Année 2024-2025 — 1ère session – 17 Décembre 2024

PG109 – Programmation Impérative

1ère année Télécom

Responsable
Philippe Swartvagher

Intervenants
Joachim Bruneau-Queyreix
Kylian Desier
Guillaume Mercier
Augusta Mukam

Durée de l'examen : 1 h

Documents autorisés	☐	sans document	☒
Calculatrice autorisée	☐	non autorisée	☒

Correction

Nom et Prénom: _____

- **N'oubliez pas d'écrire votre nom ci-dessus !**
- **Il est impératif de répondre dans les espaces prévus à cet effet : tout ce qui dépasse ne sera pas pris en compte lors de la correction.**
- **Dans les cadres où du code est attendu, il est impératif d'écrire en respectant les numéros de ligne.**
- **L'orthographe, la grammaire, la conjugaison et la syntaxe seront pris en compte lors de la correction.**
- **Écrivez le code comme si vous écriviez dans un fichier source : la syntaxe, le style (indentation, noms de variables cohérents, ...) seront pris en compte lors de la correction. Il va sans dire que le code que vous écrivez doit compiler.**
- **N'oubliez pas d'écrire votre nom ci-dessus !**

Exercice 1 – QCM

Chaque question possède une et une seule bonne réponse.

Barème :

- 1 point par bonne réponse
- **-0,5 point par mauvaise réponse**
- 0 point si absence de réponse

On considérera que chaque extrait de code présenté est correctement intégré dans une fonction `main()` et que tous les `#include` nécessaires sont présents.

1. Comment peut-on connaître le nombre d'éléments dans n'importe quel tableau ?

   ☐ Avec l'opérateur `sizeof`

   ☐ Avec la fonction `len()`

   ☐ Avec la fonction `strlen()`

   ■ On ne peut pas

   ☐ On parcourt le tableau jusqu'à atteindre une valeur particulière qui indique la fin du tableau

   > Il n'est pas possible de connaître la taille d'un tableau en ne possédant que l'adresse du début du tableau. C'est pourquoi on a besoin de stocker le nombre d'éléments dans un tableau séparément, dans une variable dédiée, par exemple.

2. Que va afficher le code suivant ?

```c
int t[4] = {3, 1, 4, 2};
int* u = t + 1;
int* v = &t[3];
u[1] = *v;
*v = 4;
printf("%d\n", t[2]);
```

   ☐ 3

   ☐ 1

   ☐ Une autre valeur

   ☐ 4

   ■ 2

   > `u` contient l'adresse de `t[1]`.
   > `v` contient l'adresse de `t[3]`.
   > `u[1]` correspond donc à `t[2]`, et on y met la valeur à l'adresse contenue dans `v`, donc la valeur dans `t[3]`, donc 2.
   > On met à l'adresse contenue dans `v`, soit dans `t[3]`, la valeur 4.
   > Finalement, on demande à afficher le contenu de `t[2]`, qui contient maintenant 2.

3. Que va afficher le code suivant ?

```c
char s[10] = {'H', 'e', 'l', 'l', 'o', '\0' };
printf("%ld\n", strlen(s));
```

   ☐ 10

   ■ 5

   ☐ 6

   ☐ On ne peut pas utiliser `strlen()` sur `s`.

   > La fonction `strlen()` compte le nombre de caractères jusqu'à rencontrer le caractère de fin de chaînes de caractères `'\0'`, en l'excluant du décompte.

4. Que va afficher le code suivant ?

```
#define VALUE 3 + 4

printf("%d\n", 2 * VALUE);
```

- ■ 10
- ☐ 14
- ☐ Rien, une erreur survient à la compilation
- ☐ Le double de la valeur à l'adresse `0x7`

> Le préprocesseur va réaliser une substitution textuelle de `VALUE` : l'appel à `printf()` devient donc `printf("%d\n", 2 * 3 + 4)`. Avec les règles de priorité des opérateurs, `2 * 3 + 4` est équivalent à `(2 * 3)+ 4` et donc 10 est affiché.

Exercice 2 – Questions de cours

1. Quelle condition faut-il utiliser dans le `if` pour tester l'égalité des variables `a` et `b` ?

```
float a, b;

// ... des opérations sur a et b ...

if (???)
{
  printf("a et b ont la même valeur : %f = %f\n", a, b);
}
```

> Pour tester l'égalité de deux flottants, on teste que leur différence est inférieure à un certain seuil : `if (fabs(b - a)<1e-10)`.

2. Quel le problème avec le code suivant ?

```
int* une_fonction(void)
{
  int une_variable = 12;

  return &une_variable;
}
```

> Cette fonction renvoie l'adresse d'une variable locale à la fonction. On n'a donc aucune garantie que cette zone mémoire va rester réservée pour cette variable, et le contenu de cette zone mémoire risque d'être écrasé par l'exécution d'une autre fonction.

3. On souhaite définir un fichier d'en-tête nommé `interface.h`.
   (a) Quel code est-il recommandé de mettre au début et à la fin de ce fichier d'en-tête ?

```
#ifndef __INTERFACE_H
#define __INTERFACE_H

/* ... contenu du fichier interface.h ... */

#endif // !__INTERFACE_H
```

(b) Pourquoi ?

> Pour éviter les doubles inclusions et que les définitions dans `interface.h` apparaissent plusieurs fois, ce qui poserait une erreur de compilation.

Exercice 3 – Extraction de nombres

Inspiré de https://adventofcode.com/2024/day/1

On souhaite écrire une fonction qui extrait d'une chaîne de caractères deux nombres séparés par une espace. Par exemple, le code suivant :

```
char s[] = "254 18";
int x1 = 0;
int x2 = 0;
extract_int(s, &x1, &x2);
printf("%d\n%d\n", x1, x2);
```

affichera :

```
254
18
```

Écrivez le code de la fonction `extract_int()`. On fera les hypothèses suivantes :

- la chaîne de caractères passée en paramètre est toujours formattée comme attendue (un nombre, suivi d'une espace, suivi d'un nombre) ;
- les deux nombres à extraire peuvent avoir n'importe quel nombre de chiffres ;
- il est possible de modifier la chaîne de caractères passée en paramètre ;
- les pointeurs passés en paramètre sont toujours différents de `NULL`.

```
void extract_int(char* s, int* x1, int* x2)
{
    int space_pos = 0;
    while (s[space_pos] != ' ')
    {
        space_pos++;
    }
    s[space_pos] = '\0';
    *x1 = atoi(s);
    *x2 = atoi(s+space_pos+1);
}
```

Exercice 4 – Vérification du tri d'un tableau

On souhaite écrire un programme qui nous indique si les n nombres entiers passés comme arguments du programme sont triés dans un ordre croissant ou pas. Par exemple :

```
./tab_is_sorted 1 2 3 4 5 6
Les nombres sont dans un ordre croissant
./tab_is_sorted 1 2 3 5 4 6
Les nombres ne sont pas dans un ordre croissant
```

1. Écrire une fonction `is_sorted()` qui prend en paramètre un tableau de nombres entiers (et peut-être d'autre(s) paramètre(s)) et renvoie `1` si les valeurs de ce tableau sont triées dans l'ordre croissant, et `0` si ce n'est pas le cas.

```c
int is_sorted(int* tab, int n)
{
    for (int i = 0; i < n-1; i++)
    {
        if (tab[i] > tab[i+1])
        {
            return 0;
        }
    }
    return 1;
}
```

2. Écrire **tout** le code nécessaire pour avoir un tel programme : instructions de préprocesseur, fonction `main()`, *etc.*
   La fonction `main()` récupère les arguments passés au programme, appelle la fonction `is_sorted()` et affiche le message indiquant le résultat. La définition de la fonction `is_sorted()` sera omise, seul **son emplacement dans le code source sera signifié par un commentaire**. On considérera que le programme est toujours bien utilisé (nombre et type d'arguments, ...) et que les fonctions réussissent toujours, ce n'est donc pas la peine de vérifier le nombre d'arguments du programme ou les valeurs de retour des fonctions.

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

// is_sorted() définie ici

int main(int argc, char* argv[])
{
    const int n = argc - 1;

    int* tab = malloc(n*sizeof(int));

    for (int i = 0; i < n; i++)
    {
        tab[i] = atoi(argv[i+1]);
    }

    printf("Les nombres %s dans un ordre croissant\n", is_sorted(tab, n) ?
    ↪  "sont" : "ne sont pas");

    free(tab);

    return EXIT_SUCCESS;
}
```

Exercice 5 – Déplacement de robots

Inspiré de https://adventofcode.com/2024/day/6

Dans cet exercice, il est question d'implémenter une interface pour gérer le déplacement de robots sur une grille 2D. Pour savoir dans quelle case de la grille est situé un robot, on utilise un couple de coordonnées (x, y), x représentant l'axe des abscisses et y représentant l'axe des ordonnées. La figure 1 donne un exemple d'une telle grille.
L'interface utilisera les types suivants :

```
struct coord_s {
    int x;
    int y;
};

enum direction_e {
    HAUT, BAS, // Pour se déplacer sur l'axe des ordonnées
    GAUCHE, DROITE, // Pour se déplacer sur l'axe des abscisses
};

struct robot_s {
    struct coord_s coord; // Où est situé le robot
    enum direction_e direction; // Dans quelle direction le robot va avancer
};
```

1. Écrire la fonction qui initialise un robot situé aux coordonnées passées en paramètre. À l'initialisation, un robot est toujours dirigé vers le haut. On considérera que les allocations de mémoire dynamiques réussissent toujours : ce n'est donc pas nécessaire de gérer les cas où une erreur surviendrait.

```
struct robot_s* init_robot(struct coord_s init_coord)
{
    struct robot_s* robot = malloc(sizeof(struct robot_s));
    robot->coord = init_coord;
    robot->direction = HAUT;

    return robot;
}
```

2. Écrire la fonction qui permet de libérer les ressources allouées par la fonction `init_robot()`.

```
void free_robot(struct robot_s* robot)
{
    free(robot);
}
```

3. Écrire la fonction qui permet de changer la direction d'un robot.

```
void change_direction(struct robot_s* robot, enum direction_e
    ↪ new_direction)
{
    robot->direction = new_direction;
}
```

4. Écrire la fonction qui fait avancer un robot d'une case dans la direction où il est orienté. On considérera que le robot peut toujours avancer dans sa direction.

```c
void move(struct robot_s* robot)
{
    switch (robot->direction){
        case HAUT:
            robot->coord.y++;
            break;
        case BAS:
            robot->coord.y--;
            break;
        case GAUCHE:
            robot->coord.x--;
            break;
        case DROITE:
            robot->coord.x++;
            break;
    }
}
```

5. Écrire la fonction qui à partir d'un tableau de robots, indique si la case dont les coordonnées sont passées en paramètre est libre (`return 1`) ou non (`return 0`, parce déjà occupée par un autre robot).

```c
int is_free_position(struct coord_s coord, struct robot_s* robots[], int
    ↪ n_robots)
{
    for (int i = 0; i < n_robots; i++)
    {
        if (robots[i]->coord.x == coord.x && robots[i]->coord.y == coord.y
            ↪ )
        {
            return 0;
        }
    }

    return 1;
}
```

6. (Bonus) Écrire la fonction qui permet de déplacer le robot pour qu'il atteigne la case de cordonnées passées en paramètre. La trajectoire du robot fera au maximum un angle droit (*cf* figure 1) : on déplace le robot d'abord sur l'axe des abscisses, puis sur l'axe des ordonnées. On ne cherche pas à minimiser la distance parcourue. Cette fonction fera appel aux fonctions `change_direction()` et `move()`.

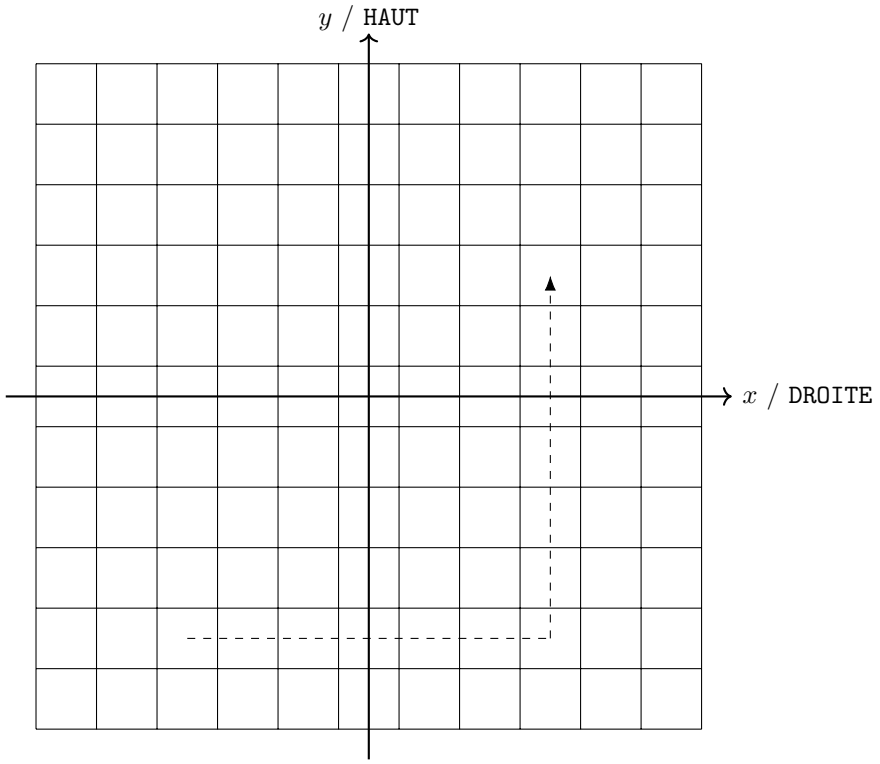

Figure 1: Trajectoire à suivre si le robot initialement positionné en $(-3, -4)$ doit aller à la case $(3, 2)$.

```c
void reach_position(struct robot_s* robot, struct coord_s target_coord)
{
    // X axis:
    if (target_coord.x < robot->coord.x)
    {
        change_direction(robot, GAUCHE);
    }
    else
    {
        change_direction(robot, DROITE);
    }

    while (robot->coord.x != target_coord.x)
    {
        move(robot);
    }

    // Y axis:
    if (target_coord.y < robot->coord.y)
    {
        change_direction(robot, BAS);
    }
    else
    {
        change_direction(robot, HAUT);
    }

    while (robot->coord.y != target_coord.y)
    {
        move(robot);
    }
}
```