Université Bordeaux I — **Module MIAS 302**
Travaux dirigés — **Feuille 1**

Variations sur le calcul de $n!$

1. Ecrire la fonction C `puissance` qui, pour deux entiers a et n donnés, calcule a^n (on suppose n positif, et on ne se préoccupe pas du cas où n est négatif).
2. Ecrire une fonction qui calcule un coefficient binomial, en utilisant la fonction `factorielle` vue en cours, et la formule :
$$C_n^k = \frac{n!}{k!(n-k)!}$$
Expliquer l'inconvénient majeur de cette formule, et transformer la fonction pour appliquer la formule :
$$C_n^k = \frac{n(n-1)\cdots(n-k+1)}{1\cdot 2\cdots k}$$
de façon à éviter tout débordement inutile.
3. Un dérangement est une permutation sans point fixe. Le nombre d_n de dérangements de n éléments vérifie l'étonnante récurrence suivante :
$$d_n = nd_{n-1} + (-1)^n$$
Quelle est la valeur de d_1 ? Ecrire la fonction C `derangement` qui calcule d_n.

Université Bordeaux I **Module MIAS 302**
Travaux dirigés **Feuille 2**

Boucles et tests

1. Ecrire une fonction C `maximum` qui calcule le maximum de trois entiers `x`, `y` et `z` donnés.
2. Soit la fonction C suivante :

   ```
   int qui_suis_je( int n){

       int d = 0;
       int k;

       for( k = n; k > 0; k = k - 2)
          d = d + k;

       return d;
   }
   ```

   Quelles sont les valeurs de `qui_suis_je(0)`, `qui_suis_je(1)`, `qui_suis_je(2)`, et `qui_suis_je(3)` ? Que calcule cette fonction ? Donnez une expression de `qui_suis_je(n)`.
3. Ecrire une fonction C `somme` qui calcule la somme des `n` premiers entiers, pour `n` un entier donné.
4. Ecrire une fonction qui calcule la somme des entiers pairs inférieurs ou égaux à un entier `n` donné.
5. Ecrire une fonction qui, étant donné un nombre entier `m`, calcule la somme des chiffres de `m`.
6. Ecrire une fonction qui calcule la somme des diviseurs d'un entier `n`. Remarque : `p` divise `n` s'écrit `n % p == 0`.

Nombres en virgule flottante

Calculer le développement, en numération binaire, du nombre 1/5, ie. la suite de chiffres binaires b_i telle que :

$$\frac{1}{5} = \frac{b_1}{2} + \frac{b_2}{4} + \frac{b_3}{8} + \ldots = \sum_{i=1}^{\infty} \frac{b_i}{2^i}$$

Si l'on ne garde que les huit premiers chiffres de ce développement, quel est le résultat du calcul (approché) de :

$$\frac{1}{5} + \frac{1}{5} + \frac{1}{5} + \frac{1}{5} + \frac{1}{5} \ ?$$

Calcul de séries

1. Ecrire une fonction `ln_un_plus_x(double x)` qui calcule $\ln(1+x)$ à partir de son développement en série :

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \ldots = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{x^i}{i}$$

   Quel problème mathématique rencontre-t-on ? En déduire une fonction qui calcule $\ln y$ pour tout réel $y > 0$.

2. Ecrire une fonction qui calcule $(1+x)^\alpha$ (pour α réel) à partir du développement :

$$\begin{aligned} (1+x)^\alpha &= 1 + \alpha x + \frac{\alpha(\alpha-1)}{2} x^2 + \frac{\alpha(\alpha-1)(\alpha-2)}{3!} x^3 + \ldots \\ &= \sum_{i=0}^{\infty} \frac{\alpha(\alpha-1)\ldots(\alpha-i+1)}{i!} x^i \end{aligned}$$

Exercices complémentaires
Il est conseillé de faire les exercices suivants :

1. Ecrire une fonction qui calcule $\sin x$ à partir de son développement en série usuel (2 versions, une utilisant le rang, l'autre en fonction de la contribution du terme) :

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!}$$

2. Ecrire une fonction qui calcule $\cos x$ à partir de son développement en série usuel :

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!}$$

Université Bordeaux I — **Module MIAS 302**
Travaux dirigés — **Feuille 4**

Suites récurrentes, méthodes numériques

1. Ecrire une fonction C qui calcule le terme u_n de la suite définie par :

$$\begin{array}{lcl} u_0 & = & 0 \\ u_{n+1} & = & \frac{1}{1+u_n} \end{array}$$

2. Ecrire une fonction C qui calcule le terme u_n de la suite de Fibonacci, définie par :

$$\begin{array}{lcl} u_0 & = & 0 \\ u_1 & = & 1 \\ u_{n+1} & = & u_n + u_{n-1} \end{array}$$

3. Ecrire une fonction qui calcule le terme u_n (pour $n \geq 2$) pour la suite définie par $u_0 = u_1 = 1$ et

$$u_{n+1} = \begin{cases} u_n + u_{n-1} \text{ si } n+1 \text{ est pair} \\ u_n * u_{n-1} \text{ sinon} \end{cases}$$

4. Pour calculer une valeur approchée de l'intégrale :

$$\int_a^b f(x)dx$$

on peut diviser l'intervalle $[a\ b]$ en n segments *égaux*, délimités par les points :

$$x_0 = a,\ x_1,\ x_2,\ \ldots,\ x_n = b$$

et remplacer f, sur chaque segment $[x_i\ x_{i+1}]$, par la fonction constante égale à $f(x_i)$; en d'autres termes on approche f par une fonction en escalier.

Ecrire, en langage C, la fonction `integrale` qui effectue ce calcul approché et dont le prototype est :

```
double integrale(double f(double), double a, double b, int n)
```

Une meilleure approximation est obtenue en remplaçant f, sur $[x_i\ x_{i+1}]$, par la fonction constante égale à :

$$\frac{f(x_i) + f(x_{i+1})}{2} \ ;$$

écrire la version modifiée correspondante de la fonction `integrale`, en veillant à ne pas calculer deux fois la même valeur $f(x)$.

Université Bordeaux I — **Module MIAS 302**
Travaux dirigés — **Feuille 5**

Structures

1. En utilisant la définition et les fonctions suivantes :

```
struct complexe {
  double reel, imaginaire;
};

struct complexe somme (struct complexe u, struct complexe v) {
  struct complexe z;

  z.reel = u.reel + v.reel;
  z.imaginaire = u.imaginaire + v.imaginaire;
  return z;
}

struct complexe produit (struct complexe u, struct complexe v) {
  struct complexe z;

  z.reel = u.reel * v.reel - u.imaginaire * v.imaginaire;
  z.imaginaire = u.reel * v.imaginaire + u.imaginaire * v.reel;
  return z;
}
```

   écrire une fonction qui calcule l'exponentielle d'un nombre complexe z, comme somme de la série :

$$\sum_{n=0}^{\infty} \frac{z^n}{n!}$$

2. On appelle *séquence* toute suite d'entiers terminée par un zéro.

   Faire une fonction qui lit une séquence et retourne le minimum, le maximum et la moyenne d'une séquence. Pour cela, on définira une structure MMM composée de deux entiers (min et max) et d'un flottant (moyenne).

   Utiliser cette fonction pour faire un programme qui lit au clavier une suite de séquences (terminée par la séquence vide qui ne contient que 0) et qui affiche le min, le max et la moyenne de chaque séquence.

Université Bordeaux I — **Module MIAS 302**
Travaux dirigés — **Feuille 6**

Tableaux

1. Écrire une fonction qui calcule le produit scalaire de deux vecteurs u et v, dans un espace de dimension n.
2. Écrire des fonctions *min* et *max* qui calculent les éléments minimum et maximum, dans un tableau t de taille n. Vaut-il mieux calculer la valeur du minimum (resp. maximum), ou son indice ?
3. Écrire une fonction *moyenne* qui calcule la moyenne des éléments d'un tableau d'entiers t de taille n.
4. Écrire une fonction *indice* qui calcule l'indice d'un élément x dans un tableau t de taille n ; si x figure plusieurs fois dans t, la fonction retournera l'indice de la première occurrence. Quelle valeur doit-elle retourner si x ne figure pas dans t ?
5. Écrire une fonction qui calcule le nombre d'occurrences d'un élément x dans un tableau t de taille n.
6. On considère la fonction suivante :

```
#define MAX 20
 void f (int t[], int u[], int n) {
  int i, j;
  for (j = 0; j <= MAX; j++)
    u [j] = 0;
  for (i = 0; i < n; i++) {
    j = t [i];
    u [j]++;
  }
}
```

   On suppose donné un tableau `notes`, contenant les (dix) valeurs suivantes : 12 ; 14 ; 10 ; 8 ; 12 ; 12 ; 15 ; 8 ; 13 ; 10, et déclaré un tableau :

   `int a [MAX + 1] ;`

   Quel est le résultat de l'appel : `f (notes, a, 10)` ?

   Que calcule, de façon générale, la fonction f ?

   Cette fonction comporte une instruction dangereuse : laquelle ? Que faudrait-il ajouter pour rendre cette fonction sûre ?
7. On appelle *inversion*, dans un tableau d'entiers t, tout couple (i, j) d'indices tel que : $i < j$ et $t_i > t_j$. Écrire une fonction C qui calcule le nombre d'inversions d'un tableau t de taille n.
8. Soit un tableau t de n entiers, et soit f la fonction $f : i \mapsto t_i$. Écrire en C la fonction *injection* qui détermine si f est injective. Écrire la fonction *bijection* qui détermine si f est une bijection $[0, \ldots, n-1] \rightarrow [0, \ldots, n-1]$.

Université Bordeaux I — **Module MIAS 302**
Travaux dirigés — **Feuille 7**

Chaînes de caractères

Écrire les fonctions suivantes :

1. `void insert(char chaine[], int pos, char destination[])` qui insère chaîne en position pos dans destination.

2. `void remove(int debut, int n, char source[])` qui supprime n caractères à partir de début.

3. `int match(char motif[], char source[])` qui donne la première position de la chaîne motif dans source (retourne -1 si le motif n'apparaît pas).

4. `void replace(char motif[], char par[], char source[])` qui remplace toutes les occurrences de motif dans source par "par".

Université Bordeaux I | **Module MIAS 302**
Travaux dirigés | **Feuille 8**

Matrices

On suppose le type *matrice* défini par :

```
#define HAUTEUR 10
#define LARGEUR 20

typedef int matrice [HAUTEUR][LARGEUR];
```

1. Ecrire une fonction C qui construit la matrice *Identité* de dimension n.

2. Une loi de composition $\otimes$, sur l'ensemble des entiers compris entre 0 et $n-1$, est donnée sous forme de matrice ; on la note ci-dessous multiplicativement ($M_{ij} = i \otimes j$) :

$\otimes$	0	1	2	3
0	0	0	0	0
1	1	1	1	1
2	1	2	0	0
3	1	3	1	3

$$i \otimes j = (i^j) \bmod 4$$

   (a) Ecrire des fonctions C qui vérifient si cette loi est commutative (resp. associative).
   (b) Ecrire une fonction C qui calcule l'élément neutre à gauche e, s'il existe (ie. pour tout $x, \quad ex = x$) ; idem pour l'élément neutre à droite.
   (c) Ecrire une fonction C qui vérifie si la loi est simplifiable à gauche (ie. pour tout $a, \quad ax = ay \Rightarrow x = y$). idem pour la simplification à droite.

3. Ecrire une fonction C qui transforme une matrice en sa transposée.

4. En supposant définies les fonctions (vues en cours) `produit(P,A,B,la,ca_lb,cb)` et `copier(destination,source,l,c)` définir une fonction qui calcule la n-ième puissance d'une matrice (carrée).

Tableaux & Structures

1. Définir une structure de donnée `Point` permettant de représenter un point dans un plan ; déclarer un tableau de `CARD_MAX` points.
2. Ecrire une fonction `distance(a,b)` qui calcule la distance euclidienne entre les points a et b.
3. Un tableau de points peut coder un polygone $(p_0, p_1, ..., p_{k-1})$; pour cela, il suffit de ranger le i^e sommet dans la i^e case du tableau et de garder en mémoire le nombre de sommets du polygone (k).
   (a) Écrire une fonction qui vérifie si un polygone est un triangle équilatéral : la fonction retournera 1 si tous les cotés sont de même longueur, 0 sinon.
   (b) Écrire une fonction qui calcule le périmètre d'un polygone.
   (c) Écrire une fonction qui vérifie si un polygone est équilatéral (renvoyer 0 pour faux, 1 sinon).
4. Écrire une fonction `voisins(x, P, n, d)` qui calcule le nombre de points du tableau `P` (de taille `n`) situés à une distance inférieure à `d` du point `x`.
5. Écrire une fonction `a_un_triangle_equilateral(P,n)` qui retourne 1 si l'ensemble de n points contenus dans `P` possède trois points formant un triangle équilatéral et 0 sinon.

Université Bordeaux I — **Module MIAS 302**
Travaux dirigés — **Feuille 10**

Polynômes

On suppose le type *polynome* défini par :

```
#define DEGRE_MAX 100

typedef double polynome [DEGRE_MAX + 1];
```

1. Les polynômes de Tchebycheff T_n sont définis par :

$$\cos nx = T_n(\cos x) \ ,$$

et vérifient la récurrence :

$$T_{n+1} = 2XT_n - T_{n-1} \ .$$

avec les conditions initiales $T_0 = 1$, $T_1 = X$. Ecrire une fonction qui calcule T_n.

2. On définit la division euclidienne du polynôme A par le polynôme B avec la formule :

$$A = BQ + R$$

et la condition : degré $(R) <$ degré (B). Ecrire la fonction qui calcule le quotient Q et le reste R.

3. La division selon les puissances croissantes à l'ordre n est définie par :

$$A = BQ + R$$

où R ne contient pas de terme de degré inférieur à n. Ecrire la fonction qui calcule le quotient Q et le reste R.

4. Polynômes de Bernoulli.

   (a) Ecrire en langage C la fonction qui, à un polynôme A, associe :

$$\int_0^1 A(t)dt$$

   (b) Ecrire en langage C la fonction f qui, à un polynôme A, associe le polynôme B défini par :

$$\begin{cases} B' = A \\ \displaystyle\int_0^1 B(t)dt = 0 \end{cases}$$

   (c) Le polynôme de Bernoulli B_n est défini par la récurrence :

$$B_n = f(nB_{n-1}), \qquad B_0 = 1$$

   Ecrire une fonction C qui calcule B_n.

Factorielle

1. Editer, compiler, et exécuter le programme suivant :

```
long factorielle (int n) {
  int k;
  long f = 1;

  for (k = 2; k <= n; k++)
    f = f * k;
  return f;
}

void main () {
  int n;

  printf ("Entrer un entier : ");
  scanf ("%d", &n);
  printf ("%d! = %ld\n", n, factorielle (n) );
}
```

2. Quelle est la plus petite valeur de n pour laquelle la valeur calculée par la fonction `factorielle` n'est pas correcte ? Explication ?

   Recommencer en remplaçant `long` par `short`.

3. Examiner l'évolution des valeurs des variables k et f, en utilisant les outils de mise au point de Visual C.

Calculs sur les entiers

1 Coefficients du binôme

1. Ecrire un programme qui comporte la fonction *factorielle* et une fonction *binome1* qui calcule :
$$C_n^k = \frac{n!}{k!(n-k)!}$$
La fonction principale doit lire la valeur de n, et afficher toutes les valeurs de C_n^k, pour $0 \leq k \leq n$.
Quelle est la plus petite valeur de n pour laquelle la fonction *binome1* est incorrecte (à cause d'un débordement) ?
2. Ajouter une fonction *binome2* qui, par exemple, calcule
$$C_7^3 = \frac{7 \times 6 \times 5}{3 \times 2 \times 1}$$
sous la forme `7 / 3 * 6 / 2 * 5 / 1` (respecter l'ordre des opérations).
Examiner les résultats en remplaçant *binome1* par *binome2* dans la fonction principale, et expliquer.
3. Ajouter une fonction *binome3* qui, par exemple, calcule C_7^3 sous la forme `5 / 1 * 6 / 2 * 7 / 3`.
Examiner les résultats en remplaçant *binome2* par *binome3* dans la fonction principale. Expliquer pourquoi ce mode de calcul est toujours correct (tant qu'il n'y a pas de débordement).
4. Ajouter une fonction *binome4* qui, par exemple, calcule C_7^3 sous la forme `7 / 1 * 6 / 2 * 5 / 3`.
Examiner les résultats en remplaçant *binome3* par *binome4* dans la fonction principale. Ce mode de calcul est-il toujours correct (au sens de la question précédente) ?
5. Quelle est la plus petite valeur de n pour laquelle il existe k tel que la la fonction *binome3* calcule une valeur incorrecte de C_n^k (à cause d'un débordement) ?

2 Arithmétique circulaire

1. Ecrire la fonction *puissance*, qui calcule a^n, pour a et n entiers naturels, en utilisant le type `int`.
Ecrire une fonction principale qui calcule et affiche tous les entiers compris entre $2^{31} - 8$ et $2^{31} + 7$, en format normal (`%d`), sans signe (`%u`), et hexadécimal (`%x`).
Observer et expliquer les résultats.
2. Recommencer en remplaçant l'exposant 31 par 32. Observer et expliquer les résultats.
3. Recommencer en remplaçant le type `int` par `short`, l'exposant 31 par 15, et 32 par 16.

Université Bordeaux I **Module MIAS 302**
Travaux pratiques **Feuille 3**

Nombres en virgule flottante

1. La fonction `exponentielle` vue en cours s'écrit :

```
#include <math.h>

double exponentielle (double x, double epsilon) {
  int n;
  double s = 1, u = 1;

  for (n = 1; fabs (u) > epsilon; n++) {
    u = u * x / n;
    s = s + u;
  }
  return s;
}
```

   Comparer les résultats fournis par cette fonction, pour $x < 0$ et "grand" en valeur absolue, à ceux fournis en ajoutant, au début du corps de la fonction, la ligne :

```
  if (x < 0)
    return 1 / exponentielle (-x, epsilon);
```

   Utiliser la fonction `exp` de la bibliothèque standard mathématique, pour obtenir la valeur "exacte" (écrire une fonction `main` qui affiche les valeurs des trois fonctions).

2. Expliquer les résultats produits par le programme suivant :

```
void main (void) {
  double q, e;

  for (e = 1e-10 ; e > 1e-18 ; e = e /10 ) {
    q = 1 + e;
    q = q - 1;
    q = q / e;
    printf("%g %g\n", e, q);
  }
}
```

3. La valeur 1 est racine de l'équation

$$x = 5(4(x - \frac{1}{5}) - 3)$$

   Expliquer les résultats produits par le programme suivant :

```
void main (void) {
  int i;
  double x = 1;
```

```
  for (i = 1 ; i < 20 ; i++ ) {
    x = 5 * ( 4 * (x - 1 / 5.0) - 3);
    printf("%g\n", x);
  }
}
```

4. Compléter la fonction suivante pour calculer $\cos x$ à ϵ près, en utilisant le développement en série usuel :

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots + (-1)^n \frac{x^{2n}}{(2n)!} + \cdots$$

```
#include <math.h>

double cosinus (double x, double epsilon) {
  int n;
  double u = ...;
  double s = ...;

  for (n = ... ; fabs(u) > epsilon ; n = ...) {
    u = ...
    s = ...
  }
  return s;
}
```

Ecrire la fonction `main` pour comparer les résultats avec ceux obtenus par la fonction `cos` de la bibliothèque mathématique standard.

5. On considère la *suite de Fibonacci* définie par :

$$\begin{cases} F_0 = 0 \ , \ F_1 = 1 \ , \\ F_{n+1} = F_n + F_{n-1} \ , \ n > 0 \end{cases}$$

Ecrire une fonction qui calcule F_n, pour n donné, et une seconde fonction qui calcule :

$$u_n = \frac{\Phi^n}{\sqrt{5}} \quad \text{avec} \quad \Phi = \frac{1+\sqrt{5}}{2}$$

(Φ est le nombre d'or). Ecrire un programme qui compare F_n et u_n.

Université Bordeaux I — **Module MIAS 302**
Travaux pratiques — **Feuille 4**

Méthode dichotomique

1. La fonction `dichotomie` vue en cours s'écrit :

```
double dichotomie (
  double f (double),
  double a, double b,
  double epsilon
) {
  double c;

  while (b - a > epsilon) {
    c = (a + b) / 2;
    if (f(c) < 0)
      a = c;
    else
      b = c;
  }
  return b;
}
```

   Elle est correcte si $f(a) < 0$ et $f(b) > 0$. Modifier aussi simplement que possible cette fonction pour qu'elle soit correcte dès que $f(a)$ et $f(b)$ sont de signes contraires.

2. Vérifier expérimentalement que le nombre d'itérations nécessaires pour calculer une valeur approchée à 10^{-n} près, par la méthode de dichotomie, est proportionnel à n : afficher (ou noter à la main, en utilisant les outils de déboguage de Visual C) les résultats obtenus (nombres d'itérations) pour $n = 2, 4, 6, 8, 10, 12, 14$ et les équations suivantes :

$$\sin x = \frac{1}{2}\ ,\ x \in [0\ 1.5]$$

$$x^2 = 2\ ,\ x \in [1\ 2]$$

$$e^x = 2\ ,\ x \in [0\ 1]$$

3. On désire, pour le jeu du nombre mystérieux, comparer l'efficacité d'une *stratégie dichotomique* par rapport à une *stratégie aléatoire*. Pour cela, on approxime le nombre moyen d'essais nécessaires à la découverte du nombre mystérieux en faisant la moyenne d'un grand nombre d'expériences sur différents intervalles de recherche ([0,511], [0,1023], [0,2047], [0, 4095],...).

   Compléter les fonctions `coups_dicho()` et `coups_hasard()` qui retournent le nombre d'essais qu'il a fallu réaliser pour déterminer la valeur du nombre mystérieux `sol` (compris entre `a` et `b`) en utilisant soit une stratégie dichotomique, soit une stratégie aléatoire.

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

```
#define INTER_DEPART 511
#define NB_INTER 7
#define ITERATION 1000

long hasard(long a, long b){ /* renvoie un entier appartenant à [a,b] */
   return a + rand(b - a + 1) ;
}

long coups_dicho(long a, long b, long sol){
...
}


long coups_hasard(long a, long b, long sol){
...
}

void main(){
  long cpt_has, cpt_dicho ;
  long maximum = INTER_DEPART ;
  long ite, i , x, inter ;
  srand(...) ; /* initialisation du generateur */
  printf("intervalle\t\t dicho\t hasard \n") ;
  for(inter = 1 ; inter <= NB_INTER ; inter++){ /* traiter un intervalle */
     cpt_has = 0 ;
     cpt_dicho = 0 ;
     for(ite = 1 ; ite <= ITERATION ; ite++){
       x = hasard(0,maximum) ;
       cpt_has =  cpt_has + coups_hasard(0,maximum, x) ;
       cpt_dicho =  cpt_dicho + coups_dicho(0,maximum, x) ;
     }
     printf("[0, %ld]\t\t  %5.2lf\t  %5.2lf\n",
            maximum,
            ((double)cpt_dicho) / ITERATION,
            ((double)cpt_has) / ITERATION) ;
     maximum = maximum * 2 + 1 ;
  }/* for(inter...} */
}/* main*/
```

Université Bordeaux I **Module MIAS 302**
Travaux pratiques **Feuille 5**

Structures

1. On appelle *séquence* toute suite d'entiers terminée par un zéro.

   Faire une fonction qui lit une séquence et retourne le minimum, le maximum et la moyenne d'une séquence. Pour cela, on définira une structure MMM composée de deux entiers (min et max) et d'un flottant (moyenne).

   Utiliser cette fonction pour faire un programme qui lit au clavier une suite de séquences (terminée par la séquence vide qui ne contient que 0) et qui affiche le min, le max et la moyenne de chaque séquence.

2. On utilise la définition et la fonction suivantes :

   ```
   struct complexe {
     double reel, imaginaire;
   };

   struct complexe produit (struct complexe u, struct complexe v) {
     struct complexe z;

     z.reel = u.reel * v.reel - u.imaginaire * v.imaginaire;
     z.imaginaire = u.reel * v.imaginaire + u.imaginaire * v.reel;
     return z;
   }
   ```

   Ecrire une fonction qui convertisse un couple (r, θ) en un complexe de module r et d'argument θ (en degrés).

   Tester en affichant les puissances du nombre complexe de module 2 et d'argument 30 degrés : ces puissances successives seront calculées avec la fonction `produit` ci-dessus.

3. Ecrire une fonction qui calcule l'exponentielle d'un nombre complexe z, comme somme de la série :

   $$\sum_{n=0}^{\infty} \frac{z^n}{n!}$$

   Tester en vérifiant que, pour φ réel, $e^{i\varphi} = \cos\varphi + i \sin\varphi$.

Université Bordeaux I | **Module MIAS 302**
Travaux pratiques | **Feuille 6**

Tableaux

On appelle *tirage* toute séquence d'entiers choisis aléatoirement entre 0 et 9999. Pour stocker un tirage, on utilise un tableau (strictement plus grand que le tirage) et un marqueur de fin de tirage, ici -1. Le marqueur de fin délimite la zone utilisée par le tirage dans le tableau et doit donc, obligatoirement, figurer dans le tableau. Exemples :

```
int tab1[] =[1] {1,9,3,-1,4,5,-1,0,3,8,-5}; /* tirage 1 9 3 */
int tab2[] = {1,1,1,2,0,-1}; /* tirage 1 1 1 2 0 */
int tab3[] = {-1,3,1,-8,14,78}; /* tirage vide */
int tab4[] = {1,3,-4,-1}; /* erreur : nombre négatif dans le tirage*/
```

1. Écrire et tester (sur `tab1`, `tab2` et `tab3`) les fonctions qui calculent les mesures suivantes :
   - `longueur(t)` d'un tirage,
   - `max(t)` : indice de l'élément maximal (tirage vide : -1),
   - `min(t)` : indice de l'élément minimal (tirage vide : -1),
   - `moyenne(t)` (tirage vide : 0),
   - `ecart_type(t)` d'un tirage (tirage vide : 0)

$$\sigma(t) = \sqrt{\sum_{i=1}^{i=n} \frac{s_i^2}{n} - \left(\sum_{i=1}^{i=n} \frac{s_i}{n}\right)^2}$$

2. Écrire une fonction `tirer_alea(a,b,t)` qui place dans `t` un tirage de longueur choisie aléatoirement entre `a` et `b`. Pour ce faire, on utilisera la fonction prédéfinie `rand()`[2] qui retourne un entier choisi aléatoirement entre 0 et un entier positif `RAND_MAX` prédéfini dans la bibliothèque `stdlib.h`.
3. Afficher la longueur, la moyenne, le min, le max et l'écart type de tirages de longueur comprise entre 10 et 1000.
4. Écrire une fonction `echange(t,i,j)` qui échange le contenu des cases `t[i]` et `t[j]`.

[1] Cette affectation n'est permise qu'à l'initialisation.

[2] Par exemple, si vous désirez générer aléatoirement un entier compris entre 1 et 10 vous pouvez utiliser l'instruction : `1 + (int) (10.0*rand()/(RAND_MAX+1.0));`

Chaînes de caractères

On ne considère ici que les caractères codés en ASCII (non accentués).

1. Afficher la table des codes ASCII avec la boucle :

```
for (c = 32; c < 128; c++)
  printf ("%3d %c\n", c, c) ;
```

2. Écrire et tester les fonctions suivantes :

```
void saisir_chaine(char chaine[]){
  scanf("%s",chaine) ;
}
void afficher_chaine(char chaine[]){
 printf("%s", chaine) ;
}
```

3. Écrire une fonction `char en_majuscule(char c)` qui retourne la lettre majuscule correspondante lorsque `c` représente une lettre minuscule et retourne `c` autrement (indication : 'A'-'a' == ... == 'Z'-'z').

4. Faire un programme principal qui saisisse une chaîne puis affiche cette chaîne en lettres majuscules.

5. Pour chiffrer un message on utilisait (au temps de César) la technique dite du *décalage*. Cette technique consiste à associer à la x-ième lettre de l'alphabet, la *$(k+x)$*-ième lettre de l'alphabet (modulo 26), par exemple :
pour $k = 1$ 'A' est codé 'B', 'B' est codé 'C',..., 'Z' est codé 'A' ;
pour $k = 2$ 'A' est codé 'C', 'B' est codé 'D',..., 'Z' est codé 'B'.

   On supposera que les chaînes manipulées ne contiennent que des lettres majuscules (il n'y a ni minuscule, ni espace).

   (a) Écrire une fonction `encoder(k, destination, source)` qui place dans destination la chaîne source codée avec un décalage de k.

   (b) Écrire une fonction `decoder(k, destination, source)` qui place dans destination le résultat du décodage de la chaîne source supposée encodée par un décalage de k.

   (c) Faire un programme principal qui utilise les fonctions `encoder` et `decoder`. Montrer qu'il n'est pas très difficile de *casser* ce type de chiffrement (ie. de savoir décoder sans connaître k).

Université Bordeaux I — **Module MIAS 302**
Travaux pratiques — **Feuille 8**

Tableau à deux dimensions

Soit le type `matrice` et la fonction `afficher` définis par :

```
#define HAUTEUR 15
#define LARGEUR 20

typedef int matrice [HAUTEUR][LARGEUR] ;

void afficher(matrice M, int ligne, int col){
 int l, c ;
 for(l = 0 ; l < ligne ; l++){
  for(c = 0 ; c < col ; c++)
    printf("%4d",M[l][c]) ;
  printf("\n") ;
  }
}
```

1. Écrire (et tester) une fonction `table_multi(matrice M, int n)` qui place dans M la table de multiplication des n premiers nombres non nuls. La séquence `table_multi(M,3) ; affiche(M,3,3) ;` affichera

1	2	3
2	4	6
3	6	9

2. Écrire (et tester) une fonction `jusqu_a_lc(matrice M,int l,int c)` qui place dans M les entiers allant de 1 à $l.c$; La séquence `jusqu_a_lc(M, 3,5) ; affiche(M,3,5) ;` affichera

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

3. Écrire (et tester) une fonction `table_puiss(matrice M, int l, int c)` qui place dans M la table des c premières puissances des l premiers nombres non nuls. La séquence `puissance(M,3,5) ; affiche(M,3,5) ;` affichera

1	1	1	1	1
1	2	4	8	16
1	3	9	27	81

4. Écrire (et tester) une fonction qui construit la matrice *Identité* de dimension n.
5. Écrire (et tester) une fonction `transpose(matrice M, int l, int c)` qui place dans M sa transposée.
6. Écrire (et tester) une fonction `est_symetrique(matrice M, int dim)` qui retourne 1 si le matrice (carré de taille dim) est symétrique et 0 sinon.
7. Écrire (et tester) une fonction `somme(matrice res, matrice A, matrice B, int l , int c)` qui place dans res la somme des matrices A et B.
8. Écrire (et tester) une fonction qui place dans une matrice res le produit de deux matrices A et B.

Université Bordeaux I **Module MIAS 302**
Travaux pratiques **Feuille 9**

Tableaux & Structures

On considère les définitions suivantes :

```
#include <stdlib.h>
#define NMAX 2000

typedef struct { double x, y ; } Point ;

Point tirer_point()
{ struct point p ;
   p.x = 2.0*rand()/RAND_MAX - 1;
   p.y = 2.0*rand()/RAND_MAX - 1;
  return p ;
}
```

1. Écrire une fonction `initialiser(T,n)` qui range dans chacun des n premiers éléments de T un point tiré aléatoirement.
2. Écrire un programme principal qui initialise un tableau de 100 points.
3. Écrire une fonction `distance(a,b)` qui calcule la distance euclidienne entre les points a et b.
4. Écrire une fonction `voisins(p, T, n, d)` qui calcule le nombre de points du tableau `T` (de taille `n`) situés à une distance inférieure à `d` du point `p`.
5. Écrire un programme principal qui calcule 4k/n, où k est la valeur retournée par l'appel `voisins({0,0}, T, n, 1)`. Tester votre programme avec 75 points.
6. Modifier le programme principal précédent afin d'afficher une approximation de π en fonction de n (faire varier n de 10 à 10000).
7. Écrire une fonction `distance_droite(p,a,b)` qui retourne la distance du point `p` à la droite définie par les points `a` et `b`.
8. Écrire une fonction `intersecte(a,b,T,n)` qui retourne le nombre de points de T appartenant à la droite définie par les points `a` et`b`.
9. Écrire un programme principal qui affiche le nombre maximal de points alignés dans un ensemble de points T tiré aléatoirement.

Université Bordeaux I **Module MIAS 302**
Travaux pratiques **Feuille 10**

Divisibilité

1. Complexité.
   Saisir, compiler et exécuter le programme suivant :

```
#include <stdio.h>
#include <math.h>
#include <time.h>
void diviseur_naif (long n){
   long i;
   for (i=1; i<=n ; i++)
      if ( n%i==0 )
         printf( "%10ld",i);
   printf("\n");
}

void diviseur (long n){
   long i;
   long rac = (long) sqrt((double)n);
   for (i=1; i<rac ; i++)
      if ( n%i==0 )
         printf( "%10ld%10ld",i,n/i);
   if (n==rac*rac)
      printf("%10ld",rac);
   printf("\n");
}

void main(){
   long n;
   time_t debut,fin;
   for (n=1000;n<1E9;n*=10){
      printf ("\nn=%ld fonction diviseur naif\n",n);
      debut= time(NULL);
         diviseur_naif(n);
      fin=time(NULL);
      printf("\ntemps de calcul :%f s",difftime(fin,debut));
      printf ("\nn=%ld fonction diviseur\n",n);
      debut= time(NULL);
         diviseur(n);
      fin=time(NULL);
      printf("\ntemps de calcul :%f s",difftime(fin,debut));
   }
}
```

   Comparer les temps d'exécution des deux fonctions.(un temps de 0 s signifie temps inférieur à une seconde) Expliquer la différence de temps d'exécution. Que peut-on en conclure ?

2. Décomposition d'un nombre en facteurs primaires.
   Tout nombre entier peut être décomposé de manière unique en un produit de facteurs primaires, chacun de ces facteurs étant constitué d'une valeur (nombre premier) et d'un exposant.
   Exemple : 58212 est le produit des facteurs 2^2, 3^3 et 7^2 et 11.

   (a) Définir un type facteur permettant de stocker un facteur primaire.
   (b) Définir un type nombre permettant de stocker tous les facteurs primaires d'un nombre entier. (on remarquera que le nombre de facteurs primaires d'un entier de type long est forcement inférieur à 11). On fera suivre le dernier facteur par une sentinelle dont la valeur sera négative.
   (c) Écrire une fonction décomposer qui associe à un nombre entier l'ensemble des facteurs primaires dont il est le produit.
   (d) Écrire une fonction ppcm qui à partir des facteurs primaires de deux entiers a et b, calcule le ppcm de ces deux entiers.
   (e) Écrire une fonction pgcd qui à partir des facteurs primaires de deux entiers a et b, calcule le pgcd de ces deux entiers.