

Exercices d'informatique pour la 1^{re} B

Exercice 1 : Opérations sur les polynômes

Développer une application Delphi avec une interface graphique conviviale qui permet d'effectuer les opérations usuelles sur les polynômes d'une variable réelle, à coefficients réels :

- *évaluation* d'un polynôme en un réel x , par la méthode de Horner
- *addition* et *soustraction* de deux polynômes
- *multiplication* de deux polynômes
- *puissance* n^e d'un polynôme, où $n \in \mathbb{N}$
- *dérivée* et *primitive* d'un polynôme

Eventuellement aussi :

- *division euclidienne* d'un polynôme par un autre

On définira une constante `max` représentant le degré maximal d'un polynôme et le type `poly` pour travailler avec les polynômes :

```
const max = 100;

type poly = record
    c:array[0..max] of extended;
    d:integer
end;
```

Exercice 2 : Une petite calculatrice sur les fractions

Développer une application Delphi avec une interface graphique conviviale qui permet d'effectuer les 4 opérations $+$, $-$, $*$, $/$ sur les fractions. L'utilisateur entrera son calcul dans la boîte d'édition `edtInput` dans le format suivant :

`<fraction1>{<espace><opération><espace><fraction2>}`

Un clic sur le bouton '=' affiche le résultat dans le libellé `lblResultat`. Si l'opérateur entre seulement une fraction dans la boîte d'édition, mais pas d'opération, alors la fraction en question sera simplifiée. Chaque fraction entrée par l'utilisateur ainsi que le résultat du calcul, affiché dans le libellé `lblResultat` seront écrits dans le format

`<numérateur>{/<dénominateur>} (*)`,

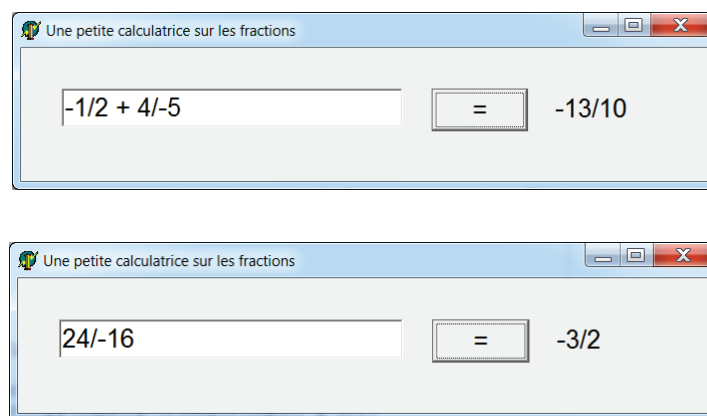
où le dénominateur est omis lorsqu'il est égal à 1. Les numérateurs et dénominateurs peuvent être des entiers positifs ou négatifs. On suppose que l'utilisateur n'entre pas un dénominateur 0.

On utilisera le type suivant pour travailler avec les fractions :

```
type fraction = record
    n:integer;
    d:integer
end;
```

Le programme comportera obligatoirement

- a) une procédure **simplifier** qui permet de simplifier une fraction donnée de type **fraction** ;
- b) des fonctions **somme**, **difference**, **produit** et **quotient** qui permettent d'effectuer les quatre opérations sur les objets de type **fraction** ;
- c) une fonction **strtofrac** qui prend en entrée un string au format (*) et retourne une fraction de type **fraction** ;
- d) une fonction **fractostr** qui prend en entrée un objet de type fraction et retourne le string représentant cette fraction au format (*).



Exercice 3 : Opérations sur les Chaînes de caractères

Développer une application Delphi avec une interface graphique conviviale qui permet d'effectuer les opérations suivantes sur une chaîne de caractères :

- (1) Renverser l'ordre des lettres de la chaîne (p.ex. "bonjour" devient "ruojnob").
- (2) Dédoubler chaque lettre d'une chaîne (p. ex. "bonjour" devient "bboonnjjjoouurr").
- (3) Mettre les lettres d'une chaîne dans un ordre aléatoire (p.ex. "bonjour" devient "oujbnor").

- (4) Enlever toutes les voyelles d'une chaîne (p.ex. "bonjour" devient "bnjr").
- (5) Trier les lettres d'une chaîne par ordre lexicographique (p.ex. "bonjour" devient "bjnooru").

Exercice 4 : Opérations sur les nombres complexes

Développer une application Delphi avec une interface graphique conviviale qui permet d'effectuer les opérations usuelles sur les nombres complexes :

- addition, soustraction, multiplication et division de deux nombres complexes ;
- module, inverse, puissance n^e d'un nombre complexe ;
- conversion de la forme trigonométrique en la forme algébrique et réciproquement.

On utilisera le type suivant pour travailler avec les nombres complexes :

```
type complex = record
    re:extended;
    im:extended
end;
```

Exercice 5

Développer une application Delphi avec une interface graphique conviviale qui demande à l'utilisateur d'entrer une phrase dans une boîte d'édition. Le programme décomposera cette phrase en ses différents mots, qui seront stockés dans une liste. On convient que deux mots sont séparés par un ou plusieurs des caractères suivants, dont le code ASCII se trouve entre parenthèses : espace (32), apostrophe (39), trait d'union (45), virgule (44). La fin de la phrase est marquée par un point.

Le programme devra également analyser la phrase : nombre de mots, mot le plus long, longueur moyenne d'un mot, fréquence d'un caractère entré par l'utilisateur.

Exercice 6

Développer une application Delphi avec une interface graphique conviviale qui permet de fusionner deux listes de nombres entiers triés par ordre croissant en une seule liste de nombres triés. Par exemple :

1^{re} liste : (1,1,3,4,4,8)

2^e liste : (2,3,5)

Liste fusionnée : (1,1,2,3,3,4,4,5,8)

Exercice 7 : Crible d'Erathosthène

Développer une application Delphi avec une interface graphique conviviale qui dresse la liste de tous les nombres premiers $\leq N$, où N est donné par l'utilisateur, par la méthode du *crible d'Erathosthène*, dont on rappelle ici le principe :

- (1) On écrit la liste de tous les entiers allant de 2 jusqu'à N .
- (2) On garde 2 et on élimine tous les autres multiples de 2.
- (3) On garde 3 qui est le premier élément non éliminé après 2 et on élimine tous les autres multiples de 3.
- (4) On garde 5 qui est le premier élément non éliminé après 3 et on élimine tous les autres multiples de 5.
- (5) On réitère le procédé jusqu'à la partie entière de la racine de N .

Les nombres non éliminés sont les nombres premiers $\leq N$.

Exercice 8

Développer une application Delphi avec une interface graphique conviviale qui recherche dans une liste de personnes toutes celles a) dont le nom commence par une lettre donnée ou un groupe de lettres données et b) celles dont le nom ne contient aucune des lettres d'un groupe donné.

Exercice 9 : Code de César

Le *code de César* est la méthode de cryptographie la plus ancienne communément admise par l'histoire. Il consiste en une substitution mono-alphabétique, où la substitution est définie par un décalage de lettres. Par exemple, si on remplace A par D, on remplace B par E, C par F, D par G, etc... Voici le code de César avec un décalage de 3 lettres :

Texte clair : A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Texte codé : D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Le texte que nous souhaitons coder est par exemple le suivant :

**JE LEVAI LES YEUX VERS LE SOLEIL IL ETAIT BAS ; DANS
MOINS D'UNE HEURE IL ARRIVERAIT JUSTE AU-DESSUS DES
BRANCHES SUPERIEURES DU VIEUX CHENE.**

Le texte codé est alors :

**MH OHYDL OHV BHXA YHUV OH VROHLO LO HWDLW EDV ; GDQV
PRLQV G'XQH KHXUH LO DUULYHUDLW MXVWH DX-GHVVXV GHV
EUDQFKHV VXSHULHXUHV GX YLHXA FKHQH.**

Exemple d'application. Essayer de décrypter le message suivant :

Exercice 10 : Suite de Matt Frank

$$\begin{cases} f(1) = 7 \\ f(n) = f(n-1) + \text{pgcd}(n, f(n-1)), n \geq 2 \end{cases}$$

- $f(1) = 7$
- $f(2) = 7 + \text{pgcd}(2,7) = 8$
- $f(3) = 8 + \text{pgcd}(3,8) = 9$
- $f(4) = 9 + \text{pgcd}(4,9) = 10$
- $f(5) = 10 + \text{pgcd}(5,10) = 15$
- \dots

$1, 1, 1, 5, 3, 1, 1, 1, 1, 11, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 23, 3, 1, 1, 1, 1, 1, 1, 1,$
 $1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 47, 3, 1, 5, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,$
 $1, 1,$
 $1, 1, 1, 1, 1, 1, 101, 3, 1, 1 \dots$

5

5, 3, 11, 3, 23, 3, 47, 3, 5, 3, 101, 3, 7, 11, 3, 13, 233, 3, 467, 3, 5, 3, 941, 3, 7, 1889, 3, 3779, 3, 7559, 3, 13, 15131, 3, 53, 3, 7, 30323, 3, 60647, 3, 5, 3, 101, 3, 121403, 3, 242807, 3, 5, 3, 19, 7, 5, 3, 47, 3, 37, 5, 3, 17, 3, 199, 53, 3, 29, 3, 486041, 3, 7, 421, 23, ...

Développer une application Delphi avec une interface conviviale qui affiche cette liste de nombres premiers. Le programme permettra également à l'utilisateur d'étudier ce qui se passe lorsqu'on prend une autre valeur pour $f(1)$.

Exercice 11 : Triangle de Pascal

Créer une application Delphi qui remplit un tableau avec les coefficients binomiaux : la cellule ayant comme *indice de ligne* n et comme *indice de colonne* k (avec $0 \leq k \leq n$) devra contenir C_n^k , les autres cellules resteront vides. La grille pourra être remplie en utilisant les propriétés suivantes de ces coefficients :

$$C_n^0 = C_n^n = 1, n \geq 0 ;$$

$$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}, n \geq 2 \text{ et } 1 \leq k \leq n-1.$$

Exercice 12 : Conjecture de Gilbreath

Voici un tableau dont on conjecture que la première colonne **commence toujours par le chiffre 1** (à partir de la 2^e ligne) :

2	3	5	7	11	13	17	19
1	2	2	4	2	4	2	
1	0	2	2	2	2		
1	2	0	0	0			
1	2	0	0				
1	2	0					
1	2						
1							

Ce tableau est obtenu de la manière suivante :

- Dans la 1^{re} ligne on écrit tous les nombres premiers, jusqu'à un entier donné, par exemple 19 ;
- Dans la 2^e ligne on calcule la valeur absolue de la différence entre deux voisins de la 1^{re} ligne ;
- On recommence ce calcul sur les ligne suivantes jusqu'à épuisement des possibilités.

On demande d'écrire une application Delphi avec une interface conviviale qui teste la conjecture de Gilbreath jusqu'à un entier fourni par l'utilisateur.

Exercice 13

Développer une application Delphi avec une interface conviviale qui détermine le plus grand écart entre deux éléments dans un tableau d'entiers donné. (L'écart entre deux entiers est la valeur absolue de leur différence.)

Exercice 14

Développer une application Delphi avec une interface conviviale qui détermine le nombre d'éléments différents dans un tableau donné.

Exercice 15

Développer une application Delphi avec une interface graphique contenant une matrice carrée $N \times N$ (N peut être choisi par l'utilisateur dans une boîte d'édition) et 3 boutons : a) Le bouton « Remplir » permet de placer les entiers de 1 à N^2 dans la matrice, dans un ordre aléatoire, b) le bouton « Trier les lignes » permettra de trier par ordre croissant toutes les lignes de la matrice et c) le bouton « Trier les colonnes » permettra de trier par ordre croissant toutes les colonnes de la matrice. Tester le programme en cliquant alternativement sur les deux boutons de tri. (Pour le tri, on pourra utiliser l'un des tris qui figurent au programme.)

Exercice 16 : Suites palindromiques

Un nombre palindromique (ou palindrome) est un entier naturel qui reste le même qu'on le lise de droite à gauche ou de gauche à droite. Par exemple : 47374, 101, 6 et 9449 sont des palindromes.

Partant d'un entier naturel quelconque N , on obtient la *suite palindromique* associé à cet entier de la manière suivante :

- On inverse l'ordre des chiffres de N et on ajoute N à ce nombre inversé.
- On recommence l'opération avec le nouveau nombre jusqu'à ce qu'on obtienne un palindrome.

Exemples :

N	Suite palindromique
13	13, 44
29	29, 121
1048	1048, 9449
64	64, 110, 121
87	87, 165, 726, 1353, 4884
89	89, 187, 968, 1837, ... , 8813200023188 (24 étapes en tout !)
196	196, 887, 1675, ... (suite infinie ??)

On ne sait pas à ce jour si la suite palindromique de 196 se termine par un palindrome ou non, même après plus de 700 millions d'itérations de renversement-addition ...

Développer une application Delphi avec une interface conviviale qui, à partir d'un entier naturel N donné détermine sa suite palindromique. Le programme devra s'arrêter lorsque le nombre d'éléments de cette suite dépasse un seuil que l'utilisateur pourra fixer. Pour plus d'informations, voir <http://www.jasondoucette.com/worldrecords.html>.

Exercice 17 : Opérations sur les matrices

Développer une application Delphi avec une interface graphique conviviale qui permet d'effectuer les opérations addition, soustraction et multiplication sur les matrices.

On utilisera le type suivant pour définir les opérations sur les matrices :

```
type matrice = record
    items:array[1..N,1..N] of extended;
    rowCount,colCount:integer
end;
```

où la constante N fixe le nombre maximal de colonnes ou de lignes d'une matrice. Elle est déclarée avec la valeur 50 par exemple.

Des procédures `stringGridToMatrice` et `matriceToStringGrid` permettront respectivement de transférer les éléments d'un `stringgrid` vers un élément de type `matrice` et réciproquement.

Le programme devra vérifier si l'opération choisie par l'utilisateur est possible : on rappelle que l'addition et la soustraction ne sont possibles que si les deux matrices ont exactement les mêmes dimensions. La multiplication d'une matrice de type (n,m) (n lignes et m colonnes) par une matrice de type (q,r) n'est possible que si $m=q$, c.-à-d. si le nombre de colonnes de la 1re est égal au nombre de lignes de la seconde.

Des boutons `Compléter A` et `Compléter B` permettront de remplir les deux matrices A et B du formulaire par des entiers aléatoires dans $[-10,...,10]$. Bien sûr l'utilisateur pourra aussi entrer manuellement les valeurs de type `extended` dans les matrices. Pour cela, on mettra l'option `goEditing` des deux `stringgrids` sur `true`.

Données
Entrez le genre des matrices A et B, cliquez ensuite sur Redessiner, puis entrez les éléments des matrices.

A
Nbr lignes : 3
Nbr colonnes : 7
Redessiner

-9	-8	-6	-4	10	1	-3
1	5	-10	9	-1	-1	5
5	8	3	7	-6	-7	3

B
Nbr lignes : 7
Nbr colonnes : 5
Redessiner

-3	-5	6	-2	9
-1	-9	-4	6	5
6	-4	-7	-4	6
8	-1	-5	4	-4
-6	-10	-4	4	7
-3	5	3	-3	-3
-4	-4	3	-10	-5

Opérations

+
-
X
Compléter A
Compléter B
Clear A
Clear B

Sorties

Matrice résultat : Ax B

Nbr lignes : 3
Nbr colonnes : 5

-84	62	-6	45	-59
-7	-34	27	53	-91
96	-103	-46	21	39

Exercice 18 : Jeu de la vie

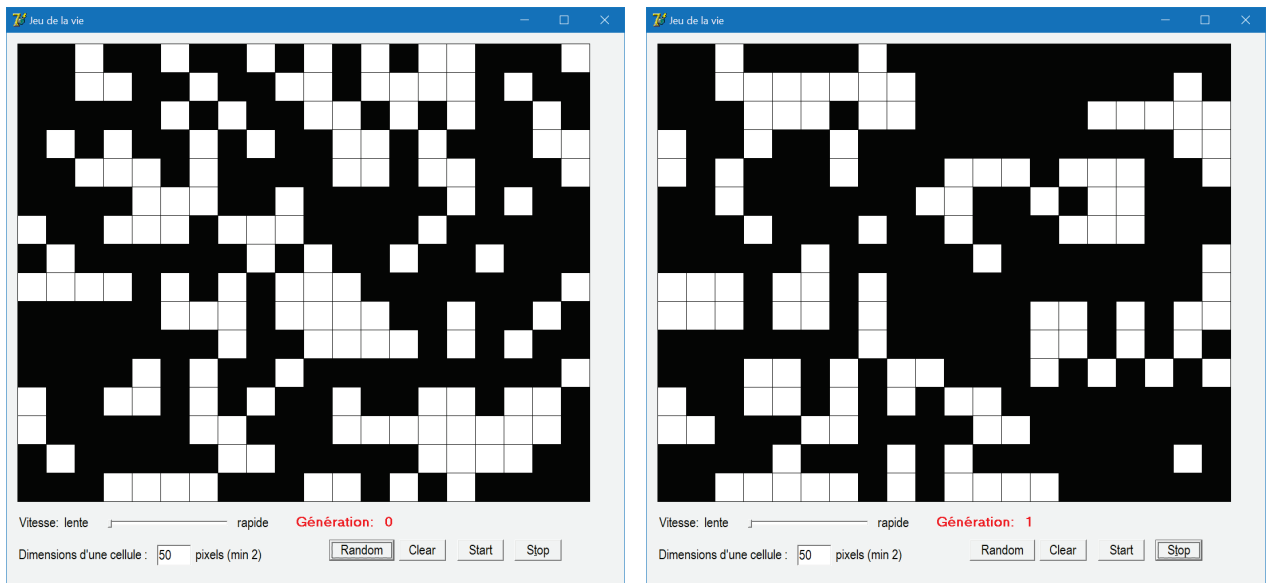
Le mathématicien américain John Conway (né en 1937) a imaginé vers 1970 un jeu, appelé « Jeu de la vie » qui met en scène des cellules susceptibles de se reproduire, de disparaître ou de survivre lorsqu'elles obéissent à des règles quelquefois appelées « génétiques ». Ces cellules sont représentées par des carrés blancs sur une image noire. Chaque cellule est donc entourée de huit carrés susceptibles d'accueillir d'autres cellules.

Les règles sont les suivantes :

- La **survie** : chaque cellule ayant **2** ou **3** cellules adjacentes survit jusqu'à la génération suivante.
- La **mort** : chaque cellule ayant ≥ 4 cellules adjacentes meurt **par surpopulation**. Chaque cellule ayant **0** ou **1** cellule adjacente meurt **d'isolement**.
- La **naissance** : chaque case vide ayant exactement **3** cellules adjacentes, fait naître une nouvelle cellule pour la génération suivante.
- Toutes les naissances et toutes les morts ont lieu **en même temps** au cours d'une génération.

Programmation : Créer une application Delphi qui simule le jeu de la vie sur le canevas d'une image. Une cellule vivante est représentée par un carré blanc entouré d'un bord noir, dont les dimensions peuvent être choisies par l'utilisateur. L'utilisateur pourra soit choisir la configuration initiale à l'aide de la souris, soit laisser l'ordinateur choisir cette configuration au hasard. Un bouton START permettra de lancer le calcul des générations suivantes. Le bouton STOP arrêtera la simulation. Un libellé donnera le numéro de la génération actuelle. On pourra créer la 1^{re} génération (d'indice 0) soit aléatoirement (bouton RANDOM), soit à l'aide de la souris (clic gauche pour dessiner une cellule vivante, clic droit pour effacer une cellule.) Le bouton CLEAR permet de réinitialiser le jeu. **Facultatif** : Un slider du type Trackbar (sous l'onglet WIN32) permettra de régler la vitesse de la simulation. Essayer de minimiser le nombre de calculs afin que le passage d'une génération à la suivante puisse se faire à grande vitesse !

Exemple d'exécution :



Exercice 19 : Automates cellulaires en dimension 1

Le « Jeu de la vie » présenté à l'Exercice 18 est ce qu'on appelle un automate cellulaire en 2 dimensions. Le but de cet exercice est d'étudier les automates cellulaires unidimensionnels. Ils consistent en une grille unidimensionnelle de cellules ne pouvant prendre que deux états (« 0 » ou « 1 »), avec un voisinage constitué, pour chaque cellule, d'elle-même et des deux cellules qui lui sont adjacentes.

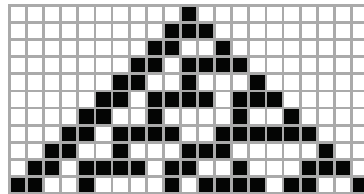
Chacune des cellules pouvant prendre deux états, il existe $2^3 = 8$ configurations possibles d'un tel voisinage. Le fonctionnement de l'automate cellulaire est défini par une règle qui détermine l'état d'une cellule, à la génération suivante, en fonction de son voisinage. Il y a $2^8 = 256$ façons différentes de s'y prendre, soit donc 256 automates cellulaires différents pouvant être simulés sur une telle grille.

Exemple. Considérons l'automate cellulaire défini par la table suivante, qui donne la règle d'évolution :

Motif initial	111	110	101	100	011	010	001	000
Valeur suivante de la cellule centrale	0	0	0	1	1	1	1	0

(Cela signifie que si par exemple, à un temps t donné, une cellule est à l'état « 1 », sa voisine de gauche à l'état « 1 » et sa voisine de droite à l'état « 0 », au temps $t+1$ elle sera à l'état « 0 ».)

Si l'on part d'une grille initiale où toutes les cellules sont à l'état « 0 » sauf une, on aboutit à :



où chaque ligne est le résultat de la ligne précédente.

Cette règle est souvent nommée « règle 30 », car 30 s'écrit 00011110 en binaire (cf deuxième ligne du tableau ci-dessus, décrivant la règle d'évolution).

Programmation. Créer une application Delphi qui simule l'évolution d'un automate cellulaire 1-dimensionnel à partir de l'une des 256 règles possible. L'utilisateur a le droit de choisir la règle, le nombre de cellules dans une ligne, le nombre de générations à calculer et la configuration initiale de l'automate. Un bouton permettra d'engendrer une configuration initiale au hasard.

Application en biologie : Les motifs de certains coquillages, comme les cônes et les cymbiolae, sont générés comme des automates cellulaires naturels.

Les cellules responsables de la pigmentation sont situées sur une bande étroite le long de la bouche du coquillage. Chaque cellule sécrète des pigments selon la sécrétion (ou l'absence de sécrétion) de ses voisines et l'ensemble des cellules produit le motif de la coquille au fur et à



mesure de sa croissance. Par exemple, l'espèce *conus textile* présente un motif ressemblant à la règle 30 décrite plus haut. (Source : Wikipedia)

Exercice 20 : Systèmes triangulaires

Créer une application Delphi avec une interface conviviale qui résout un système triangulaire de n équations à n inconnues, écrit sous forme matricielle.

Exercice 21 : Carrés magiques

Un carré magique est un carré de nombres entiers tel que la somme des éléments d'une ligne, d'une colonne ou d'une diagonale soit toujours la même. Un algorithme de construction des carrés magiques d'**ordre impair** > 3 développé par De la Loubère, envoyé de Louis XIV au Siam de 1687 à 1688, est le suivant :

- a - Mettre le 1 dans une cellule au choix ;
- b - Se déplacer selon le mouvement du cavalier aux échecs : 1 pas à droite et 2 vers le haut et y mettre le 2. Même chose pour les nombres suivants ;
- c - Lorsqu'on déborde du carré, continuer comme si le carré était bouclé : le côté droit collé à celui de gauche et le haut avec le bas ;
- d - Si cette règle aboutit à placer un nombre dans une cellule déjà occupée, il faut le placer dans la cellule immédiatement en dessous du dernier nombre écrit (multiple de l'ordre) ;
- e - Continuer comme en 2 jusqu'à remplir le carré.

Par exemple, voici un carré magique d'ordre 5, construit suivant cet algorithme :

1	14	22	10	18
7	20	3	11	24
13	21	9	17	5
19	2	15	23	6
25	8	16	4	12

Créer une application Delphi qui construit un carré magique d'ordre n impair, où n est donné par l'utilisateur. L'utilisateur aura aussi la possibilité de choisir l'emplacement du 1. Le programme affichera dans un libellé la **somme magique** du carré : c'est la somme constante des lignes, des colonnes et des diagonales. (Elle peut être calculée selon la formule :

$$\frac{n(n^2 + 1)}{2}$$

Par exemple, pour le carré magique d'ordre 5 ci-dessus, la somme magique vaut 65).

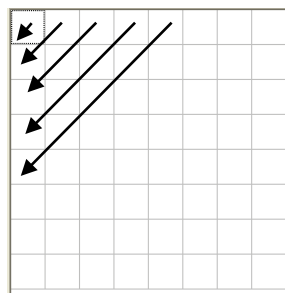
Exercice 22

On appelle MINIMAX d'une matrice d'entiers la valeur minimale des maxima de chaque ligne. Développer une application Delphi avec une interface conviviale qui donne a) le MINIMAX d'une matrice donnée d'entiers et b) l'emplacement d'un MINIMAX d'une telle matrice.

Exercice 23

Développer une application Delphi avec une interface graphique conviviale qui remplit un tableau $M \times N$ avec **tous** les entiers compris entre 1 et $M \cdot N$ (chaque entier devra donc figurer une et une seule fois dans le tableau). Chacune des méthodes de remplissage suivantes sera associée à un bouton :

- a) **Colonne par colonne** : les entiers sont placés dans l'ordre dans le tableau et les colonnes sont remplies l'une après l'autre de haut en bas, en commençant par la colonne d'indice 0.
- b) **Ligne par ligne** : les entiers sont placés dans l'ordre dans le tableau et les lignes sont remplies l'une après l'autre de gauche à droite, en commençant par la ligne d'indice 0.
- c) **Diagonale par diagonale** : les diagonales sont remplies l'une après l'autre, comme l'indique la figure ci-dessous :



- d) **Au hasard** : les entiers sont placés aléatoirement dans le tableau.

Exercice 24

Développer une application Delphi avec une interface conviviale qui remplit un tableau $n \times n$ en spirale avec les entiers de 1 à n^2 . Par exemple :

1	2	3	4
12	13	14	5
11	16	15	6
10	9	8	7

Exercice 25 : Méthode dichotomique

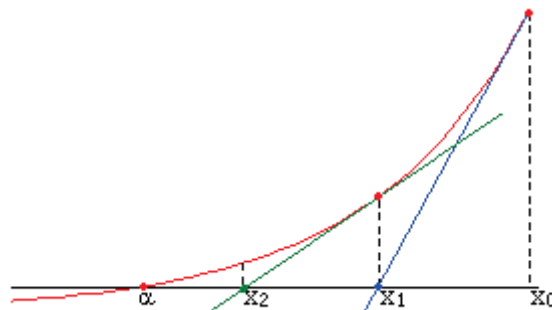
Développer une application Delphi avec une interface graphique conviviale qui *recherche une racine* d'un polynôme p par la méthode dichotomique :

- L'utilisateur entre deux réels $a < b$ tels que $p(a) \cdot p(b) < 0$, c.-à-d. tels que $p(a)$ et $p(b)$ soient de signes opposés. (Le programme devra vérifier si cette condition est bien remplie et afficher au cas contraire un message d'erreur).
- Le programme calcule $p\left(\frac{a+b}{2}\right)$: il y a alors 3 possibilités :
 - ou bien $p\left(\frac{a+b}{2}\right) = 0$, alors $\frac{a+b}{2}$ est une racine de p et le problème est résolu ;
 - ou bien $p\left(\frac{a+b}{2}\right) \cdot p(a) > 0$, alors le programme posera $a := \frac{a+b}{2}$;
 - ou bien $p\left(\frac{a+b}{2}\right) \cdot p(b) > 0$ alors le programme posera $b := \frac{a+b}{2}$;

Cet algorithme se poursuit jusqu'à ce que $b - a < \varepsilon$, où ε est l'erreur tolérée sur la racine, fixée par l'utilisateur dans une boîte d'édition.

Exercice 26 : Méthode de Newton

La méthode de Newton est un *procédé récurrent* permettant de déterminer une *valeur approchée* d'une racine d'une fonction *dérivable* f . La figure suivante explicite cette méthode :



On part d'un réel x_0 du domaine de f et on considère la tangente à \mathcal{G}_f au point d'abscisse x_0 , d'équation : $y = f'(x_0)(x - x_0) + f(x_0)$. Si $f'(x_0) \neq 0$, on montre facilement que cette tangente coupe l'axe des abscisses en

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

On recommence la même opération en partant de x_1 . Si $f'(x_1) \neq 0$, la tangente à \mathcal{G}_f au point d'abscisse x_1 coupe l'axe des abscisses en

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}.$$

Et ainsi de suite. On construit donc une suite récurrente $(x_k)_{k \geq 0}$ définie de la manière suivante :

$$\begin{cases} x_0 \text{ est quelconque,} \\ x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k \geq 1 \end{cases} \quad (*)$$

Cette suite, appelée *suite de Newton partant de* x_0 , est bien définie à condition que tous les $f'(x_k)$ sont $\neq 0$. Elle converge *souvent* vers une racine α de f (c.-à-d. une solution de l'équation $f(x) = 0$). Insistons sur le fait que la suite ne converge pas toujours, comme par exemple dans le cas où f n'a pas de racine !

Programmation : Implémenter dans Delphi la méthode de Newton, mais uniquement pour des fonctions polynômiales f . Le programme, avec une interface conviviale, comportera obligatoirement les éléments suivants :

- (1) Une fonction **eval** qui calcule l'image d'un polynôme donné **f** en un réel donné **x**, par exemple en utilisant le procédé de Horner.

- (2) Une procédure **derivee** qui détermine le polynôme dérivé d'un polynôme **f**.
- (3) Une fonction **newton** qui, partant d'un réel **x** et d'un polynôme **f** donnés, retourne
 - le terme suivant de la suite de Newton si $f'(\mathbf{x}) \neq 0$ et
 - **x** si $f'(\mathbf{x}) = 0$ (ceci afin que le terme suivant soit toujours défini).

Le programme calculera, à partir d'un polynôme **f**, d'un premier terme **x** et d'une précision **epsilon** donnés, les termes suivants de la suite de Newton et leurs images (dans deux listes) *jusqu'à ce que* l'une des trois conditions suivantes soit réalisée :

- la suite de Newton *converge* vers une racine de **f** c.-à-d. $|x_{k+1} - x_k| \leq \mathbf{epsilon}$ et $|f(x_k)| \leq \mathbf{epsilon}$, après un certain nombre k d'itérations ;
- le *nombre maximal d'itérations* (choisi par l'utilisateur) est atteint sans que la suite semble converger, c.-à-d. sans que l'une des conditions précédentes soit réalisée.
- la suite de Newton devient stationnaire après un certain nombre d'itérations c.-à-d. l'un de ses termes est égal au suivant et par conséquent à *tous* les termes suivants ; cela arrive lorsque l'on tombe sur terme x_k tel que $f'(x_k) = 0$ (à cause de notre définition de la fonction **newton**) ou tel que $f(x_k) = 0$.

Le nombre maximal d'itérations **maxiter**, par défaut égal à 100, pourra être précisé par l'utilisateur dans une boîte d'édition. Les conclusions possibles seront données dans un ou plusieurs libellés.

Exercice 27

Développer une application Delphi avec une interface conviviale qui permet de calculer l'intégrale définie d'une fraction rationnelle en utilisant la méthode des trapèzes. L'utilisateur entrera les coefficients réels du numérateur et du dénominateur de la fraction rationnelle, ainsi que les bornes d'intégration a et b et le nombre n d'intervalles. Le programme devra tester si le dénominateur de la fraction rationnelle ne s'annule pas sur $[a, b]$, par exemple en utilisant la méthode dichotomique (voir exercice 25).

Exercice 28

Développer une application Delphi avec une interface graphique conviviale qui permet de trier un tableau de données multi-dimensionnelles, par exemple un fichier à trois colonnes contenant respectivement les noms, les prénoms et les lieux de naissance des employés d'une entreprise. Le tableau sera trié d'abord suivant la 1^{re} colonne, puis suivant la 2^e etc., toujours par ordre croissant, comme dans l'exemple ci-contre.

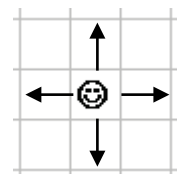
Nom	Prénom	Lieu de naissance
A	A	A
A	A	C
A	B	A
A	B	A
A	B	B
A	C	A
A	C	B
A	C	C
B	A	A
B	A	C
B	B	C
B	C	B
B	C	C
C	A	B
C	A	B
C	A	C
C	B	B
C	B	C
C	C	A

Exercice 29 : Marche de l'ivrogne

Développer une application Delphi avec une interface conviviale qui permet de simuler la marche d'un ivrogne sur un damier $N \times N$, la dimension N pouvant être choisie par l'utilisateur dans une boîte d'édition. L'utilisateur pourra fixer également le nombre d'obstacles que peut rencontrer l'ivrogne au cours de sa marche.

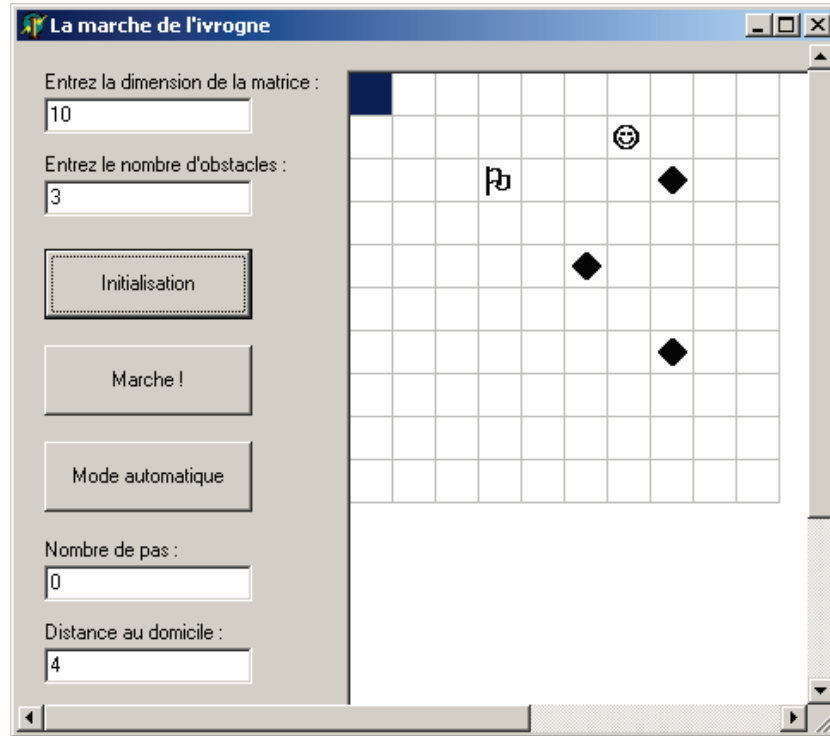
Programmation :

- Un clic sur le bouton « Initialisation » aura comme effet de tracer le damier et d'y placer au hasard l'ivrogne, sa maison et les obstacles. (Choisir dans le damier (stringgrid) la police « Wingdings » et utiliser les symboles adéquats pour représenter l'ivrogne, sa maison et les obstacles.)
- Un clic sur le bouton « Marche » aura comme effet de mouvoir l'ivrogne au hasard d'une case dans l'une des quatre directions : nord, sud, ouest, est. Bien sûr, l'ivrogne ne pourra pas s'arrêter sur un obstacle, ni quitter le damier. Un message informera l'utilisateur lorsque l'ivrogne est finalement rentré.



- Facultatif : un clic sur le bouton « Mode automatique » automatisera la marche de l'ivrogne (utiliser le composant timer).

Le programme affichera à tout moment le nombre de pas effectués par l'ivrogne ainsi que sa distance au domicile.



Exercice 30 : Test du chi-deux

Développer une application Delphi avec une interface graphique conviviale qui permet de simuler n lancers d'un dé. Les résultats des lancers seront sauvegardés dans une liste. Un bouton « Dé équilibré ? » permet de décider si le dé est pipé (truqué) ou non à l'aide d'un *test statistique* appelé *test d'adéquation du chi-deux* :

- (1) On compte le nombre d'occurrences O_i de chaque résultat i ($1 \leq i \leq 6$) dans la série des n lancers où l'on supposera $n \geq 30$.
- (2) On calcule le chi-deux : $\chi^2 = \frac{6}{n} \sum_{i=1}^6 \left(O_i - \frac{n}{6} \right)^2$.
- (3) On détermine la probabilité p pour que le dé soit équilibré à l'aide du tableau suivant :

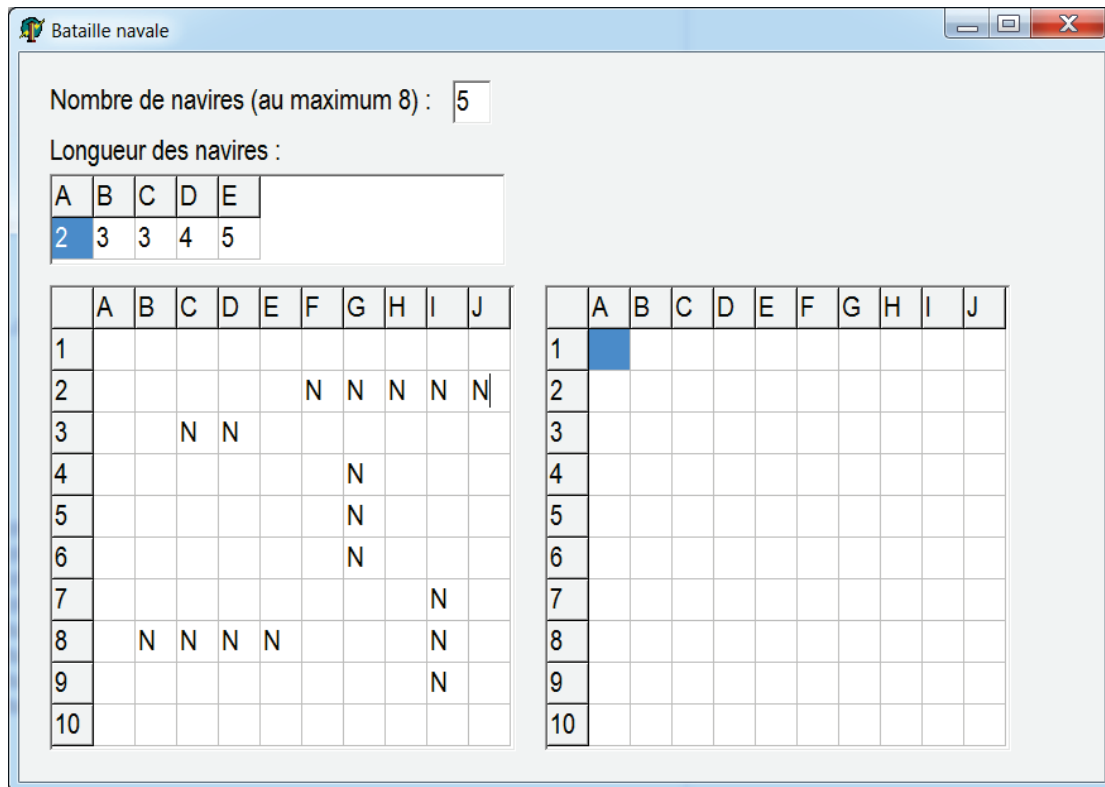
χ^2	0,55	1,15	1,61	4,35	9,24	11,07	15,09	20,52
p	0,99	0,95	0,90	0,50	0,10	0,05	0,01	0,001

En d'autres termes :

- Si $\chi^2 \leq 0,55$, la conclusion sera : « Le dé est équilibré avec une probabilité de 99 %. »
- Si $0,55 < \chi^2 \leq 1,15$, la conclusion sera : « Le dé est équilibré avec une probabilité de 95 %. »
- Si $1,15 < \chi^2 \leq 1,61$, la conclusion sera : « Le dé est équilibré avec une probabilité de 90 %. »
- ...
- Si $20,52 < \chi^2$, la conclusion sera : « Le dé est équilibré avec une probabilité inférieure à 0,1 %. »

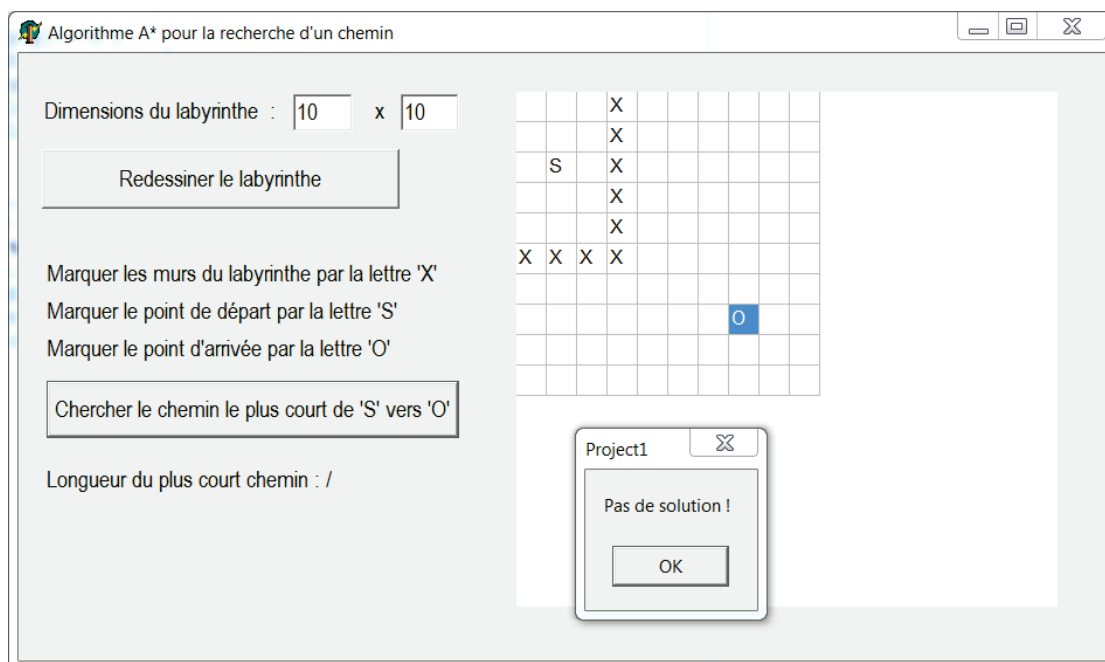
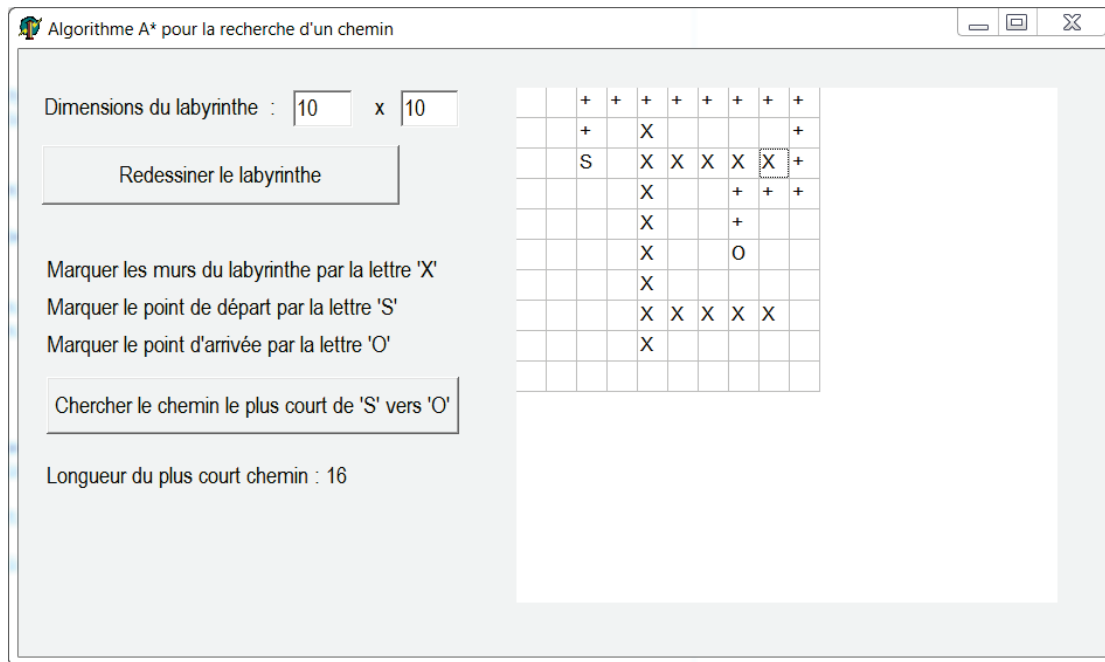
Exercice 31 : Bataille navale

Implémenter dans Delphi une version du jeu "Bataille navale", dans lequel deux joueurs doivent placer des "navires" sur une grille tenue secrète et tenter de toucher les navires adverses. L'ordinateur prend le rôle de l'adversaire. Le gagnant est celui qui parvient à torpiller complètement les navires de l'adversaire avant que tous les siens ne le soient. Chaque joueur possède les mêmes navires, dont le nombre et le type dépendent des règles du jeu choisies. Dans la configuration par défaut, il y a 5 navires A, B, C, D et E de longueurs respectives 2, 3, 3, 4 et 5. Avant de commencer la partie, chaque joueur positionne ses navires horizontalement ou verticalement sur la grille à gauche. Le joueur dessinera un navire de longueur n sur sa grille en y écrivant n fois la lettre N (horizontalement ou verticalement, dans des cases adjacentes). L'ordinateur positionne ses navires de la même façon, au hasard. (On placera pour l'ordinateur deux grilles invisibles sur le formulaire.) Pour tirer, le joueur clique dans la grille à droite à l'endroit souhaité. L'ordinateur contrôle alors si le joueur a touché l'un de ses navires et met dans la case correspondante : un 'O' si le navire a été touché, un 'X' sinon. A la fin de la partie l'ordinateur avertit le joueur qui des deux a gagné la partie. Le programmeur veillera à ce que l'ordinateur pose ses tirs de façon intelligente c.-à-d. : il tirera au hasard jusqu'à ce qu'il ait touché un bateau, puis il coulera ce bateau en entier. D'autres stratégies plus fines peuvent être envisagées. Que le meilleur gagne !



Exercice 32 : Algorithme A* pour la recherche d'un chemin

Réaliser un programme avec une interface conviviale qui permet de trouver le chemin le plus court d'un point de départ S à un point d'arrivée O dans un labyrinthe. L'algorithme à utiliser est l'algorithme A* décrit sur internet à l'adresse <http://en.wikipedia.org/wiki/Pathfinding>. Le programme devra également détecter si le problème n'admet pas de solution, comme sur la deuxième figure ci-dessous.



Exercice 33 : Le jeu Puissance 4

On demande de programmer une version du jeu Puissance 4 avec une interface graphique conviviale.

Les règles du Puissance 4 sont :

- établir un plateau de 7 colonnes sur 6 lignes,
- chaque joueur possède 21 jetons de sa couleur,
- chaque jeton joué est lâché au sommet d'une colonne et prend sa place dans la première cellule libre en partant du bas,

d) Le premier joueur alignant 4 jetons dans sa couleur (horizontal, vertical ou diagonal) a gagné ; si les 42 jetons sont joués sans qu'un joueur ait réussi à aligner 4 jetons, la partie est nulle.

On demande que deux joueurs puissent jouer l'un contre l'autre sur le même ordinateur. Le programme doit remarquer quand un joueur a gagné ou quand il y a match nul.

Exercice 34 La diagonale des sous

Un casino imaginaire propose le jeu suivant, intitulé « La diagonale des sous ».

Vous misez 5 €. Un programme aléatoire dépose alors toutes les sommes de 1 à 100 centimes d'euros sur les cases d'un damier de taille 10 sur 10, mais en respectant la contrainte suivante : deux valeurs successives doivent obligatoirement se trouver sur des cases ayant un côté commun. Vous remportez le total des euros qui se trouvent sur la diagonale représentée en gris.

Réaliser un programme avec une interface conviviale qui permet de simuler ce jeu. Le joueur pourra jouer plusieurs parties de suite. Il inscrira son avoir initial dans une boîte d'édition. Sa fortune variant au cours des parties suivantes sera toujours affichée en € et Cents, par exemple 3 € 24 Cents, au lieu de 324 Cents. Pour simplifier, nous convenons que si le programme n'arrive plus à placer le montant suivant sur la grille, comme sur la figure à droite, alors il s'arrête et la somme gagnée est simplement la somme des cases grises non vides.

(Au fait, quelle somme maximale pouvez-vous espérer gagner dans une partie ?)

D'après Elisabeth Busser et Gilles Cohen, Affaire de logique, Le Monde