Correction series 01: C Basics

Exercise 1: Ball bounces

(file rebond1.c)

```c
#include <stdio.h>
#include <math.h>

/* On déclare g constante. Elle ne peut plus être modifiée dans le   *
 * reste du code                                                     */
double const g = 9.81;

int main(void)
{
    // Déclarations

    double v = 0.0, v1 = 0.0;   // vitesses avant et après le rebond
    double h = 0.0, h1 = 0.0;   // hauteur avant le rebond, hauteur de remontée
    double H0 = 1.0, eps = 0.1; // hauteur initiale, coefficient de rebond
    int NBR = -1;               // nombre de rebonds voulus

    /*
     * Entrée des valeurs par l'utilisateur,
     * avec test de validité
     */

    do {
        printf("Coefficient de rebond (0 <= coeff < 1) :\n");
        scanf("%lf", &eps);

    /* répétition tant que l'utilisateur n'entre pas une valeur
     * correcte pour eps                                         */
    } while ( (eps < 0.0) || (eps >= 1.0) );

    do {
        printf("Hauteur initiale      (0 <= H0)        :\n");
        scanf("%lf", &H0);

    /* répétition tant que l'utilisateur n'entre pas une valeur
     * positive pour H0                                          */
    } while ( H0 < 0.0);

    do {
        printf("Nombre de rebonds     (0 <= N )        :\n");
        scanf("%d", &NBR);

    /* répétition tant que l'utilisateur n'entre pas une valeur
     * positive pour NBR                                         */
    } while (NBR < 0);

    // ========== Boucle de calcul ==========

    // Au départ (0 rebond), la hauteur de rebond vaut H0
    h = H0;

    for (int nombre = 0; nombre < NBR; ++nombre) {
        v  = sqrt(2.0 * g * h);
        v1 = eps * v;              // vitesse après le rebond
        h1 =(v1 * v1) / (2 * g);   // hauteur à laquelle elle remonte...
        h  = h1;                   // ...laquelle devient la nouvelle hauteur initiale

        printf("rebond %d : %f\n", nombre+1, h);
    }
```

```
    // Affichage du résultat
    printf("Au %dème rebond, la hauteur sera de %f m.\n", NBR, h);

    return 0;
}
```

Note:

in the `for` loop of this code, the `nombre` variable is initialized to `0`. However, on the first pass through this loop, we're already calculating the *first* (1) bounce. This is why we have:

`printf("rebond %d : %f\n", nombre+1, h);`

where the number of bounces displayed is `nombre+1` and not `nombre`. A solution to avoid this addition would be to initialize `nombre = 1` directly in the loop, but but *warning!*, this would mean modifying the loop's condition as well.

Exercise 2: Prime numbers

(file premiers.c)

```
// C99, pour changer
#include <stdio.h>
#include <math.h>

int main(void)
{
    // Saisie du nombre à tester
    int n = 2;
    do {
        printf("Entrez un nombre entier > 1 :\n");
        scanf("%d", &n);
    } while (n <= 1);

    int diviseur = 1;  // Diviseur trouvé. Si c'est 1, alors le nombre est premier

    if (0 == (n % 2)) {
        // Le nombre est pair
        if (n != 2) {
            diviseur = 2; // n n'est pas premier (2 est diviseur)
        }
    } else {
        const double borne_max = sqrt((double) n);
        for (int i = 3; (diviseur == 1) && (i <= borne_max); i += 2) {
            if (0 == (n % i)) {
                diviseur = i; // n n'est pas premier (i est diviseur)
            }
        }
    }

    printf("%d", n);

    if (diviseur == 1) {
        printf(" est premier");
    } else {
        printf(" n'est pas premier, car il est divisible par %d", diviseur);
    }
    printf("\n");
    return 0;
}
```

A few explanations:

The value `n` entered by the user must be an integer. However, if we want to calculate the square root of this number, it is necessary to temporarily transform it into a double. This operation is performed as shown on the line:

```
const double borne_max = sqrt((double) n);
```

by specifying the desired type in parentheses before the variable.

In the `for` loop at the end of the program, there is a double condition:

1. we check that we did not find a divisor in the previous iteration by testing the value of `diviseur`. We could avoid this test by inserting a `break` in the last line of the `if` condition, but the use of `break` and `continue` is strongly discouraged.
2. we check that we have not yet exceeded the `borne_max`.

Results:

```
2 est premier
16 n'est pas premier, car il est divisible par 2
17 est premier
91 n'est pas premier, car il est divisible par 7
589 n'est pas premier, car il est divisible par 19
1001 n'est pas premier, car il est divisible par 7
1009 est premier
1299827 est premier
2146654199 n'est pas premier, car il est divisible par 46327
```

Remarks:

If you want to test numbers larger than `2147483647` (i.e. $2^{31}-1$, which by the way is prime!), replace `int n;` with `unsigned long n;` You also need to change the declaration of `i`: `unsigned long i;`

You can then test up to `4294967295` (Try for example `4292870399`).

For those who would like to test even larger numbers, you can replace the previous integers (`int`, then `unsigned long`) with: `unsigned long long n;` (don't forget to do this for `n` and for `i`; note that this extended type is only available since C99).

This allows you to go up to `18446744073709551615` (Try for example `18446744073709551577` or `18446744073709551557` (you have to wait for a little while though)).

The `printf()` format for `long unsigned` is `"%lu"` and for `long long unsigned`, `"%llu"`.

Exercise 3: Solving a third-degree equation

(file deg3.c)

```c
/* C89 */
#include <stdio.h>

/* ligne pour avoir M_PI (= pi). A mettre AVANT le include de math.h. */
#define  _USE_MATH_DEFINES
#include <math.h>
/* Si vraiment le compilateur respecte strictement le standard, même *
 * _USE_MATH_DEFINES ne fera pas l'affaire. On fait alors le travail *
 * nous même :                                                       */
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

int main(void)
{
  double a0 = 0.0, a1 = 0.0, a2 = 0.0,
    z1 = 0.0, z2 = 0.0, z3 = 0.0,
    D = 0.0, Q = 0.0, R = 0.0, S = 0.0, T = 0.0;
```

```
  printf("Entrez a2, puis a1, puis a0 :\n");
  scanf("%lf %lf %lf", &a2, &a1, &a0);

  Q = (3.0 * a1 - a2*a2) / 9.0;
  R = (9.0 * a2 * a1 - 27.0 * a0 - 2.0 * a2*a2*a2) / 54.0;
  D = Q*Q*Q + R*R;
  printf("(pour info D = %f)\n", D);

  if (D < 0.0) { /* test du déterminant */
     /* cas de trois racines réelles */

     T  = acos( R / sqrt(-Q*Q*Q) );
     z1 = 2.0 * sqrt(-Q) * cos(T/3.0)             - a2/3.0;
     z2 = 2.0 * sqrt(-Q) * cos( (T+2*M_PI) / 3.0 ) - a2/3.0;
     z3 = 2.0 * sqrt(-Q) * cos( (T+4*M_PI) / 3.0 ) - a2/3.0;
     printf("Trois racines ( %f , %f , %f )\n", z1, z2, z3);
  } else {
     /* cas de moins de trois racines réelles */

     /* calcul de S */
     double s = R+sqrt(D);
     const double un_tiers = 1.0/3.0;
     if (0.0 == s)     { S = 0.0;                }
     else if (s < 0.0) { S = -pow(-s, un_tiers); }
     else if (s > 0.0) { S =  pow( s, un_tiers); }

     /* calcul de T */
     s = R-sqrt(D);
     if (0.0 == s)     { T = 0.0;                }
     else if (s < 0.0) { T = -pow(-s, un_tiers); }
     else if (s > 0.0) { T =  pow( s, un_tiers); }

     printf("(pour info S = %f, T = %f, S+T= %f)\n", S, T, S+T);

     /* calcul des solutions */
     z1 = -a2 / 3.0 + S + T;
     if ((0.0 == D) && (S+T != 0.0)) {
       z2 = -a2 / 3.0 - (S + T) / 2.0;
       printf("Deux racines...\n");
       printf("   l'une simple   : %f\n", z1);
       printf("   l'autre double : %f\n", z2);
     } else {
       printf("Une seule racine : %f\n", z1);
     }
  }

  return 0;
}
```

Note: the proposed version is not numerically "clean". You certainly already know that we should ***never*** make equality test (`==`) with `double` (it doesn't make sense numerically because of lack of precision on the values). It would be better to define a function (e.g. `double_eq` or `double_cmp`) that takes three `doubles` (the two numbers to compare and a precision) and returns `0` if the absolute value of the difference between the two numbers is greater than the precision, and non-zero otherwise.

I suggest you implement this improvement as a complementary exercise.

Exercise 4: Approximate calculation of an integral

(file integrale.c)

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

```c
double f(double x) {
  return sin(x);
}

double demander_nombre(void)
{
  double res = 0;
  printf("Entrez un nombre réel : ");
  scanf("%lf", &res);
  return res;
}

double integre(double a, double b)
{
  double res =
       41.0 * ( f(a) + f(b) )
    + 216.0 * ( f((5*a+b)/6.0) + f((5*b+a)/6.0) )
    +  27.0 * ( f((2*a+b)/3.0) + f((2*b+a)/3.0) )
    + 272.0 * f((a+b)/2.0) ;

  res *= (b-a)/840.0;

  return res;
}


int main(void)
{
  double
    a = demander_nombre(),
    b = demander_nombre();
  printf("Integrale de sin(x) entre %f et %f : %.12f\n", a, b, integre(a, b));
  return 0;
}
```

Exercise 5: Exchange

(file swap.c)

```c
void echange(int* a, int* b)
{
  /* on sauvegarde la valeur pointée par a pour ne pas la perdre */
  int const copie = *a;

  /* la valeur pointée par a prend la valeur pointée par b       */
  *a = *b;

  /* la valeur pointée par b prend la valeur de copie            */
  *b = copie;
}
```

Exercise 6: Date stories

1. The bug:

   When `days` is 366 and `year` is a leap year (i.e. the last day of a leap year, such as the famous December 31, 2008), the code stays in the `while (days > 365)`: it passes the first `if (IsLeapYear(year))` but does not satisfy `if (days > 366)`. So, we exit the first `if` and start another round in the (infinite) loop.

2. A solution:

The problem comes from mixing the loop for calculating the number of years and the number of days in the year (365 or 366). The solution is to clearly separate the two. Here's one possible solution:

(file zune.c)

```c
/*
 * zune.c
 * ANSI C89
 */

#include <stdio.h>
#include <stdlib.h> /* atoi(3) */

#define MICROSOFT_EPOCH_YEAR 1980
#define december_31_2008 10593

typedef enum {
    JANUARY = 1,
    FEBRUARY,
    MARCH,
    APRIL,
    MAY,
    JUNE,
    JULY,
    AUGUST,
    SEPTEMBER,
    OCTOBER,
    NOVEMBER,
    DECEMBER
} Month;

int IsLeapYear(int y)
{

    return (((y % 4 == 0) && (y % 100 != 0)) || (y % 400 == 0));
}

/* on a besoin de l'année (year) pour le cas spécial (Février) */
int DaysForMonth(int year, Month month)
{
    int days = 31;

    switch (month) {
    case FEBRUARY: /* cas spécial */
        if (IsLeapYear(year))
            days = 29;
        else
            days = 28;
        break;
    case APRIL:
    case JUNE:
    case SEPTEMBER:
    case NOVEMBER:
        days = 30;
        break;
    default:
        days = 31;
    }

    return days;
}

int main(void)
{
```

```c
    int year = MICROSOFT_EPOCH_YEAR;
    int days = 1, d = 31;
    Month month = JANUARY;

    printf("Entrez le nombre de jours écoulés depuis le 31/12/1979 : ");
    scanf("%d", &days);

    /* calcul de year */
    while (days > 0) {
        if (IsLeapYear(year))
            days -= 366;
        else
            days -= 365;
        ++year;
    }
    --year;
    if (IsLeapYear(year))
        days += 366;
    else
        days += 365;

    /* calcul de month */
    month = JANUARY;
    d = DaysForMonth(year, month);
    while (days > d) {
        days -= d;
        d = DaysForMonth(year, ++month);
    }

    printf("%02d/%02d/%d\n", days, month, year);

    return 0;
}
```

3. Based on what has been done above:

   (file unix-time.c)

```c
/*
 * unix-time.c
 * ANSI C89
 */

#include <stdio.h>
#include <time.h>

#define UNIX_EPOCH_YEAR 1970

/* copié de zune.c */
typedef enum {
    JANUARY = 1,
    FEBRUARY,
    MARCH,
    APRIL,
    MAY,
    JUNE,
    JULY,
    AUGUST,
    SEPTEMBER,
    OCTOBER,
    NOVEMBER,
    DECEMBER
} Month;
```

```c
int IsLeapYear(int y)
{

    return (((y % 4 == 0) && (y % 100 != 0)) || (y % 400 == 0));
}

/* on a besoin de l'année (year) pour le cas spécial (Février) */
int DaysForMonth(int year, Month month)
{
    int days = 31;

    switch (month) {
    case FEBRUARY: /* cas spécial */
        if (IsLeapYear(year))
            days = 29;
        else
            days = 28;
        break;
    case APRIL:
    case JUNE:
    case SEPTEMBER:
    case NOVEMBER:
        days = 30;
        break;
    default:
        days = 31;
    }

    return days;
}

int main(void)
{
    time_t now = time(NULL);
    time_t seconds, minutes, hours;
    int year = UNIX_EPOCH_YEAR;
    time_t days = 1, d = 31;
    Month month = JANUARY;

    printf("%u secondes se sont ecoulees depuis le 1.1.1970 a minuit.\n", now);

    seconds = now % 60;
    now /= 60;
    minutes = now % 60;
    now /= 60;
    hours = now % 24;
    now /= 24;

    /* copié de zune.c */
    /* calcul de year */
    days = now;
    year = UNIX_EPOCH_YEAR;
    while (days > 0) {
        if (IsLeapYear(year))
            days -= 366;
        else
            days -= 365;
        ++year;
    }
    --year;
    if (IsLeapYear(year))
        days += 366;
    else
```

```
        days += 365;

    /* calcul de month */
    month = JANUARY;
    d = DaysForMonth(year, month);
    while (days > d) {
        days -= d;
        d = DaysForMonth(year, ++month);
    }
    ++days; /* 0 corespond au 1er Janvier 1970, il faut faire +1 */

    printf("Nous sommes donc le %02d/%02d/%d a %02d:%02d:%02d\n",
            days, month, year, hours, minutes, seconds);

    return 0;
}
```

Yes, there is a one hour delay. The `time(2)` function returns the UTC (Coordinated Universal Time) time while in Switzerland the time zone is GMT+1 (in winter), which explains why the `unix-time` program displays a time delayed compared to Swiss time.

Exercise 7: Multiplication of matrices

(file matrices.c)

```
/* C89 */
#include <stdio.h>

#define N 10

typedef struct {
   double m[N][N];
   size_t lignes;
   size_t colonnes;
} Matrice;

Matrice lire_matrice(void);
void affiche_matrice(const Matrice);
Matrice multiplication(const Matrice a, const Matrice b);

/* ---------------------------------------------------------------------- */
int main(void)
{
  Matrice M1 = lire_matrice();
  Matrice M2 = lire_matrice();

  if (M1.colonnes != M2.lignes)
    printf("Multiplication de matrices impossible !\n");
  else {
    printf("Résultat :\n");
    affiche_matrice(multiplication(M1, M2));
  }
  return 0;
}

/* ---------------------------------------------------------------------- */
Matrice lire_matrice(void)
{
  Matrice resultat;
  size_t lignes = 2;
  size_t colonnes = 2;

  printf("Saisie d'une matrice :\n");
```

```
  do {
    printf("  Nombre de lignes (< %d) : ", N+1);
    scanf("%lu", &lignes); /* "%zu" en C99 ; c'est mieux. */
  } while ((lignes < 1) || (lignes > N));

  do {
    printf("  Nombre de colonnes (< %d) : ", N+1);
    scanf("%lu", &colonnes);
  } while ((colonnes < 1)  || (colonnes > N));

 resultat.lignes = lignes;
 resultat.colonnes = colonnes;
 { size_t i, j;
  for (i = 0; i < lignes; ++i)
    for (j = 0; j < colonnes; ++j) {
      printf("  M[%lu, %lu]=", i+1, j+1);
      scanf("%lf", &resultat.m[i][j]);
    }
  }

  return resultat;
}

/* ---------------------------------------------------------------------- */
Matrice multiplication(const Matrice a, const Matrice b)
{
  Matrice resultat = a; /* Disons que par convention on retourne a si la
                         * multiplication ne peut pas se faire.
                         */
  size_t i, j, k; /* variables de boucle */

  if (a.colonnes == b.lignes) {
    resultat.lignes = a.lignes;
    resultat.colonnes = b.colonnes;

    for (i = 0; i < a.lignes; ++i)
      for (j = 0; j < b.colonnes; ++j) {
        resultat.m[i][j] = 0.0;
        for (k = 0; k < b.lignes; ++k)
      resultat.m[i][j] += a.m[i][k] * b.m[k][j];
      }
  }

  return resultat;
}

/* ---------------------------------------------------------------------- */
void affiche_matrice(const Matrice matrice)
{
  size_t i, j;
  for (i = 0; i < matrice.lignes; ++i) {
    for (j = 0; j < matrice.colonnes; ++j)
      printf("%g ", matrice.m[i][j]);
    putchar('\n');
  }
}
```

Note

The `Matrix` type is defined as an array of size `N x N` (with `N = 10`). This implies that as soon as we create a `Matrix`, we allocate memory space for 10 x 10 numbers, even if the user then decides that its size will be lower. However, we cannot do otherwise at the moment, since in C, we are not allowed to declare an array without specifying its size.

To do better, you have to wait for the pointers...

Note: we could have used VLAs, but

1. we could not make it a global type (like `Matrix` here), which is still the primary spirit of this exercise;
2. as mentioned in class, VLAs are very open to criticism (and criticized) for cases like this (uncontrolled size), because they are allocated on the stack without any verification.

Exercise 8: Complex numbers (Level 1)

(file complexe.c)

```c
#include <stdio.h>

typedef struct {
  double x;
  double y;
} Complexe;

/* Solution simple */
void affiche(const Complexe z)
{
  printf("(%g,%g)", z.x, z.y);

  /* autre solution :  printf("%g+%gi", z.x, z.y); */
}

/* Solution plus complexe mais plus élégante */
void affiche2(const Complexe z)
{
  double y_affiche = z.y;

  if ((z.x == 0.0) && (z.y == 0.0)) {
    printf("0");
    return;
  }

  if (z.x != 0.0) {
    printf("%g", z.x);
    if (z.y > 0.0)
      printf("+"); /* ou putchar('+'); */
    else if (z.y < 0.0) {
      putchar('-');
      y_affiche = -z.y;
    }
  }
  if (y_affiche != 0.0) {
    if ((z.x == 0.0) && (y_affiche == -1.0))
      putchar('-');
    else if (y_affiche != 1.0)
      printf("%g", y_affiche);
    putchar('i');
  }
}

/* ------------------------------------ */
Complexe addition(const Complexe z1, const Complexe z2)
{
  Complexe z;
  z.x = z1.x + z2.x;
  z.y = z1.y + z2.y;
  return z;
}
```

```c
/* ------------------------------------ */
Complexe soustraction(const Complexe z1, const Complexe z2)
{
  Complexe z;
  z.x = z1.x - z2.x;
  z.y = z1.y - z2.y;
  return z;
}

/* ------------------------------------ */
Complexe multiplication(const Complexe z1, const Complexe z2)
{
  Complexe z;
  z.x = z1.x * z2.x - z1.y * z2.y;
  z.y = z1.x * z2.y + z1.y * z2.x;
  return z;
}

/* ------------------------------------ */
Complexe division(const Complexe z1, const Complexe z2)
{
  const double r = z2.x*z2.x + z2.y*z2.y;
  Complexe z;
  z.x = (z1.x * z2.x + z1.y * z2.y) / r;
  z.y = (z1.y * z2.x - z1.x * z2.y) / r;
  return z;
}

/* ------------------------------------ */
int main(void)
{
  Complexe un = { 1.0, 0.0 };
  Complexe i  = { 0.0, 1.0 };
  Complexe j;
  Complexe z;
  Complexe z2;

  j = addition(un, i);

  affiche(un); printf(" + "); affiche(i); printf(" = ");
  affiche(j); putchar('\n');

  z = multiplication(i,i);
  affiche(i); printf(" * "); affiche(i); printf(" = ");
  affiche(z); putchar('\n');

  z = multiplication(j,j);
  affiche(j); printf(" * "); affiche(j); printf(" = ");
  affiche(z); putchar('\n');

  z2 = division(z,i);
  affiche(z); printf(" / "); affiche(i); printf(" = ");
  affiche(z2); putchar('\n');

  z.x = 2.0; z.y = -3.0;
  z2 = division(z,j);
  affiche(z); printf(" / "); affiche(j); printf(" = ");
  affiche(z2); putchar('\n');

  return 0;
}
```

Exercise 9: Complex numbers revisited (Level 2)

(file complexe2.c)

```c
// C99
#include <stdio.h>
#include <math.h>

// ------------------------------------------------------------------------
typedef struct {
  double x;
  double y;
} Complexe;

typedef struct {
  Complexe z1;
  Complexe z2;
} Solutions;

// ------------------------------------------------------------------------
void affiche(const Complexe z);

Complexe addition      (const Complexe z1, const Complexe z2);
Complexe soustraction  (const Complexe z1, const Complexe z2);
Complexe multiplication(const Complexe z1, const Complexe z2);
Complexe division      (const Complexe z1, const Complexe z2);
Complexe racine        (const Complexe z);

Solutions resoudre_second_degre(const Complexe b, const Complexe c);

// ========================================================================
int main(void)
{
  Complexe b  = { 3.0, -2.0 };
  Complexe c  = { -5.0, 1.0 };

  Solutions s = resoudre_second_degre(b, c);

  printf("Avec b="); affiche(b);
  printf(" et c=");  affiche(c);
  printf(" on a :\n");
  printf("  z1="); affiche(s.z1); putchar('\n');
  printf("  z2="); affiche(s.z2); putchar('\n');

  return 0;
}

// ========================================================================
void affiche(const Complexe z)
{
  if ((z.x == 0.0) && (z.y == 0.0)) {
    printf("0");
    return;
  }

  double y_affiche = z.y;

  if (z.x != 0.0) {
    printf("%g", z.x);
    if (z.y > 0.0)
      putchar('+');
    else if (z.y < 0.0) {
```

```
      putchar('-');
      y_affiche = -z.y;
    }
  }
  if (y_affiche != 0.0) {
    if ((z.x == 0.0) && (y_affiche == -1.0))
      putchar('-');
    else if (y_affiche != 1.0)
      printf("%g", y_affiche);
    putchar('i');
  }
}

// =============================================================================
Complexe addition(const Complexe z1, const Complexe z2)
{
  return (Complexe) { z1.x + z2.x, z1.y + z2.y };
}

// =============================================================================
Complexe soustraction(const Complexe z1, const Complexe z2)
{
  return (Complexe) { z1.x - z2.x, z1.y - z2.y };
}

// =============================================================================
Complexe multiplication(const Complexe z1, const Complexe z2)
{
  return (Complexe) {
    z1.x * z2.x - z1.y * z2.y ,
    z1.x * z2.y + z1.y * z2.x
  };
}

// =============================================================================
Complexe division(const Complexe z1, const Complexe z2)
{
  const double r = z2.x*z2.x + z2.y*z2.y;
  return (Complexe) {
    (z1.x * z2.x + z1.y * z2.y) / r ,
    (z1.y * z2.x - z1.x * z2.y) / r
  };
}

// =============================================================================
Complexe racine(const Complexe z)
{
  const double r = sqrt(z.x * z.x + z.y * z.y);
  Complexe retour;

  retour.x = sqrt((r + z.x) / 2.0);
  if (z.y >= 0.0)
    retour.y = sqrt((r - z.x) / 2.0);
  else
    retour.y = - sqrt((r - z.x) / 2.0);

  return retour;
}

// =============================================================================
Solutions resoudre_second_degre(const Complexe b, const Complexe c)
{
  // Pour faciliter l'écriture
```

```
  const Complexe deux   = { 2.0, 0.0 };
  const Complexe quatre = { 4.0, 0.0 };

  // delta^2 = b^2 - 4c
  const Complexe sdelta = racine(soustraction(multiplication(b, b),
                                              multiplication(quatre, c)));

  // Calcule -b (ou alors faire une fonction "oppose")
  const Complexe mb = { -b.x, -b.y };

  // Réponse = -b +- delta / 2
  return (Solutions) {
    division( soustraction(mb, sdelta) , deux) ,
    division( addition    (mb, sdelta) , deux)
  };
}
```

Exercise 10: Writing to a file

Here is a relatively complete code. The biggest difficulty is to correctly handle the various possible anomalies when entering input. If you don't do this, your code will obviously be much shorter (but much less robust). So look carefully at the correction below.

The most subtle difficulty for correct processing is to remove the newline (`\n`) which is still hanging around in the input buffer after the `scanf()` on the age.

(file ecriture.c)

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

/* taille maximale pour un nom */
#define TAILLE_NOM 1024

int main(void)
{
  char const nom_fichier[] = "data.dat"; /* le nom du fichier */
  FILE* sortie;
  int taille_lue;

  char nom[TAILLE_NOM+1]; /* pour stocker le "nom" à lire depuis le clavier */
  unsigned int age;       /* pour stocker l'"âge" à lire depuis le clavier */

  /* Ouverture de data.dat en écriture (w=write) */
  sortie = fopen(nom_fichier, "w");

  /* on teste si l'ouverture du flot s'est bien réalisée */
  if (sortie == NULL) {
    fprintf(stderr,
            "Erreur: le fichier %s ne peut etre ouvert en écriture !\n",
            nom_fichier);
    return 1; /* retourne un autre chiffre que 0 car il y a eu une erreur */
  }

  /* itération sur les demandes à entrer :
     on continue tant que stdin est lisible */
  while (! feof(stdin)) {

    /* tant qu'un nom vide est entré */
    do {
      printf("Entrez un nom (CTRL+D pour terminer) : "); fflush(stdout);
      fgets(nom, TAILLE_NOM+1, stdin);
      taille_lue = strlen(nom) - 1;
```

```
      if ((taille_lue >= 0) && (nom[taille_lue] == '\n'))
        nom[taille_lue] = '\0';
    } while (!feof(stdin) && (taille_lue < 1));

    if (! feof(stdin)) {
      /* L'utilisateur a bien saisi un nom, on peut donc lui demander
       * de saisir l'age.
       */
      printf("âge : "); fflush(stdout);
      taille_lue = scanf("%u", &age);

      if (taille_lue != 1) {
        printf("Je vous demande un age (nombre entier positif) pas "
               "n'importe quoi !\nCet enregistrement est annulé.\n");
        while (getc(stdin) != '\n'); /* vide le tampon d'entrée */
      } else {
        getc(stdin); /* récupère le \n résiduel */
        /* ecriture dans le fichier */
        fprintf(sortie, "%s %d\n", nom, age);
      }
    }
  }

  /* purisme : retour a la ligne pour finir proprement la question */
  putchar('\n');

  fclose(sortie); /* fermeture du fichier */

  return 0;
}
```

Exercise 11: Reading from a file

No difficulty here.

(file lecture.c)

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

/* on affiche les noms sur 15 caractères, comme spécifié dans la donnée */
#define TAILLE_NOM 15

int main(void)
{
  char const nom_fichier[] = "data.dat"; /* le nom du fichier */
  FILE* entree;
  int taille_lue;

  char nom[TAILLE_NOM+1]; /* la donnée "nom" à lire depuis le fichier */
  unsigned int age;       /* la donnée "age" à lire depuis le fichier */

  /* variables nécessaires aux différents calculs */
  unsigned int nb = 0;
  unsigned int age_max = 0;
  unsigned int age_min = (unsigned int) -1; /* truc : -1 sera toujours le plus
                                               grand nombre représentable */
  double total = 0.0;

  /* ouverture du fichier en lecture (r=read) */
  entree = fopen(nom_fichier, "r");

  /* on teste si l'ouverture du flot s'est bien réalisée */
```

```
  if (entree == NULL) {
    fprintf(stderr,
            "Erreur: le fichier %s ne peut etre ouvert en lecture !\n",
            nom_fichier);
    return 1; /* retourne autre chose que 0 car ça s'est mal passé */
  }

  /* On commence par l'affichage du cadre */
  printf("+-----------------+-----+\n");

  /*
   * Et on boucle directement sur la condition de lecture correcte
   * du couple <nom,age> (en fait, sur la condition de lecture correcte
   * de 'age', mais comme il n'est pas possible de lire 'age' si la
   * lecture de 'nom' à échoué...)
   */

  do {
    taille_lue = fscanf(entree, "%15s %u", nom, &age);

    if (taille_lue == 2) { /* la lecture s'est bien passée */
      ++nb; /* nombre de personnes + 1 */
      total += age; /* pour faire la moyenne plus tard */
      /* on vérifie si l'âge lu est le plus grand/petit lu jusqu'ici  */
      if (age_min > age) age_min = age; /* */
      if (age_max < age) age_max = age; /**/

      /* Affichage */
      /* le signe "-" permet d'aligner à gauche */
      printf("| %-15s | %3d |\n", nom, age);
    }
  } while (! feof(entree));

  /* Partie finale */

  fclose(entree);  /* ne pas oublier de fermer le  fichier ! */

  printf("+-----------------+-----+\n");
  printf("  âge minimum     : %3d\n", age_min);
  printf("  âge maximal     : %3d\n", age_max);

  printf("%d personnes, âge moyen : %5.1f ans\n", nb, total/nb);
  /* l'âge moyen est sur 5 positions dont un chiffre après la virgule */

  return 0;
}
```

Exercise 12: Binary files

1. Initial version:

   (file encode-bin-1.c)

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

int main(void) {
  FILE* entree;
  char nom_fichier[] = "./a_lire.bin";
  int taille_lue;
  int code;

  /* Ouverture du fichier binaire en lecture */
```

```
  entree = fopen(nom_fichier, "rb");
  if (entree == NULL) {
      fprintf(stderr,
              "Erreur : je ne peux pas ouvrir le fichier %s en lecture.\n",
              nom_fichier);
      return 1;
  }

  while (!feof(entree)) {
    /* Pour lire un fichier binaire, on utilise fread.
     * Ici, on va lire un int et le sauver dans code.
     */
    taille_lue = fread(&code, sizeof(int), 1, entree);

    if (taille_lue == 1) {
      printf("%d -> %c\n", code, (char) sqrt((double) code));
    }
  }

  fclose(entree);
  return 0;
}
```

2. Full version:

(file decode-bin.c)

```
#include <stdio.h>
#include <string.h>
#include <math.h>

#define TAILLE 1025

int main(void) {

  FILE* entree;
  char nom_fichier[FILENAME_MAX+1] = "";
  int taille_lue;
  int code;

 /* Le fichier à lire est donné par l'utilisateur */
  do {
    printf("Quel fichier voulez vous lire ?\n");
    fgets(nom_fichier, FILENAME_MAX+1, stdin);
    taille_lue = strlen(nom_fichier) - 1;
    if ((taille_lue >= 0) && (nom_fichier[taille_lue] == '\n'))
      nom_fichier[taille_lue] = '\0';
  } while ((taille_lue < 1) && !feof(stdin));

  if (nom_fichier[0] == '\0') return 1;

  entree = fopen(nom_fichier, "rb");
  if (entree == NULL) {
      fprintf(stderr,
              "Erreur : je ne peux pas ouvrir le fichier %s en lecture.\n",
              nom_fichier);
      return 1;
  }

  while ( ! feof(entree) ){
    taille_lue = fread(&code, sizeof(int), 1, entree);

    if (taille_lue == 1) {
      /* on transforme code en double avant de prendre sa racine carrée,
```

```
       * puis on transforme la racine carrée en char avant de l'afficher */
      printf("%c", (char) sqrt((double) code));
    }
  }
  printf("\n");
  fclose(entree);

  return 0;
}
```

3. Encoder:

   (file encode-bin.c)

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

#define TAILLE 1024

void demander_chaine(char* reponse, int taille)
{
  int taille_lue;

  do {
    fgets(reponse, taille+1, stdin);
    taille_lue = strlen(reponse) - 1;
    if ((taille_lue >= 0) && (reponse[taille_lue] == '\n'))
      reponse[taille_lue] = '\0';
  } while ((taille_lue < 1) && !feof(stdin));
}

int main(void) {

  char nom_fichier[FILENAME_MAX+1];
  char phrase[TAILLE+1];

  puts("Dans quel fichier voulez vous écrire ?\n");
  demander_chaine(nom_fichier, FILENAME_MAX);
  if (nom_fichier[0] == '\0') return 1;

  printf("Entrez une phrase (<= %d caractères) :\n", TAILLE);
  demander_chaine(phrase, TAILLE);

  if (phrase[0] != '\0') {
    /* écriture et codage */
    int i;
    int taille;
    FILE* sortie;
    int ecrits;
    unsigned int code;

    /* On va écrire du binaire */
    sortie = fopen(nom_fichier, "wb");
    if (sortie == NULL) {
      fprintf(stderr,
              "Erreur : je ne peux pas ouvrir le fichier %s en écriture.\n",
              nom_fichier);
      return 1;
    }

    taille = strlen(phrase);
    for (i = 0; i < taille; ++i) {
```

```
      code = (unsigned char) phrase[i];
      code *= code;
      printf("%c -> %u -> %d\n", phrase[i], (unsigned char) phrase[i], code);
      ecrits = fwrite(&code, sizeof(int), 1, sortie);

      if (ecrits != 1)  {
        fprintf(stderr,
                "Erreur : je n'ai pas pu écrire plus que %d entiers (sur %d) !\n",
                i, taille);
        return 3;
      }
    }

    fclose(sortie);
  }

  return 0;
}
```

Exercise 13: Statistics on a file

This correction contains several "practical" aspects and should therefore be studied more specifically and well understood.

(file stat.c)

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

/* ===== CONSTANTES ===== */

/* nombre maximum de demandes en cas d'erreur */
#define NB_DEMANDES 3

/* taille maximum d'une Statistique : au plus 256 car il n'y a pas plus
   que 256 char. */
#define TAILLE 256

/* bornes sur les caractères à prendre en compte */
#define start  32
#define stop  253


/* ===== DEFINITIONS DE TYPES ====== */

typedef unsigned long int Statistique[TAILLE];


/* ===== FONCTIONS ====== */
FILE* demander_fichier(void);

void initialise_statistique(Statistique a_initialiser);
/* Rappel : les tableaux sont toujours passés par référence. Pas
   besoin de pointeur supplémentaire ici */

unsigned long int collecte_statistique(Statistique a_remplir,
                                       FILE* fichier_a_lire);

void affiche(Statistique a_afficher, unsigned long int total,
             unsigned short int taille);

/* ========================================================================== */
int main(void)
{
```

```
  FILE* fichier = demander_fichier();
  if (fichier == NULL) {
    printf("=> j'abandonne !\n");
    return 1;
  } else {
    Statistique stat;
    initialise_statistique(stat);
    affiche(stat, collecte_statistique(stat, fichier), stop - start + 1);
    fclose(fichier);
  }

  return 0;
}

/* ============================================================================
 * Fonction demander_fichier
 * ----------------------------------------------------------------------------
 * In:   Un fichier (par référence) à ouvrir.
 * Out:  Ouvert ou non ?
 * What: Demande à l'utilisateur (au plus NB_DEMANDES fois) un nom de fichier
 *       et essaye de l'ouvrir en lecture.
 * ========================================================================== */
FILE* demander_fichier(void)
{
  FILE* f = NULL;
  char nom_fichier[FILENAME_MAX+1];
  size_t taille_lue = 0;
  unsigned short int nb = 0;

  do {
    ++nb;

    /* demande le nom du fichier */
    do {
      printf("Nom du fichier à lire : "); fflush(stdout);
      fgets(nom_fichier, FILENAME_MAX+1, stdin);
      taille_lue = strlen(nom_fichier);
      if ((taille_lue >= 1) && (nom_fichier[taille_lue-1] == '\n'))
        nom_fichier[--taille_lue] = '\0';
    } while ((taille_lue == 0) && !feof(stdin));

    if (nom_fichier[0] == '\0') {
      return NULL;
    }

    /* essaye d'ouvrir le fichier */
    f = fopen(nom_fichier, "r");

    /* est-ce que ça a marché ? */
    if (f == NULL) {
      printf("-> ERREUR, je ne peux pas lire le fichier \"%s\"\n",
             nom_fichier);
    } else {
      printf("-> OK, fichier \"%s\" ouvert pour lecture.",
             nom_fichier);
    }
  } while ((f == NULL) && (nb < NB_DEMANDES));

  /* la valeur de retour est le résultat du test entre (): 0 ou 1 */
  return f;
}
```

```c
/* ========================================================================
 * Fonction initialiser_statistique
 * ------------------------------------------------------------------------
 * In:   Une Statistique à initialiser.
 * What: Initialiser tous les éléments d'une Statistique à zéro.
 * ======================================================================== */
void initialise_statistique(Statistique stat)
{
  int i;
  for (i = 0; i < TAILLE; ++i) {
    stat[i] = 0;
  }
}

/* ========================================================================
 * Fonction collecte_statistique
 * ------------------------------------------------------------------------
 * In:   Une Statistique à remplir et le fichier à lire.
 * Out:  Le nombre d'éléments comptés dans la Statistique.
 * What: Lit tous les caractères dans le fichier et compte dans la Statistique
 *       combien de fois chaque caractère apparait dans le fichier.
 * ======================================================================== */
unsigned long int collecte_statistique(Statistique stat, FILE* f)
{
  int c;                    /* le caractère lu             */
  unsigned long int nb = 0; /* le nombre d'éléments comptés */

  while ((c = getc(f)) != EOF) {
    /* est-ce que le caractère lu est dans l'intervalle qu'on étudie ? */
    if (( ((unsigned char) c) >= start) &&
        ( ((unsigned char) c) <= stop ) ) {
      ++(stat[c - start]); /* on incrément la statistique pour ce caractère */
      ++nb; /* on incrémente le nombre total d'éléments comptés */
    }
  }

  return nb;
}

/* ========================================================================
 * Fonction affiche
 * ------------------------------------------------------------------------
 * In:   La Statistique à afficher, le nombre par rapport auquel on affiche
 *       les pourcentages (si 0 recalcule ce nombre comme la somme des
 *       éléments) et la taille du tableau.
 * What: Affiche tous les éléments d'une Statistique (valeurs absolue et
 *       relative).
 * ======================================================================== */
void affiche(Statistique stat, unsigned long int nb, unsigned short int taille)
{
  unsigned short int i;

  if (nb == 0) { /* on doit calculer la somme si nb == 0 */
    for (i = 0; i < taille; ++i)
      nb += stat[i];
  }

  printf("STATISTIQUES :\n");
  for (i = 0; i < taille; ++i) {
    /* on n'affiche que les résultats pour des statistiques supérieures à 0 */
    if (stat[i] != 0) {
      printf("%c : %10lu - %6.2f%%\n", (char) (i+start), stat[i],
             100.0 * stat[i] / (double) nb);
```

```
    }
  }
}
```