Algorithmes et programmation en Pascal

Faculté des Sciences de Luminy — Edouard Thiel

Annales corrigées

Deug 1 Mass MA
Module de 75 heures
Examens de 1998 à 2003

Sommaire

Faculté des Sciences de Luminy — 9 juin 1998
Deug 1 MASS MA — Durée 3 heures

Épreuve d'Informatique

Documents, calculettes et ordinateurs sont interdits.
Les exercices sont indépendants, à réaliser en langage Pascal.

1. Tableau de sortie *(4 points)*

Faire le tableau de sortie de chaque programme (c'est-à-dire le tableau des valeurs successives des variables et expressions) et en déduire ce qui est affiché à l'écran.

1.a
```
var i : integer;
begin
  i := 1; while i < 10 do i := 11 - 2*i; writeln (i);
end.
```

1.b
```
const r = 'mercredi'; var i : integer; m, e : char;
begin
  m := r[1]; e := r[2];
  for i := 2 to 7 do if m <= r[i]
                     then begin e := m; m := r[i]; end;
  writeln (e = m);
end.
```

2. Trouver les erreurs *(5 points)*

Les programmes suivants devraient tous afficher 1998, or ils sont truffés d'erreurs (il peut y en avoir plusieurs dans chaque programme). Trouvez-les et proposez éventuellement une correction.

2.a
```
procedure p (var x : integer);
begin
  writeln(x+4);
end;
Begin
  p(1994);
End.
```

2.b
```
procedure q (var x : integer);
begin
  for i := 1 to 1998 do x := x+1;
end;
Var t : integer;
Begin
  q(t); writeln(t);
End.
```

2.c
```
function r (a, b : char) : boolean;
begin
  if a <> b then r := 999;
end;
Begin
  r('a','b'); writeln(2*r);
End.
```

2.d
```
procedure s (var u, v, w : integer);
begin
  w := u; v := w; u := v;
end;
Var x, y : integer;
Begin
  x := 4 and y := 8; s(x,y);
  writeln('199'x);
End.
```

3. Tableaux de matchs *(6 points)*

Le sélectionneur de l'équipe de France veut faire des statistiques sur les matchs de $1^{ère}$ division. Il dispose des types suivants pour mémoriser le nom des équipes :

```
CONST MaxEquipe = 25;
TYPE  nom_t = string[63];
      TabNom_t = array [1..MaxEquipe] of nom_t;
```

3.a Écrire la procédure `SaisieNoms (var tN : TabNom_t ; var nN : integer);` qui lit au clavier une suite de noms (1 par ligne) terminée par une ligne vide, mémorise ces noms dans `tN` et le nombre résultant dans `nN`.

3.b Les équipes sont désormais numérotées dans leur ordre de saisie. On introduit maintenant des types pour mémoriser les scores des matchs, qui ont lieu chaque fois entre une équipe *locale* et une équipe *extérieure* :

```
CONST MaxMatch = 1000;
TYPE  Match_t = record
                  n_loc, n_ext,           { numéro équipe locale, extérieure }
                  s_loc, s_ext : integer; { score équipe locale, extérieure  }
                end;
      TabMatch_t = array [1..MaxMatch] of Match_t;
```

Écrire la fonction `QuiGagne (m : Match_t) : integer;` qui pour un match `m` donné, renvoie 1, -1 ou 0 selon que la gagnante est l'équipe locale, extérieure ou que le match est nul.

3.c Écrire la procédure `AffiGagnants (tM : TabMatch_t; nM : integer; tN : TabNom_t; nN : integer);` qui affiche pour chacun des `nM` matchs le nom de l'équipe gagnante, en appelant éventuellement la fonction `QuiGagne`.

3.d Écrire la fonction `DiffLocalExt (tM : TabMatch_t; nM : integer) : integer;` qui renvoie pour l'ensemble des `nM` matchs, la différence entre le nombre de matchs gagnés par l'équipe locale et le nombre de matchs gagnés par l'équipe extérieure, en appelant éventuellement la fonction `QuiGagne`.

4. Fusion de 2 fichiers triés *(5 points)*

Soit le type `ligne_t = string[255];`

On dispose de 2 fichiers texte `nomf1` et `nomf2` qui contiennent chacun exactement 1 mot par ligne (lettres minuscules sans espacement), sauf à la fin où il peut y avoir une ligne vide. Ces fichiers sont triés sur les mots dans l'ordre croissant.

On se propose de fusionner ces 2 fichiers en 1 fichier unique `nomf3`, qui contiendra tous les mots de `nomf1` et `nomf2`, et sera trié.

Exemple

`nomf1`	`nomf2`		`nomf3`
choisir	bien		bien
fait	des		choisir
germer	exemples		des
la		⟶	exemples
solution			fait
			germer
			la
			solution

Il s'agit de lire alternativement parfois dans `nomf1`,
parfois dans `nomf2`, et de décider chaque fois quel mot on écrit dans `nomf3`.

On rappelle que l'on n'a pas le droit de lire dans un fichier sans avoir vérifié au préalable que sa fin n'est pas atteinte.

4.a Écrire la procédure `Lire (var f : text; var s : ligne_t);` qui reçoit en paramètre un fichier `f` déjà ouvert, puis lit une ligne `s` dans `f` si `f` n'est pas vide, sinon rend `s` vide.

4.b Écrire la procédure `Fusion (nomf1, nomf2, nomf3 : ligne_t);` qui réalise la fusion des fichiers `nomf1` et `nomf2` dans le fichier `nomf3`, et s'occupe de l'ouverture et de la fermeture des 3 fichiers.

Correction

Exercice 1 : sur 4 points.
1.a : 1 point pour le tableau, 0,5 point pour l'affichage.
1.b : 2 points pour le tableau, 0,5 point pour l'affichage, + bonus 1 point pour l'interprétation du résultat.

```
1.a   i = 1   init
      i = 9
      i = -7
      i = 25
```

Le programme affiche 25.

```
1.b                         m = 'm'  e = 'e'   init
      i = 2  r[i] = 'e'  m = 'm'  e = 'e'
      i = 3  r[i] = 'r'  m = 'r'  e = 'm'
      i = 4  r[i] = 'c'  m = 'r'  e = 'm'
      i = 5  r[i] = 'r'  m = 'r'  e = 'r'
      i = 6  r[i] = 'e'  m = 'r'  e = 'r'
      i = 7  r[i] = 'd'  m = 'r'  e = 'r'
```

Le programme affiche true.
(e est le précédent max et le test renvoie true si il y a plusieurs occurences du max).

Exercice 2 : sur 5 points, soit 0,5 points par erreur * 10 erreurs.

```
2.a procedure p (var x : integer);           enlever le var
    begin
      writeln(x+4);
    end;
    Begin
      p(1994);
    End.

2.b procedure q (var x : integer);
    begin                                     déclarer i
      for i := 1 to 1998 do x := x+1;
    end;
    Var t : integer;
    Begin
      q(t); writeln(t);                       initialiser t := 0;
    End.

2.c function r (a, b : char) : boolean;      : integer;
    begin
      if a <> b then r := 999;                la fct ne renvoie pas
    end;                                       toujours un résultat
    Begin
      r('a','b'); writeln(2*r);               writeln(2*r('a','b'))
    End.

2.d procedure s (var u, v, w : integer);     mettre w en local
    begin
      w := u; v := w; u := v;                 mauvais échange, il faut
    end;                                       changer l'ordre des :=
    Var x, y : integer;
    Begin
      x := 4 and y := 8; s(x,y);              remplacer and par ;
      writeln('199'x);                        il manque la virgule
    End.
```

Exercice 3 : sur 6 points, 1,5 point par question.

```
3.a Procedure SaisieNoms (var tN : TabNom_t ; var nN : integer);
    Var s : nom_t;
    Begin
      nN := 0; readln (s);
      while s <> '' do       { ou length(s) > 0 }
      begin nN := nN+1; tN[nN] := s; readln (s); end;
    End;

3.b Function QuiGagne (m : Match_t) : integer;
    Begin
      if m.s_loc > m.s_ext then QuiGagne := 1
      else if m.s_loc < m.s_ext then QuiGagne := -1
      else QuiGagne := 0;
    End;

3.c Procedure AffiGagnants (tM : TabMatch_t; nM : integer;
                            tN : TabNom_t; nN : integer );
    Var i, g : integer;
    Begin
      for i := 1 to nM do
      begin
        g := QuiGagne (tM[i]);
        if g > 0 then writeln (tN[tM[i].n_loc])
        else if g < 0 then writeln (tN[tM[i].n_ext]);
      end;
    End;

3.d Function DiffLocalExt (tM : TabMatch_t; nM : integer) : integer;
    Var i, g : integer;
    Begin
      g := 0;
      for i := 1 to nM do g := g + QuiGagne (tM[i]);
      DiffLocalExt := g;
    End;
```

Exercice 4 sur 5 points : 4.a sur 1 point ; 4.b sur 4 points, dont 1 pour les assign/close.

```
4.a Procedure Lire (var f : text; var s : ligne_t);
    Begin
        if eof (f) then s := ''
                   else readln (f, s);
    End;

4.b Procedure Fusion (nomf1, nomf2, nomf3 : ligne_t);
    Var f1, f2, f3 : text;
        m1, m2 : ligne_t;
    Begin
        assign (f1, nomf1); reset (f1);
        assign (f2, nomf2); reset (f2);
        assign (f3, nomf3); rewrite (f3);

        Lire (f1, m1); Lire (f2, m2);

        while (m1 <> '') and (m2 <> '') do
          if m1 < m2
          then begin writeln (f3, m1); Lire (f1, m1); end
          else begin writeln (f3, m2); Lire (f2, m2); end;

        while m1 <> '' do
          begin writeln (f3, m1); Lire (f1, m1); end;

         while m2 <> '' do
          begin writeln (f3, m2); Lire (f2, m2); end;

        close (f1); close (f2); close (f3);
    End;
```

Faculté des Sciences de Luminy — 1^{er} septembre 1998
Deug 1 MASS MA — Durée 3 heures

ÉPREUVE D'INFORMATIQUE

Documents, calculettes et ordinateurs sont interdits.

Problème Le jeu du pendu

Le problème est à réaliser en langage Pascal. Pour répondre à une question, vous avez le droit de vous servir des types ou d'appeler les procédures des questions précédentes, même si vous ne les avez pas traitées. Il est conseillé de lire le problème en entier avant de commencer. Le barème est aussi une indication de la difficulté des questions.

1. Question de cours *(1 point)*

On introduit le type suivant :

```
TYPE chaine_t = string[63];
```

Soit `s` une variable du type `chaine_t`. Sur combien d'octets est codée `s` ? Où est stockée la longueur courante de `s` ? Quelles sont les 2 façons pour connaître la longueur courante de `s` ?

2. Saisie *(1 point)*

Écrire une procédure `saisie_secret` qui demande à l'utilisateur de rentrer une chaîne secrète au clavier. Cette chaîne sera stockée dans un paramètre résultat de type `chaine_t`.

3. Contrôle *(3 points)*

Écrire une fonction `controle_saisie` prenant en paramètre une chaîne de type `chaine_t`, et donnant en résultat de fonction un booléen qui dit si la chaîne respecte les règles suivantes : elle ne doit être composée que de minuscules, de majuscules ou d'espaces, et ses caractères extrêmes (le premier et le dernier) ne doivent pas être un espace.

4. Masquage *(3 points)*

On dispose de la fonction `MinusCar (c : char) : char` qui renvoie `c` en minuscule si `c` est une majuscule, sinon renvoie `c` inchangé. (Ne pas écrire cette fonction !)

Écrire une procédure `masquage` qui prend en paramètre la chaîne secrète `se`, et en paramètre résultat la chaîne masquée `ma`. L'opération consiste à recopier `se` dans `ma` puis à remplacer dans `ma` tous les caractères internes qui ne sont pas des espaces par le caractère `'_'`. Attention, les caractères internes qui sont égaux (en minuscules ou majuscules) aux caractères extrêmes ne doivent pas être masqués. Exemple (`'␣'` symbolise le caractère espace) :

```
se := 'Bien␣le␣bonjour'   ⟶   ma := 'B___␣__␣b_____r'
```

5. Dévoiler (*3 points*)

Écrire la fonction `devoiler` qui prend en paramètres la chaîne secrète `se`, la chaîne masquée courante `ma` et un caractère `c`. Si `c` est un caractère de `se` non déjà dévoilé, alors la fonction renvoie `TRUE` et dévoile les occurences de `c` dans `ma`. Attention, il faut gérer l'égalité indépendamment des minuscules ou majuscules. Exemple :

```
se := 'Bien␣le␣bonjour'                    ma := 'B__n␣__␣b_n___r'
ma := 'B___␣__␣b_____r'    ⟶            devoiler := TRUE
c := 'N'
```

6. Gagner (*2 points*)

On introduit le type `partie_t` dans lequel le champ `se` est la chaîne secrète, `ma` est la chaîne masquée courante, et `pendu` est le nombre de tentatives restant avant d'être pendu.

```
TYPE partie_t = record
                  se, ma : chaine_t;
                  pendu  : integer;
                end;
```

Écrire la fonction `gagne` qui prend en paramètre `p` de type `partie_t`, puis renvoie `TRUE` si tous les caractères ont été dévoilés.

7. Une partie (*3 points*)

Écrire la procédure `une_partie` qui reçoit en paramètre `p` de type `partie_t` correctement initialisé. Cette procédure gère le déroulement de toute une partie. À chaque étape, elle affiche la chaîne masquée courante et le nombre de tentatives restant avant d'être pendu, puis lit au clavier un caractère correspondant à la tentative du joueur. La procédure s'arrête lorsque le joueur a gagné ou lorsque le joueur est pendu (mais sans afficher « gagné » ou « pendu » : c'est le programme principal qui s'en charge, en se servant de `p`).

8. Programme principal (*4 points*)

Faire un programme principal permettant de jouer au pendu. Le programme demande de rentrer une chaîne secrète au clavier, contrôle la saisie, et recommence si le contrôle est faux. Le programme masque ensuite les caractères internes et initialise le nombre de tentatives à 10, puis fait jouer la partie. À la fin de la partie, le programme affiche si le joueur a gagné ou s'il est pendu. Enfin, le programme propose de jouer une nouvelle partie.

Correction

1.

s est codée sur 64 octets ; la longueur courante est codée dans s[0] ;
on y accède par length(s) ou ord(s[0])

2.

```
Procedure saisie_secret (var se : chaine_t);
Begin
    write ('Rentrez une chaîne secrète : ');
    readln (se);
End;
```

3.

Un while serait mieux mais on ne recherche pas l'efficacité.

```
Function controle_saisie (se : chaine_t) : boolean;
Var i : integer;
    r : boolean;
Begin
    r := true;
    if (se[1] = ' ') or (se[length(se)] = ' ')
    then r := false
    else for i := 1 to length(se) do
           if not (    (se[i] >= 'a') and (se[i] <= 'z')
                    or (se[i] >= 'A') and (se[i] <= 'Z')
                    or (se[i] = ' ')
                  )
           then r := false;
    controle_saisie := r;
End;
```

4.

```
Procedure masquage (se : chaine_t; var ma : chaine_t);
Var i : integer;
Begin
    ma := se;
    for i := 2 to length(se)-1 do
      if  (ma[i] <> ' ')
      and (MinusCar(ma[i]) <> MinusCar(se[1]))
      and (MinusCar(ma[i]) <> MinusCar(se[length(se)]))
      then ma[i] := '_';
End;
```

5.

```
Function devoiler (    se : chaine_t;
                   var ma : chaine_t; c : char) : boolean;
Var i : integer;
    r : boolean;
Begin
    r := false; c := MinusCar(c);
    for i := 2 to length(se)-1 do
      if (ma[i] = '_') and (MinusCar(se[i]) = c)
      then begin
             r := true; ma[i] := se[i];
           end;
    devoiler := r;
End;
```

6.

```
Function gagne (p : partie_t) : boolean;
Var i : integer;
    r : boolean;
Begin
    r := true;
    for i := 2 to length(p.se)-1 do
      if (p.ma[i] = '_') then r := false;
    gagne := r;
End;
```

7.

```
Procedure une_partie (var p : partie_t);
Var c : char;
Begin
    while (p.pendu > 0) and not gagne(p) do
    begin
      write (p.ma, '  ', p.pendu:2, '  ? '); readln (c);
      if not devoiler (p.se, p.ma, c)
      then p.pendu := p.pendu-1;
    end;
End;
```

8.

```
PROGRAM pendu;

{ Ici les types et fonctions }

VAR
    p   : partie_t;
    rep : char;
BEGIN
    writeln ('Le jeu du pendu'); writeln;
    repeat
      repeat
        saisie_secret (p.se);
      until controle_saisie (p.se);

      masquage (p.se, p.ma);
      p.pendu := 10;
      une_partie (p);

      if p.pendu = 0 then writeln ('Pendu !!')
                     else writeln ('Gagné !!');
      writeln ('La solution était : ''', p.se, '''');

      write ('Voulez-vous rejouer (o/n) ? ');
      readln (rep);
    until (MinusCar(rep) <> 'o')
END.
```

Faculté des Sciences de Luminy — 7 juin 1999
Deug 1 MASS MA — Durée 3 heures

Épreuve d'Informatique

Documents, calculettes et ordinateurs sont interdits.
Les exercices sont indépendants, à réaliser en langage Pascal.

1. Tableau de sortie *(4 points)*

Faire le tableau de sortie de chaque programme, c'est-à-dire le tableau des valeurs successives des variables et expressions.

1.a
```
var a : integer;
begin
  a := 3;
  repeat
    a := a+4;
    if a > 5 then a := a-11;
  until not (a < 1) or (a > 3);
end.
```

1.b
```
var a, b : (c, d, e, f, g);
begin
  a := g;
  for b := c to a do
    if (a > b)
    then a := pred(a);
end.
```

2. Date de naissance dans Pi *(8 points)*

Soit le type `vec_t = array [1..6] of char;`

On dispose du fichier texte `'pi.txt'`, qui contient le premier million de décimales de π, codées sous la forme de caractères entre `'0'` et `'9'`, sans espacement et sans saut de ligne.

On se propose de rechercher si une date de naissance, codée sur 6 caractères, figure ou non dans le fichier sur 6 caractères consécutifs.

2.a Écrire la procédure `Saisie (var vda : vec_t);` qui demande de rentrer une date, lit au clavier une chaîne de caractères puis recopie les 6 premiers caractères de la chaîne dans `vda`.

2.b Écrire la procédure `DecaleVec (var vpi : vec_t; d : char);` qui décale les 5 caractères de droite de `vpi` de 1 cran vers la gauche, puis insère `d` à la fin de `vpi`. Cette procédure permettra de connaître à tout instant les 6 derniers caractères lus dans le fichier.

2.c Écrire la fonction `CompareVec (vpi, vda : vec_t) : boolean;` qui compare les 2 vecteurs `vpi` et `vda`, et renvoie vrai si ils sont égaux. Par souci d'efficacité, la comparaison doit s'arrêter dès que l'on est sûr que le résultat est faux. On rappelle que l'on n'a pas le droit de sortir d'un vecteur, et qu'une expression est toujours complètement évaluée.

2.d Faire un programme principal, qui s'occupe de l'ouverture et la fermeture du fichier `'pi.txt'`, saisit une date de naissance, puis parcourt le fichier à la recherche de la première apparition de la date.

La recherche s'interrompt dès que la date est trouvée, et le programme affiche alors la plage des numéros des décimales de π correspondant à la date trouvée ; sinon le programme affiche un message d'échec et dit combien de décimales de π le programme a parcourues en vain.

On rappelle que l'on n'a pas le droit de lire dans un fichier sans avoir vérifié au préalable que sa fin n'est pas atteinte, et que les caractères sont lus l'un après l'autre, sans retour possible en arrière.

3. Les invités du mariage *(8 points)*

Pour aider un couple à organiser son mariage, un étudiant en Deug Mass propose de réaliser un logiciel pour gérer les invités et placer les convives parmi les tables du banquet.

Il dispose des types suivants pour mémoriser le nom d'un invité :

```
TYPE Sexe_t = (homme, femme);
     Invite_t = record
                  nom, prenom : string[63];
                  adresse : string[255];
                  sexe : Sexe_t;
                  a_deja_une_place : boolean;
                end;
```

3.a Faire une procédure `AffiInvite (i : Invite_t);` qui affiche sur une seule ligne le nom et le prénom de l'invité `i`.

3.b Les invités sont mémorisés dans un tableau dans leur ordre de saisie :

```
CONST MaxInvite = 200;
TYPE  ListeInvite_t = array [1..MaxInvite] of Invite_t;
```

Écrire une fonction `CombienDe (li,ni,s)` qui renvoie le nombre d'invités de sexe `s : Sexe_t` dans le tableau `li : ListeInvite_t` parmi les `ni` invités.

3.c Pour chaque table on mémorise le nombre de places libres et occupées, ainsi que le tableau des numéros des invités à cette table :

```
CONST MaxConvive = 12;
TYPE Table_t = record
                 nb_places_libres, nb_places_occupees : integer;
                 convives : array [1..MaxConvive] of integer;
               end;
```

Faire une procédure `RemplisTable (ta,li,ni)` qui reçoit une table vide `ta : Table_t` avec un nombre connu de places libres, choisit des convives dans la liste `li : ListeInvite_t` parmi les `ni` invités, et mémorise le numéro de chaque convive.

La procédure ne peut choisir un invité qui `a_deja_une_place`, et se charge de mettre à jour ce champ lorsqu'elle a choisi un invité.

La règle est de placer alternativement un `homme` et une `femme`, et d'essayer de remplir au maximum la table. S'il n'y a plus assez d'un genre ou d'un autre, la procédure s'arrête.

3.d L'organisation des tables est stockée dans le tableau suivant :

```
CONST MaxTable = 25;
TYPE  Banquet_t = array[1..MaxTable] of Table_t;
```

Écrire une procédure `AffiTables (b,li)` qui affiche pour chaque table non inoccupée de `b : Banquet_t` le numéro de la table, puis le nom et le prénom de chacun des convives inscrits à cette table, pris dans la liste `li : ListeInvite_t`.

Correction

Exercice 1 : 2 + 2 points.

1.a Évaluation d'une expression booléenne : le `not` ne s'applique pas au `or`, donc l'expression est équivalente à `(a >= 1) or (a > 3)`, qui est équivalente à `a >= 1`.

a	3	7	-4	0	4

Remarque : si l'expression était `not (a < 1) or not (a > 3)`, on aurait `3, 7, -4`; si l'expression était `not ((a < 1) or (a > 3))`, on aurait `3, 7, -4, 0, 4, 8, -3, 1`!!

1.b Type énuméré : il n'y a que 2 variables! Les bornes du `for` sont figées avant l'itération; la fonction `pred` est vue en cours.

a	g	f	e	e	e	e
b		c	d	e	f	g

Exercice 2 : 1 + 1 + 2 + 4 points.

```
2.a procedure Saisie (var vda : vec_t);
    var s : string[10];
        i : integer;
    begin
      write ('date ? '); readln (s);
      for i := 1 to 6 do vda[i] := s[i];
    end;

2.b procedure DecaleVec (var vpi : vec_t; d : char);
    var i : integer;
    begin
      for i := 1 to 5 do vpi[i] := vpi[i+1];
      vpi[6] := d;
    end;
```

Vu en cours (§ insertion dans un vecteur trié); pour décaler à gauche on fait la boucle vers la droite.

```
2.c function CompareVec (vpi, vda : vec_t) : boolean;
    var i : integer;
    begin
      i := 1;
      while (i < 6) and (vpi[i] = vda[i]) do
        i := i+1;
      CompareVec := vpi[i] = vda[i];
    end;
```

Vu en cours (§ recherche dans un vecteur trié); il est essentiel de faire le test inférieur strict pour ne pas sortir du vecteur dans l'expression. On peut contourner la difficulté avec un booléen.

```
2.d VAR f : text;
        i : integer;
        c : char;
        trouve : boolean;
        vpi, vda : vec_t;
    BEGIN
      Saisie (vda);

      assign (f, 'pi.txt');
      reset(f);

      i := 0; trouve := false; vpi[6] := ' ';
      while not eof(f) and not trouve do
      begin
        read (f, c); i := i+1;
        DecaleVec(vpi, c);
        if CompareVec (vpi, vda) then trouve := true;
      end;
      if trouve
      then writeln ('trouve en position ', i-5, '..', i)
      else writeln ('non trouve dans les ', i, ' premieres decimales');

      close(f);
    END.
```

On intègre la lecture des 5 premiers caractères dans la boucle car elle se charge de tester `eof(f)`. On peut initialiser vpi comme on veut, l'essentiel est que les 5 premières fois la recherche échoue.

Exercice 3 : 0.5 + 1.5 + 4 + 2 points.

3.a
```
procedure AffiInvite (i : Invite_t);
begin
  writeln (i.nom, ' ', i.prenom);
end;
```

3.b
```
function CombienDe (li : Liste_Invite_t; ni : integer; s : Sexe_t) : integer;
var i, res : integer;
begin
  res := 0;
  for i := 1 to ni do
    if li[i].sexe = s then res := res+1;
  CombienDe := res;
end;
```

3.c Attention aux `var`!!

```
procedure RemplisTable (var ta : Table_t; var li : Liste_Invite_t; ni : integer);
var i, k : integer; consecutif : boolean;
begin
  { Init : aucun convive }
  for k := 1 to ta.nb_places_libres do
    ta.convives[k] := 0;

  { Recherche des hommes, stockés en k = 1, 3, 5, ..}
  k := 1; i := 1;
  while (i <= ni) and (k <= ta.nb_places_libres) do
  begin
    if (li[i].sexe = homme) and not li[i].a_deja_une_place
    then begin ta.convives[k] := i; k := k + 2; end;
    i := i + 1;
  end;

  { Recherche des femmes, stockées en k = 2, 4, 6, ..}
  k := 2; i := 1;
  while (i <= ni) and (k <= ta.nb_places_libres) do
  begin
    if (li[i].sexe = femme) and not li[i].a_deja_une_place
    then begin ta.convives[k] := i; k := k + 2; end;
    i := i + 1;
  end;

  { Recherche du nombre de places consécutives qu'on a pu occuper }
  k := 1; consecutif := true;
  while consecutif and (ta.nb_places_libres >= 0) do
    if ta.convives[k] = 0 then consecutif := false
    else begin
           { On assoit le convive k }
           ta.nb_places_libres := ta.nb_places_libres - 1;
           ta.nb_places_occupees := ta.nb_places_occupees + 1;
           li[ta.convives[k]].a_deja_une_place := true;
           k := k + 1;
         end;
end;
```

3.d
```
procedure AffiTables (b : banquet_t; li : Liste_Invite_t);
var i, j : integer;
begin
  for i := 1 to MaxTable do
  if b[i].nb_places_occupees > 0
  then begin
         writeln ('Table n. ', i);
         for j := 1 to b[i].nb_places_occupees do
           AffiInvite ( li[ b[i].convives[j] ] );
       end;
end;
```

Faculté des Sciences de Luminy — 6 septembre 1999
Deug 1 MASS MA — Durée 3 heures

Épreuve d'Informatique

Documents, calculettes et ordinateurs sont interdits.

Problème La collection de timbres

Le problème est à réaliser en langage Pascal. Pour répondre à une question, vous avez le droit de vous servir des types ou d'appeler les procédures des questions précédentes, même si vous ne les avez pas traitées.

1. Type *(2 points)*

Écrire un type enregistrement `timbre_t` comprenant les champs `descriptif`, `oblitere`, `etat` et `cote`. Le champ `descriptif` est une chaîne de 79 caractères ; `oblitere` est un booléen ; `etat` est un énuméré parmi `Neuf`, `Moyen` ou `Abime` ; `cote` est un réel, qui donne la valeur estimée du timbre.

2. Affichage *(3 points)*

Écrire une procédure `affi_timbre(t:timbre_t)`, qui affiche à l'écran, pour chaque champ de `t`, une ligne comprenant le nom du champ, puis `':'`, puis la valeur du champ. Afficher `'oui'` ou `'non'` selon l'oblitération ; utiliser un `case of` pour afficher l'état ; afficher la cote sur 8 chiffres avec 2 chiffres après la virgule.

3. Thème *(3 points)*

On introduit le type suivant pour mémoriser un ensemble de timbres faisant partie d'un même thème, tel que *animaux*, *fleurs*, *hommes célèbres*, *mission Apollo*, etc. Le champ `ntimbre` est le nombre de timbres du thème, qui sont stockés dans le champ `vtimbre`.

```
CONST MAXTIMBRE = 1000;
TYPE theme_t = Record
                 titre : string[79];
                 vtimbre : array [1..MAXTIMBRE] of timbre_t;
                 ntimbre : integer;
               End;
```

Écrire une fonction `plus_grande_cote(t:theme_t)` qui renvoie l'indice, dans le tableau du champ `vtimbre`, du timbre qui a la plus grande cote. S'il n'y a pas de timbre dans le thème, la fonction renvoie 0.

4. Collection *(3 points)*

Une collection de timbres est mémorisée sous la forme d'un tableau de thèmes ; le champ `ntheme` est le nombre de thèmes de la collection, qui sont stockés dans le champ `vtheme`.

```
CONST MAXTHEME = 100;
TYPE collec_t = Record
                  proprietaire : string[79];
                  vtheme : array [1..MAXTHEME] of theme_t;
                  ntheme : integer;
                End;
```

Écrire une procédure `affi_meilleures_cotes(c:collec_t)` qui, pour chaque thème de la collection, affiche le titre du thème, puis les informations concernant le timbre ayant la meilleure cote, ou `'pas de timbre'`.

5. Comparaison *(4 points)*

La procédure `compare_collec(c1,c2:collec_t; obli:boolean);` affiche le nom du propriétaire de la collection ayant la plus grande valeur, ou affiche `'même cote totale'` s'ils sont *ex aequo*. Cette valeur est la somme des cotes des timbres dont l'oblitération est égale à `obli`.

5.a Découper la procédure `compare_collec` en plusieurs procédures ou fonctions, spécialisées par exemple dans le calcul d'une somme de cotes pour un thème ou une collection ; donner leurs en-têtes avec paramétrage complet.

5.b Programmer complètement ces procédures et fonctions.

6. Vente *(5 points)*

Le propriétaire d'une collection `c` décide de vendre un lot de timbres, que l'on représente par :

```
CONST MAXLOT = 2000;
TYPE lot_t = Record
               itheme, itimbre : array [1..MAXLOT] of integer;
               nlot : integer;
             End;
```

Le champ `nlot` est le nombre de timbres dans le lot. Le timbre numéro `i` dans le lot `lo` est le timbre numéro `lo.itimbre[i]` dans le thème numéro `lo.itheme[i]` de la collection `c`.

6.a Quelle est l'expression Pascal donnant l'état de ce timbre.

6.b Écrire une procédure `enleve_de_la_collec(c,lo)` qui supprime de la collection `c` tous les timbres répertoriés dans le lot `lo` (paramétrer correctement la procédure).

La procédure doit vérifier que les timbres répertoriés figurent bien dans la collection, c'est-à-dire que les indices ne sortent pas des tableaux.

La suppression d'un timbre correspond à la suppression dans un vecteur <u>non</u> trié (on remplit le trou avec le dernier élément).

Attention, cette pratique peut fausser les indices dans le lot (puisqu'on déplace des timbres en remplissant les trous) ; il faut donc procéder en 2 temps : d'abord on parcourt le lot et on marque dans la collection les timbres à supprimer (par exemple en mettant leur cote à `-1`) ; puis on balaie la collection et on supprime les timbres marqués.

Correction

1. Sur 2 points.

```
TYPE timbre_t = Record
                  descriptif : string[79];
                  oblitere : boolean;
                  etat : (Neuf, Moyen, Abime);
                  cote : real;
                End;
```

2. Sur 3 points.

```
Procedure affi_timbre (t:timbre_t);
Begin
  writeln ('Descriptif : ', t.descriptif);

  write   ('Oblitere   : ');
  if t.oblitere then writeln ('oui') else writeln ('non');

  write   ('Etat       : ');
  case t.etat of
    Neuf  : writeln ('Neuf');
    Moyen : writeln ('Moyen');
    Abime : writeln ('Abime');
  end;

  writeln ('Cote       : ', t.cote:8:2);

End;
```

3. Sur 3 points.

```
Function plus_grande_cote (t:theme_t) : integer;
Var k, i : integer;
Begin
  if t.ntimbre = 0 then k := 0 else k := 1;

  { cette boucle ne fait rien si il y a 0 ou 1 timbre }
  for i := 2 to t.ntimbre do
    if t.vtimbre[i].cote > t.vtimbre[k].cote
    then k := i;

  plus_grande_cote := k;
End;
```

4. Sur 3 points.

```
Procedure affi_meilleures_cotes (c:collec_t);
Var k, i : integer;
Begin
  for i := 1 to c.ntheme do
  begin
    writeln ('Theme : ', c.vtheme[i].titre);

    k := plus_grande_cote (c.vtheme[i]);
    if k > 0
    then affi_timbre (c.vtheme[i].vtimbre[k])
    else writeln ('pas de timbre');

    writeln;
  end;
End;
```

5. Sur 4 points.

5.a En-têtes : cf ci-dessous.

```
5.b Function somme_timbres_obli (t : theme_t; obli : boolean) : real;
    Var s : real; i : integer;
    Begin
      s := 0;
      for i := 1 to t.ntimbre do
        if t.vtimbre[i].oblitere = obli then s := s + t.vtimbre[i].cote;
      somme_timbres_obli := s;
    End;

    Function total_collec_obli (c : collec_t; obli : boolean) : real;
    Var s : real; i : integer;
    Begin
      s := 0;
      for i := 1 to c.ntheme do
        s := s + somme_timbres_obli (c.vtheme[i], obli);
      total_collec_obli := s;
    End;

    Procedure compare_collec (c1, c2 : collec_t; obli : boolean);
    Var s1, s2 : real;
    Begin
      s1 := total_collec_obli (c1, obli);
      s2 := total_collec_obli (c2, obli);
      if s1 = s2 then writeln ('même cote totale')
      else if s1 > s2 then writeln (c1.proprietaire)
                      else writeln (c2.proprietaire);
    End;
```

6. 1 + 4 points.

6.a État : `c.vtheme[ lo.itheme[i] ].vtimbre[ lo.itimbre[i] ].etat`

```
6.b Procedure enleve_de_la_collec (var c : collec_t; lo : lot_t);
    Var i, j, k : integer;
    Begin
      { marquage des cotes à -1 }
      for i := 1 to lo.nlot do
      begin
        j := lo.itheme[i]; k := lo.itimbre[i];
        if (j >= 1) and (j <= c.ntheme)
        then { attention, bien séparer pour que j ne sorte pas de c.vtheme[ ] }
             if (k >= 1) and (k <= c.vtheme[j].ntimbre)
             then c.vtheme[j].vtimbre[k].cote := -1;
      end;

      { suppression de la collection des timbres marqués }
      for j := 1 to c.ntheme do
      begin
        { on ne peut pas employer de for car ntimbre peut varier }
        k := 1;
        while k <= c.vtheme[j].ntimbre do
        begin
          if c.vtheme[j].vtimbre[k].cote = -1
          then begin
                 { suppression : on copie le dernier, mais on n'incrémente pas k }
                 { car le timbre copié est peut-etre marqué ; test au prochain tour }
                 c.vtheme[j].vtimbre[k] :=
                   c.vtheme[j].vtimbre[ c.vtheme[j].ntimbre ];
                 c.vtheme[j].ntimbre := c.vtheme[j].ntimbre - 1;
               end
          else k := k+1;
        end;
      end;
    End;
```

Faculté des Sciences de Luminy — 13 juin 2000
Deug 1 MASS MA — Durée 3 heures

Épreuve d'Informatique

Documents, calculettes et ordinateurs sont interdits.
Les exercices sont indépendants, à réaliser en langage Pascal.

1. Archéologie *(3 points)*

En l'an 4096, un archéologue spécialiste de Pascal ancien trouve sur un bas relief le fragment de programme suivant ; aider l'archéologue en reconstituant dans la mesure du possible tous les types manquants.

```
TYPE  { *** types illisibles }
VAR la : souris_t;
    i  : integer;
BEGIN
    { *** début illisible }
    for i := 1 to 10 do
        if la.forme[i].est_ronde or la.forme[i].est_ovale
        then case la.teinte[i] of
               Beige, Grise : la.n := la.n+1;
               Fluo         : la.n := la.n-1;
             end;
    { *** suite illisible }
END.
```

2. Programme mystère *(4 points)*

Faire le tableau de sortie du programme (c'est-à-dire le tableau des valeurs successives des variables et expressions), et en déduire ce que le programme affiche.

```
PROGRAM mystere_j00;

TYPE ligne_t = string[255];
VAR  SanA : ligne_t;

PROCEDURE hommage (var s : ligne_t);
VAR i : integer;
    c : char;
BEGIN
    for i := 1 to length(s) do
    begin
        c := s[i];
        case c of
            'a' .. 'z' : c := chr (ord('a') + ord('z') - ord(c));
            'A' .. 'Z' : c := chr (ord('A') + ord('Z') - ord(c));
        end;
        s[i] := c;
    end;
END;

BEGIN
    SanA := 'Uivwvirx Wziw';
    hommage (SanA);
    writeln (SanA);
END.
```

3. Le jeu des 12 erreurs *(3 points)*

Le programme suivant est truffé d'erreurs ; trouvez-les et proposez une correction.

```
Programme mal_ecrit:

fich_t = file of text;

Begin
    assigne (f, "toto");
    for i = 1 to 10 do
        writeln (f, i);
End;
```

4. Carrés magiques *(12 points)*

Un carré magique est un tableau d'entiers, où les sommes des colonnes, des lignes et des diagonales sont toutes égales. Si l'on en croit une légende chinoise, l'exemple ci-contre fut révélé à l'empereur Yü sur le dos d'une tortue, au XXIII^{ème} siècle avant J-C. Le total commun, ou « somme magique », de ce carré est 15, et sa taille, ou « ordre », est 3.

4	9	2
3	5	7
8	1	6

4.a On considère une constante `OrdreMax` valant 10, et un type `Carre_t` qui est un tableau d'entiers en 2 dimensions, dont les indices de ligne et de colonne vont de 1 à `OrdreMax`. Quelles sont les 2 façons de déclarer `Carre_t` ?

4.b Écrire la procédure `AffiCarre(c : Carre_t; n : integer);` qui reçoit en paramètres un carré `c` de taille `n`×`n` et l'affiche à l'écran.

4.c Écrire une fonction `CarreMagique(c : Carre_t; n : integer) : boolean;` qui reçoit en paramètres un carré `c` de taille `n`×`n`, puis renvoie `TRUE` si les sommes des colonnes, des lignes et des diagonales de `c` sont toutes égales, `FALSE` sinon. (Conseil : les 2 diagonales sont parcourues avec `c[i,i]` et `c[i,n-i+1]`, respectivement).

4.d On dit qu'un carré est « normal » si les entiers qui sont dans le carré n'apparaissent qu'une fois et sont tous compris entre 1 et $\texttt{n}^2$ (c'est le cas de l'exemple ci-dessus). Écrire une fonction `CarreNormal(c : Carre_t; n : integer) : boolean;` qui reçoit en paramètres un carré `c` de taille `n`×`n`, puis renvoie `TRUE` si le carré est normal, `FALSE` sinon. (Conseil : utiliser un tableau de booléens pour marquer les entiers déjà utilisés).

4.e Intéressons-nous maintenant au cas des carrés magiques d'ordre 3 ; nous souhaitons écrire une procédure qui les trouve tous. L'algorithme que l'on demande de programmer part de l'idée suivante : on constate que tout carré magique peut se décomposer sous la forme ci-contre, à partir de 4 entiers e, f, g, h et pour une somme magique s. Il suffit donc de générer toutes les combinaisons d'entiers e, f, g, h entre 1 et 9 (soit 4 boucles imbriquées) et de compléter chaque carré comme indiqué. Les carrés générés ne sont pas forcément magiques ni normaux (il faudra faire les tests), mais le nombre de carrés générés n'est que de $9^4 = 6561$. Écrire une procédure `CaMaNo3(s : integer);` qui cherche avec cet algorithme tous les quadruplets formant un carré magique normal de somme magique s et affiche les carrés solution.

e	f	$s-e-f$
g	h	$s-g-h$
$s-e-g$	$s-f-h$	$s-e-h$

4.f Écrire un programme principal qui demande une somme magique puis appelle `CaMaNo3`. Indiquer en commentaire l'emplacement des types et fonctions (mais sans les recopier).

Correction

Exercice 1 : 3 points.

```
CONST
    MaxSouris = 10;
TYPE
    forme_t = Record
                est_ronde, est_ovale : boolean;
              End;
    tabforme_t = array [1..MaxSouris] of forme_t;

    teinte_t = (Beige, Grise, Fluo);
    tabteinte_t = array [1..MaxSouris] of teinte_t;

    souris_t = Record
                 forme : tabforme_t;
                 teinte : tabteinte_t;
                 n : integer;
               End;
```

Exercice 2 : 4 points.

i	s avant	c avant	c après	s après
1	'Uivwvirx Wziw'	'U'	'F'	'Fivwvirx Wziw'
2	'Fivwvirx Wziw'	'i'	'r'	'Frvwvirx Wziw'
3	'Frvwvirx Wziw'	'v'	'e'	'Frewvirx Wziw'
4	'Frewvirx Wziw'	'w'	'd'	'Fredvirx Wziw'
5	'Fredvirx Wziw'	'v'	'e'	'Fredeirx Wziw'
6	'Fredeirx Wziw'	'i'	'r'	'Frederrx Wziw'
7	'Frederrx Wziw'	'r'	'i'	'Frederix Wziw'
8	'Frederix Wziw'	'x'	'c'	'Frederic Wziw'
9	'Frederic Wziw'	' '	' '	'Frederic Wziw'
10	'Frederic Wziw'	'W'	'D'	'Frederic Dziw'
11	'Frederic Dziw'	'z'	'a'	'Frederic Daiw'
12	'Frederic Daiw'	'i'	'r'	'Frederic Darw'
13	'Frederic Darw'	'w'	'd'	'Frederic Dard'

Le programme affiche `'Frederic Dard'`.

Exercice 3 : 12 erreurs * 0.25 = 3 points.

```
Programme mal_ecrit:           1.  program et non programme
                               2.  ";" et non ":"
                               3.  il manque :  type
fich_t = file of text;         4.  text et non "file of text"
                               5.  il manque :  var i : integer;
                               6.                   f : fich_t;
Begin
    assigne (f, "toto");       7.  assign et non assigne
                               8.  'toto' et non "toto"
                               9.  il manque :  rewrite (f);
    for i = 1 to 10 do        10.  ":=" et non "="
        writeln (f, i);
                              11.  il manque :  close (f);
End;                          12.  "." et non ";"
```

Exercice 4 : $1 + 1 + 3 + 3 + 3 + 1 = 12$ points.

```
4.a TYPE
        Carre_t = array [1..OrdreMax] of array [1..OrdreMax] of integer;
    ou
        Carre_t = array [1..OrdreMax , 1..OrdreMax] of integer;
```

```
4.b PROCEDURE AffiCarre(c : Carre_t; n : integer);
    VAR i, j : integer;
    BEGIN
        for i := 1 to n do
        begin
          for j := 1 to n do write (c[i,j], ' ');
          writeln;
        end;
        writeln;
    END;
```

```
4.c FUNCTION CarreMagique(c : Carre_t; n : integer) : boolean;
    VAR i, j, s, magic : integer;
        res : boolean;
    BEGIN
        res := TRUE;
        { calcul 1ere somme sur diagonale }
        s := 0;
        for i := 1 to n do s := s + c[i,i];
        magic := s;

        { verif sur 2eme diagonale }
        s := 0;
        for i := 1 to n do s := s + c[i,n-i+1];
        if s <> magic then res := FALSE;

        { verif sur lignes i }
        for i := 1 to n do
        begin
          s := 0;
          for j := 1 to n do s := s + c[i,j];
          if s <> magic then res := FALSE;
        end;

        { verif sur colonnes j }
        for j := 1 to n do
        begin
          s := 0;
          for i := 1 to n do s := s + c[i,j];
          if s <> magic then res := FALSE;
        end;

        { résultat }
        CarreMagique := res;
    END;
```

```
4.d FUNCTION CarreNormal(c : Carre_t; n : integer) : boolean;
    VAR i, j, k: integer;
        res : boolean;
        tab : array [1..100] of boolean;   {OrdreMax*OrdreMax}
    BEGIN
        res := TRUE;
        { init de tab }
        for k := 1 to n*n do tab[k] := FALSE;

        { parcours tableau }
        for i := 1 to n do
          for j := 1 to n do
          begin
            k := c[i,j];
            if (k < 1) or (k > n*n)    { verif l'intervalle, sinon on  }
            then res := FALSE          { on risque de sortir de tab !! }
            else if tab[k]
                 then res := FALSE     { k n'est pas unique }
                 else tab[k] := TRUE;  { on marque k }
          end;

        { résultat }
        CarreNormal := res;
    END;
```

```
4.e PROCEDURE CaMaNo3(s : integer);
    VAR e, f, g, h : integer;
        c : Carre_t;
    BEGIN
        for e := 1 to 9 do
         for f := 1 to 9 do
          for g := 1 to 9 do
           for h := 1 to 9 do
           begin
             c[1,1] := e;     c[1,2] := f;     c[1,3] := s-e-f;
             c[2,1] := g;     c[2,2] := h;     c[2,3] := s-g-h;
             c[3,1] := s-e-g; c[3,2] := s-f-h; c[3,3] := s-e-h;
             if CarreMagique (c, 3) and CarreNormal (c,3)
             then AffiCarre (c, 3);
           end;
    END;
```

```
4.f PROGRAM magique;

    {Ici les const et les types }

    VAR s : integer;

    { Ici les fonctions et la procédure}

    BEGIN
      write ('somme magique = '); readln (s);
      writeln ('Carrés magiques normaux d''ordre 3 :');
      CaMaNo3 (s);
    END.
```

Faculté des Sciences de Luminy
 Deug 1 MASS MA

7 septembre 2000
 Durée 3 heures

ÉPREUVE D'INFORMATIQUE

Documents, calculettes et ordinateurs sont interdits.

Problème Édition de factures

Le problème est à réaliser en langage Pascal. Pour répondre à une question, vous avez le droit de vous servir des types ou d'appeler les procédures des questions précédentes, même si vous ne les avez pas traitées.

1. Type *(1 point)*

Écrire un type enregistrement `article_t` comprenant les champs `ref` (entier), `desi` (chaîne de 39 caractères) et `pu` (réel). Le champ `ref` est le code de référence de l'article ; `desi` est la désignation de l'article ; `pu` est le prix unitaire de l'article.

2. Affichage *(1 point)*

Écrire une procédure `affi_article(a:article_t)` qui affiche à l'écran les champs de `a` sur une seule ligne, puis fait un saut de ligne. Afficher le code sur 5 caractères, la désignation sur 40 caractères, et le prix unitaire sur 8 caractères dont 2 chiffres après la virgule.

3. Lecture *(1 point)*

Écrire une procédure `lire_article(var f:text; var a:article_t)` qui lit un code d'article, un prix unitaire et une désignation dans le fichier `f`, et les mémorise dans les champs de `a`. On suppose que le fichier `f` est déjà ouvert en lecture, et que sa fin n'est pas atteinte.

4. Catalogue *(3 points)*

On introduit le type suivant pour mémoriser l'ensemble des articles vendus dans un magasin. Le champ `na` est le nombre d'articles, stockés dans le champ `va`.

```
CONST MAXART = 10000;
TYPE catalogue_t = Record
                     va : array [1..MAXART] of article_t;
                     na : integer;
                   End;
```

Écrire une procédure `lire_catalogue(nomf:string; var cat:catalogue_t)` qui reçoit un nom de fichier `nomf`, ouvre le fichier texte correspondant en lecture, initialise le champ `na`, puis lit et mémorise dans `va` tous les articles présents dans le fichier en se servant de `lire_article`, enfin referme le fichier. On rappelle que l'on n'a pas le droit de déborder d'un tableau ; il faudra donc faire les tests lors de la lecture. Enfin, on suppose que toutes les opérations sur le fichier se déroulent sans erreur.

5. Recherche dichotomique *(4 points)*

Écrire une fonction `rech_article(ref:integer; cat:catalogue_t):integer` qui reçoit un catalogue `cat` trié dans l'ordre croissant des codes d'articles, fait une recherche dichotomique de l'article ayant le code `ref`, puis renvoie l'indice de l'article trouvé, ou -1 si l'article n'a pas été trouvé.

6. Commande *(3 points)*

On introduit le type suivant pour mémoriser une commande. Le champ `vref` mémorise le code des articles commandés, le champ `vqte` mémorise la quantité de chaque article commandé. Le champ `nref` est le nombre de références commandées.

```
CONST MAXCOM = 100;
TYPE commande_t = Record
                    vref, vqte : array [1..MAXCOM] of integer;
                    nref : integer;
                  End;
```

Écrire avec un paramétrage correct une procédure `com_article` qui reçoit un catalogue `cat` et une commande en cours `com`. La procédure vérifie en tout premier lieu s'il n'y a plus de place dans `com`, auquel cas elle affiche un message d'erreur et se termine. La procédure demande de rentrer une nouvelle référence, vérifie l'existence de la référence ou affiche un message d'erreur et se termine. Ensuite, la procédure demande la quantité à commander. La procédure mémorise la référence et la quantité dans `com` si la quantité est strictement positive.

7. Facture *(4 points)*

Écrire avec un paramétrage correct une procédure `impr_facture` qui reçoit un catalogue `cat` et une commande `com` ne contenant que des articles référencés, puis affiche la facture à l'écran. Pour chaque article commandé, la procédure affiche la référence, la désignation et le prix unitaire sur une ligne, en se servant des procédures déjà programmées ; si la quantité excède 1, la procédure affiche sur une ligne supplémentaire la quantité et le prix total. À la fin de la facture, la procédure affiche le montant total.

8. Programme principal *(3 points)*

Écrire un programme principal qui lit un catalogue d'articles dans le fichier `'invent.air'`, puis affiche : « N nouvelle commande », « C commande article », « A afficher facture », « Q quitter ». Le programme lit un caractère au clavier, exécute l'action correspondante ou affiche un message d'erreur, puis se remet en attente d'un nouveau caractère. Le programme principal devra utiliser l'instruction `case`. Indiquer en commentaires l'emplacement des types et des fonctions (mais sans les recopier).

Correction

1. Sur 1 point.

```
TYPE article_t = Record
                   ref  : integer;
                   desi : string[39];
                   pu   : real;
                 End;
```

2. Sur 1 point.

```
PROCEDURE affi_article (a : article_t);
BEGIN
  writeln (a.ref:5, ' ', a.desi:40, ' ', a.pu:8:2);
END;
```

3. Sur 1 point.

```
PROCEDURE lire_article (var f : text; var a : article_t);
BEGIN
  readln (f, a.ref, a.pu, a.desi);
END;
```

4. Sur 3 points ; vu en cours.

```
PROCEDURE lire_catalogue (nomf : string; var cat : catalogue_t);
VAR f : text;
BEGIN
  assign (f, nomf);
  resef (f);

  cat.na := 0;
  while not eof(f) and (cat.na < MAXART) do
    begin cat.na := cat.na+1; lire_article (f, cat.va[cat.na]); end;

  close (f);
END;
```

5. Sur 4 points ; vu en cours.

```
FUNCTION rech_article (ref : integer; cat : catalogue_t) : integer;
VAR inf, sup, m : integer;
    trouve      : boolean;
BEGIN
  trouve := false; inf := 1; sup := cat.na;

  while (inf <= sup) and not trouve do
  begin
    m := (inf + sup) div 2;
    if cat.va[m].ref = ref then trouve := true
    else if cat.va[m].ref < ref then inf := m+1 else sup := m-1;
  end;

  if trouve then rech_article := m else rech_article := -1;
END;
```

6. Sur 3 points.

```
PROCEDURE com_article (cat : catalogue_t; var com : commande_t);
VAR ref, qte : integer;
BEGIN
    if com.nref >= MAXCOM
    then writeln ('Erreur, commande pleine')
    else begin
           write ('Référence : '); readln(ref);
           if rech_article (ref,cat) < 0
           then writeln ('Erreur, article inexistant')
           else begin
                  write ('Quantité : '); readln(qte);
                  if qte <= 0
                  then writeln ('Erreur, mauvaise quantité')
                  else begin
                         com.nref := com.nref+1;
                         com.vref[com.nref] := ref;
                         com.vqte[com.nref] := qte;
                       end;
                end;
         end;
END;
```

7. Sur 4 points.

```
PROCEDURE impr_facture (cat : catalogue_t; com : commande_t);
VAR i, j  : integer;
    somme : real;
BEGIN
  somme := 0;
  writeln ('Facture :'); writeln;
  for i := 1 to com.nref do
  begin
    j := rech_article (com.vref[i], cat);
    affi_article (cat.va[j]);
    if com.vqte[i] > 1
    then writeln ('     * ', com.vqte[i]:5,
                  '     = ', com.vqte[i]*cat.va[j].pu);
    somme := somme + com.vqte[i]*cat.va[j].pu;
  end;
  writeln; writeln ('Montant total : ', somme);
END;
```

8. Sur 3 points.

```
PROGRAM facture;

{ ici les const et les types }

VAR cat : catalogue_t; com : commande_t; rep : char;

{ ici les procédures et fonctions }

BEGIN
  lire_catalogue ('invent.air', cat);
  repeat
    writeln ('N nouvelle commande   C commande article');
    writeln ('A afficher facture    Q quitter');
    write ('Votre choix : '); readln (rep);
    case rep of
      'N' : com.nref := 0;
      'C' : com_article (cat, com);
      'A' : impr_facture (cat, com);
      'Q' : ;
      else writeln ('Erreur, mauvais choix');
    end;
  until rep = 'Q';
END.
```

Épreuve d'Informatique

Documents, calculettes et ordinateurs sont interdits.
Les exercices sont indépendants, à réaliser en langage Pascal.

1. Les 16 erreurs *(4 points)*

Le programme suivant devrait afficher 2001, or il est truffé d'erreurs (il peut y en avoir plusieurs dans chaque ligne). Trouvez-les et proposez chaque fois une correction.

```
PROGRAM : Archifaux
TYPE a : record of b, c : integer;
VAR d := a;
PROCEDURE e (f)
BEGIN
  writeln (g(f)'1');
END;
FUNCTION g (h)
BEGIN
  g = h.b^2 + h.c;
END;
BEGIN
  d.a := 14 and d.b := 4 ; e(d);
END;
```

2. Programme mystère *(4 points)*

Faire le tableau de sortie du programme (c'est-à-dire le tableau des valeurs successives des variables et expressions), et en déduire ce que le programme affiche.

```
PROGRAM lsm6;

PROCEDURE areteton (var a, b, c, d : char);
VAR e : integer;
BEGIN
    e := a; a := b; b := c; c := d; d := e;
END;

PROCEDURE acuire (n : integer);
VAR c, h, a, r : char;
BEGIN
  c := 'l'; h := 'd'; a := 'u'; r := 'j';
  while (n > 5) do
  begin
    areteton (c, h, a, r);
    c := chr(ord(c)-1); a := chr(ord(a)+3);
    n := n div 10;
  end;
  writeln (c, h, a, r);
END;

BEGIN
  acuire (2001);
END.
```

3. Score d'un jeu au tennis *(5 points)*

On se propose de gérer l'évolution du score d'un jeu au tennis. Ce score sera codé au moyen de deux variables de type énuméré. Nous rappelons que les opérateurs de comparaison `<`, `<=`, etc, les fonctions `pred` et `succ` et l'instruction `case of` sont utilisables sur des énumérés.

3.a Déclarer un type énuméré `score_jeu_t` comprenant dans l'ordre `Zero`, `Quinze`, `Trente`, `Quarante`, `Avantage`, `Gagne`.

3.b Écrire la procédure `point_jeu (var score1, score2 : score_jeu_t);` qui rajoute un point au `score1` (du joueur 1 qui marque le point) et met éventuellement à jour le `score2` (perte de l'avantage du joueur 2 par exemple), selon les règles du tennis que nous rappelons ci-dessous. On suppose qu'avant l'appel, d'une part le jeu n'est pas déjà gagné, et d'autre part que les scores respectent bien les règles.

Règles du tennis concernant le score d'un jeu : au début d'un jeu le score est 0 à 0. Le joueur qui marque un point passe à 15, au point suivant il passe à 30, puis à 40. Si le joueur qui marque a déjà 40 et que son adversaire à moins de 40, il gagne. Les joueurs sont dits à égalité s'ils ont tous les deux 40. Si les joueurs sont à égalité, celui qui marque prend l'avantage. Si le joueur qui a l'avantage marque, il gagne. Si le joueur qui n'a pas l'avantage marque, ils reviennent à égalité (codé 40 partout).

4. Recherche dans un fichier *(7 points)*

On se propose de faire un programme qui demande le nom d'un fichier texte et une chaîne de caractères à rechercher, puis affiche toutes les lignes du fichier qui contiennent au moins une occurence de la chaîne de caractère.

4.a Écrire le programme principal, appelé `grep`. Déclarer le type `ligne_t` comme une chaîne de 255 caractères. Le programme demande à l'utilisateur le nom du fichier et une chaîne à rechercher, puis appelle la procédure `cherche_dans_fichier` décrite ci-dessous.

4.b Écrire la procédure `cherche_dans_fichier (nomf, rech : ligne_t);` qui ouvre le fichier texte de nom `nomf`, lit le fichier ligne à ligne, puis ferme le fichier. Au cours de la lecture, appeler pour chaque ligne la fonction `chaine_dans_ligne` décrite ci-dessous, et si le résultat de la fonction est `true`, afficher la ligne précédée de son numéro dans le fichier.

4.c Écrire la fonction `chaine_dans_ligne (rech, ligne : ligne_t) : boolean;` qui renvoie `true` si la chaîne de caractères `rech` est présente quelque part dans la chaîne de caractères `ligne`, autrement dit si `rech` est une sous-chaîne de `ligne`. La comparaison de `rech` et d'une sous-chaîne de `ligne` est faite caractère à caractère, et doit s'arrêter aussitôt que possible. La recherche doit s'arrêter dès qu'une occurence de `rech` est trouvée dans `ligne`. *Bien entendu, vous ne devez pas vous servir de la fonction* `pos` *du langage Pascal pour répondre à cette question.*

Correction

Exercice 1 : 4 points (pour 16 erreurs et corrections)

```
PROGRAM : Archifaux                     ':' en trop   manque ';'
TYPE a : record of b, c : integer;      ':' -> '='    'of' en trop   manque 'end;'
VAR d := a;                             ':=' -> ':'
PROCEDURE e (f)                         manque ':a'   manque ';'
BEGIN
  writeln (g(f)'1');                    manque ','    g pas encore déclaré
END;
FUNCTION g (h)                          manque ':a'   manque ':integer;'
BEGIN
  g = h.b^2 + h.c;                      '=' -> ':='   '^2' -> '*h.b'
END;
BEGIN
  d.a := 14 and d.b := 4 ; e(d);        'and' -> ';'  '.a' -> '.b'   '.b' -> '.c'
END;                                    ';' -> '.'
```

Exercice 2 : 4 points

c	h	a	r	c := chr(ord(c)-1)	a := chr(ord(a)+3)	n
'l'	'd'	'u'	'j'			2001
'd'	'u'	'j'	'l'	'c'	'm'	200
'u'	'm'	'l'	'c'	't'	'o'	20
'm'	'o'	'c'	't'	'l'	'f'	2

La procédure affiche : `loft`

Exercice 3 : 1 + 4 points.

```
3.1 TYPE score_jeu_t = (Zero, Quinze, Trente, Quarante, Avantage, Gagne);
```

```
3.2 PROCEDURE point_jeu (var score1, score2 : score_jeu_t);
    BEGIN
      case score1 of
        Zero .. Trente : score1 := succ(score1);
        Quarante       : case score2 of
                           Zero .. Trente : score1 := Gagne;
                           Quarante       : score1 := Avantage;
                           Avantage       : score2 := Quarante;
                         end;
        Avantage       : score1 := Gagne;
      end;
    END;
```

Exercice 4 : 2 + 2 + 3 points.

4.1
```
PROGRAM grep;

TYPE ligne_t = string[255];
VAR nomf, rech : ligne_t;

BEGIN
  write ('Entrez un nom de fichier : '); readln (nomf);
  write ('Entrez une chaîne à rechercher : '); readln (rech);

  cherche_dans_fichier (nomf, rech);
END.
```

4.2
```
PROCEDURE cherche_dans_fichier (nomf, rech : ligne_t);
VAR f : text; ligne : ligne_t; n : integer;
BEGIN
  assign (f, nomf);
  reset (f);

  n := 1;
  while not eof(f) do
  begin
    readln (f, ligne);
    if chaine_dans_ligne (rech, ligne)
    then writeln (n:4, '  ', ligne);
    n := n+1;
  end;

  close (f);
END;
```

4.3
```
FUNCTION chaine_dans_ligne (rech, ligne : ligne_t) : boolean;
VAR i, j, imax, jmax : integer;
BEGIN
  i := 1; imax := length(rech);
  j := 1; jmax := length(ligne) - imax + 1;

  while (i <= imax) and (j <= jmax) do
  begin
    if rech[i] = ligne[j+i-1]
    then i := i+1
    else begin i := 1; j := j+1; end;
  end;

  chaine_dans_ligne := i > imax;
END;
```

Faculté des Sciences de Luminy — 3 septembre 2001
Deug 1 MASS MA — Durée 3 heures

Épreuve d'Informatique

Documents, calculettes et ordinateurs sont interdits.

Problème Formatage de texte

Le problème est à réaliser en langage Pascal. Pour répondre à une question, vous avez le droit de vous servir des types ou d'appeler les procédures des questions précédentes, même si vous ne les avez pas traitées.

On se propose d'écrire un programme qui lit un texte dans un fichier, le décompose en mots, élimine les espaces et saut de lignes inutiles, calcule le nombre de mots que l'on peut mettre dans chaque nouvelle ligne, puis réécrit le texte en le justifiant (= alignant) à gauche.

1. Types *(1 point)*

Déclarer une constante `tab_max` à 200. Écrire un type chaîne de caractère `ligne_t` de 255 caractères. Écrire un type enregistrement `mot_t` comprenant les champs entiers `debut` et `fin`. Écrire un type enregistrement `tab_t` comprenant un champ entier `nb` et un champ `mot`, ce dernier étant un tableau de `mot_t` indicé de 1 à `tab_max`.

2. Test d'un caractère *(1 point)*

Un mot est une suite de lettres, et tous les autres caractères sont des séparateurs (espace et caractères de contrôle, dont saut de ligne, tabulation, etc).
Écrire une fonction booléenne `est_lettre` qui reçoit un caractère `c` en paramètre, puis renvoie vrai si le code Ascii de `c` est strictement plus grand que 32 et inférieur ou égal à 255, sinon renvoie faux.

3. Détection d'un mot *(4 points)*

Écrire une procédure `detecte_mot(s : ligne_t; var m : mot_t)` qui détecte les indices de début et de fin du prochain mot à droite de la position début dans `s`, puis les mémorise dans `m`. À l'appel, le champ `debut` est soit l'indice de la première lettre du mot (il n'y a plus qu'à chercher la fin), soit l'indice d'un séparateur (il faut chercher le début puis la fin). Si aucun mot n'est détecté, le champ `debut` est mis à -1.

4. Analyse d'une ligne *(4 points)*

Écrire la procédure `analyse_ligne(s : ligne_t; var v : tab_t)` qui reçoit une chaîne `s`, puis détecte tous les mots de `s` et mémorise leur indice de début et de fin ainsi que leur nombre dans `v`. On n'oubliera pas d'initialiser la détection au début de `s`, et le nombre de mots détectés à 0.

5. Formatage d'une ligne *(4 points)*

Écrire la procédure `formate_ligne(f : text; s : ligne_t; var nc : integer; max : integer)` qui reçoit un fichier `f` déjà ouvert en écriture, une ligne `s` à analyser, un nombre de caractères `nc` déjà écrits sur la ligne courante de `f`, et la largeur `max` de justification du texte. La procédure analyse la ligne, puis pour chaque mot détecté : calcule la largeur du mot ; teste si elle peut encore écrire le mot (et un espace) dans `f` sans dépasser `max` caractères sur la ligne courante de `f` ; si c'est possible elle écrit un espace dans `f`, sinon elle écrit un saut de ligne ; enfin elle écrit le mot dans `f` (caractère à caractère) et met à jour `nc`.

6. Formatage du texte *(3 points)*

Écrire la procédure `formate_texte(f1, f2 : text; max : integer)` qui reçoit un fichier `f1` déjà ouvert en lecture, un fichier `f2` déjà ouvert en écriture, et la largeur `max` de justification du texte. La procédure lit le fichier `f1` ligne à ligne, et pour chaque ligne, appelle la procédure de formatage de ligne, de telle sorte que celle-ci écrive dans `f2`, et tienne à jour dans une variable locale `nc` le nombre de caractères écrits sur la ligne courante de `f2` (on aura pris soin d'initialiser `nc` à 0).

7. Programme principal *(3 points)*

Écrire le programme principal `aligauche` qui demande le nom du fichier à traiter, le nom du fichier résultat et la largeur de justification, ouvre les fichiers, formate le texte, et enfin ferme les fichiers.

Correction

1. Sur 1 point.

```
CONST
    tab_max = 200;

TYPE
    ligne_t = string[255];

    mot_t = Record
              debut, fin : integer;
            End;

    tab_t = Record
              nb : integer;
              mot : array [1 .. tab_max] of mot_t;
            End;
```

2. Sur 1 point.

```
FUNCTION est_lettre (c : char) : boolean;
BEGIN
    est_lettre := (ord(c) > 32) and (ord(c) <= 255);
END;
```

Exemple d'autres expressions admises :

```
ord(c) > 32         car ord(c) est toujours <= 255
c > #32             raccourci
c > ' '             comparaison entre chars
```

3. Sur 4 points.

```
PROCEDURE detecte_mot (s : ligne_t; var m : mot_t);
VAR i : integer;
BEGIN
    i := m.debut;
    while (i <= length(s)) and not est_lettre(s[i]) do i := i+1;
    if i > length(s)
    then m.debut := -1
    else begin
           m.debut := i;
           while (i <= length(s)) and est_lettre(s[i]) do i := i+1;
           m.fin := i-1;
         end;
END;
```

4. Sur 4 points.

```
PROCEDURE analyse_ligne (s : ligne_t; var v : tab_t);
VAR m : mot_t;
BEGIN
    v.nb := 0;
    m.debut := 1;
    detecte_mot (s, m);

    while m.debut <> -1 do
    begin
      v.nb := v.nb+1;
      v.mot[v.nb] := m;
      m.debut := m.fin + 1;
      detecte_mot (s, m);
    end;
END;
```

Remarque : pas besoin ici de tester si `v.nb <= tab_max` car le nombre maximum de mots est de 255/2, donc le tableau est toujours assez grand.

5. Sur 4 points.

```
PROCEDURE formate_ligne (f : text; s : ligne_t;
                         var nc : integer; max : integer);
VAR v : tab_t;
    i, j, lg : integer;
BEGIN
    analyse_ligne (s, v);

    for i := 1 to v.nb do
    begin
      lg := v.mot[i].fin - v.mot[i].debut + 1;

      if nc > 0
      then if nc+1+lg <= max
           then begin write (f, ' '); nc := nc+1; end
           else begin writeln (f);    nc := 0;    end;

      for j := v.mot[i].debut to v.mot[i].fin do write (f, s[j]);

      nc := nc + lg;
    end;
END;
```

6. Sur 3 points.

```
PROCEDURE formate_texte (f1, f2 : text; max : integer);
VAR s : ligne_t;
    nc : integer;
BEGIN
    nc := 0;
    while not eof(f1) do
    begin
      readln (f1, s);
      formate_ligne (f2, s, nc, max);
    end;
END;
```

7. Sur 3 points.

```
PROGRAM aligauche;
VAR
    f1, f2 : text;
    nom1, nom2 : ligne_t;
    max : integer;
BEGIN
    write ('Nom du fichier à formater : '); readln (nom1);
    write ('Nom du fichier résultat : '); readln (nom2);
    write ('Largeur max : '); readln (max);
    assign (f1, nom1); reset (f1);
    assign (f2, nom2); rewrite (f2);
    formate_texte (f1, f2, max);
    close (f1); close (f2);
END.
```

Faculté des Sciences de Luminy — 12 juin 2002
Deug 1 MASS MA — Durée 3 heures

Épreuve d'Informatique

Documents, calculettes et ordinateurs sont interdits.
Les exercices sont indépendants, à réaliser en langage Pascal.

1. Le jeu des erreurs *(4 points)*

Le programme suivant devrait afficher 2002, or il est truffé d'erreurs (il peut y en avoir plusieurs dans chaque ligne). Trouvez-les et proposez chaque fois une correction.

```
PROGRAM ('docteur');
TYPE t : [a, b, c, d];
BEGIN
  for v = (d .. a) do liteul (v);
END;
FUNCTION liteul (kevin)
BEGIN
  case kevin of
  begin
    b to c  : print ("0");
    a and d : print ("2");
  end;
END.
```

2. Programme mystère *(4 points)*

Faire le tableau de sortie du programme (c'est-à-dire le tableau des valeurs successives des variables et expressions), et en déduire ce que le programme affiche.

```
PROGRAM alenvers;
VAR f : TEXT; c, d : char; n, m : integer;
BEGIN
  n := 0;
  assign (f, 'les_gars.txt');
  reset (f);
  while not eof(f) do
  begin
    readln (f, c, m);
    n := 2*n + m; d := chr(ord(c)+n);
    write (d);
  end;
  close (f);
END.
```

Voici le fichier `'les_gars.txt'` :

```
M 2
V -4
B -1
T 0
A 5
```

3. Compter les BUTS *(4 points)*

Écrire une fonction `compter_buts` qui lit au clavier une suite de caractères terminée par `'.'` et qui renvoie le nombre d'occurences de quadruplets consécutifs `'B'`,`'U'`,`'T'`,`'S'` dans la suite. La fonction devra être capable de lire une suite de caractères de taille quelconque.

4. Législatives *(8 points)*

Au terme du premier tour des élections législatives 2002, la loi électorale stipule qu'un candidat ayant plus de 50% des *votes exprimés* est élu dès le premier tour. Dans le cas contraire un second tour a lieu, auquel seuls les candidats ayant rassemblé au moins 12.5% des *électeurs inscrits* peuvent participer.

4.a Déclarer un type enregistrement `candidat_t` comprenant les champs `nom`, `prenom`, `parti` (chaînes de caractères) et `nb_voix` (entier). Déclarer une constante `MAX_CANDI` de valeur 30, représentant le nombre maximum de candidats possibles dans une même circonscription. Déclarer un second type enregistrement `circonscription_t` comprenant les champs `nb_inscrits` (entier), `nb_exprimes` (entier), `tab_candi` (tableau de `candidat_t`) et `nb_candi` (entier).

4.b Écrire une procédure `affi_candidat` qui reçoit un candidat `can` et qui affiche sur une même ligne le contenu des 4 champs de `can`.

4.c Écrire une procédure `affi_un_score` qui reçoit une circonscription `cir` et un numéro de candidat `num`. La procédure affiche en se servant de `affi_candidat` les informations sur le candidat. Elle affiche ensuite les pourcentages réalisés par le candidat, d'une part par rapport aux suffrages exprimés, et d'autre part par rapport au nombre d'électeurs inscrits.

4.d Écrire une procédure `premier_tour` qui reçoit une circonscription `cir`. La procédure détermine si l'un des candidats est élu et affiche son score, sinon elle affiche le score de chacun des candidats admis au second tour.

Correction

Exercice 1 : 4 points

```
PROGRAM ('docteur');                suppr (' et ')
TYPE t : [a, b, c, d];              : -> =    [] -> ()
                                    manque VAR v : t;
BEGIN
  for v = (d .. a) do liteul (v);   = -> :=    suppr ()    .. -> downto
END;                                ; -> .

FUNCTION liteul (kevin)             FUNCTION -> PROCEDURE    manque :t
                                    manque ;    mettre avant prog princ
BEGIN
  case kevin of
  begin                             suppr begin
    b to c  : print ("0");          to -> ..    print -> write    " -> '
    a and d : print ("2");          and -> ,    print -> write    " -> '
  end;
END.                                . -> ;
```

Exercice 2 : 4 points

c	m	n	d
'M'	2	2	'0'
'V'	-4	0	'V'
'B'	-1	-1	'A'
'T'	0	-2	'R'
'A'	5	1	'B'

Le programme affiche `'OVARB'`, c'est-à-dire BRAVO à l'envers.

Exercice 3 : 4 points

```
FUNCTION compter_buts : integer;
VAR s : string[10];
    c : char;
    p, res : integer;
BEGIN
  p := 1; res := 0; s := 'BUTS';

  read (c);
  while c <> '.' do
  begin
    if c = s[p] then p := p+1 else p := 1;
    if p > 4 then begin p := 1; res := res + 1; end;
    read (c);
  end;

  compter_buts := res;
END;
```

Exercice 4 : 2 + 1 + 2 + 3 points

```
4.1 CONST
      MAX_CANDI = 30;

    TYPE
      candidat_t = Record
                     nom, prenom, parti : string[255];
                     nb_voix : integer;
                   End;

      circonscription_t = Record
                            nb_inscrits, nb_exprimes : integer;
                            tab_candi : array [1..MAX_CANDI] of candidat_t;
                            nb_candi : integer;
                          End;

4.2 PROCEDURE affi_candidat (can : candidat_t);
    BEGIN
      write (can.nom, ' ', can.prenom, ' (',
             can.parti, ') ', can.nb_voix, ' voix');
    END;

4.3 PROCEDURE affi_un_score (cir : circonscription_t; num : integer);
    VAR can : candidat_t;
    BEGIN
      can := cir.tab_candi[num];  { par commodité }
      affi_candidat (can);
      write (' ', can.nb_voix * 100.0 / cir.nb_exprimes, ' % exprimés ',
                  can.nb_voix * 100.0 / cir.nb_inscrits, ' % inscrits');
    END;

4.4 PROCEDURE premier_tour (cir : circonscription_t);
    VAR i : integer;
        trouve : boolean;
    BEGIN
      trouve := false;

      { Premier passage pour trouver s'il y a un élu }
      for i := 1 to cir.nb_candi do
      if cir.tab_candi[i].nb_voix * 100.0 / cir.nb_exprimes > 50.0
      then begin
             write ('Elu : ');
             affi_un_score (cir, i);
             writeln;
             trouve := true;
           end;

      { Second passage pour trouver les candidats au 2e tour }
      if not trouve then
      begin
        writeln ('Admis au second tour :');
        for i := 1 to cir.nb_candi do
        if cir.tab_candi[i].nb_voix * 100.0 / cir.nb_inscrits >= 12.5
        then begin
               affi_un_score (cir, i);
               writeln;
             end;
      end;
    END;
```

Faculté des Sciences de Luminy — 13 septembre 2002
Deug 1 MASS MA — Durée 3 heures

Épreuve d'Informatique

Documents, calculettes et ordinateurs sont interdits.

Problème La discothèque

Le problème est à réaliser en langage Pascal. Pour répondre à une question, vous avez le droit de vous servir des types ou d'appeler les procédures des questions précédentes, même si vous ne les avez pas traitées.

1. Types *(2 points)*

Déclarer une constante `MAX_CHANSON` à 30. Écrire un type enregistrement `chanson_t` comprenant les champs `titre` (une chaîne de 63 caractères) et `duree_sec` (un entier). Écrire un type enregistrement `album_t` comprenant les champs `liste` (un tableau de `chanson_t` indicé de 1 à `MAX_CHANSON`), `nb_chanson` et `annee` (des entiers), `titre` et `artiste` (des chaînes de 63 caractères).

2. Affichage d'une chanson *(1 point)*

Écrire une procédure `affi_chanson(c : chanson_t)` qui affiche sur une seule ligne le titre de la chanson `c` et sa durée en secondes, puis fait un saut de ligne.

3. Affichage d'un album *(2 points)*

Écrire une procédure `affi_album(a : album_t)` qui affiche le titre, l'artiste et l'année de l'album `a`, puis la liste des chansons de l'album. La liste des chansons est affichée en faisant appel à `affi_chanson`.

4. Durée totale d'un album *(3 points)*

Écrire une fonction `duree_totale(a : album_t)` qui renvoie sur un entier la durée totale en secondes de l'album `a`.

5. Album le plus long *(4 points)*

On introduit le type suivant pour mémoriser une discothèque :

```
CONST MAX_ALBUM = 200;
TYPE disco_t = Record
                 catalogue : array [1..MAX_ALBUM] of album_t;
                 nb_album : integer;
                 proprietaire : string[63];
               End;
```

Écrire une fonction `plus_long_album(d : disco_t)` qui renvoie sur un entier l'indice dans la discothèque `d` de l'album dont la durée totale est la plus longue.

6. Insertion d'un album *(3 points)*

Écrire et paramétrer correctement la fonction booléenne `inserer_album` qui reçoit en paramètres un album `a` et une discothèque `d`. La fonction `inserer_album` mémorise l'album `a` à la fin du catalogue de `d` et renvoie vrai si l'insertion est possible, sinon elle renvoie faux.

7. Fusion de discothèques *(5 points)*

Écrire la procédure `fusion_disco(d1, d2 : disco_t; var d3 : disco_t)`. On suppose que `d1` et `d2` sont triées sur l'année des albums et que `d3` n'est pas initialisée. La procédure `fusion_disco` recopie tous les albums de `d1` et `d2` dans `d3` en se servant de `inserer_album`, et en choisissant l'ordre de recopie afin que `d3` soit triée sur l'année des albums.

Correction

1. Sur 2 points.

```
CONST
    MAX_CHANSON = 30;

TYPE
    chanson_t = Record
                  titre : string[63];
                  duree_sec : integer;
                End;

    album_t = Record
                liste : array [1..MAX_CHANSON] of chanson_t;
                nb_chanson, annee : integer;
                titre, artiste : string[63];
              End;
```

2. Sur 1 point.

```
PROCEDURE affi_chanson (c : chanson_t);
BEGIN
  writeln (c.titre, ' - ', c.duree_sec, ' s');
END;
```

3. Sur 2 points.

```
PROCEDURE affi_album (a : album_t);
VAR i : integer;
BEGIN
  writeln ('Titre   :', a.titre);
  writeln ('Artiste :', a.artiste);
  writeln ('Année   :', a.annee);
  for i := 1 to a.nb_chanson do
    affi_chanson (a.liste[i]);
END;
```

4. Sur 3 points.

```
FUNCTION duree_totale (a : album_t) : integer;
VAR i, res : integer;
BEGIN
  res := 0;
  for i := 1 to a.nb_chanson do
    res := res + a.liste[i].duree_sec;
  duree_totale := res;
END;
```

5. Sur 4 points.

```
FUNCTION plus_long_album (d : disco_t) : integer;
VAR i, duree, duree_max, ind_max: integer;
BEGIN
  duree_max := 0; ind_max := 0;
  for i := 1 to d.nb_album do
  begin
    duree := duree_totale (d.catalogue[i]);
    if duree > duree_max then
      begin duree_max := duree; ind_max := i; end;
  end;
  plus_long_album := ind_max;
END;
```

6. Sur 3 points.

```
FUNCTION inserer_album (a : album_t; var d : disco_t) : boolean;
VAR ok : boolean;
BEGIN
  ok := true;
  if d.nb_album < MAX_ALBUM
  then begin
         d.nb_album := d.nb_album + 1;
         d.catalogue[d.nb_album] := a;
       end
  else ok := false;
  inserer_album := ok;
END;
```

7. Sur 5 points.

```
PROCEDURE fusion_disco (d1, d2 : disco_t; var d3 : disco_t);
VAR i1, i2 : integer;
    ok : boolean;
BEGIN
  d3.nb_album := 0;
  i1 := 1; i2 := 1; ok := true;

  while (i1 <= d1.nb_album) and (i2 <= d2.nb_album) and ok do
  if d1.catalogue[i1].annee <= d2.catalogue[i2].annee
  then begin
         ok := inserer_album (d1.catalogue[i1], d3);
         i1 := i1 + 1;
       end
  else begin
         ok := inserer_album (d2.catalogue[i2], d3);
         i2 := i2 + 1;
       end;

  while (i1 <= d1.nb_album) and ok do
  begin
    ok := inserer_album (d1.catalogue[i1], d3);
    i1 := i1 + 1;
  end;

  while (i2 <= d2.nb_album) and ok do
  begin
    ok := inserer_album (d2.catalogue[i2], d3);
    i2 := i2 + 1;
  end;
END;
```

Faculté des Sciences de Luminy — 12 juin 2003
Deug 1 MASS MA — Durée 3 heures

Épreuve d'Informatique

Documents, calculettes et ordinateurs sont interdits.
Les exercices sont indépendants, à réaliser en langage Pascal.

1. Le jeu des erreurs *(3 points)*

Le programme suivant devrait afficher 2003, or il est truffé d'erreurs (il peut y en avoir plusieurs dans chaque ligne). Trouvez-les et proposez chaque fois une correction.

```
PROGRAM
TYPE  chif_t := (deux, mille, trois);
BEGIN
  var chif : integer;

  for chif := deux to trois do
    case of
      chif = deux  :: write ('2');  end;
      chif = mille :: write ('00'); end;
      chif = trois :: write ('3');  end;
    end;
  end;

  writeln.
END.
```

2. Programme mystère *(4 points)*

Faire le tableau de sortie du programme (c'est-à-dire le tableau des valeurs successives des variables et expressions), et en déduire ce que le programme affiche.

```
PROGRAM clint;
VAR a, b, c : string[80]; d : integer;
BEGIN
  a := 'jkaihbcaeg'; b := 'wagontedos'; c := ''; d := 7;
  while d <> 0 do
    begin c := c + b[d]; d := ord(a[d]) - ord('a'); end;
  writeln (c);
END.
```

3. Entrelacement *(4 points)*

Écrire un programme `Entrelace` qui saisit le nom de 3 fichiers texte, puis recopie depuis le 1^{er} fichier les lignes de rang pair dans le $2^{ème}$ fichier et les lignes de rang impair dans le $3^{ème}$ fichier.

T.S.V.P.

4. Timbres poste *(9 points)*

Le problème des timbres poste, encore appelé problème de Frobenius, est le suivant : on suppose que l'on ne dispose, pour affranchir une lettre, que de 2 sortes de timbres, les uns de valeur a, les autres de valeur b (a et b sont entiers et strictement positifs, exprimés en centimes). On cherche à déterminer les tarifs d'affranchissement que l'on peut réaliser à l'aide de ces 2 sortes de timbres uniquement. Par exemple pour $a = 3$ et $b = 4$, on a souligné toutes les valeurs réalisables entre 1 et 10 : 1 2 $\underline{3}$ $\underline{4}$ 5 $\underline{6}$ $\underline{7}$ $\underline{8}$ $\underline{9}$ $\underline{10}$.

4.a On considère le type suivant :

```
CONST VMAX = 10000;
TYPE  vec_t = array [1..VMAX] of boolean;
```

Écrire une procédure `crible_poste` qui reçoit en paramètres un vecteur `v:vec_t`, une borne `vn:integer` telle que $1 \leq$ `vn` $\leq$ `VMAX`, et les valeurs de timbres `a,b:integer`. La procédure initialise toutes les cases de `v[1..vn]` à `false`, puis fait un crible sur les cases de `v[1..vn]` pour mettre à `true` les cases dont l'indice est une valeur réalisable.

Indication pour le crible : si une valeur x est réalisable, alors $x+a$ et $x+b$ sont aussi réalisables. En exploitant cette idée on peut marquer toutes les valeurs réalisables en un seul parcours du vecteur `v`.

4.b On appelle *conducteur* de a et b le plus petit entier k tel que tous les entiers $\geq k$ peuvent être réalisés. Dans l'exemple ci-dessus avec $a = 3$ et $b = 4$, on voit que $k = 6$.
Écrire une fonction `conducteur` qui reçoit en paramètres les valeurs des timbres `a,b:integer`, puis renvoie le conducteur de a et b. La fonction appelle la procédure `crible_poste` sur un vecteur `v`, puis cherche le plus grand entier k tel que la case $k - 1$ soit à `false` dans `v`, enfin vérifie que les cases de `v[k..k+min(a,b)-1]` sont toutes à `true`. Si ce dernier test réussit, la fonction renvoie `k`, sinon elle renvoie `-1` pour signifier que le conducteur n'a pu être trouvé.

4.c Quelles sont les raisons pour lesquelles la recherche du conducteur peut échouer ?

Le corrigé sera visible après l'épreuve sur : `http://www.lim.univ-mrs.fr/~thiel/ens/pascal`

Correction

Exercice 1 : 3 points

La difficulté vient du fait que l'énuméré et le for sont corrects !

```
PROGRAM                                      manque "titre;"
TYPE  chif_t := (deux, mille, trois);        ":=" -> "="
BEGIN
  var chif : integer;                        mettre cette ligne avant BEGIN
                                             "integer" -> "chif_t"
  for chif := deux to trois do
    case of                                  "case of" -> "case chif of"
      chif = deux  :: write ('2');  end;     supprimer "chif = "
      chif = mille :: write ('00'); end;     "::" -> ":"
      chif = trois :: write ('3');  end;     supprimer les "end;"
    end;
  end;                                       supprimer cette ligne

  writeln.                                   "." -> ";"
END.
```

Exercice 2 : 4 points

d	b[d]	c := c + b[d]	a[d]	d := ord(a[d])-ord('a')
7	'e'	'e'	'c'	2
2	'a'	'ea'	'k'	10
10	's'	'eas'	'g'	6
6	't'	'east'	'b'	1
1	'w'	'eastw'	'j'	9
9	'o'	'eastwo'	'e'	4
4	'o'	'eastwoo'	'i'	8
8	'd'	'eastwood'	'a'	0

Le programme affiche : `eastwood` (d'où le nom du programme).

Exercice 3 : 4 points

Attention : si on fait 2 `read` dans la même boucle il faut chaque fois tester `eof`. La solution proposée évite ce piège.

```
PROGRAM Entrelace;
VAR
  f1, f2, f3 : text;
  n1, n2, n3, ligne : string[255];
  i : integer;
BEGIN
  write ('Fichier 1 : '); readln (n1);
  write ('Fichier 2 : '); readln (n2);
  write ('Fichier 3 : '); readln (n3);
  assign (f1, n1); assign (f2, n2); assign (f3, n3);
  reset (f1); rewrite (f2); rewrite (f3);

  i := 0;
  while not eof(f1) do
  begin
    readln (f1, ligne);       { toujours readln avec un string }
    i := i+1;
    if i mod 2 = 1
    then writeln (f2, ligne)
    else writeln (f3, ligne);
  end;

  close (f1); close (f2); close (f3);
END.
```

Exercice 4 : $4 + 4 + 1$ points

4.a Attention : à `var v` dans l'entête, et au respect de la borne `vn` dans le corps.

```
PROCEDURE crible_poste (var v : vec_t; vn, a, b : integer);
VAR i : integer;
BEGIN
  { Init crible }
  for i := 1 to vn do v[i] := false;
  if a <= vn then v[a] := true;
  if b <= vn then v[b] := true;

  { Crible }
  for i := 1 to vn do if v[i] then
  begin
    if i+a <= vn then v[i+a] := true;
    if i+b <= vn then v[i+b] := true;
  end;
END;
```

4.b Remarque : il existe une formule directe : $k = ab - a - b + 1$.

```
FUNCTION conducteur (a, b : integer) : integer;
VAR v : vec_t; k, i, j1, j2 : integer;
BEGIN
  crible_poste (v, VMAX, a, b);

  { Recherche conducteur }
  k := 1;                       { init : tous les entiers sont réalisables }
  for i := 2 to VMAX do
    if not v[i-1] then k := i;  { i-1 n'est pas réalisable }

  { Vérification }
  j1 := k;
  if a < b then j2 := k+a-1 else j2 := k+b-1;
  if j2 > VMAX then k := -1
  else for i := j1 to j2 do
         if not v[i] then k := -1;

  conducteur := k;
END;
```

4.c La recherche du conducteur peut échouer si `v` est trop petit, soit pour contenir le conducteur, soit pour que la vérification puisse être entièrement faite.
La recherche échoue en particulier si le conducteur a une valeur infinie, ce qui est le cas si `a` et `b` ne sont pas premiers entre eux. Par exemple si $a = 6$ et $b = 8$, aucun entier impair n'est réalisable, donc le conducteur n'a pas une valeur finie.

Septembre 2003 : à venir ..