

```
program TraductionBinaire;

const bit_poids_fort = 0;
      bit_poids_faible = 7;
      (* Définition du numéro des bits de poids fort et de poids faible *)

type binaire8bits = array [bit_poids_fort .. bit_poids_faible] of integer;
      (* Définition du type entiers binaires codés sur 8 bits,
      * les bits de poids faibles sont à droite et
      * les bits de poids fort à gauche *)

var valeur_decimale : integer;
      (* Valeur en décimal du nombre *)
      valeur_binaire : binaire8bits;
      (* Valeur en binaire du nombre *)
      j : integer;
      (* compteur de boucle *)
      aleat : integer;
      (* Variable pour le générateur aléatoire *)

function Bin2Dec (n : binaire8bits) : integer;
var val : integer;
      (* La valeur calculée *)
i : integer;
      (* compteur de boucle (valide dans la fonction) *)
begin
  val := 0;          (* Initialisation de la valeur *)

  for i:= bit_poids_fort to bit_poids_faible do
    (* Calcul de la valeur *)
    val := val * 2 + n[i];

    (* Retour de la valeur calculée *)
    Bin2Dec := val
end;

(* Insertion du code des fonctions aléatoires (fichier hasard.p) *)
#include "hasard.p"

(* Début du programme principal *)
begin
  init_hasard;      (* Initialisation du générateur aléatoire *)

  (* Initialisation du nombre binaire *)
  for j := bit_poids_fort to bit_poids_faible do
    valeur_binaire[j] := entier_hasard (0, 1);

  valeur_decimale := Bin2Dec (valeur_binaire);

  (* Affichage des valeurs binaire et décimale sous la forme :
  * "Le nombre binaire 01001011 est égal à 75 en décimal"
  *)
  write ('Le nombre binaire: ');
  for j := bit_poids_fort to bit_poids_faible do
    write (valeur_binaire[j]:1);
  writeln (' est égal a ', valeur_decimale:2, ' en decimal')
end.
```

```
program Echange;

(* Les bornes du tableau d'entiers *)
const borneInf = 1;
      borneSup  = 10;

(* Définition du type intervalle et du type monTableau *)
type monTableau = array [borneInf..borneSup] of integer;

var leTableau1  : monTableau;
    leTableau2  : monTableau;
    i1, i2, Aleat : integer;

(* Procédure qui échange les éléments p1 et p2 du tableau t *)
procédure EchangeElementsTableau (var t : monTableau; p1, p2 : intervalle);
var tmp : integer;
begin
  tmp := t[p1];
  t[p1] := t[p2];
  t[p2] := tmp;
end;

(* Procédure qui échange les éléments p1 et p2 du tableau t
 * (version sans var) *)
procédure EchangeElementsTableau2 (t : monTableau; p1, p2 : intervalle);
var tmp : integer;
begin
  tmp := t[p1];
  t[p1] := t[p2];
  t[p2] := tmp;
end;

#include "hasard.p"

begin
  (* Initialisation du tableau *)
  for il := borneInf to borneSup do
    begin
      leTableau1[il] := entier_hasard(0,100);
      leTableau2[il] := leTableau1[il];
      writeln ('case ', il:2, ':', leTableau1[il]:3);
    end;

  write ('Tapez deux indices de tableau (dans l''intervalle ',
        borneInf:0, ', ', borneSup:0, '):');
  readln (i1,i2);

  while (i1 < borneInf) and (i1 > borneSup) and
        (i2 < borneInf) and (i2 > borneSup) do
    begin
      write ('Tapez deux indices de tableau (dans l''intervalle ',
            borneInf:0, ', ', borneSup:0, '):');
      readln (i1,i2);
    end;

  EchangeElementsTableau (leTableau1, i1, i2);
  EchangeElementsTableau2 (leTableau2, i1, i2);

  (* Affichage du tableau *)
  for il := borneInf to borneSup do
    begin
      writeln ('case ', il:2, ':', leTableau1[il]:3, ' ', leTableau2[il]:3);
    end;
  end.
end.
```

```
(* Insertion du code des fonctions aléatoires (fichier hasard.p) *)

function hasard: real;
const  a = 103767;
       b   = 9671;
       m   = 5787;

begin
  Aleat := (Aleat*a + b) mod m;
  hasard := Aleat/m;
end;

procedure init_hasard ;
(* procedure qui initialise Aleat a une valeur entiere et positive *)
const m = 5787;
begin
  Aleat:= 1000 mod m
  (* Aleat est un entier positif et strictement plus petit que m *)
end;

function entier_hasard (p, q: integer): integer;
(* les parametres p et q sont des entiers, q est plus grand que p *)
(* entier_hasard (p, q) donne un entier aleatoire de l'intervalle [p, q] *)
(* les entiers sont tires de facon equiprobable *)
begin
  entier_hasard:= p + trunc(hasard * (q+1-p))
end;
```

```

program Traduction;

const bit_poids_fort = 0;
      bit_poids_faible = 3;
      (* Définition du numéro des bits de poids fort et de poids faible *)

type hexa32bits = array [bit_poids_fort .. bit_poids_faible] of char;
      (* Définition du type entiers hexadécimaux codés sur 32 bits,
      * les bits de poids faibles sont à droite et
      * les bits de poids fort à gauche *)

var choix : integer;
    fin : boolean;

(* Fonction de conversion d'un caractère hexadécimal en décimal *)
function DigitHex2Dec (d : char) : integer;
begin
    if (d='a') then DigitHex2Dec := 10
    else if (d='b') then DigitHex2Dec := 11
    else if (d='c') then DigitHex2Dec := 12
    else if (d='d') then DigitHex2Dec := 13
    else if (d='e') then DigitHex2Dec := 14
    else if (d='f') then DigitHex2Dec := 15
    else DigitHex2Dec := ord(d) - ord('0')
end; { DigitHex2Dec }

(* Fonction de conversion d'un caractère décimal en hexadécimal *)
function DigitDec2Hex (v : integer) : char;
begin
    if (v=10) then DigitDec2Hex:='a'
    else if (v=11) then DigitDec2Hex:='b'
    else if (v=12) then DigitDec2Hex:='c'
    else if (v=13) then DigitDec2Hex:='d'
    else if (v=14) then DigitDec2Hex:='e'
    else if (v=15) then DigitDec2Hex:='f'
    else DigitDec2Hex:=chr(v+ord('0'));
end; { DigitDec2Hex }

(* Fonction de conversion en décimal de l'hexadécimal h *)
function Hex2Dec (h : hexa32bits) : integer;
var i : integer; (* Variable de parcours du tableau *)
    val : integer; (* Valeur décimale du nombre hexa *)
begin
    val := 0; (* Initialisation de la valeur *)

    for i := bit_poids_fort to bit_poids_faible do
        val := val * 16 + DigitHex2Dec (h[i]);

    Hex2Dec := val
end; { Hex2Dec }

(* Fonction de conversion en hexadécimal du décimal v *)
function Dec2Hex (v : integer) : hexa32bits;
var t : hexa32bits; (* Le tableau contenant le chiffre hexa à renvoyer *)
    i : integer; (* Variable de parcours du tableau *)
    res : integer; (* Divison entière par 16 *)
begin

```

```

    res := v;
    for i := bit_poids_faible downto bit_poids_fort do
        begin
            t[i] := DigitDec2Hex (res mod 16);
            res := res / 16
        end;

    Dec2Hex := t
end;

(* Procédure qui demande un nombre hexadécimal, le traduit en décimal
* et affiche le résultat de la traduction *)
procedure TraductionHexDec;
var n : hexa32bits;
begin
    writeln;
    write ('Saisissez un nombre hexadecimal sur 4 digits : ');
    read (n);

    writeln ('La traduction en decimal de ', n, ' est ', Hex2Dec(n):0);
end;

(* Procédure qui demande un nombre décimal, le traduit en hexadécimal
* et affiche le résultat de la traduction *)
procedure TraductionDecHex;
var n : integer;
begin
    writeln;
    write ('Saisissez un nombre decimal : ');
    read (n);

    writeln ('La traduction en decimal de ', n:0, ' est ', Dec2Hex(n));
end;

begin
    fin := false;

    while not(fin) do
        begin
            writeln;
            writeln ('1) Traduction hexa vers decimal');
            writeln ('2) Traduction decimal vers hexa');
            writeln ('0) quitter');
            readln (choix);

            while (choix <> 1) and (choix <> 2) and (choix <> 0) do
                begin
                    writeln ('1) Traduction hexa vers decimal');
                    writeln ('2) Traduction decimal vers hexa');
                    writeln ('0) quitter');
                    readln (choix);
                end;

            if (choix = 1) then
                TraductionHexDec
            else
                if (choix = 2) then
                    TraductionDecHex
                else
                    fin := true
            end
        end
    end
end

```

end.

```
program InitialisationAleatoire;

const borneInfLignes = 1;
      borneSupLignes  = 10;
      borneInfColonnes = 1;
      borneSupColonnes = 10;
      (* Définition des bornes pour les deux dimensions du tableau *)

type intervalleLignes = borneInfLignes..borneSupLignes;
   intervalleColonnes = borneInfColonnes..borneSupColonnes;
   monTableau         = array [intervalleLignes, intervalleColonnes] of real;

var Aleat : integer;
    tab   : monTableau;
    l     : intervalleLignes;
    c     : intervalleColonnes;

(* Insertion du code des fonctions aléatoires (fichier hasard.p) *)
#include "hasard.p"

procedure InitTableau (var t : monTableau);
var lig : intervalleLignes;
    col : intervalleColonnes;
    (* Les variables pour parcourir le tableau *)
begin
  for lig := borneInfLignes to borneSupLignes do
    for col := borneInfColonnes to borneSupColonnes do
      t[lig, col] := hasard
    end;
  end;

(* Début programme principal *)
begin
  init_hasard;

  InitTableau (tab);

  for l := borneInfLignes to borneSupLignes do
    begin
      for c := borneInfColonnes to borneSupColonnes do
        write(tab[l,c]:0:2, ' ');
        writeln
      end
    end
  end.
end.
```

```

program JeuVie;

const real_lmin = 0;
      real_lmax = 15;
      real_cmin = 0;
      real_cmax = 15;
      lmin      = real_lmin + 1;
      lmax      = real_lmax - 1;
      cmin      = real_cmin + 1;
      cmax      = real_cmax - 1;

(* real... representent les lignes et colonnes extremes du tableau, elles
 * sont toujours vides : elles constituent un bord commode pour évaluer les
 * générations successives mais ne seront jamais considérées faisant partie
 * des cases observées. Ces cases observées sont limitées par
 * [lmin..lmax, cmin..cmax] *)

type t_grille = array[real_lmin..real_lmax, real_cmin..real_cmax] of boolean;

var cellules          : integer;
    nb_generations    : integer;
    laGrille          : t_grille;
    generation_courante : integer;
    Aleat              : integer;

#include "hasard.p"

(* Fonction qui retourne le nombre de voisins de la cellules i,j
 * Comme i,j sont contenus dans [lmin..lmax, cmin..cmax], nous ne
 * considérons pas les bords de l'échiquier *)
function NombreVoisines (t : t_grille; i,j:integer) : integer;
var l,c : integer;
    cpt : integer;
begin
    cpt := 0;
    for l:=i-1 to i+1 do
        for c:=j-1 to j+1 do
            if t[l,c] then cpt := cpt + 1;

            (* On retire la case i,j si elle a été comptée *)
            if t[i,j] then cpt := cpt - 1;
            NombreVoisines := cpt
end; { NombreVoisines }

(* Fonction qui vérifie que la case est inoccupée et qu'elle remplit
 * la condition pour qu'à la génération suivante une cellule naisse *)
function Naissance (t : t_grille; i,j : integer) : boolean;
begin
    Naissance := not(t[i,j]) and (NombreVoisines(t,i,j)=3)
end; { Naissance }

(* Fonction qui vérifie que la case est occupée et qu'elle vérifie la
 * condition pour qu'à la génération suivante la cellule survive *)
function Survie (t : t_grille; i,j : integer) : boolean;
begin
    Survie := t[i,j] and
              ((NombreVoisines(t,i,j)=2) or (NombreVoisines(t,i,j)=3))
end; { Survie }

```

```

(* Procédure d'initialisation de la grille par nbcellules cellules *)
procedure InitGrille (var t : t_grille; nbcellules : integer);
var i,j,cpt : integer;
begin
    for i := real_lmin to real_lmax do
        for j := real_cmin to real_cmax do
            t[i,j] := false;

            for cpt := 1 to nbcellules do
                begin
                    i := entier_hasard(lmin, lmax);
                    j := entier_hasard(cmin, cmax);
                    while t[i,j] do
                        begin
                            i := entier_hasard(lmin, lmax);
                            j := entier_hasard(cmin, cmax);
                        end;
                    t[i,j] := true
                end
            end; { InitGrille }

(* Procédure d'affichage de la grille *)
procedure Affichage (t : t_grille);
var i,j : integer;
begin
    (* Affichage des bords du jeux *)
    for j := cmin to cmax do
        write ('-');
        writeln ('--');

        for i := lmin to lmax do
            begin
                write ('|');
                for j := cmin to cmax do
                    if t[i,j] then
                        write ('0')
                    else
                        write (' ');
                    writeln('|')
                end;

                (* Affichage des bords du jeux *)
                for j := cmin to cmax do
                    write ('-');
                    writeln ('--');
                end; { Affichage }

(* Procédure de calcul de la nouvelle generation *)
procedure NouvelleGeneration (var t : t_grille; var cellules : integer);
var g : t_grille;
    (* Grille temporaire utilisée pour calculer la génération suivante *)
    i,j : integer;
begin
    cellules := 0; (* Réinitialisation du nombre de cellules *)
    for i := lmin to lmax do
        for j := cmin to cmax do
            begin
                if Naissance(t,i,j) or Survie(t,i,j) then

```

```
begin
  g[i,j] := true;
  cellules := cellules + 1;
end
else
  g[i,j] := false
end;

(* Copie de la grille temporaire dans la grille t *)
for i := lmin to lmax do
  for j := cmin to cmax do
    t[i,j] := g[i,j]
  end;
end; { NouvelleGeneration }

(* Début du programme principal *)
begin
  write ('Donnez le nombre de cellules de l'echiquier: ');
  readln (cellules);
  InitGrille (laGrille, cellules);

  write ('Donnez le nombre de generations: ');
  readln (nb_generations);

  Affichage (laGrille);
  generation_courante := 1;
  while (generation_courante <= nb_generations) and (cellules <> 0) do
  begin
    NouvelleGeneration (laGrille, cellules);
    generation_courante := generation_courante + 1;
    writeln ('Generation', generation_courante);
    Affichage (laGrille);
    writeln ('Reste:', cellules:0, ' cellules');
    (* Attente d'un retour chariot *)
    readln;
    writeln
  end
end.
end.
```

```
program Minimum;

(* Les bornes du tableau d'entiers *)
const borneInf = 1;
      borneSup  = 10;

(* Définition du type intervalle et du type monTableau *)
type intervalle = borneInf..borneSup;
   monTableau   = array [intervalle] of integer;

var i : intervalle;
    (* Variable de parcours du tableau *)
    t : monTableau;
    (* le tableau considéré *)
    Aleat : integer;
    (* Variable pour le générateur aléatoire *)

function Minimum (tab : monTableau) : integer;
var i : intervalle;
    (* Variable de parcours du tableau *)
    min : integer;
    (* Le minimum courant *)
begin
    (* Initialisation du minimum par le premier élément *)
    min := tab[borneInf];

    (* Parcours du tableau *)
    for i := borneInf + 1 to borneSup do
        if tab[i] < min then
            min := tab[i];

    (* Retour de la valeur trouvée *)
    Minimum := min
end; { Minimum }

(* Insertion du code des fonctions aléatoires (fichier hasard.p) *)
#include "hasard.p"

begin
    (* Initialisation du générateur aléatoire *)
    init_hasard;

    (* Initialisation et affichage du tableau *)
    for i:= borneInf to borneSup do
        begin
            t[i] := entier_hasard(0,50);
            write (t[i]:3)
        end;
    writeln;

    (* Calcul et affichage du minimum *)
    writeln ('Le minimum du tableau est ', Minimum(t):2)
end.
```

```
program MinMax;

(* Les bornes du tableau d'entiers *)
const borneInf = 1;
      borneSup  = 10;

(* Définition du type intervalle et du type monTableau *)
type intervalle = borneInf..borneSup;
   monTableau  = array [intervalle] of integer;

var tab          : monTableau;
    lemin, lemax : integer;
    Aleat, i     : integer;

procedure MinMax (t : monTableau; var min, max : integer);
var i : intervalle;          (* Variable de parcours du tableau *)
begin
  (* Initialisation du max et du min au premier élément *)
  min := t[borneInf];
  max := t[borneInf];

  (* Parcours des autres éléments *)
  for i := borneInf+1 to borneSup do
    if t[i] < min then
      min := t[i]
    else
      if t[i] > max then
        max := t[i]
  end;

  (* Insertion du code des fonctions aléatoires (fichier hasard.p) *)
  #include "hasard.p"

begin
  (* Initialisation du générateur aléatoire *)
  init_hasard;

  (* Initialisation et affichage du tableau *)
  for i:= borneInf to borneSup do
    begin
      tab[i] := entier_hasard(0,50);
      write (tab[i]:3)
    end;
  writeln;

  MinMax(tab, lemin, lemax);

  writeln ('Le minimum du tableau est ', lemin:2);
  writeln ('Le maximum du tableau est ', lemax:2)
end.
```

```

program Recherche;

(* Les bornes du tableau d'entiers *)
const borneInf = 1;
      borneSup = 10;

(* Définition du type intervalle et du type monTableau *)
type intervalle = borneInf .. borneSup;
      monTableau = array [intervalle] of integer;

var tab : monTableau;
      Aleat : integer;
      v, pos : integer;

#include "hasard.p"

procedure InitAfficheTableau (var t : monTableau);
var i : intervalle;
begin
  init_hasard;

  for i := borneInf to borneSup do
    write (i:3);
    writeln;

    for i := borneInf to borneSup do
      begin
        t[i] := entier_hasard (0,100);
        write (t[i]:3)
      end;
    writeln
  end;

(* Fonction qui recherche la première occurrence de e dans le tableau t *)
function RechercheElement (t : monTableau; e : integer) : integer;
var i : integer;
      trouve : boolean;
      (* trouve est vrai si on a trouvé l'élément e dans le tableau *)
begin
  (* Initialisation des variables *)
  i := borneInf;
  trouve := false;

  while (i <= borneSup) and not (trouve) do
    begin
      trouve := t[i] = e;
      i := i + 1
    end;

  (* Renvoi de la valeur trouvée
  * NB: lorsque la valeur trouve est passée à vrai, la valeur de
  * i a été incrémentée : ne pas oublier de décrémenter *)
  if trouve then
    RechercheElement := i-1
  else
    RechercheElement := -1
  end;

(* Fonction qui recherche la première occurrence de e dans le tableau t *)
function RechercheElementPos (t : monTableau; e : integer; var pos : integer) : boolean;

```

```

var i : integer;
      trouve : boolean;
      (* trouve est vrai si on a trouvé l'élément e dans le tableau *)
begin
  (* Initialisation des variables *)
  i := borneInf;
  trouve := false;

  while (i <= borneSup) and not (trouve) do
    begin
      trouve := t[i] = e;
      i := i + 1
    end;

  (* Renvoi de la valeur trouvée
  * NB: lorsque la valeur trouve est passée à vrai, la valeur de
  * i a été incrémentée : ne pas oublier de décrémenter
  * si la valeur n'est pas trouvée, on retourne une valeur quelconque *)

  RechercheElementPos := trouve;
  pos := i-1
end;

(* Début du programme principal *)
begin
  InitAfficheTableau (tab);

  write ('Tapez la valeur a rechercher : ');
  readln (v);

  write ('Methode 1: ');
  pos := RechercheElement (tab, v);
  if pos = -1 then
    writeln ('Element non trouve')
  else
    writeln ('Element trouve a la position ', pos:0);

  write ('Methode 2: ');
  if RechercheElementPos (tab, v, pos) then
    writeln ('Element trouve a la position ', pos:0)
  else
    writeln ('Element non trouve')
  end.
end.

```

```
program SommeColonne;

const borneInfLignes = 1;
      borneSupLignes  = 10;
      borneInfColonnes = 1;
      borneSupColonnes = 10;
      (* Définition des bornes pour les deux dimensions du tableau *)

type interLignes = borneInfLignes .. borneSupLignes;
     interColonnes = borneInfColonnes .. borneSupColonnes;
     monTableau   = array [interLignes, interColonnes] of integer;

var leTableau : monTableau;
    c         : interColonnes;
    Aleat     : integer;

#include "hasard.p"

(* Procédure pour initialiser et afficher le tableau *)
procédure InitAfficheTableau (var t : monTableau);
var lig : interLignes;
    col : interColonnes;
    (* Les variables pour parcourir le tableau *)
begin
    init_hasard;

    for lig := borneInfLignes to borneSupLignes do
    begin
        for col := borneInfColonnes to borneSupColonnes do
        begin
            t[lig, col] := entier_hasard (0,100);
            write (t[lig, col]:5);
        end;
        writeln;
    end
end;

(* Fonction qui calcule la somme des éléments du tableau t et de la
 * colonne col *)
function SommeColonne (t : monTableau; col : interColonnes) : integer;
var lig : interLignes;
    s   : integer;      (* Somme des éléments de la colonne *)
begin
    s := 0;
    (* Parcours de toutes les lignes du tableau *)
    for lig := borneInfLignes to borneSupLignes do
        s := s + t[lig, col];
    end
    SommeColonne := s
end;

begin
    InitAfficheTableau (leTableau);

    for c := borneInfColonnes to borneSupColonnes do
        write ('-----');
        writeln;

        for c := borneInfColonnes to borneSupColonnes do
            write (SommeColonne (leTableau, c):5);
            writeln
        end
    end
end;
```

```
end.
```

```
program TestVoyelle;
var unCaractere : char;

function EstVoyelle (c : char) : boolean;
(* Fonction qui retourne vrai si le caractère c est une voyelle *)
begin
EstVoyelle := (c = 'a') or
(c = 'e') or
(c = 'i') or
(c = 'o') or
(c = 'u') or
(c = 'y')
end; { EstVoyelle }

begin
write ('Saisissez une lettre: ');
read (unCaractere);
writeln;

if EstVoyelle(unCaractere) then
writeln('Il s'agit d'une voyelle')
else
writeln('Il ne s'agit pas d'une voyelle')
end.
```