Année 2023-2024 — 1ère session – 18 Janvier 2024

PG109 – Programmation Impérative

1ère année Télécom

Joachim Bruneau-Queyreix
Kylian Desier
Guillaume Mercier
Augusta Mukam
Philippe Swartvagher

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.**
- **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. Qu'affiche le code suivant ?

```c
int a;
printf("%d\n", a);
```

- ☐ Rien, parce que la compilation échoue
- ☐ Rien, parce que l'exécution échoue
- ☐ Rien.
- ☐ `0`
- ☐ `1`
- ■ Il n'est pas possible de prédire la valeur qui sera affichée

> Lorsqu'une variable est déclarée sans qu'une valeur lui soit directement affectée, cette variable interprète le contenu de la zone mémoire qui lui est attribuée comme sa valeur. Or le contenu de cette zone mémoire n'est pas prédictible (il peut s'agir de la mémoire utilisée par un programme précédent, par exemple). Il n'est donc pas possible de prédire la valeur d'une variable non initialisée.

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

- ☐ Avec la fonction `strlen()`
- ■ On ne peut pas
- ☐ Avec l'opérateur `sizeof`
- ☐ On parcourt le tableau jusqu'à atteindre une valeur particulière qui indique la fin du tableau
- ☐ Avec la fonction `len()`

> 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.

3. Qu'affiche le code suivant ?

```c
int t[4] = {100, 200, 300, 400};
int* u = t;
int* v = u + 2;
int* z = &u[1];
v[1] = 500;
printf("%d\n", z[1]);
```

- ☐ `200`
- ■ `300`
- ☐ `400`
- ☐ `500`
- ☐ Une autre valeur

`u` contient l'adresse de `t[0]`.
`v` contient l'adresse de `u[2]`, donc l'adresse de `t[2]`.
`z` contient l'adresse de `u[1]`, donc l'adresse de `t[1]`.
`v[1]`, soit `t[3]`, se voit attribuer la valeur `500`, qui remplace donc la valeur `400`.
On affiche `z[1]`, soit `t[1+1]`, donc `t[2]`, donc `300`.

4. Quelle ligne du code suivant pose problème ?

```
1 char t[] = "Bonjour\n" ;
2 char* s = "Salut!";
3 t[1] = 'Z';
4 s[1] = 'Z';
5 s = t;
6 s[1] = 'A';
```

- ☐ Aucune
- ☐ Ligne 3
- ■ Ligne 4
- ☐ Ligne 5
- ☐ Ligne 6

`t` est un tableau déclaré sur la pile qui contient les valeurs pour former la chaîne de caractères `"Bonjour\n"`. `s` contient l'adresse d'une chaîne de caractères constante, qu'on ne peut donc pas modifier. Ainsi, `s[1] = 'Z';` est impossible. La dernière ligne est possible, car `s` contient alors l'adresse de `t`, qu'il est possible de modifier.

5. Qu'affiche le code suivant ?

```
float x = 'b';
printf("%f\n", x);
```

- ☐ `b`
- ☐ Rien, une erreur se produit à la compilation ou à l'exécution
- ☐ `b.000000`
- ■ `98.000000`

La valeur ASCII du caractère `'b'` est `98`, qui est converti en flottant lors de l'affectation à une variable de type `float`.

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. Soient `a` et `b` deux variables de type `int`.

   (a) Quelle est la différence entre `if (a & b)` et `if (a && b)` ?

   > `a & b` réalise un ET **binaire** entre les valeurs de `a` et `b` et évalue le résultat comme condition. `a && b` réalise un ET **logique** entre les deux valeurs.

   (b) Donner un exemple de valeurs pour `a` et `b` qui illustre cette différence.

   > Lorsque `a=1` et `b=2`, `a & b` vaut 0 donc est évalué à `faux` ; alors que `a && b` est évalué à `vrai` (`a` et `b` s'évaluent chacun à `vrai`).

3. Avec le code suivant, pourquoi peut-on définir le type `struct foo_s`, mais pas `struct bar_s` ?

```
struct foo_s { // Structure valide
    struct foo_s* s;
    // ...
};

struct bar_s { // Structure invalide
    struct bar_s s;
    // ...
};
```

> Lorsqu'on définit une structure, le compilateur a besoin de connaître la taille des champs qui la composent. La taille de `struct foo_s*` est connue, c'est la taille d'un pointeur (un pointeur a toujours la même taille, quelque soit le type vers lequel il pointe), donc la définition de `struct foo_s` est valide. La taille de `struct bar_s` comme membre de la structure `struct bar_s` n'est pas connue, puisqu'on est en train de définir cette structure !

4. (a) Pourquoi le code suivant générera une erreur de segmentation ?

```
char* s1 = "Carottes râpées";
char* s2;
strcpy(s2, s1);
```

   > `strcpy` copie la chaîne de caractères qui commence à l'adresse contenue dans `s1` vers la zone mémoire dont l'adresse de début est stockée dans `s2`. Ici `s2` est définie mais pas initialisée, donc cette variable peut contenir n'importe quoi comme adresse et il y a donc fort à parier qu'on n'ait pas le droit d'écrire à cette adresse.

   (b) Que faudrait-il faire pour corriger cette erreur ?

   > Il faut que `s2` contienne l'adresse d'une zone mémoire où on peut écrire et qui soit suffisamment grande pour contenir la chaîne de caractères `s1`, par exemple `char s2[100];` ou `char* s1 = malloc(sizeof(char)*(strlen(s1)+1));`.

Exercice 3 – Écrivons un peu de code

On souhaite écrire un programme qui affiche le maximum des nombres entiers qui lui sont passés en paramètres. Par exemple :

```
./prog 3 7 8 2 1
8
```

1. Écrire une fonction `max_tab()` qui prend en paramètre un tableau de chaînes de caractères représentant des nombres (et peut-être d'autre(s) paramètre(s)) et renvoie le plus grand nombre dans le tableau, en utilisant le type `int`. On considérera que le tableau possède toujours au moins un élément et que les chaînes de caractères du tableau représentent toujours des nombres entiers.

```
int max_tab(char** tab, int n)
{
    int m = atoi(tab[0]);
    for (int i = 1; i < n; i++)
    {
        int nb = atoi(tab[i]);
        if (nb > m)
        {
            m = nb;
        }
    }
    return m;
}
```

2. Écrire le reste du code nécessaire pour avoir un tel programme : instructions de préprocesseur, fonction `main()`, *etc.* Seule la fonction `max_tab()` sera omise, son emplacement dans le code source sera signifié par un commentaire. On considérera que le programme est toujours bien utilisé (nombre de paramètres, ...), ce n'est donc pas la peine de vérifier le nombre de paramètres du programme.

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

// fonction max_tab ...

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

    int m = max_tab(argv+1, n);

    printf("%d\n", m);

    return EXIT_SUCCESS;
}
```

Exercice 4 – Une interface pour un jeu de pendu

Dans cet exercice, il est question d'implémenter une interface pour gérer un jeu de pendu. Voici un exemple de partie :

```
./pendu telecom

# le contenu du terminal est effacé (pour ne pas voir le mot à trouver !)

Il faut trouver: -------
Proposez une lettre : a
Il n'y a pas de lettre 'a' dans le mot à trouver...
Il faut trouver: -------
Proposez une lettre : e
Oui ! La lettre 'e' est présente 2 fois !
Il faut trouver: -e-e---
Proposez une lettre : c
Oui ! La lettre 'c' est présente 1 fois !
Il faut trouver: -e-ec--
Proposez une lettre : o
```

```
Oui ! La lettre 'o' est présente 1 fois !
Il faut trouver: -e-eco-
Proposez une lettre : e
La lettre 'e' a déjà été trouvée...
Il faut trouver: -e-eco-
Proposez une lettre : m
Oui ! La lettre 'm' est présente 1 fois !
Il faut trouver: -e-ecom
Proposez une lettre : t
Oui ! La lettre 't' est présente 1 fois !
Il faut trouver: te-ecom
Proposez une lettre : l
Oui ! La lettre 'l' est présente 1 fois !
Bravo ! Vous avez trouvé le mot en 6 coups ! Le mot était 'telecom'.
```

L'interface utilisera cette structure pour gérer l'état du jeu :

```c
struct pendu_s {
    char* mot; // mot à trouver
    int len_mot; // nombre de lettres dans le mot à trouver
    int* lettres_trouvees; // tableau de taille len_mot qui indique quelles
       ↪ lettres ont déjà été trouvées
    int nb_essais; // nombre de propositions de lettres
    int nb_lettres_trouvees; // nombre de lettres déjà trouvées dans le mot
};
```

Le tableau `lettres_trouvees` contient des booléens : si `lettres_trouvees[i]` vaut `vrai`, cela signifie que la lettre `mot[i]` a été trouvée.

1. Écrire la fonction qui permet d'initialiser la structure `struct pendu_s`. On considérera que les allocations de mémoire dynamiques réussissent toujours : ce n'est donc pas nécessaire de gérer les cas où une erreur surviendrait.

```c
struct pendu_s* init_pendu(char* mot)
{
    struct pendu_s* pendu = malloc(sizeof(struct pendu_s));
    pendu->mot = mot;
    pendu->len_mot = strlen(mot);
    pendu->nb_essais = 0;
    pendu->nb_lettres_trouvees = 0;
    pendu->lettres_trouvees = calloc(pendu->len_mot, sizeof(int));

    return pendu;
}
```

2. Écrire la fonction qui permet de libérer la mémoire allouée par la fonction `init_pendu()`.

```c
void pendu_libere(struct pendu_s* pendu)
{
    free(pendu->lettres_trouvees);
    free(pendu);
}
```

3. Écrire la fonction qui permet de proposer une lettre. Si la lettre proposée appartient au mot à trouver, le tableau `lettres_trouvees` est mis à jour en conséquence. Cette fonction renvoie le nombre de lettres découvertes ou `-1` si la lettre a déjà été trouvée. On ne se préoccupera pas des différences de casse (lettre majuscule ou minuscule). Ne pas oublier de mettre également à jour les différents compteurs.

```c
int pendu_tester_lettre(struct pendu_s* pendu, char lettre)
{
    int nb = 0;
```

```
    for (int i = 0; i < pendu->len_mot; i++)
    {
        if (pendu->mot[i] == lettre)
        {
            if (pendu->lettres_trouvees[i])
            {
                return -1;
            }

            pendu->lettres_trouvees[i] = 1;
            nb++;
        }
    }

    pendu->nb_lettres_trouvees += nb;
    pendu->nb_essais++;

    return nb;
}
```

4. Écrire la fonction qui permet de savoir si on a gagné, *i.e.*, si toutes les lettres du mot ont été découvertes.

```
int pendu_gagne(struct pendu_s* pendu)
{
    return pendu->nb_lettres_trouvees == pendu->len_mot;
}
```

5. Écrire la fonction qui affiche le mot avec les lettres découvertes et en remplaçant les autres lettres par un tiret -.

```
void pendu_affiche_mot(struct pendu_s* pendu)
{
    for (int i = 0; i < pendu->len_mot; i++)
    {
        if (pendu->lettres_trouvees[i])
        {
            printf("%c", pendu->mot[i]);
        }
        else
        {
            printf("-");
        }
    }
}
```

6. Écrire la fonction qui affiche le message de victoire en indiquant le nombre de coups utilisés.

```
void pendu_affiche_victoire(struct pendu_s* pendu)
{
    assert(pendu_gagne(pendu));

    printf("Bravo ! Vous avez trouvé le mot en %d coups ! Le mot était '%s
        ↪ '.\n", pendu->nb_essais, pendu->mot);
}
```

7. (Bonus) En utilisant les fonctions précédentes, compléter la fonction `main()` qui permet de jouer au pendu. Le mot à trouver est passé comme paramètre du programme. On considérera que le programme est toujours bien utilisé (nombre de paramètres, ...) et que l'utilisateur saisit toujours une lettre.

   Pour effacer le contenu du terminal, on pourra utiliser le code suivant :

```
printf("\e[1;1H\e[2J");
```

Pour demander à l'utilisateur de saisir une lettre, on pourra utiliser le code suivant :

```
printf("Proposez une lettre : ");
char lettre;
scanf(" %1c", &lettre);
```

```
int main(int argc, char* argv[])
{
    struct pendu_s* pendu = init_pendu(argv[1]);

    printf("\e[1;1H\e[2J");

    while (!pendu_gagne(pendu))
    {
        printf("Il faut trouver: ");
        pendu_affiche_mot(pendu);
        printf("\n");

        printf("Proposez une lettre : ");
        char lettre;
        scanf(" %1c", &lettre);

        int nb_trouvees = pendu_tester_lettre(pendu, lettre);
        if (nb_trouvees == 0)
        {
            printf("Il n'y a pas de lettre '%c' dans le mot à trouver...\n
                ↪ ", lettre);
        }
        else if (nb_trouvees == -1)
        {
            printf("La lettre '%c' a déjà été trouvée...\n", lettre);
        }
        else
        {
            printf("Oui ! La lettre '%c' est présente %d fois !\n", lettre
                ↪ , nb_trouvees);
        }
    }

    pendu_affiche_victoire(pendu);

    pendu_libere(pendu);

    return EXIT_SUCCESS;
}
```