doublons (exemplaires multiples d'une valeur) d'un tableau.

Au lieu de saisir une valeur, on procède à l'élimination des doublons en sélectionnant une par une les valeurs situées dans le tableau.

```
  entier tab[50]←{1,2,3,4,6,4,3,2,9,5,8,7,4,5,2};
  entier util;  // taille utile du tableau
  entier cpt, cpt2;  // compteurs pour parcourir tableau
  entier nb_renc;  // nb de fois où l'on rencontre la valeur
  entier val_saisi;  // valeur saisie pour élimination des doublons

entier ind_depart; // indice pour parcourir et selectionner les
                   // valeurs a supprimer

  util ← 15;

  pour cpt de 0 a util-1 faire
  {
    afficher(tab[cpt]);
  }
  afficher("\n");

pour ind_depart de 0 à util-1 faire
{
     val_saisi ← tab[ind_depart];

     nb_renc ← 0; // pas encore rencontree
```

```
      pour cpt de 0 à util faire
      {
      // on teste si la valeur voulue est dans la
      // variable d'indice cpt du tableau

      si (val_saisi = tab[cpt]) // si oui
     {
         nb_renc ← nb_renc+1; // on indique qu'on la rencontre

         si (nb_renc > 1) // il s'agit donc d'un doublon
         {
                 pour cpt2 de cpt à util-2 faire
                 {
                    tab[cpt2]←tab[cpt2+1];
                 }

                 util←util-1; // on a supprimé une valeur du tableau
         }
     }
   }
}
  afficher("la valeur ", val_saisi, " a ete rencontree ",nb_renc
," fois\n");
```

Retour sur la recherche précédente :

Imaginez un algorithme qui, en utilisant la suppression de doublons et éventuellement un tri, permet de faire rapidement la recherche de la deuxième plus grande valeur d'un tableau.

Réponse simple : 1) supprimer les doublons

2) faire un tri (plus rapide s'il n'y a pas de doublons) par ordre décroissant

3) la deuxième valeur la plus grande se trouve forcément dans la variable d'indice 1 du tableau !

Utilisation des tableaux pour représenter des polynômes

Un polynôme $P=a_n.x^n+a_{n-1}.x^{n-1}+\ldots+a_2.x^2+a_1.x+a_0$ est en fait complètement donné par la liste $(a_0,a_1,\ldots a_{n-1},a_n)$.

Ecrire un programme qui saisit le degré d'un polynôme *P*, ses coefficients, et qui l'affiche sous la forme : $P = a_n.x^n+a_{n-1}.x^{n-1}+\ldots+a_2.x^2+a_1.x+a_0$

```
programme polynomes
```

```
reel poly[20];
entier degre, compt;

afficher("entrez le degre du polynome:");
saisir(degre);

pour compt de 0 a degre faire
{
     afficher("entrez le coefficient de degre ",compt," :");
     saisir(poly[compt]);
}

// affichage du polynome p[x]

  pour compt de degre a 1 faire
  {
  afficher(poly[compt],".x^",compt," + ");
  }

     // attention a l'affichage du dernier !

  afficher(poly[0],"\n");
```

A la suite de ce programme, écrire une partie de programme qui saisit une valeur de *x* et calcule la valeur du polynôme *P* en *x*, valeur que l'on note *P*(*x*).

Pour améliorer ce calcul, on propose d'utiliser la méthode dite de Hörner, qui se base sur l'égalité suivante (il s'agit d'une ré écriture) :

$a_n.x^n+a_{n-1}.x^{n-1}+\ldots+a_2.x^2+a_1.x+a_0 = a_0+ x\ (\ a_1 + x\ (\ldots(\ a_{n-2}+ x\ (\ a_{n-1}+ x\ a_n\ ))\ldots))$

```
     // saisie de la valeur de x et calcul de la valeur du polynome
// en x

  afficher("entrez la valeur de x:");
  saisir(x);

  val_poly ← poly[degre];

  pour compt de degre-1 à 0 faire
  {
  val_poly ← val_poly*x + poly[compt];
  }

  afficher("en ",x," le polynome vaut : ",val_poly,"\n");
```

Quelle relation existe-t-il entre le degré du polynôme stocké et la taille utile du tableau ? **la taille utile est égale au degré du polynome+1.**

On considère maintenant 2 polynômes *P* et *Q*, dont les coefficients sont déjà saisis et dont les degrés respectifs sont donnés par des variables *p* et *q*. Selon les valeurs des

coefficients, quel sera le degré de $P+Q$? Ecrire un programme qui calcule la somme des polynômes $P(x)+Q(x)$, ainsi que le degré de ce polynôme.

On aurait tendance à écrire simplement que d°(P+Q) = max(p,q), mais certains coefficients peuvent s'annuler !

L'égalité précédente n'est vraie que si p ≠ q, dans le cas contraire, on se base sur la valeur des coefficients pour trouver le degré. A partir de p, on cherche le premier coefficient non nul du polynôme somme.

```
programme somme_poly

reel p[50], q[50]; // pour stocker les coeffs de p et q
entier p,q,compt;     // les degres et un compteur
entier deg_max, deg_min; // le plus grand entre p et q, et le plus
                         // petit entre p et q

reel sommepoly[50]; // coeffs de la somme
entier s;           //degre de la somme

// saisie des degres et des coeffs des polynomes p et q

// traitement de la somme : determiner le plus grand entre p et q
// pour savoir quelles valeurs ranger dans s
si (p>q) alors
{
     degmax ← p;
     degmin ← q;
}
sinon
{
     degmax ← q;
     degmin ← p;
}

// premiere partie : remplissage de la partie de la somme fournie
// par le polynôme de plus haut degre, car on n'a pas stocke de 0
// dans les coefficients de meme degre de l'autre polynôme.

pour compt de degmax a degmin+1 faire
{
     si (degmax=p) alors  // ce sont les coeffs de p qu'on doit
                          // recopier
     {
          sommepoly[compt] ← p[compt];
     }
     sinon
     {
          sommepoly[compt] ← q[compt];
     }
}
// deuxieme partie : somme des coeffs pour les degres inferieurs
```

```
pour compt de degmin à 0 faire
{
      sommepoly[compt] ← p[compt]+q[compt];
}

// calcul du degre de la somme

si (p ≠ q) alors
{
      s ← degmax; // on a directement le degre max
}
sinon // il faut calculer ce degre
{
      s ← degmax;

      tant que ((s>=0) ET (sommepoly[s] =0.0))
      {
            s ← s-1;
      }
}

afficher("le polynôme P+Q a pour degre ",s);

// affichage des coefficients du polynôme
```

Ecrire un programme qui, à partir d'un polynôme *P*, calcule son polynôme dérivé *P'*.

A partir du fait que la dérivée de $a.x^n = n.a.x^{n-1}$, on recalcule les coefficients en les décalant vers la gauche.

```
programme deriv_poly

reel poly[50], deriv[50];
entier deg_p, deg_d;
entier compt;

// saisie du degre et des coefficients de p

deg_d ← deg_p –1;

si (deg_d>=0) alors
{
      pour compt de 0 deg_d faire
      {
            deriv[compt] ← (compt+1)*poly[compt];
      }
}

// afficher le polynôme résultat
```

Ecrire un programme qui, à partir d'un polynôme *P*, calcule son polynôme intégral Π (c'est à dire le polynôme Π tel que $\Pi' = P$), sachant que $\Pi(0) = K$, *K* étant une valeur réelle arbitraire.

On constate que le monôme intégral de $a.x^n$ est $(a/n+1).x^{n+1}$. De plus, le coefficient de degré 0 du polynôme Π est égal à K.

```
programme integre

reel poly[50], integ[50];
entier deg_p, deg_int, compt;
reel k;

// saisie du degre et des coefficients de P

afficher("valeur de PI(0) ? :");
saisir(k);

deg_int ← deg_p+1;

integ[0] ← k;

pour compt de 1 a deg_int faire
{
     integ[compt] ← poly[compt-1] / compt;
}

// affichage du polynôme integ
```

Tris et recherches

Tri du joueur de cartes :

On cherche à trier les valeurs d'un tableau en utilisant le tri dit du joueur de carte : on procède par tri de sous-tableaux successifs en partant de la gauche.

Exemple :

4	0	2	7	1	5	?

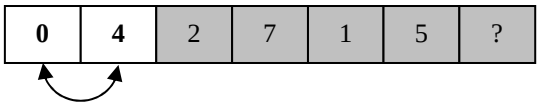
Première étape : on considère le sous tableau comportant 1 variable : on considère que ce sous tableau est trié : la première valeur est bien classée.

4	0	2	7	1	5	?

Deuxième étape : on considère le sous tableau comportant 2 variables :

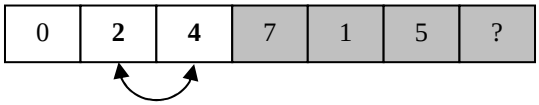
4	0	2	7	1	5	?

On échange la valeur de droite avec les valeurs situées à sa gauche jusqu'à ce qu'elle soit bien classée. Donc, les deux premières valeurs sont bien classées après ces échanges



Troisième étape : on considère le sous tableau comportant 3 variables :

On échange la valeur de droite avec les valeurs situées à sa gauche jusqu'à ce qu'elle soit bien classée. Donc, les trois premières valeurs sont bien classées.




```
programme tri_joueur de cartes
entier tab[100];
entier util, cpt1, cpt2;
entier temp;

// saisie des valeurs du tableau et calcul de la taille utile

pour cpt1 de 1 à util-1 faire
{
      cpt2 ← cpt1;
      tant que ((cpt2>0) ET (tab[cpt2]<tab[cpt2-1]))
      {
            temp ← tab[cpt2];
            tab[cpt2] ← tab[cpt2-1];
            tab[cpt2-1] ← temp;
            cpt2 ← cpt2-1;
      }
}
```

recherche par dichotomie : programmation

Nous avons abordé en cours la méthode de recherche par dichotomie dans un tableau dont les valeurs sont triées. Ecrire le programme de recherche par dichotomie dans un tableau dont les valeurs sont triées par ordre croissant. N'oubliez pas de vous aider à l'aide de schémas.

```
Programme rech_dicho
```

```
entier tab_v[512]; // tableau des valeurs

entier taille;    // taille utile du tableau
entier init;      // pour stocker les valeurs à ranger dans le
                  // tableau
entier debut, fin, milieu, indice; // indices pouir la recherche
entier valeur;    // valeur a rechercher

taille ← 512;
init ← 0;

// initialisation du tableau avec des valeurs croissantes

pour indice de 0 à taille-1 faire
{
     tab_v[indice] ← init;
     init=init + hasard(3);
}

// saisie de la valeur à rechercher

afficher("Entrez la valeur a rechercher :");
saisir(valeur);

// initialisation des indices pour la recherche

debut ← 0;
fin ← taille;
milieu ← (debut+fin)/2;

tant que ((debut < fin) ET (valeur ≠ tab_v[milieu]))
    {
        si (valeur > tab_v[milieu]) alors
        {
                debut ← milieu+1;
        }
        sinon
        {
                fin ← milieu-1;
        }

        milieu ← (debut+fin)/2;
    }

    si (valeur = tab_v[milieu]) alors
    {
        afficher("valeur ", valeur," trouvee a l'indice ", milieu, "
: ", tab_v[milieu]);
    }
    sinon
    {
        afficher("valeur non trouvee\n");
    }
```

Fusion de tableaux triés

On dispose de deux tableaux triés, on désire réunir les valeurs de ces deux tableaux dans un troisième tableau qui devra lui aussi être trié. On veut éviter de recopier les valeurs du premier tableau, puis à la suite les valeurs du deuxième tableau puis de trier le tout. Proposez un algorithme qui insère les valeurs dans la troisième tableau directement à leur bonne place, et écrivez le programme correspondant.

```
entier tab1[20] ← {1,3,5,9,12,16,20};
entier tab2[20] ← {0,2,3,4,8};
entier tab3[40];

entier tai1, tai2, tai3;
entier i1, i2, i3;

tai1 ← 7;
tai2 ← 5;

// affichage de tab1 et tab2

tai3 ← tai1+ tai2;

i1 ← i2 ← i3 ← 0;

tant que ((i1 < tai1) ET (i2 < tai2))
{
    si (tab1[i1] < tab2[i2]) alors
    {
        tab3[i3] ← tab1[i1];
        i1 ← i1+1;
    }
    sinon
    {
        tab3[i3] ← tab2[i2];
        i2 ← i2+1;
    }

    i3 ← i3+1;
}

tant que (i1 < tai1)
{
    tab3[i3] ← tab1[i1];
    i1 ← i1+1;
    i3 ← i3+1;
}

tant que (i2 < tai2)
{
    tab3[i3] ← tab2[i2];
    i2 ← i2+1;
```

```
    i3 ← i3+1;
}

afficher("tableau fusionne : ");

pour i3 de 0 à tai3-1 faire
{
    afficher(" ",tab3[i3]);
}
afficher("\n");
```

THEME 3 : tableaux de caractères : travailler avec du texte

Message personnalisé

Ecrire un programme qui saisit un nom d'utilisateur (sous la forme d'un texte) et qui répond par un message de bienvenue comportant le nom de l'utilisateur saisi précédemment. Le nom de l'utilisateur doit se trouver au milieu du message de bienvenue.

```
programme personnalise

caractere nom_ut[50];
caractere debut_mess[50] ← "Bonjour, ";
caractere fin_mess[50] ← ". Bienvenue sur notre systeme.\n";
caractere mess_complet[150];

afficher("entrez votre nom :");
lire(nom_ut);

copier(mess_complet, debut_mess);
concat(mess_complet, nom_ut);
concat(mess_complet, fin_mess);

ecrire(mess_complet);
```

Noms de fichiers

Lorsque l'on écrit des programmes utilisant des fichiers, on accède à ces fichiers en précisant leur nom sous la forme de texte, donc de tableaux de caractères. En général, l'utilisateur entre un nom de fichier court (ou relatif par rapport au répertoire courant), mais l'ordinateur a besoin de connaître le nom long ou absolu du fichier.

Ecrire un programme qui saisit un nom de disque dur (sous la forme d'une lettre pour désigner le disque), le nom d'un répertoire où sont stockés des fichiers de l'utilisateur (ce peut être un nom comportant plusieurs répertoires si on a besoin de parcourir plusieurs niveaux dans l'arborescence), et le nom d'un fichier (celui que veut manipuler l'utilisateur). Le programme doit afficher le nom absolu (nom long) du fichier.

```
programme nom_long_fic

caractere disque[2], chemin[100], fichier[20];
caractere nom_long[150];

afficher("entrez le nom de disque :");
lire(disque);
copier(nom_long, disque);

afficher("entrez le chemin :");
lire(chemin);
concat(nom_long, chemin);
```

```
afficher("entrez le nom du fichier :");
lire(fichier);
concat(nom_long, fichier);
afficher("le nom de votre fichier est :");
ecrire(nom_long);
```

Y a-t-il un nombre ?

Ecrire un programme qui est capable d'indiquer si un texte contient un nombre (une suite de chiffres) entier dans un premier temps, puis à virgule dans un deuxième temps. Un nombre est défini comme une suite de chiffres entouré par des espaces. Ainsi, dans le texte "Il loge dans la chambre 92A", on considère qu'il n'y a pas de nombre, mais il y en a dans le texte suivant : "La nuitée coûte 50 €.".

```
programme nombre_present

caractere texte[50]←"Il loge dans la chambre 92A, la nuitee coute 50
euros";

  entier compt;   // indice pour parcourir le texte
  entier nombre_pres; // indique si un nombre est present
  entier limite;  // taille utile du texte
  entier debut, chiffre; // debut : indique si on a trouve le
                        // debut d'un mot
                         // chiffre : indique si l'on a une
                        // suite de chiffres
  caractere c;           // caractere en cours de traitement

  // initialisation des variables

  limite ← longueur_texte(texte);
  nombre_pres ← 0;
  debut←0;
  chiffre←0;
  compt←0;

  // tant que l'on n'a pas trouve et qu'on n'est pas arrive a la fin
du texte

  tant que ((nombre_pres=0) ET (compt<limite))
  {
    // initialiser c

    c ← texte[compt];

    // cas a distinguer : un espace, un chiffre, ou autre chose

    si (c=' ') alors // espace
    {
        si (chiffre=1) alors // si l'on avait une suite de chiffres,
                             // alors on a un nombre
        {
```

```
            nombre_pres ← 1;
        }
        sinon    // sinon, cet espace est le debut d'un nouveau mot
        {
            debut ← 1;
        }
    }
    sinon si ((c>='0') ET (c<='9')) // cas d'un caractere numerique
    {
        si (debut=1) alors // si on etait au debut d'un mot
        {
            chiffre←1; // alors on est en train de traiter une suite
                       // de chiffres
        }
    }
    sinon // tout autre caractere
    {
        debut←0; // ce n'est pas le debut d'un mot
        chiffre←0;  // ce n'est pas une suite de chiffres
    }

    compt ← compt+1; // passage au caractere suivant
}

si (nombre_pres=1) alors
{
  afficher("il y a un nombre\n");
}
sinon
{
  afficher("il n'y a pas de nombre\n");
}
```

Correcteur de ponctuation

On cherche à écrire un programme qui est capable de corriger un texte pour lui appliquer les règles de ponctuation suivante :

De manière générale, un point est un point simple, un point d'exclamation ou un point d'interrogation.

Une phrase commence par une majuscule, toutes les autres lettres sont en minuscule (on suppose qu'il n'y a pas de nom propre dans le texte);

Toute virgule, point ou point virgule est précédé d'un et un seul espace;

Il n'y a qu'un espace entre deux mots;

Le début d'une phrase est repérée par un point;

Il n'y a pas d'espace avant un point, une virgule ou un point-virgule.

Soit un tableau de caractères, nommé texte_orig, et contenant le texte suivant :

" ceci Est un tExte à bieN meTtre en forme,en respECTAnt les règles de la ponctuation .donc,attention à bien Traiter les virgules,les points . Et les points virgules ? "

Ecrivez un programme qui permette d'appliquer les règles de ponctuation listées et de ranger le texte résultant dans un tableau de caractères nommé text_format.

char text_orig[200]=" ceci Est un tExte a bieN meTtre en forme,en respECTAnt les regles de la ponctuation .donc,attention a bien Traiter les virgules,les points . Et les points virgules ? ";

```
caractere text_orig[1000];
caractere text_format[1000];
caractere carac;

entier compteur1, compteur2; // compteur1 pour se déplacer dans le
// tableau contenant le texte a corriger, compteur2 pour le texte
// formaté

entier espace, est_point; // espace indique si un espace a déjà été
// rencontré et est en cours de traitement, est_point indique si un
// point quelconque a été rencontré et est en cours de traitement

entier limite; // taille du tableau texte

afficher("entrez votre texte :");
lire(text_orig);

limite ← longueur_texte(texte_orig);

compteur1 ← compteur2 ← 0;

est_point ← 1; // on commence par une majuscule
espace ← 0; // pour manger les espaces

// première etape : ignorer les espaces de début de texte

  tant que (compteur1 < limite) ET (text_orig[compteur1]=' '))
  {
    compteur1 ← compteur1+1;
  }

  tant que (compteur1 < limite)
  {

     // cas à traiter : rencontre d'une suite d'espaces alors qu'on
```

```
      // a déjà renconré un espace
    si (espace=1) alors
    {
      tant que (text_orig[compteur1]=' ')
      {
            compteur1 = compteur1+1;
      }
    }

// traitons les points : on indique qu'on a rencontré un point, on
// le recopie dans le texte formaté, et on indique qu'on doit
// rencontrer un espace (regle de ponctuation)
      si  ((text_orig[compteur1]='.')  OU  (text_orig[compteur1]='!')
OU (text_orig[compteur1]='?'))
        {
        est_point ← 1;
        text_format[compteur2] ← text_orig[compteur1];
        compteur2 ← compteur2+1;
        espace ← 1;
        }
      sinon si (text_orig[compteur1]=' ') // les espaces 'normaux'
      {
        espace ← 1;  // on ne le recopiera qu'après
      }
      sinon             si             ((text_orig[compteur1]=',')OU
(text_orig[compteur1]=';')) // les ponctuations , ;
      {
        text_format[compteur2] ← text_orig[compteur1];
        compteur2 ← compteur2+1;
      // on recopie les ponctuations, elles serotn suivies d'un
      // espace
        espace←1;
      }
      else // le dernier cas : un caractère quelconque
      {
            carac ← text_orig[compteur1];
            si (espace=1) alors
            {
           // s'il y avait un espace, on le place maintenant
               text_format[compteur2] ← ' ';
               compteur2 ← compteur2+1;
            }
            espace ← 0; // on indique qu'il a été traité
            si (est_point = 1) alors // s'il y avait un point
            {
                // passage minuscule - majuscule
                si ((carac >='a') ET (carac <='z')) alors
                {
                   carac ← carac-'a'+'A';
                }

                // on place le caractere et on indique que le point
                // a ete traite
                text_format[compteur2] ← carac;
```

```
                compteur2 ← compteur2+1;
                est_point ← 0;
            }
            sinon // pas un debut de phrase : passage en minuscule
            {
                si ((carac >='A') ET (carac <='Z')) alors
                {
                   carac ← carac-'A'+'a';
                }
                text_format[compteur2] ← carac;
                compteur2 ← compteur2+1;

            }
        }

       // fini ? passage à la lettre suivante
        compteur1 ← compteur1+1;
    }

    text_format[compteur2]←'\0'; // pour terminer le texte formaté

    afficher(texte_orig,"\n formate en:\n", text_format,"\n");
```

Loi de Zipf

La loi de Zipf est une loi empirique que les textes respectent plus au moins. On cherche d'abord à classer les mots par fréquence : on compte le nombre de fois où chaque mot apparaît dans le texte. Ensuite, on classe les fréquences de tous les mots par ordre décroissant, et on devrait observer que la deuxième fréquence est la moitié de la première, que la troisième est la moitié de la deuxième, et ainsi de suite.

Partie 1) recherche d'un mot quelconque dans un texte.

Quelques précautions avant de traiter le texte :

On considère que le texte que l'on veut analyser pour vérifier la pertinence de la loi de Zipf est obtenue par un simple copier/coller à partir d'une page web où d'un document électronique quelconque. Une partie du travail d'analyse va consister à découper le texte en mots pour pouvoir réaliser leur comptage. Comment peut-on repérer qu'un ensemble de caractères constitue un mot ?

Un mot est un ensemble de caractères entouré par : des espaces, ou des signes de ponctuation, on un de chaque. On peut choisir, dans un premier temps, de remplacer toutes les ponctuations (, ; . ? !) par des espaces pour simplifier la recherche par la suite.

Si on ne fait pas cette substitution, il faudra tester la présence d'un séparateur (espace ou signe de ponctuation).

Exemple pour résoudre cette question : dans le texte suivant, quels sont les mots présents et quels sont leur(s) fréquences respectives ?

```
"Une personne m'a dit le mot bonjour. J'ai répondu bonjour à cette personne! Bonjour? Quel joli mot!"
```

bonjour : fréquence 3

personne : fréquence 2

mot : fréquence 2

tous les autres mots : fréquence 1.

mots et sous-mots.

Lors de la recherche d'un mot, il faut bien faire attention à ne pas compter les occurrences d'un mot lorsqu'il est inclus dans un autre mot. Comment être sur de ne compter que les mots isolés et, pour l'exemple suivant, de ne compter le mot 'café' que 2 fois.

```
"sais-tu où je peux prendre un café ? oui, pour un café, il faut aller à la cafétéria !"
```

Pour cela, on ne recherche pas uniquement les caractères du mot, mais on doit vérifier qu'il est bien entouré de séparateurs : si l'on a choisi de remplacer tous les séparateurs par des espaces, alors, pour le mot 'café' de l'exemple, ce que l'on recherche, c'est la chaîne ' café ' (mot entouré de deux espaces). Il faut donc prévoir que le texte commence par un espace et finit également par un espace.

a) écrire un programme qui recherche, dans une tableau de caractères, le nombre de fois où apparaît un mot quelconque (qui peut être saisi au clavier).

b) Améliorer ce programme pour qu'il "supprime" du tableau le mot recherché et en range un exemplaire dans un nouveau tableau où seront stockés les mots. Pour supprimer un mot du tableau, on remplace toutes ses lettres par le caractère '#'.

Remarque : on peut aussi remplacer les caractères du mot trouvé par des espaces, le fait de mettre des '#' peut être utile pour la ise au point du programme.

Illustration : dans le texte "le magasin ouvrira le 12 juin et le 14 juin.", on compte 3 fois le mot "le" le but est d'arriver au texte :

"## magasin ouvrira ## 12 juin et ## 14 juin.", et de stocker le mot "le" dans un nouveau texte (que l'on appellera dictionnaire).

De même, le texte contient 2 fois le mot "juin". Le but est d'arriver au texte :

"## magasin ouvrira ## 12 #### et ## 14 ####.", et au dictionnaire

"le juin"

rangement dans le dictionnaire : immédiat; suppression : il suffit de modifier un peu le parcours, dès que la première lettre du mot recherché est reconnue, on marque l'indice de cette première lettre. Si la correspondance est complète, alors on remplace les lettres

c) A partir des questions a et b, on peut calculer la fréquence de chaque mot du texte. Ecrire un programme qui permet de calculer ces fréquences et qui les range dans un tableau d'entiers.

Illustration avec le texte précédent ; en ayant traité les mots "le" et "juin", voici respectivement : le texte, le dictionnaire, et le tableau d'entiers:

"## magasin ouvrira ## 12 #### et ## 14 ####."

"le juin"


```
le principe est non plus de saisir les mots à repérer, mais de les
prendre les uns après les autres dans le texte modifié, en passant
les espaces et les '#'.

Nouvelle version :
```

d) Améliorer le programme pour qu'il affiche : les fréquences dans l'ordre décroissant et les mots associés à chacune des fréquences, à partir du dictionnaire.

Pour cela, on cherchera l'indice de la valeur maximum du tableau des fréquences (ind_max), puis on affiche la fréquence correspondante, et on met cette fréquence à 0 pour ne plus la traiter par la suite. A partir de cette valeur ind_max, on

parcoure le dictionnaire, dans lequel on comptera les mots en repérant les espaces de séparation.

THEME 4 : tableaux à plusieurs dimensions

Tri selon une ligne

On cherche à écrire un logiciel de gestion d'une compétition de tir à l'arc. Cette compétition accueille au maximum 15 concurrents, chaque concurrent se voit attribuer un numéro de dossard compris entre 1 et 15. La compétition de déroule en 10 volées de 3 flèches, en respectant la séquence suivante :

Pour les 5 premières volées, les concurrents passent à tour de rôle suivant leur numéro de dossard. A chaque volée de 3 flèches, leur score est augmenté du total des points (on suppose qu'ils visent une cible concentrique avec le 1 à l'extérieur et le 10 au centre).

A l'issue de ces 5 premières volées, l'ordre de passage est modifié : on fait passer les concurrents par ordre croissant de score (le concurrent qui est en tête après la cinquième volée tirera donc en dernier pour les volées suivantes).

A l'issue des dix volées, on souhaite appeler un par un les concurrents selon l'ordre inverse du classement final pour leur remettre un lot.

Ecrire un programme qui :

Affiche le numéro du dossard du prochain tireur (attention à l'ordre de passage !), met à jour son score, et organise l'appel des concurrents pour la remise des lots. On définit un tableau à deux dimensions pour stocker les résultats de ce concours de tir à l'arc. Ce tableau est constitué de deux lignes et de 15 colonnes (il y a au maximum 15 concurrents). La première ligne contient les numéros de dossard, la deuxième contient les scores.

Le principe de cet exercice est relativement simple : il faut tout d'abord initialiser le tableau avec en première ligne le numéro de dossard puis des 0 pour le score en deuxième ligne.

```
programme concours_tir_arc

entier tab_dos_scores[2][15];
entier nb_conc, indice, score_volee;
entier num_volee;

// variables pour le tri

entier etape, cpt;
entier temp;

faire
```

```
{
afficher("entrez le nombre de concurrents :")
saisir(nb_conc);
} tant que ((nb_conc <2) OU (nb_conc>15));

// initialisation des n° de dossard
// et des scores à 0
pour indice de 0 à nb_conc-1 faire
{
     tab_dos_scores[0][indice] ← indice+1;
     tab_dos_scores[1][indice] ← 0;

}
// premiere serie de 5 volees : appel et saisie du score
pour num_volee de 1 à 5 faire
{
     pour indice de 0 à nb_conc-1 faire
     {
          afficher("concurrent  numero,"  tab_dos_scores[0][indice],
" appele sur le pas de tir\n");
     afficher("score de la volee :");
     saisir(score);
     tab_dos_scores[1][indice] ← tab_dos_scores[1][indice]+score;
     }
}

// tri du tableau selon les scores croissants

pour etape de 0 à nb_conc-1 faire
{
     pour cpt de 0 à nb_conc-2 faire
     {
          si  (tab_dos_scores[1][cpt]  >  tab_dos_scores[1][cpt+1])
alors
          {
          temp = tab_dos_scores[0][cpt];
          tab_dos_scores[0][cpt] = tab_dos_scores[0][cpt+1];
          tab_dos_scores[0][cpt+1] = temp;

          temp = tab_dos_scores[1][cpt];
          tab_dos_scores[1][cpt] = tab_dos_scores[1][cpt+1];
          tab_dos_scores[1][cpt+1] = temp;

          }
     }
}

// il suffit de recopier la première partie du programme

// 5 volees puis classement de nouveau

// appel pour les lots

pour indice de 0 à nb_conc-1 faire
{
```

```
    afficher("concurrent ",tab_dos_scores[0][indice]," venez
chercher votre lot !\n");
}
```

Pour les 5 premières volées, on parcoure le tableau en affichant les numéros de dossard croissants et on met à jour les scores correspondants. Au bout de la 5ème volée, on trie le tableau (les deux lignes) suivant les valeurs croissantes de la deuxième ligne, puis on recommence à appeler les concurrents dans l'ordre ainsi établi, en mettant les scores à jour. Au bout de la dixième volée, on trie à nouveau le tableau suivant la deuxième ligne pour procéder à la remise des lots.

Calcul matriciel

Une matrice en deux dimensions est définie en mathématiques comme un ensemble de coefficients repérés par leur numéro de ligne et numéro de colonne. Une matrice *M* à *n* lignes et *p* colonnes est un ensemble de coefficients m_{ij}, avec *i* compris entre 1 et *n* et *j* compris entre 1 et *p*.

$$M= \underbrace{\begin{pmatrix} m_{11} & m_{12} & & m_{1p} \\ m_{21} & \dots & & \\ & & & \\ & & & \\ m_{n1} & & & m_{np} \end{pmatrix}}_{p \text{ colonnes}} \Bigg\} n \text{ lignes}$$

On peut naturellement les représenter par des tableaux à 2 dimensions.

Si M est une matrice ayant le même nombre de lignes que de colonnes (c'est à dire que *n=p*), on dit que la matrice est carrée et on peut calculer sa trace $\mathrm{Tr}(M) = \sum_{i=1}^{i=n} m_{ii}$

a) Ecrire un programme qui initialise une matrice carrée avec des valeurs aléatoires et qui calcule sa trace.

Pour les valeurs aléatoires, on dispose de la commande hasard(n) qui produit un entier compris entre 0 et n. initialisation avec une double boucle pour, calcul de la trace avec une seule boucle pour.

```
programme init_et_trace
entier mat[50][50];

entier tai, lig, col;
entier trace;
```

```
afficher("entrez la taille de la matrice carree M:");
saisir(tai);

pour lig de 0 à tai-1 faire
{
     pour col de 0 à tai-1 faire
     {
          mat[lig][col]=hasard(10);
     }
}

trace ← 0;

pour lig de 0 à tai-1 faire
{
     trace ← trace+met[lig][lig];
}

afficher("Tr(M)=",trace);
```

On peut multiplier entre elles deux matrices *A* et *B* pour obtenir le produit *A.B* (attention avec les matrices la multiplication n'est plus une opération commutative) à la seule condition que le nombre de colonnes de la matrice *A* soit égal au nombre de lignes de la matrice *B*.

Si *A* est une matrice à *n* lignes et *p* colonnes et *B* une matrice à *p* lignes et *q* colonnes, alors le produit *A.B* est une troisième matrice (nommons la *C*) telle que :

$$\forall i \in [1..n], \forall j \in [1..p], c_{ij} = \sum_{k=1}^{k=p} a_{ik}.b_{kj}$$

Où a_{ik}, b_{kj} et c_{ij} sont des coefficients des matrices *A*, *B* et *C*.

b) Ecrivez un programme qui initialise deux matrices avec des entiers au hasard et qui réalise le produit de ces deux matrices. Vous pouvez vous aider d'un schéma pour matérialiser comment se fait le produit avant de tenter d'écrire le programme.

```
programme multi_mat

entier a[50][50];
entier b[50][50];
entier c[50][50];

entier n,p,q;
entier cpt_i, cpt_j, cpt_k;
entier inter;

afficher("entrez le nombre de lignes de la matrice A:");
saisir(n);
```

```
afficher("entrez le nombre de colonnes de la matrice A:");
saisir(p);
afficher("la matrice B comporte donc ",p," lignes\n");
afficher("entrez le nombre de colonnes de la matrice B:");
saisir(q);

// initialisation des matrices a et b
// reprise du programme précédent pour l'initialisation

// multiplication de matrices

pour cpt_i de 0 à n-1 faire
{
     pour cpt_j de 0 à q-1 faire
     {
     inter ← 0;

     pour cpt_k de 0 à p-1 faire
     {
          inter ← inter + a[cpt_i][cpt_k] * b[cpt_k][cpt_j];
     }

     c[cpt_i][cpt_j] ← inter;
     }
}
// affichage de la matrice c avec une double boucle pour
```

Jeu de dames

On souhaite réaliser un jeu de dames en utilisant un damier de *n* cases sur *n* cases, il s'agit donc d'un cas un peu plus général que le jeu classique, pour lequel *n*=10. On imposera tout de même, pour simplifier, que *n* soit pair, et qu'il soit compris entre 6 et 20.

On choisit de représenter l'état d'une des cases du damier par un nombre entier, en utilisant la convention suivante :

0 représente une case noire vide

1 représente une case blanche vide

2 représente une case blanche occupée par un pion blanc

3 représente une case blanche occupée par un pion noir

4 représente une case blanche occupée par une dame blanche

5 représente une case blanche occupée par une dame noire

a) Ecrire un programme dans lequel on trouve la définition d'un tableau à deux dimensions permettant de jouer avec n'importe quelle taille d'échiquier autorisée.

```
programme jeu_dames
entier damier[20][20];
// on prévoit juste la plus grande taille possible.
```

b) Ecrire un morceau du programme (pour lequel on ne répètera pas la définition du tableau) qui initialise le damier vide avec l'alternance : case noire / case blanche

```
programme jeu_dames
entier damier[20][20];
// on prévoit juste la plus grande taille possible.

entier tai_dam;
entier ligne, colonne;

faire
{
      afficher("entrez une taille paire comprise entre 6 et 20");
      saisir(tai_dam);
}     tant que ((tai_dam <6) OU (tai_dam>20) OU (tai-dam%2 ≠ 0));

pour ligne de 0 à tai_dam-1 faire
{
      pour colonne de 0 a tai_dam-1 faire
      {
            damier[ligne][colonne] ← (ligne+colonne)%2;
      }
}
```

c) A la suite, écrire un morceau du programme qui affiche le damier. Note : pour afficher une case noire, on affiche un caractère espace, dont le code ascii est 32, pour les cases blanches, on affiche le caractère dont le code ascii est 219.

```
pour ligne de 0 à tai_dam-1 faire
{
      pour colonne de 0 a tai_dam-1 faire
      {
            si (damier[ligne][colonne]=0) alors
            {
                  afficher(" ");
            }
            sinon
            {
                  afficher((caractere)219);
            }
      }
      afficher("\n"); // ne pas oublier le passage à la ligne
}
```

d) Pour positionner les pions suivant les positions de départ, on utilise la règle suivante : les pions blancs sont positionnées sur toutes les cases blanches des (*n*/2

– 1) premières lignes du damier, les pions noires sur toutes les cases blanches des (n/2 – 1) dernières lignes du damier. Ecrivez la partie du programme qui réalise ce positionnement.

On remplacera toutes les cases blanches vides (valeurs 1) par des valeurs 2 (pion blanc sur case blanche) pour les lignes d'indice 0 à (tai-dam/2)-2 pour les pions blancs.

On remplacera toutes les cases blanches vides (valeur 1) par des valeurs 3 (pion noir sur une case blanche) pour les lignes d'indice tai_dam/2+1 à tai_dam-1 pour les pions noirs.

```
pour ligne de 0 à (tai_dam/2-2) faire
{
      pour colonne de 0 à tai_dam-1 faire
      {
            si (damier[ligne][colonne]=1) alors
            {
                  damier[ligne][colonne] ← 2;
            }
      }
}


pour ligne de (tai_dam/2+1) à (tai_dam-1) faire
{
      pour colonne de 0 à tai_dam-1 faire
      {
            si (damier[ligne][colonne]=1) alors
            {
                  damier[ligne][colonne] ← 3;
            }
      }
}
```

e) On cherche à savoir si un pion peut être déplacé (attention, dans le jeu de dames, les pions ne peuvent se déplacer que dans un sens, c'est à dire vers le camp adverse). Si la position d'un pion est repéré par son numéro de ligne *i* et son numéro de colonne *j*, quel(s) test(s) faut-il effectuer pour savoir s'il peut se déplacer ? Ecrivez le(s) test(s) correspondants en langage algorithmique.

Donc les pions blancs se déplacent vars le 'bas' du damier (indices de ligne croissants), et les pions noirs vers le 'haut' (indices de ligne décroissants);

```
entier peut_bouger, case_libre, pas_au_bord;

si (damier[i][j] = 2) alors // pion blanc
{
```

```
     // on doit tester que l'on ne sortira pas du damier
     // et qu'il existe une case libre comme destination

     pas_au_bord ← (i < tai_dam-1);

     case_libre ← ((j>0) ET (damier[i+1][j-1]=1));
     case_libre ← case_libre OU ((j<tai_dam-1) ET
damier[i+1][j+1]=1));

}
sinon // c'est un pion noir
{
     pas_au_bord ← i>0;
     case_libre((j>0) ET damier[i-1][j-1]=1));
     case_libre←case_libre OU ((j<tai_dam-1) ET (damier[i-
1][j+1]=1));
}

peut_bouger ← ((case_libre=1) ET (pas_au_bord=1);
```

f) Prise : de la même manière, quel(s) test(s) faut-il effectuer pour savoir si un pion peut prendre un pion adverse ? Ecrivez ces tests en langage algorithmique.

On retrouve le même type de test, en un peu plus compliqué ! il faut tester que : dans l'une des diagonales, on trouve successivement : un pion adverse puis une case libre, et que l'on ne sortira pas du damier.

```
entier peut_prendre, case_libre, pas_au_bord;

si (damier[i][j] = 2) alors // pion blanc
{
     // on doit tester que l'on ne sortira pas du damier
     // et qu'il existe une case libre comme destination

     pas_au_bord ← (i < tai_dam-2); // on anticipe de 2 lignes

     case_libre ← ((j>1) ET (damier[i+1][j-1]=3) ET (damier[i+2][j-
2]=1)); // 3 pour un pion noir et 1 pour une case vide
     case_libre ← case_libre OU ((j<tai_dam-2) ET
damier[i+1][j+1]=3) ET (damier[i+2][j+2]=1));

}
sinon // c'est un pion noir
{
     pas_au_bord ← i>1;
     case_libre((j>1) ET damier[i-1][j-1]=2) ET (damier[i-2][j-
2]=1));
     case_libre←case_libre OU ((j<tai_dam-2) ET (damier[i-
1][j+1]=2) ET (damier[i-2][j+2]=1));
}
```

```
peut_prendre ← ((case_libre=1) ET (pas_au_bord=1);
```

g) Comment repère-t-on qu'un pion se transforme en dame ? Ecrire le test correspondant en langage algorithmique.

On repère qu'un pion se transforme en dame lorsqu'il atteint, suite à un mouvement, le côté opposé de l'échiquier, soit : ligne=0 pour un pion noir et ligne=tai_dam-1 pour un pion blanc.

h) Comment repère-t-on la fin de la partie ? Ecrire le(s) test(s) correspondants en langage algorithmique.

La fin de la partie est repérée par le fait qu'il n'y a plus de pion d'une certaine couleur sur le damier. Il faut donc ajouter; pour chaque camp, le nombre total de pions de la couleur concernée sur l'échiquier. A chaque prise, ce nombre est diminué de 1. On devra donc tester la fin de la partie après chaque prise de pion.

THEME 5 : Synthèse

Simulation d'un adressage réseau avec un tableau

On cherche à simuler le cheminement d'un message à travers un réseau en utilisant un tableau à plusieurs dimensions qui joue le rôle d'une table de routage : chaque machine possède une adresse IP (Internet Protocol) unique qui permet de l'identifier de façon non ambiguë.

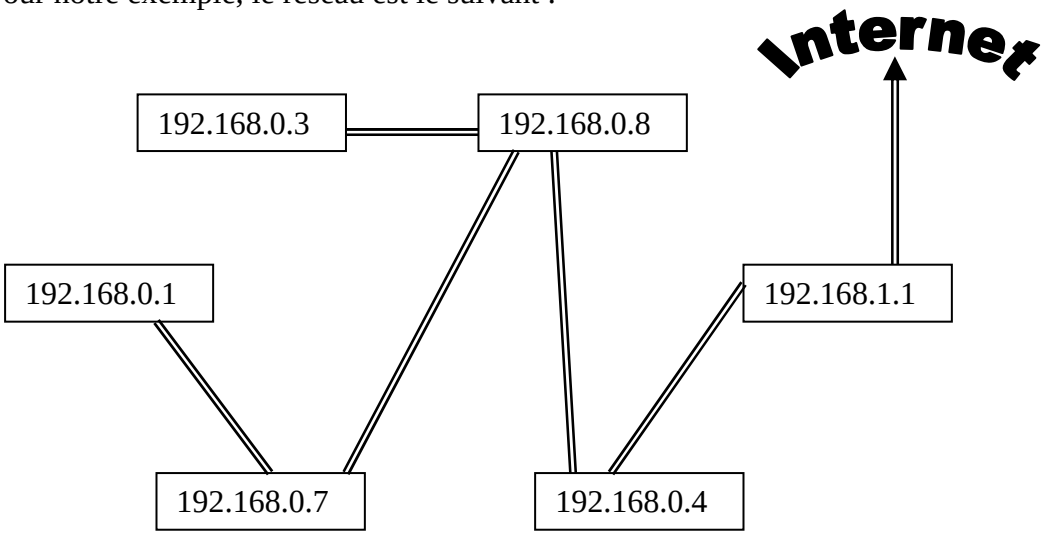
Dans l'exemple que nous allons traiter, notre réseau comporte 6 machines, numérotées de 1 à 6, et dont les adresse IP sont les suivantes :

1) 192.168.0.1
2) 192.168.0.3
3) 192.168.0.4
4) 192.168.0.7
5) 192.168.0.8
6) 192.168.1.1

Ces adresses IP sont celles d'un réseau dit local (intranet), où seule la machine 6 a un accès à Internet.

Une table de routage est une table qui permet de faire l'expédition correcte d'un message depuis une machine vers une autre machine. Ainsi, si l'on fait le schéma des câbles existants pour transmettre les informations, on peut matérialiser quelles machines sont en contact avec quelle machines.

Pour notre exemple, le réseau est le suivant :



On remarque déjà qu'aucune machine n'est isolée, mais que chaque machine n'est pas systématiquement reliée à toutes les autres.

a) Comment peut-on matérialiser les liens existants entre les machines à l'aide d'un tableau à deux dimensions ? quelles valeurs indiquer dans ce tableau ? Ce tableau à deux dimensions est aussi appelé matrice d'adjacence du réseau (il indique quelles sont les liens existants). Pour simplifier la création de ce tableau, reportez sur le schéma précédent les numéros des machines correspondant aux adresses IP du schéma.

Le but est d'initialiser le tableau 2D avec des 0 et de reporter les 1 correspondants à des liaisons existantes.

b) On cherche à réaliser une table de routage à l'aide d'un tableau à deux dimensions T_rout. Cette table de routage contient les informations suivantes : T_rout[i][j] contient l'adresse IP de la machine par laquelle il fait passer pour transmettre un message de la machine i à la machine j. par exemple, pour émettre un message de la machine 1 (192.168.0.1) à la machine 5 (192.168.0.8), il faut passer par la machine d'adresse 192.168.0.7. Quel type de données utiliser pour stocker une adresse IP ? **normalement, ce devrait être du texte ! or on un tableau à deux**

dimensions comportant des textes devient un tableau à 3 dimensions comportant des caractères…ce que nous allons simplifier.

c) Comment simplifier la table de routage pour qu'elle ne contienne plus les adresses IP des machines mais leur numéro ? (attention, on doit tout de même garder le lien entre numéro de machine et adresse IP de la machine).

On aura donc une table de routage comportant des numéros de machine, mais aussi un nouveau tableau qui fait le lien entre le numéro de la machine et son adresse IP sous la forme d'un texte : c'est un tableau de tableaux de caractères, soit un tableau 2D que l'on peut fournir aux étudiants pour terminer cette question.

d) On décide de faire jouer à la machine d'adresse 192.168.1.1 le rôle de proxy : elle sert de passerelle pour que toutes les autres machines du réseau local aient accès à Internet. De plus, toute adresse IP ne commençant pas par 192 est considérée comme l'adresse d'une machine Internet (hors du réseau local). Cela signifie que, si une machine du réseau local (expéditeur) veut envoyer un message à une autre machine (destinataire), alors, il faut regarder le début de l'adresse du destinataire :

Si cette adresse est du type 192.xxx.yyy.zzz, il faut regarder si la machine correspondante existe dans le réseau local.

Si cette adresse est du type ttt.xxx.yyy.zzz avec ttt différent de 192, alors le destinataire est une machine internet, et donc la communication doit obligatoirement passer par la machine d'adresse 192.168.1.1 (la passerelle)

Ecrire un programme qui, à partir de la table de routage et de la matrice d'adjacence, est capable d'indiquer le chemin emprunté par un message issu d'une machine expéditeur du réseau local à destination d'une autre machine. Ce programme devra saisir les adresses des machines expéditrice et destinataire, et gérer les cas où ces adresses ne sont pas correctes.

Exemples :

si la machine 192.168.0.8 veut envoyer un message à la machine 80.162.254.20 (sur internet), alors le programme doit afficher :

Message de 192.168.0.8
passe par : 192.168.0.4
passe par : 192.168.1.1
arrive à : 80.162.254.20

Si la machine 192.168.0.1 veut envoyer un message à la machine 192.168.0.3

Message de : 192.168.0.1

passe par : 192.168.0.7

passe par : 192.168.0.8

arrive à : 192.168.0.3

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    char tab_adresses[6][15]={
    "192.168.0.1",
    "192.168.0.3",
    "192.168.0.4",
    "192.168.0.7",
    "192.168.0.8",
    "192.168.1.1"
    };

    long adja[6][6];
    long routage[6][6];

    long lig,col,cpt;

    char emet[15], dest[15];

    long correct; // adresse correcte ou non
    long interne; // destinataire interne ou non

    long num_emet, num_dest;    // numero des machines emettrices
                                // et destinatrices de message
    long pas_trouve;  // l'adresse demandee correspond-elle à une
                      // machine existante ?

     // initialisation des matrices d'adjacence et de routage
    for(lig=0; lig < 6; lig++)
    {
        for(col=0; col<6; col++)
        {
        adja[lig][col] = 0;
        routage[lig][col] = 0;
        }
    }

// intialisation de la matrice d'adjacence, les liens sont
// bidirectionnels, donc la matrice est symetrique

  adja[0][3] = adja[1][4] = adja[3][4] = adja[2][4]=adja[2][5] = 1;
  adja[3][0] = adja[4][1] = adja[4][3] = adja[4][2]=adja[5][2] = 1;

// initialisation de la matrice de routage : si cet exo est traité
```

```
// en TP, ce code sera rendu disponible en téléchargement

  routage[0][1]= 3;
  routage[0][2]= 3;
  routage[0][3]= -1;
  routage[0][4] = 3;
  routage[0][5] = 3;

  routage[1][0] = 4;
  routage[1][2] = 4;
  routage[1][3] = 4;
  routage[1][4] = -1;
  routage[1][5] = 4;

  routage[2][0] = 4;
  routage[2][1] = 4;
  routage[2][3] = 4;
  routage[2][4] = -1;
  routage[2][5] = -1;

  routage[3][0] = -1;
  routage[3][1] = 4;
  routage[3][2] = 4;
  routage[3][4] = -1;
  routage[3][5] = 4;

  routage[4][0] = 3;
  routage[4][1] = -1;
  routage[4][2] = -1;
  routage[4][3] = -1;
  routage[4][5] = 2;

  routage[5][0] = 2;
  routage[5][1] = -1;
  routage[5][2] = 2;
  routage[5][3] = 2;
  routage[5][4] = 2;

     // traitement de l'adresse de l'emetteur

  do
  {

  printf("entrez votre adresse de départ :");
  gets(emet);

  if (strncmp(emet,"192",3)!=0 ) // l'émetteur doit appartenir à
                                 // l'intranet : adresse 192.xxx
  {
    correct = 0;
    printf("pas de machine exterieure en emetteur\n");
  }
  else // l'adresse commence par 192
  {
  cpt = 0;
     // on recherche si cette adresse existe dans la table des
```

```
    // adresses IP des machines de l'intranet

while ((cpt < 6) && (strcmp(emet,tab_adresses[cpt])!=0 ) )
{
  cpt=cpt+1;
}

  if (cpt == 6)
  {
      correct = 0;
      printf("pas de machine à cette adresse dans le réseau\n");
  }
  else
  {   num_emet = cpt;
     correct = 1;
  }
}

} while (correct != 1);

    // saisie du destinataire, on accepte les adresses internet


do
{

printf("entrez votre adresse de destination :");
gets(dest);

if (strncmp(dest,"192",3)!=0 ) // donc internet
{
  interne = 0;// machine externe
  num_dest = 5; // doit passer par 192.168.1.1
}
else // l'adresse commence par 192
{
cpt = 0;
    // recherche de l'existence de la machine
while ((cpt < 6) && (strcmp(dest,tab_adresses[cpt])!=0 ) )
{
  cpt=cpt+1;
}

  if (cpt == 6)
  {
      correct = 0; // adresse non reconnue
      printf("pas de machine à cette adresse dans le réseau\n");
  }
  else
  {
      interne = 1;  // destinataire interne
      num_dest = cpt; // on stocke son numero
      correct = 1; // adresse reconnue
  }
}
```

```
  } while (correct != 1);

  printf("machine identifiée");

  // cherchons un chemin

  pas_trouve=1;

  printf("message de %s\n",tab_adresses[num_emet]);

  while (pas_trouve == 1)
  {
     // on teste si un lien direct existe
    if (adja[num_emet][num_dest]==0) // non, donc on route
    {
        num_emet = routage[num_emet][num_dest];
           // on truve dans la table de routage la machine
          // intermediaire qui devient le nouvel emetteur
        printf("passe par %s\n",tab_adresses[num_emet]);
    }
    else // il y a un lien direct, ok c'est termine
    {
        pas_trouve = 0;
    }

  }

  if (interne==0) // si le destinataire est externe on doit indiquer
                  // l'adresse finale en plus de la machine
                  // 192.168.1.1 (dernier destinataire interne).
  {
  printf("passe par %s\n",tab_adresses[num_dest]);
  printf("arrive a %s\n",dest);
  }
  else // un simple affichage du destinataire
  {
  printf("arrive à %s\n",tab_adresses[num_dest]);
  }

  system("PAUSE");
  return 0;
}
```