Programmation en C et structures de données

guillaume.revy@univ-perp.fr

CT, Lundi 13/01/2025 de 14h à 16h

Aucun document n'est autorisé.
Les calculatrices, ordinateurs, et téléphones portables sont interdits.

Exercice 1. Importance de l'indentation, malgré tout

$0.75 + 1.25 = 2$ *pts*

Contrairement à certains autres langages, l'indendation d'un programme C n'a pas d'impact sur le résultat de son exécution. Mais elle a une importance pour sa lisibilité.

► **1.** Que fait le programme ci-dessous ?

```c
int main(void){for(int i = 0; i < 4; i ++){for(int j = 0; j < i+1; j++) printf("%d ", j);
    printf("\n");} return 0;}
```

► **2.** Écrire un programme qui lit deux valeurs entières a et b au clavier, puis qui indique si la différence $a - b$ est positive, négative, ou nulle (en distinguant les 3 cas), sans la calculer explicitement.

Exercice 2. Simulation d'une loterie

$1.5 + 1 + 0.5 + 1 = 4$ *pts*

À l'occasion d'un jeu de loterie, un joueur mise une certaine somme d'argent et fait tourner une première fois une roue circulaire découpée en 8 secteurs de même angle numérotés de 1 à 8. Les quatre premiers secteurs sont rouges. S'il tombe sur le secteur 1, il récupère le double de sa mise, sinon (secteur 2 à 8) il relance la roue et gagne sa mise augmentée de 6 euros s'il tombe sur un secteur de couleur rouge. Dans les autres cas, sa mise est perdue.

► **1.** Écrire une fonction `jeu` qui simule une partie de loterie, et qui, étant donnée une mise, calcule et renvoie le gain, en utilisant la fonction `rand`.

► **2.** Écrire une fonction `gain_moyen`, qui, étant donnés une mise et un entier $n > 0$, calcule et retourne le gain moyen sur n parties en utilisant une boucle **`while`**.

► **3.** Modifier cette fonction pour exécuter le même traitement en utilisant une boucle **`for`**.

► **4.** Écrire un programme qui lit une valeur entière n au clavier et une mise, et qui affiche l'évolution du gain moyen sur au plus n parties.

Exercice 3. Intégration numérique

$1 + 2 + 1 = 4$ *pts*

Soit $f(x)$ une fonction continue définit sur les réels, et prenant ses valeurs dans un intervalle $I = [a, b]$. L'intégrale de la fonction $f(x)$ sur $[a, b]$, notée

$$A = \int_a^b f(x)\,\mathrm{d}x,$$

peut être interprétée comme l'aire du domaine délimité par la courbe de $f(x)$, l'axe des abscisses, et les droites verticales d'abscisses a et b. Parmi les méthodes d'intégration numérique, la méthode des rectangles consiste à découper l'intervalle $[a, b]$ en n sous intervalles, et à considérer l'intégrale A comme la somme des n rectangles, comme illustré en Figure 1 pour $f(x) = -x^2 + 4$ sur $[-2, 2]$.

► **1.** Écrire une fonction `f` qui calcule et retourne, pour un réel x donné, la valeur de $f(x) \mapsto -x^2 + 4$.

► **2.** Écrire une fonction `rectangle` qui calcule par la méthode des rectangles et retourne une approximation de l'intégrale de la fonction $f(x)$ sur un intervalle $[a, b]$ donné en paramètre. *Une attention particulière devra être apportée autour des racines de la fonction.*

► **3.** Modifier cette fonction pour calculer l'intégrale d'une fonction passée en paramètre en utilisant un pointeur de fonction.

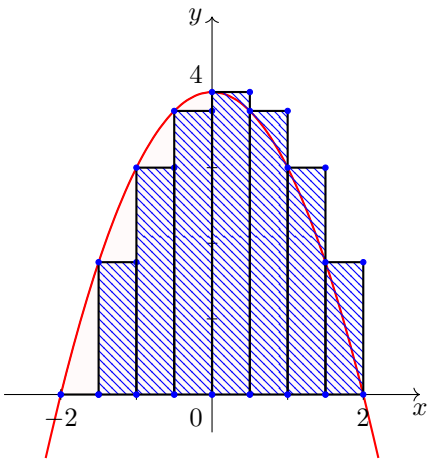

FIGURE 1 – Méthode d'intégration numérique : méthode des rectangles.

Exercice 4. Zoo Zoo Land sur un tableau

$1 + 1 + 1 = 3$ *pts*

Soit T un tableau 1D d'entiers de taille n, initialisé avec des valeurs nulles. Nous allons le remplir avec des valeurs aléatoires comprises entre 1 et 9 inclus, en partant de la case d'indice $i = 0$ de la manière suivante. Nous tirons n valeurs aléatoires. Pour chaque tirage :

— si $i = 0$ ou si la valeur est strictement supérieure à celle de la case d'indice $i - 1$, nous l'ajoutons dans la case i et nous incrémentons i,
— sinon, nous remplaçons la valeur de la case $i - 1$ par cette nouvelle valeur.

Le programme devra déterminer le plus grand indice i de la case dans laquelle une valeur a été placée.

▶ **1.** Écrire une fonction `zoo_zoo_land` qui, étant donné un tableau taille $n = 20$, construit le tableau T comme décrit ci-dessus, et détermine et retourne le plus grand indice i calculé.

▶ **2.** Illustrer son utilisation sur dans un programme principal.

▶ **3.** Donner le prototype de la fonction qui permet de faire le même traitement sur un tableau de taille n.

Exercice 5. Tableaux et préfixe

2 *pts*

Soient deux tableaux d'entiers T_1 et T_2 de taille n_1 et n_2, respectivement. Le tableau T_1 est un préfixe de T_2 si et seulement si :

— $n_2 \geq n_1$
— et $\forall\, i \in [0, n_1], T_1[i] = T_2[i]$.

▶ **1.** Écrire une fonction `prefixe` qui, étant donnés deux tableaux T_1 et T_2, détermine si le premier est un préfixe du second. L'utilisation de cette fonction sera la suivante.

```
if(prefixe(T1, n1, T2, n2))
  printf("Le tableau T1 est un prefixe du tableau T2.\n");
else
  printf("Le tableau T1 n'est pas un prefixe du tableau T2.\n");
```

Exercice 6. À l'aide !

$1 + 1 + 1 = 3$ *pts*

Un étudiant vous envoie le mail suivant : "*Je ne comprends pas, mon programme C compile sans erreur avec gcc-12, mais il plante (Segmentation fault : 11). Il est censé allouer et initialiser un tableau de taille 10 dans la fonction foo, puis de l'afficher dans la programme principal. Mais je vous assure, il n'y a pas d'erreur !*".

Pour l'aider, vous lui demandez de vous envoyer son programme (ci-dessous).

```
int* foo(int n) {
  int *T = (int*)malloc(n*sizeof(int));
  for(int i = 0; i < n; i++)
    T[i] = i;
  int tmp = T[0];
  return &tmp;
}
```

```
int main(void) {
  int *R = foo(10);
  for(int i = 0; i < 10; i++)
    printf("%d ", R[i]);
  printf("\n");
  free(R);
  return 0;
}
```

► **1.** Proposer une explication, en vous aidant d'un dessin.

► **2.** Corriger le programme, sans modifier le prototype des fonctions.

► **3.** Réécrire la fonction `foo` de telle sorte que son retour soit de type **`void`** et expliquer précisément son fonctionnement en vous aidant d'un dessin.

Exercice 7. Suite de Conway et liste chainée $1 + 1 + 2 = 4$ *pts*

La suite de Conway est une suite qui commence avec un terme $C_0 = 1$ et où un terme C_i se détermine en comptant combien de fois chaque chiffre apparaît dans le terme précédent C_{i-1}, comme ci-dessous :

— $C_1 = 11$ car dans C_0, il y a 1 chiffre 1,
— $C_2 = 21$ car dans C_1, il y a 2 chiffres 1,
— $C_3 = 1211$ car dans C_2, il y a 1 chiffre 2 et 1 chiffre 1 ...

L'idée de l'exercice est de déterminer les différents termes de la suite de Conway et de les stocker dans un tableau de listes chainées.

► **1.** Dans ce cas, quel est l'intérêt d'utiliser une liste chainée par rapport à un tableau ?

Pour implanter la liste chainée, nous utilisons la structure suivante.

```
struct list {
  int i;
  struct list *succ;
};
```

► **2.** Écrire une fonction `ajouter` qui, étant donnés une liste et un entier n, ajoute l'élément n en queue de la liste, et une fontion `detruire` qui libère la mémoire utilisée par une liste passée en paramètre.

► **3.** Écrire une fonction `conway` qui prend en paramètre un entier n, et qui construit un tableau contenant les $n + 1$ premiers termes de la suite de Conway, tous représentés par une liste chainée. (Le premier élément du tableau contiendra l'élément la liste chainée de un seul élément représentant C_0, et l'élément d'indice i, la liste représentant C_i.)