Chapitre 5

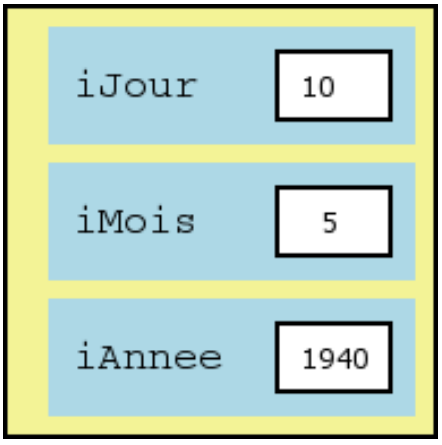
Structures

5.1 Utilité des enregistrements ou structures

Nous avons abordé précédemment les tableaux, qui facilitent les traitements répétitifs. Il existe d'autres structures complexes aussi nécessaires. Si on veut représenter une personne, dans un programme, il sera nécessaire de mémoriser, par exemple, son nom, son prénom, son adresse et sa date de naissance. Pour chacune de ces données atomiques, on pourra prévoir une variable :

```
Variable Chaîne sNom
Variable Chaîne sPrenom
Variable Chaîne sAdresse
Variable Entière iJour
Variable Entière iMois
Variable Entière iAnnee
```

Ce genre de déclarations occulte le fait que les trois dernières données forment une date et que l'ensemble de toutes ces données constitue un tout qui représente une personne. C'est ici qu'intervient la notion de *structure* parfois appelée aussi *enregistrement*.




```
Définition Structure eDate
  Champ Entier iJour
  Champ Entier iMois
  Champ Entier iAnnee
Fin de structure
```

5.2 Définitions des types et des variables structurées

Généralement, les structures sont employées plusieurs fois au sein d'un même programme, dans des variables différentes, voire même répétées dans des tableaux. On distingue donc généralement la définition d'un type de structure, sorte de moule, et la déclaration de variables structurées, réalisées à l'aide du moule.

5.2.1 Déclaration d'une structure

Une structure porte un nom (commençant par `e` comme enregistrement [1]) et se compose de *champs*, qui constituent des sortes de variables imbriquées dans la structure. Ces variables ne seront pas directement accessibles au programme, elles font partie d'une structure. Il est même possible de donner à un champ le nom d'une autre variable (bien que ce ne soit pas toujours à conseiller). Les champs peuvent être de n'importe quel type, en ce compris des tableaux et des structures .

```
Définition structure Nom_Type
    Champ Type Nom_Champ
    Champ Type Nom_Champ
Fin de structure
```

```
/* Exemple : une date */
Définition structure eDate
   Champ entier iJour
   Champ entier iMois
   Champ entier iAnnee
Fin structure
```

5.2.2 Déclaration d'une variable structurée

Comme il vient d'être dit, une structure est un simple moule à variables. Pour créer une variable, nous allons utiliser le nouveau type structuré que nous avons défini. Par exemple, nous déclarerons trois variables `uAujourdhui`, `uDemain`, `uHier`, de type `eDate`. Comme on le constate, ces variables ont un préfixe `u`, comme utilisateur.

```
Variables eDate uAujourdhui, uDemain, uHier
```

5.2.3 Déclarations de types complexes

Un champ peut être d'un type quelconque, entier, flottant, logique, caractère ou chaîne. Le champ peut également être un tableau voire même une variable structurée d'un type préalablement défini. Voici quelques exemples :

```
/* Structure contenant une variable structurée */
Définition Structure ePersonne
 Champ Chaîne #20 sNom[2]
 Champ Chaîne #20 sPrenom
 Champ Chaîne #40 sAdresse
 Champ eDate uDNaissance
Fin de structure
```

1. Rappelons que cette habitude que nous avons de préfixer nos variables n'est qu'une contrainte volontairement acceptée. Le compilateur laf2c est le seul à l'exiger. Nous verrons qu'en programmation objet quand nous aurons des dizaines de classes/types différents, cette habitude deviendra très utile. Il en est de même en programmation Windows. A cet égard, la documentation et les exemples fournis par Microsoft sont exemplaires.
2. Dans le but de ne pas gaspiller d'espace, je propose ici de donner une taille explicite à nos chaînes.

```
/* Structure contenant une structure et un tableau */
Définition structure eEtudiant
  Champ ePersonne uData
  Tableau flottants fNotes Taille 10
Fin de structure
```

Une autre manière de créer des structures complexes consiste à créer des tableaux de structures :

```
/* Tableau de dates */
Tableau eDate uJoursFeries Taille 20
/* Tableau d'étudiants */
Tableau eEtudiant uTClasse Taille 30
```

5.2.4 Initialisation des variables structurées

Il est parfois intéressant de disposer d'un moyen rapide d'initialiser des variables structurées. Nous reprendrons la technique déjà vue pour les tableaux :

```
Variable eDate uGuerre40 <* ={10,5,1940} *>
```

5.3 Utilisation des structures

Chaque champ d'une variable structurée est accessible par le nom de la variable structurée combiné au nom du champ. Dans la plupart des langages, on emploie un point pour séparer les identificateurs. Les trois champs de la variable `uGuerre40` de type `eDate` sont respectivement `uGuerre40.iJour`, `uGuerre40.iMois` et `uGuerre40.iAnnee`.

```
Variables ePersonne uP1, uP2
Tableau ePersonne uFamille Taille 10
Variable eDate uD1, uD2
```

Les variables structurées ressemblent aux tableaux par le fait qu'elles regroupent plusieurs données dans une seule variable. Elles ne permettent pas un accès direct aux données :

- impossibilité de lire ou d'afficher une variable structurée, on est obligé de manipuler chaque champ isolément :

```
uP1.sNom <- "Dupont"
uP1.sPrenom <- "Jean"
uP1.sAdresse <- "rue de l'Eglise 5"
uP1.uDNaissance.iJour <- 10
uP1.uDNaissance.iMois <- 5
uP1.uDNaissance.iAnnee <- 1940
```

  On notera les accès aux champs de la date de naissance qui utilisent deux fois le point : nous avons en effet deux structures imbriquées.
- impossibilité de les comparer (ici encore, il faut comparer les champs correspondants isolément) :

```
Si uP1.sNom==uP2.sNom ET uP1.Prenom==uP2.Prenom ET
   uP1.sAdresse==uP2.sAdresse alors
   Afficher "les deux personnes sont identiques"
Fin de si
```

Les variables structurées se distinguent néanmoins fortement des tableaux par d'autres aspects :

- les différents champs peuvent être de types distincts (ils le sont très souvent) ;
- le parcours des différents champs ne peut se faire à l'aide d'une boucle, puisque chaque champ possède un nom imprévisible ;
- la variable peut se voir affecter la valeur d'une autre variable du même type en une seule opération [3]. Nous verrons également qu'une fonction peut renvoyer une valeur affectable à une variable structuré.

```
uP1 <- uP2
```

5.4 Exemple complet

Je vais illustrer l'usage des structures dans un exemple qui se rapproche de la vie réelle. J'ai volontairement limité l'exemple afin de ne pas le rendre illisible. Voici le cahier des charges :

1. un professeur veut pouvoir encoder le nom de ses étudiants et les différentes notes obtenues lors de différents contrôles. On prévoit `MAXETUD` étudiants et pour chacun d'eux `MAXINTERRO` notes.
2. à tout moment, on doit pouvoir afficher la liste des étudiants et la moyenne des points obtenus. On suppose que le total est toujours identique (il n'y a donc pas de pondération).
3. Il doit être possible d'ajouter un étudiant.

5.4.1 Structure générale du programme

On va prévoir une boucle proposant un menu et réagissant au choix de l'utilisateur.

```
/* == Gestion d'un cahier de notes ==*/
Variable caractère cOption
/* Autres variables à ajouter */

Début du traitement

   /* Données initiales */

   /* Boucle principale */
   Initialisation
   Répéter
      Effacer écran
      Afficher
         "1. [A]fficher la moyenne de chaque étudiant\n" &
         "2. [E]ncoder un nouvel un étudiant \n" &
         "3. [Q]uitter \n\n"
      Moteur lire cOption
      Examen si cOption
         vaut '1','a','A'
            /* ======================================== */
```

3. L'affectation d'une variable structurée à une autre est possible, mais non la comparaison entre deux variables. C'est dû au fait que pour des raisons d'alignement, les champs sont parfois séparés par un ou plusieurs octets non significatifs. Lors de la copie, les octets non utilisés sont copiés sans dommage, par contre, ces mêmes octets dont la valeur est souvent aléatoire, empêche une comparaison correcte entre deux variables structurées.

```
            /* Afficher les moyennes */
            /* ======================================== */
         fin de cas
         vaut '2','e','E'
            /* ======================================== */
            /* Ajouter un étudiant */
            /* ======================================== */
         fin de cas
         vaut '3','q','Q'
            cOption <- 'q'
         fin de cas
         defaut
            Afficher " Option non reconnue"
         fin de cas
      fin examen
      Attente
   Boucler tant que cOption <> 'q'

Fin du traitement
```

Tel qu'il est écrit ce premier embryon de programme peut déjà fonctionner. On notera la gestion de la sortie, qui se produit si l'on tape '3' ou 'q' ou 'Q'. L'instruction d'attente en fin de boucle permet de lire l'écran avant son effacement.

5.4.2 Données relatives au problème

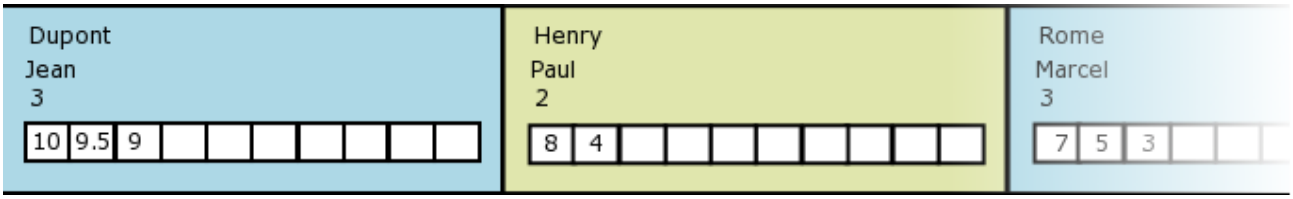
Il faut commencer par définir les deux constantes limitant la taille de la classe et la taille du cahier de notes. Le cahier de charges a spécifié leurs valeurs.

```
Constante MAXETUD = 20
Constante MAXINTERRO = 10
```

Chaque étudiant se verra représenté par une variable structurée reprenant son nom, son prénom et un tableau de flottants reprenant ses différentes notes. Comme toutes les cases du tableau ne seront pas nécessairement utilisées, il importe de prévoir un champ supplémentaire pour mémoriser le nombre de contrôles notés.

```
Définition structure ePersonne
   Champ Chaine #20 sNom
   Champ Chaine #20 sPrenom
   Champ Entier iNbInterros
   Tableau flottant fNotes taille MAXINTERRO
fin de structure
```

L'ensemble des étudiants peut se représenter à l'aide d'un tableau de personnes (`uTClasse`). Ici encore, nous allons utiliser une variable, pour indiquer le nombre d'éléments du tableau qui contiennent des données (`iNbEtudiants`).



```
Variable entière iNbEtudiants
Tableau ePersonne uTClasse taille MAXETUD
<* = {{"DUPONT","Jean",3,{10.,9.5,9.}},
      {"HENRY","Paul",2,{8.0,4.0}},
      {"ROME","Marcel",3,{7.,5.,3.}}} *>
```

Nous aurons besoin de quatre variables de travail (voir plus loin).

```
Variables indices iE,iN, iNouvelEtudiant
Variable flottante fTotal
```

5.4.3 L'affichage de la classe et le calcul de la moyenne

L'affichage des étudiants et le calcul de leur moyenne va comporter deux boucles imbriquées :

- une première boucle va parcourir le tableau des étudiants (`uTClasse`). On va l'exécuter `iNbEtudiants` fois. Pour chacun d'entre eux, on va afficher son nom (`uTClasse[iI].sNom`) et son prénom (`uTClasse[iI].sPrenom`). Le nombre d'itérations est connu (donc boucle avec compteur).
- une deuxième boucle va s'exécuter autant de fois que l'étudiant a fait d'interros (`uTClasse[iI].iNbInterro`), pour autant qu'il en ait. On y calcule la somme des points de l'étudiant, ce qui permet d'afficher la moyenne à la sortie de la boucle.

```
/* Afficher les moyennes */
/* ======================================== */
Afficher "Liste des étudiants" ~|
Afficher "-----------------------------------" ~|
/* Boucle : traiter chaque étudiant*/
Initialisation
Compter avec iE de 0 à iNbEtudiants-1
   /* Afficher nom et prénom */
   Afficher uTClasse[iE].sNom tab
   Afficher uTClasse[iE].sPrenom tab
   /* Calcul et affichage de la moyenne */
   /* Alternative : si pas d'interro, rien faire*/
   Si uTClasse[iE].iNbInterros==0 Alors
      Afficher "Pas d'interrogation"
   Sinon
      /* Boucle : additionner les notes de l'étudiant */
      Initialisation
         fTotal <- 0
      Compter avec iN de 0 à uTClasse[iE].iNbInterros -1
         fTotal <- fTotal + uTClasse[iE].fNotes[iN]
      Fin de compter
      Afficher "Moyenne obtenue "
      Afficher fTotal/uTClasse[iE].iNbInterros
   Fin de si
   Fin de ligne
   Afficher "-----------------------------------\n"
Fin de compter
```

5.4.4 L'encodage d'un nouvel étudiant

L'encodage est beaucoup plus simple puisque qu'on ne travaille que sur une seule variable structurée. Il faut néanmoins éviter deux écueils :

– encoder un étudiant quand le tableau `uTClasse` est complètement rempli.
– encoder plus de notes que prévus

Deux tests sont prévus pour éviter ces erreurs. Dans le premier cas, on envoie un petit message d'erreur. Dans le second, on ramène discrètement [4] le nombre choisi par l'utilisateur à la valeur maximale.

```
/* ======================================== */
/* Ajouter un étudiant */
/* ======================================== */
Si iNbEtudiants == MAXETUD alors
   Afficher "Le nombre maximum d'étudiants a été atteint"
sinon
   iNouvelEtudiant <- iNbEtudiants
   Incrémenter iNbEtudiants
   Afficher "Nom du nouvel étudiant: "
   Lire uTClasse[iNouvelEtudiant].sNom
   Afficher "Prénom du nouvel étudiant: "
   Lire uTClasse[iNouvelEtudiant].sPrenom
   Afficher "Nombre d'interros à encoder:"
   Lire uTClasse[iNouvelEtudiant].iNbInterros
   /* Ne pas dépasser la limite */
   Si uTClasse[iNouvelEtudiant].iNbInterros>MAXINTERRO alors
      uTClasse[iNouvelEtudiant].iNbInterros<-MAXINTERRO
   Fin de si
   /* Boucle : lire les notes du nouvel étudiant */
   Initialisation
   Compter avec iN de 0 à uTClasse[iNouvelEtudiant].iNbInterros -1
      Afficher "Interro suivante :"
      Lire uTClasse[iNouvelEtudiant].fNotes[iN]
   Fin de compter
Fin de si
```

5.4.5 Version finale et complète

```
/* == Gestion d'un cahier de notes ==*/
Constante MAXETUD = 20
Constante MAXINTERRO = 10
Définition structure ePersonne
   Champ Chaine #20 sNom
   Champ Chaine #20 sPrenom
   Champ Entier iNbInterros
   Tableau flottant fNotes taille MAXINTERRO
fin de structure
Variable entière iNbEtudiants
Tableau ePersonne uTClasse taille MAXETUD
<* = {{"DUPONT","Jean",3,{10.,9.5,9.}},
      {"HENRY","Paul",2,{8.0,4.0}},
```

4. Trop discrètement sans doute. Dans un programme mieux fini, il faudrait prévenir l'utilisateur. S'il avait réellement 11 notes à encoder, il risque d'être surpris à la fin de ne pas pouvoir encoder la dernière note.

```
      {"ROME","Marcel",3,{7.,5.,3.}}} *>
Variables indices iE,iN, iNouvelEtudiant
Variable flottante fTotal
Variable caractère cOption

Début du traitement

   /* Données initiales */
   iNbEtudiants<-3;
   /* Boucle principale */
   Initialisation
   Répéter
     Effacer écran
     Afficher
        "1. [A]fficher la moyenne de chaque étudiant\n" &
         "2. [E]ncoder un nouvel un étudiant \n" &
         "3. [Q]uitter \n\n"
     Moteur lire cOption
     Examen si cOption
         vaut '1','a','A'
            /* ======================================= */
            /* Afficher les moyennes */
            /* ======================================= */
            Afficher "Liste des étudiants" ~|
            Afficher "-------------------------------------" ~|
            /* Boucle : traiter chaque étudiant*/
            Initialisation
            Compter avec iE de 0 à iNbEtudiants-1
               /* Afficher nom et prénom */
               Afficher uTClasse[iE].sNom tab
               Afficher uTClasse[iE].sPrenom tab
               /* Calcul et affichage de la moyenne */
               /* Alternative : si pas d'interro, rien faire*/
               Si uTClasse[iE].iNbInterros==0 Alors
                  Afficher "Pas d'interrogation"
               Sinon
                  /* Boucle : additionner les notes de l'étudiant */
                  Initialisation
                     fTotal <- 0
                  Compter avec iN de 0 à uTClasse[iE].iNbInterros -1
                     fTotal <- fTotal + uTClasse[iE].fNotes[iN]
                  Fin de compter
                  Afficher "Moyenne obtenue "
                  Afficher fTotal/uTClasse[iE].iNbInterros
               Fin de si
               Fin de ligne
               Afficher "-------------------------------------"
               Fin de ligne
            Fin de compter

         fin de cas
         vaut '2','e','E'
            /* ======================================= */
            /* Ajouter un étudiant */
            /* ======================================= */
            Si iNbEtudiants == MAXETUD alors
               Afficher "Le nombre maximum d'étudiants a été atteint"
            sinon
               iNouvelEtudiant <- iNbreEtudiants
               Incrémenter iNbEtudiants
```

```
                Afficher "Nom du nouvel étudiant: "
                Lire uTClasse[iNouvelEtudiant].sNom
                Afficher "Prénom du nouvel étudiant: "
                Lire uTClasse[iNouvelEtudiant].sPrenom
                Afficher "Nombre d'interros à encoder:"
                Lire uTClasse[iNouvelEtudiant].iNbInterros
                Si uTClasse[iNouvelEtudiant].iNbInterros>MAXINTERRO alors
                   uTClasse[iNouvelEtudiant].iNbInterros<-MAXINTERRO
                Fin de si
                /* Boucle : lire les notes du nouvel étudiant */
                Initialisation
                Compter avec iN de 0 à uTClasse[iNouvelEtudiant].iNbInterros -1
                   Afficher "Interro suivante :"
                   Lire uTClasse[iNouvelEtudiant].fNotes[iN]
                Fin de compter
             Fin de si
          fin de cas
          vaut '3','q','Q'
             cOption <- 'q'
          fin de cas
          defaut
             Afficher " Option non reconnue"
          fin de cas
       fin examen
       Attente
   Boucler tant que cOption <> 'q'
Fin du traitement
```

Le programme tel qu'on vient de le lire possède trois défauts graves :

- il ne permet pas de mémoriser les données d'une exécution à l'autre (il faudrait pour cela disposer d'une possibilité d'accès aux fichiers, qui sort des limites de ce cours de programmation).
- si l'utilisateur peut ajouter un étudiant et encoder ses points, on n'a pas prévu d'ajouter une nouvelle note à un étudiant, voire à tous les étudiants. Cette possibilité aurait compromis encore un peu plus la lisibilité du programme.
- il faut convenir que le programme devient difficile à lire, notamment parce que, lors du calcul de la moyenne, on a imbriqué trois boucles. Nous verrons dans un prochain chapitre comment rendre le traitement modulaire et partant plus lisible.

5.5 Un algorithme vraiment utile

Dans la vie courante d'une entreprise, on aura souvent besoin des coordonnées d'un client, pour lui envoyer une facture ou une lettre annonçant la disponibilité d'un article ou encore une promotion spéciale. Comment faire pour retrouver le client parmi tous les autres ? Bien sûr, on disposera sans doute d'un fichier qui contiendra ces données [5]. En réalité, la gestion d'un fichier ressemblera, à bien des égards à celle d'un tableau. Pour trouver le client, nous avons deux possibilités :

- soit rechercher le client par son nom, une technique simple, mais qui posera le problème de l'homonymie et nous obligera à des vérifications basées sur le prénom et/ou l'adresse
- soit rechercher par le numéro de client, que nous trouverons dans la mémoire personnelle du client, sur sa carte de fidélité ou par un moyen externe quelconque.

5. En fait, il est plus vraisemblable qu'on dispose d'une base de données pour contenir les données de l'entreprise, mais cela c'est une autre histoire et surtout un autre cours...

5.5.1 La recherche « dichotomique »

Les méthodes de recherche que nous avons envisagées jusqu'ici ne se révélaient pas trop efficaces. En moyenne, pour trouver une donnée parmi *N* données, nous étions obligés d'en lire la moitié. En pratique, cela signifie par exemple que si un programme basé sur un tel algorithme trouve les données propres à un client en une seconde, il mettra 10 secondes si le nombre de clients est multiplié par 10. Aucun commerçant n'acceptera de prendre le risque d'utiliser un programme qui l'empêcherait d'envisager un accroissement de sa société.

Une technique plus efficace est envisageable, à condition de disposer de données préalablement triées[6]. On va se limiter à lire la donnée située au milieu du tableau et selon le cas, éliminer la partie gauche ou droite du tableau selon que la valeur lue est inférieure ou supérieure à la valeur cherchée. Nous utiliserons donc un tableau virtuel, caractérisé par son indice minimal (`iMin`) et son indice maximal (`iMax`).

Voici une version correcte de l'algorithme :

```
Constante MAX = 10
Definition structure eClient
   champ entier iNumeroClient
   champ chaine # 20 sNom
   champ chaine # 50 sAdresse
fin de structure

Tableau eClient uTClientele taille MAX
<* = {
      {10,"Dupont","adresse 10"},
      {20,"Durant","adresse 20"},
      {30,"Dupneu","adresse 30"},
      {40,"Dumortier","adresse 40"},
      {50,"Duchemin","adresse 50"},
      {60,"Durieux","adresse 60"},
      {70,"Duval","adresse 70"},
      {80,"Dubreuil","adresse 80"},
      {90,"Dupanloup","adresse 90"},
      {100,"Dumont","adresse 100"}
      }
*>
Variable indice iMin, iMilieu, iMax
Variable entière iNumero
Début du traitement

   Afficher "Numéro du client à rechercher "
   Lire iNumero
   /* Boucle : */
   Initialisation
      Min <- 0
      iMax <- iDernier-1
   Tant que
      iMin <> iMax
   Répéter
      iMilieu <- (iMin+iMax)/2
      afficher iMin & "-" &iMilieu & "-" & iMax & " ** "
```

6. Pour les besoins de la démonstration, nous prendrons des numéros espacés de 10.

```
      Moteur
          /* Alternative : */
          Si uTClientele[iMilieu].iNumeroClient< iNumero Alors
             iMin <- iMilieu + 1
          Sinon
             iMax <- iMilieu
          Fin de si
   Fin de boucle
   /* Alternative : */
   Si iMilieu < iDernier et uTClientele[iMin].iNumeroClient == iNumero Alors
         afficher iNumero & " correspond à  " &
         uTClientele[iMin].sNom & "\n"
   Sinon
         afficher iNumero & " n'existe pas\n"
   Fin de si
Fin du traitement
```

5.5.2 Variantes incorrectes

Assez curieusement, cet algorithme ne s'écrit pas toujours facilement. Il présente trois points de variation possibles qui peuvent amener des bouclages infinis si on ne fait pas attention :

- le test `iMin<>iMax` ne garantit pas toujours la fin de l'exécution de la boucle. En effet, si `iMax` devient accidentellement plus petit que `iMin`, la condition n'est pas remplie, mais on se trouve dans un tableau dont le début est situé après la fin.
- l'instruction `iMin <- iMilieu + 1` ne peut pas s'écrire `iMin <- iMilieu`. Dans ce cas, il arrive que les deux limites ne soient pas égales mais que `iMilieu` reste identique à lui-même. On répète alors le test précédent et on boucle. Par exemple, pour les valeurs 5 et 6 de `iMin` et `iMax`, `iMilieu` vaudra 5. Si la case 5 ne contient pas la valeur recherchée, on bouclera à l'infini. Le +1 est donc obligatoire.
- à l'inverse, on pourrait vouloir écrire `iMax <- iMilieu - 1` au lieu de `iMax <- iMilieu`. Toujours avec notre exemple de 5 et 6, `iMilieu` vaudra 5 et au cas où la case 5 ne contiendrait pas la valeur cherchée, on donnera à `iMax` la valeur 4, qui est plus petite que celle de `iMin`.

Tous ces cas pathologiques viennent du fait que la division entière nécessaire au calcul de `iMilieu` n'est pas toujours exacte et que le milieu qu'elle prend n'est donc pas tout-à-fait au milieu. La taille du tableau va influer sur le fonctionnement du programme, ce qui rend les tests particulièrement peu instructifs : un programme qui donne de bons résultats avec un tableau de taille 10 peut très bien boucler avec un tableau de taille 8. On voit ici que les tests ne peuvent servir qu'à prouver que le programme est mal écrit.

Une variante du programme, dans laquelle nous introduisons une boucle externe pour tester toutes les valeurs plausibles dans l'intervalle des numéros de clients, nous permettra d'examiner l'impact des changements dans le calcul de `iMilieu`.

A titre d'exemple, voici ce que donne l'exécution du programme avec un tableau de 5 nombres (respectivement 10,20,30,40,50). Le programme principal recherche systématiquement les multiples de 5 de 5 à 55.

Condition iMin<>iMax	iMin <- iMilieu + 1	iMax <- iMilieu

C'est ici le fonctionnement normal du programme.

```
[0 < 2 > 4]  [0 < 1 > 2]  [0 < 0 > 1]  Nombre 5 non trouvé
[0 < 2 > 4]  [0 < 1 > 2]  [0 < 0 > 1]  Nombre 10 trouvé en 0
[0 < 2 > 4]  [0 < 1 > 2]  [0 < 0 > 1]  Nombre 15 non trouvé
[0 < 2 > 4]  [0 < 1 > 2]  [0 < 0 > 1]  Nombre 20 trouvé en 1
[0 < 2 > 4]  [0 < 1 > 2]  Nombre 25 non trouvé
[0 < 2 > 4]  [0 < 1 > 2]  Nombre 30 trouvé en 2
[0 < 2 > 4]  [3 < 3 > 4]  Nombre 35 non trouvé
[0 < 2 > 4]  [3 < 3 > 4]  Nombre 40 trouvé en 3
[0 < 2 > 4]  [3 < 3 > 4]  Nombre 45 non trouvé
[0 < 2 > 4]  [3 < 3 > 4]  Nombre 50 trouvé en 4
[0 < 2 > 4]  [3 < 3 > 4]  Nombre 55 non trouvé
```

Condition iMin<>iMax	iMin <- iMilieu	iMax <- iMilieu

La modification de la première instruction entraîne un bouclage infini dans la presque totalité des cas [7].

```
[0 < 2 > 4]  [0 < 1 > 2]  [0 < 0 > 1]  Nombre 5 non trouvé
[0 < 2 > 4]  [0 < 1 > 2]  [0 < 0 > 1]  Nombre 10 trouvé en 0
[0 < 2 > 4]  [0 < 1 > 2]  [0 < 0 > 1]  [0 < 0 > 1]  Bouclage
[0 < 2 > 4]  [0 < 1 > 2]  [0 < 0 > 1]  [0 < 0 > 1]  Bouclage
[0 < 2 > 4]  [0 < 1 > 2]  [1 < 1 > 2]  Bouclage
[0 < 2 > 4]  [0 < 1 > 2]  [1 < 1 > 2]  Bouclage
[0 < 2 > 4]  [2 < 3 > 4]  [2 < 2 > 3]  [2 < 2 > 3]  Bouclage
[0 < 2 > 4]  [2 < 3 > 4]  [2 < 2 > 3]  [2 < 2 > 3]  Bouclage
[0 < 2 > 4]  [2 < 3 > 4]  [3 < 3 > 4]  Bouclage
[0 < 2 > 4]  [2 < 3 > 4]  [3 < 3 > 4]  Bouclage
[0 < 2 > 4]  [2 < 3 > 4]  [3 < 3 > 4]  Bouclage
```

Condition iMin<>iMax	iMin <- iMilieu+1	iMax <- iMilieu-1

Le programme ainsi modifié ne boucle pas, mais il ne trouve pas le nombre 30, qui se trouve pourtant dans le tableau.

```
[0 < 2 > 4]  [0 < 0 > 1]  Nombre 5 non trouvé
[0 < 2 > 4]  [0 < 0 > 1]  Nombre 10 trouvé en 0
[0 < 2 > 4]  [0 < 0 > 1]  Nombre 15 non trouvé
[0 < 2 > 4]  [0 < 0 > 1]  Nombre 20 trouvé en 1
[0 < 2 > 4]  [0 < 0 > 1]  Nombre 25 non trouvé
[0 < 2 > 4]  [0 < 0 > 1]  Nombre 30 non trouvé
[0 < 2 > 4]  [3 < 3 > 4]  Nombre 35 non trouvé
[0 < 2 > 4]  [3 < 3 > 4]  Nombre 40 trouvé en 3
[0 < 2 > 4]  [3 < 3 > 4]  Nombre 45 non trouvé
[0 < 2 > 4]  [3 < 3 > 4]  Nombre 50 trouvé en 4
[0 < 2 > 4]  [3 < 3 > 4]  Nombre 55 non trouvé
```

Si on augmente la taille du tableau, on constate que le nombre 50 n'est pas trouvé non plus, alors qu'il l'était dans un tableau de 5 nombres, preuve que la taille du tableau influence les résultats.

7. . J'ai ajouté une variable `iOld` pour mémoriser les valeurs précédentes de `iMilieu`. Quand `iMilieu` reste identique d'une itération à l'autre, la boucle s'interrompt et une valeur particulière est affectée à `iMin`.

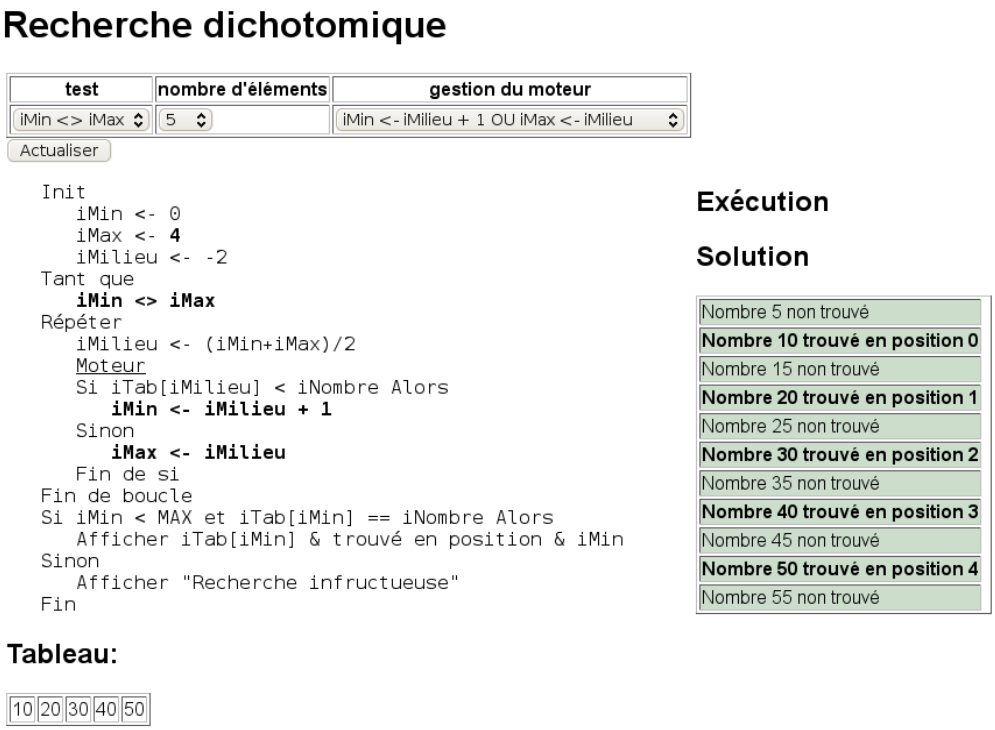
```
[0 < 2 > 5]   [0 < 0 > 1]   Nombre 5 non trouvé
[0 < 2 > 5]   [0 < 0 > 1]   Nombre 10 trouvé en 0
[0 < 2 > 5]   [0 < 0 > 1]   Nombre 15 non trouvé
[0 < 2 > 5]   [0 < 0 > 1]   Nombre 20 trouvé en 1
[0 < 2 > 5]   [0 < 0 > 1]   Nombre 25 non trouvé
[0 < 2 > 5]   [0 < 0 > 1]   Nombre 30 non trouvé
[0 < 2 > 5]   [3 < 4 > 5]   Nombre 35 non trouvé
[0 < 2 > 5]   [3 < 4 > 5]   Nombre 40 trouvé en 3
[0 < 2 > 5]   [3 < 4 > 5]   Nombre 45 non trouvé
[0 < 2 > 5]   [3 < 4 > 5]   Nombre 50 non trouvé
[0 < 2 > 5]   [3 < 4 > 5]   Nombre 55 non trouvé
[0 < 2 > 5]   [3 < 4 > 5]   Nombre 60 trouvé en 5
[0 < 2 > 5]   [3 < 4 > 5]   Nombre 65 non trouvé
```

En conclusion, l'examen de cet algorithme nous montre plusieurs choses :

- il est très délicat de mettre au point un algorithme, même si l'idée de base paraît claire ;
- l'intuition ne sert pas à grand-chose pour évaluer la correction d'un algorithme ;
- les tests sont trompeurs, puisque la taille du tableau influence les résultats. On peut, avec un peu de malchance, essayer des valeurs qui semblent justifier le bon fonctionnement du programme et croire à tort qu'il est correct.

5.5.3 Expériences et tentative de preuve

On pourra, si on le désire tester différentes taille de tableaux et les variantes de programmations sur *http://thoorens.net/Exec/dicho.php* .



Je vais tenter d'expliquer pourquoi la nouvelle de valeur de `iMilieu`. Tout tourne autours de la division par 2. Selon que les nombres sont pairs ou impairs, de subtiles variations vont intervenir. La première partie du tableau qui suit examine les quatre cas possibles pour les valeurs de `iMin` et `iMax` : dans le cas 1, les deux nombres sont pairs, dans le cas 4, les nombres sont impairs, et dans les cas 2 et 3, nous avons un pair et un impair. *a* et *b* représente des valeurs quelconques, entières et positives ou nulles. On voit en fait dans le calcul de `iMilieu`, que

seul le cas 4 prend en compte le nombre 1 des valeurs impaires. Dans le cas 1, il n'y a pas de valeur 1 et dans les 2 et 3, le 1 disparaît dans le calcul de la division entière.

Il est également intéressant d'examiner la différence entre les nombres et ce que j'appelle l'invariant, qui est une vérité constante dans la boucle : la différence entre les indicies doit être strictement positive, faute de quoi, le tableau se réduit à une seule case et la recherche est terminée. J'ai divisé les deux termes de l'inéquation par deux pour simplifier.

	Cas 1	Cas 2	Cas 3	Cas 4
iMin	2a	2a + 1	2a	2a+1
iMax	2b	2b	2b+1	2b+1
iMilieu	a+b	a+b	a+b	a+b+1
Différence	2b - 2a	2b - 2a - 1	2b + 1 - 2a	2b -2a
INVARIANT : iMax-iMin>0	b - a > 0	b - a > 0,5	b - a > -0,5	b - a > 0
iMin <- iMilieu	a+b	a+b	a+b	a+b+1
Différence	b - a	b - a	b + 1 - a	b - a
Variation	2b - 2a > b - a b - a > 0	2b - 2a - 1 > b - a **b - a > 1**	2b + 1 - 2a > b + 1 - a **b - a > 0**	2b - 2a > b- a b - a > 0
iMin <- iMilieu + 1	a + b + 1	a + b + 1	a + b + 1	a + b + 2
Différence	b - a - 1	b - a - 1	b - a	b - a - 1
Variation	2b - 2a > b - a - 1 b - a > -1	2b - 2a - 1 > b - a - 1 b - a > 0	2b + 1 - 2a > b -a b - a > - 1	2b -2a > b - a - 1 b - a > -1
iMax <- iMilieu	a + b	a + b	a + b	a + b + 1
Différence	b - a	b - a - 1	b - a	b - a
	2b - 2a > b - a b - a > 0	2b - 2a -1 > b - a -1 b - a > 0	2b + 1 - 2a > b - a b -a > -1	2b -2a > b - a b -a > 0
iMax <- iMilieu -1	a + b - 1	a + b - 1	a + b -1	a + b
Différence	b - a - 1	b - a - 2	b - a + 1	b - a - 1
Variation	2b - 2a > b - a - 1 b - a > -1	2b - 2a - 1 > b - a - 2 **b - a > -1**	2b + 1 - 2a > b - a +1 **b - a > 0**	2b -2a > b - a - 1 b -a > -1

Nous pouvons examiner maintenant les quatre affectations possibles vue plus haut. Notons que le choix de l'affectation à effectuer, à savoir modier `iMin` ou `iMax` n'entre pas ici en ligne de compte, puisqu'il dépend des valeurs contenues dans le tableau. L'important est de savoir si l'affectation fera progresser les choses, ce qui revient à demander : le moteur est-il efficace ? Pour qu'il le soit, il faut que la différence entre les deux indices limites diminue à **chaque** itération, autrement, un bouclage s'installe.

- `iMin <- iMilieu` (incorrecte)
- `iMin <- iMilieur + 1` (correcte)
- `iMax <- iMilieu` (correcte)
- `iMax <- iMilieu - 1` (correcte)

Nous allons donc comparer pour chaque nouvelle affectation la valeur de la différence entre les nouveaux indices (voir les lignes *Différence*) avec la valeur précédente de cette différence. Il en découle une inégalité puisque la différence intiale doit être plus grande que la nouvelle différence. J'ai indiqué cette inéquation dans les lignes *Variation* et je l'ai exprimée pour chaque affectation et chaque cas de départ. J'ai également simplifié l'expression sur la ligne suivante.

Comment savoir si, cette inéquation est correcte ? En fait, il y en quatre qui ne le sont pas (elles sont marquées en gris). Il suffit d'utiliser l'invariant, qui est supposé vrai. Prenons au hasard la première inéquation problématique, dans la colonne Cas2. Elle correspond à une limite inférieure impaire et une limite supérieure paire. Si nous affectons à `iMin` la valeur de `iMilieu`, la différence ne sera inférieure à la précédente que si *b-a>1*, or nous savons au mieux que *b-a>0.5*. Prenons un exemple qui respecte cela :

a	b	iMin	iMax	iMilieu	Différence	nouveau iMin	nouvelle différence	inéquation
		2a	2b	a+b	2b - 2a - 1	a+b	b-a	2b - 2a - 1 > b-a
5	6	11	12	11	1	11	1	1 > 1 FAUX

Dans cet exemple, `iMin<>iMax` et la boucle continue, mais le moteur ne progresse plus. Ceci correspondrait à la recherche de 125 dans un tableau de 20 éléments.

```
moteur
   Si iTab[iMilieu] < iNombre Alors
      iMin <- iMilieu
   Sinon
      iMax <- iMilieu
   Fin de si
Fin de boucle
Si iMin < MAX et iTab[iMin] == iNombre Alors
   Afficher iTab[iMin] & trouvé en position & iMin
Sinon
   Afficher "Recherche infructueuse"
Fin
```

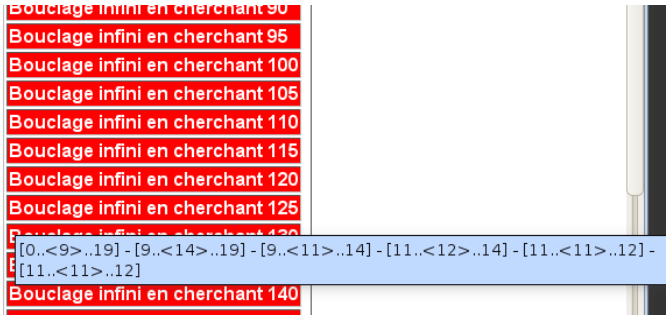


Tableau:

Vocabulaire

CHAMP, STRUCTURE

Exercices

V-1. Écrire un programme qui lit les noms et les coordonnées d'une personne, telles qu'elles ont été définies dans la structure `ePersonne` dans les notes.

V-2. Écrire un programme qui permet d'encoder les coordonnées de plusieurs clients d'une entreprise. On utilisera un tableau pour mémoriser les données.

V-3. Soit un tableau de clients, basé sur une structure plus simple ne comprenant qu'un numéro et un nom, qui contient les données suivantes :

```
<* ={{10,"Dupont"},{50,"Durant"},{25,"Dumont"},
{48,"Dumoulin"},{68,"Dubois"}} *>
```

Écrire un programme qui permette d'afficher le nom du client, sur base de son numéro de client (on notera que ces numéros ne sont pas triés).

V-4. Écrire un programme qui affiche une date, représentée dans une structure semblable à celle employée plus haut, sous la forme d'un texte en français (ex. 17 novembre 2003).

V-5. Écrire un programme qui affiche le lendemain d'une date entrée au clavier. Il faut tenir compte de la longueur des mois et de années bissextiles. Rappel : une année est bissextile si elle est multiple de 4 ou de 400 mais pas de 100.

– multiples de 4 : 1996 et 2004 sont bissextiles
  – multiples de 400 : 1600 et 2000 sont bissextiles
  – multiples de 100 : 1900 et 2100 **ne sont pas** bissextiles

V-6. Prévoir les déclarations nécessaires pour représenter à l'écran. Il n'est pas prévu d'écrire un programme pour cet exercice.

- un rectangle coloré parallèle aux bords de l'écran ;
- un cercle ;
- un rectangle coloré quelconque.