Université Mohammed V
Ecole Nationale Supérieure de l'Informatique et de l'Analyse des Systèmes
Département Génie Logiciel

Correction du TP1 – Programmation en langage C

Travaux pratiques

Langage C (M1.1.2)
Semestre 1 – 2026–2027
Correction

1 Correction du TP1

Types, controle, boucles et suites

Pr. Youness BOUTYOUR youness.boutyour@um5.ac.ma

Version du 27 septembre 2026

Table des matières

Méthode de correction **2**

Exercice 1 – Rectangle 2
Exercice 2 – Conditions et équation du second degré 2
Exercice 3 – Boucles, maximum et nombres premiers 3
Exercice 4 – Nombres parfaits 4
Exercice 5 – Nombres amicaux 5
Exercice 6 – Suite U_n 5
Exercice 7 – Suite T_n 6
Exercice 8 – Factorielle et exponentielle 7
Exercice 9 – Approximation de π 8

Méthode de correction

Chaque programme est indépendant et vérifie les données essentielles avant de les utiliser. Compiler avec`gcc -std=c11 -Wall -Wextra -Wpedantic fichier.c -lm` lorsque la bibliothèque mathématique est nécessaire. Les solutions privilégient des fonctions courtes, des bornes explicites et des tests sur les cas limites.

Exercice 1 – Rectangle

Calcul du périmètre et de la surface à partir de deux longueurs positives.

Correction complète

```c
#include <stdio.h>

int main(void) {
    double longueur, largeur;

    if (scanf("%lf %lf", &longueur, &largeur) != 2 || longueur < 0 ||
        largeur < 0) {
        return 1;
    }
    printf("perimetre = %.2f\n", 2.0 * (longueur + largeur));
    printf("surface = %.2f\n", longueur * largeur);
    return 0;
}
```

Exercice 2 – Conditions et équation du second degré

Les quatre questions sont regroupées : comparaison avec `a/A`, reconnaissance de la casse, conversion d'un jour et résolution robuste de $ax^2 + bx + c = 0$, y compris les cas dégénérés.

Correction complète

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

static void equation(void) {
    double a, b, c, delta;
    if (scanf("%lf %lf %lf", &a, &b, &c) != 3) return;
    if (a == 0.0) {
```

```c
        if (b == 0.0) puts(c == 0.0 ? "une infinite de solutions" : "aucune solution");
        else printf("x = %.6f\n", -c / b);
        return;
    }
    delta = b * b - 4.0 * a * c;
    if (delta < 0.0) puts("aucune solution reelle");
    else if (delta == 0.0) printf("x = %.6f\n", -b / (2.0 * a));
    else printf("x1 = %.6f, x2 = %.6f\n", (-b - sqrt(delta)) / (2.0 * a), (-b + sqrt(delta)) / (2.0 * a));
}

int main(void) {
    char lettre;
    int jour;

    if (scanf(" %c", &lettre) != 1) return 1;
    printf("devinette : %s\n", lettre == 'a' || lettre == 'A' ? "correct" : "incorrect");
    printf("casse : %s\n", lettre >= 'A' && lettre <= 'Z' ? "majuscule" :
           (lettre >= 'a' && lettre <= 'z' ? "minuscule" : "autre"));
    if (scanf("%d", &jour) != 1) return 1;
    if (jour >= 1 && jour <= 7) {
        const char *jours[] = {"lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche"};
        puts(jours[jour - 1]);
    } else puts("jour invalide");
    equation();
    return 0;
}
```

Exercice 3 – Boucles, maximum et nombres premiers

Lecture jusqu'à l'étoile, comptage des lettres `b` et des espaces, recherche du maximum avec sa position, puis test de primalité jusqu'à la racine carrée.

Correction complète

```c
#include <stdio.h>

static int est_premier(int n) {
    if (n < 2) return 0;
    for (int d = 2; d <= n / d; ++d) if (n % d == 0) return 0;
    return 1;
}
```

```c

int main(void) {
    char c;
    int compte_b = 0, espaces = 0;
    int valeurs[5], maximum, position = 0;
    int n;

    while (scanf("%c", &c) == 1 && c != '*') {
        putchar(c);
        if (c == 'b') ++compte_b;
        if (c == ' ') ++espaces;
    }
    printf("\nb = %d, espaces = %d\n", compte_b, espaces);
    for (int i = 0; i < 5; ++i) scanf("%d", &valeurs[i]);
    maximum = valeurs[0];
    for (int i = 1; i < 5; ++i) if (valeurs[i] > maximum) maximum = valeurs[
        position = i];
    printf("maximum = %d, position = %d\n", maximum, position + 1);
    if (scanf("%d", &n) == 1) printf("%d est %s\n", n, est_premier(n) ? "
        premier" : "compose");
    return 0;
}
```

Exercice 4 – Nombres parfaits

La somme des diviseurs propres est calculée sans tester tous les diviseurs deux fois ; elle sert ensuite à tester N et à énumérer les nombres parfaits précédents.

Correction complète

```c
#include <stdio.h>

static int somme_diviseurs_propres(int n) {
    int somme = n > 1 ? 1 : 0;
    for (int d = 2; d <= n / d; ++d) {
        if (n % d == 0) {
            somme += d;
            if (d != n / d) somme += n / d;
        }
    }
    return somme;
}

int main(void) {
    int n;
```

```c
    if (scanf("%d", &n) != 1 || n < 1) return 1;
    printf("%d est %s\n", n, somme_diviseurs_propres(n) == n ? "parfait" : "non parfait");
    for (int x = 2; x < n; ++x) if (somme_diviseurs_propres(x) == x) printf("%d ", x);
    putchar('\n');
    return 0;
}
```

Exercice 5 – Nombres amicaux

Deux entiers sont comparés par les sommes croisées de leurs diviseurs propres.

Correction complète

```c
#include <stdio.h>

static int somme_diviseurs_propres(int n) {
    int somme = n > 1 ? 1 : 0;
    for (int d = 2; d <= n / d; ++d) if (n % d == 0) {
        somme += d;
        if (d != n / d) somme += n / d;
    }
    return somme;
}

int main(void) {
    int n, m;
    if (scanf("%d %d", &n, &m) != 2 || n <= 0 || m <= 0) return 1;
    puts(somme_diviseurs_propres(n) == m && somme_diviseurs_propres(m) == n ? "amicaux" : "non amicaux");
    return 0;
}
```

Exercice 6 – Suite U_n

La récurrence est évaluée itérativement avec seulement les deux termes précédents, puis réutilisée pour afficher les cent premiers termes.

Correction complète

```c
#include <stdio.h>

static long long terme(unsigned n) {
    long long u0 = 4, u1 = 4;
    if (n < 2) return 4;
    for (unsigned i = 2; i <= n; ++i) {
        long long suivant = 3 * u1 + 4 * u0 + 6;
        u0 = u1;
        u1 = suivant;
    }
    return u1;
}

int main(void) {
    unsigned n;
    if (scanf("%u", &n) != 1) return 1;
    printf("U_%u = %lld\n", n, terme(n));
    for (unsigned i = 0; i < 100; ++i) printf("U_%u = %lld\n", i, terme(i));
    return 0;
}
```

Exercice 7 – Suite T_n

La relation de récurrence est calculée de manière itérative pour éviter une explosion du nombre d'appels récursifs.

Correction complète

```c
#include <stdio.h>

static double terme(unsigned n, double x) {
    double t0 = 1.0, t1 = x;
    if (n == 0) return t0;
    if (n == 1) return t1;
    for (unsigned k = 2; k <= n; ++k) {
        double suivant = 2.0 * x * t1 - t0;
        t0 = t1;
        t1 = suivant;
    }
    return t1;
}

int main(void) {
```

```c
    unsigned n;
    double x;
    if (scanf("%u %lf", &n, &x) != 2 || x < -1.0 || x > 1.0) return 1;
    printf("T_%u(%.6f) = %.12f\n", n, x, terme(n, x));
    return 0;
}
```

Exercice 8 – Factorielle et exponentielle

La factorielle, la somme partielle de e^x et l'arrêt à une précision de 10^{-6} sont regroupés dans des fonctions distinctes.

Correction complète

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

static double factorielle(unsigned k) {
    double resultat = 1.0;
    for (unsigned i = 2; i <= k; ++i) resultat *= i;
    return resultat;
}

static double serie_exp(double x, unsigned n) {
    double terme = 1.0, somme = 1.0;
    for (unsigned k = 1; k <= n; ++k) {
        terme *= x / k;
        somme += terme;
    }
    return somme;
}

int main(void) {
    unsigned k, n;
    double x;
    if (scanf("%u %lf %u", &k, &x, &n) != 3 || x <= 0.0) return 1;
    printf("K! = %.0f\n", factorielle(k));
    printf("somme = %.12f\n", serie_exp(x, n));
    {
        unsigned i = 0;
        double approximation = 1.0, terme = 1.0;
        do {
            ++i;
            terme *= x / i;
            approximation += terme;
```

```
        } while (fabs(terme) > 1e-6 * fabs(approximation));
        printf("exp(%.6f) = %.12f (%u termes, erreur %.3e)\n", x,
            approximation, i, fabs(approximation - exp(x)));
    }
    return 0;
}
```

Exercice 9 – Approximation de π

On utilise $\pi = \sqrt{6 \sum_{k \geq 1} 1/k^2}$ et on arrête la somme lorsque le terme ajouté devient négligeable.

Correction complète

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

int main(void) {
    double somme = 0.0, terme;
    unsigned k = 0;
    do {
        ++k;
        terme = 1.0 / ((double) k * k);
        somme += terme;
    } while (terme > 1e-8);
    printf("pi = %.12f\n", sqrt(6.0 * somme));
    printf("termes = %u\n", k);
    return 0;
}
```