Année 2023-2024 — 2ème session – 21 Mars 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.**
- **Nouveau ! 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 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.

2. Que va afficher le code suivant ?

```c
char c = 0;
while (c >= 0)
{
    c++;
}
printf("%d\n", c);
```

   ☐ `0`

   ☐ `z`

   ■ `-128`

   ☐ Rien, le programme reste bloqué dans la boucle `while`

   > Le type `char` permet de stocker des entiers de -128 à 127. Lorsque `c` vaut 127 et qu'il est incrémenté, l'incrémentation est faite au niveau binaire, mais l'ensemble de bits est alors interprété comme -128.

3. Que va afficher le code suivant ?

```c
char* txt = "Machin";
char* str = malloc(32*sizeof(char));
memset(str, 'A', 32);
memcpy(str, txt, strlen(txt)*sizeof(char)); // memcpy(dst, src, taille)
printf("%s\n", str);
free(str);
```

   ☐ `Machin`

   ☐ `AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA`

   ☐ `MachinAAAAAAAAAAAAAAAAAAAAAAAAAA`

   ■ On ne peut pas prévoir avec certitude, le `printf` va accèder à de la mémoire qu'il ne possède pas

Le `malloc` alloue un tableau de 32 `char`. On place dans chacune de ces cases `'A'` (soit la valeur numérique 65). On copie ensuite dans ce tableau le contenu de la zone mémoire de taille 6 (`strlen(txt)`) qui commence à l'adresse contenue dans `txt`. Le problème est qu'on ne copie pas le zéro terminal de la chaîne de caractères. Lorsqu'on demande d'afficher `str`, `printf` ne s'arrête pas de lire la mémoire tant qu'il ne rencontre pas un zéro et donc il sort forcément de la mémoire allouée au tableau `str`. Si on est chanceux `MachinAAAAAAAAAAAAAAAAAAAAAAAAAA` est affiché, sinon une erreur de segmentation arrêtant le programme se produit.

4. Que va afficher le code suivant ?

```
int t[4] = {3, 4, 1, 2};
int* u = t + 1;
int* v = u + 1;
printf("%d\n", t[v[0]]);
```

- ■ 4
- ☐ 3
- ☐ Une autre valeur
- ☐ 2

`u` contient l'adresse de `t[1]`.
`v` contient l'adresse de `u[1]`, donc l'adresse de `t[2]`.
`v[0]` correspond donc à `t[2]`, soit `1`. On affiche alors `t[1]`, soit `4`.

5. Que va afficher le code suivant ?

```
int a = 15;
int b = 2;
float c = a / b;
printf("%f\n", c);
```

- ☐ `7.500000`
- ■ `7.000000`
- ☐ Rien, il y a une erreur à la compilation
- ☐ On ne peut pas prévoir

`a` et `b` sont des entiers, la division de `a` par `b` est donc une division entière, dont le résultat vaut `7`. Ensuite, on stocke ce résultat dans une variable faite pour stocker un flottant, donc la valeur entière `7` est convertie en flottant.

Exercice 2 – Questions de cours

1. Donner la ligne de commande nécessaire pour compiler le fichier `main.c` en un programme `prog` avec le standard C99, tous les avertissements activés et en considérant les avertissement comme des erreurs :

```
cc -std=c99 -Wall -Werror main.c -o prog
```

2. 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)`.

3. (a) Qu'est-ce que le *prototype* d'une fonction ?

   Le prototype indique le nom de la fonction, le type de la valeur renvoyée et le type de ses paramètres. Le prototype ne comprend pas le corps de la fonction et se termine toujours par un point-virugle.

   (b) Dans quels cas a-t-on besoin d'écrire le prototype d'une fonction ? Citez-en deux.

   - On souhaite utiliser une fonction qui est définie plus bas dans le fichier : le prototype doit alors être plus haut dans le fichier.
   - Indiquer que la fonction existe, mais ailleurs, par exemple lorsqu'on place les prototypes dans des fichiers d'en-tête pour indiquer que les fonctions existent dans un autre fichier objet (fichier `.o` ou bibliothèque).

4. Comment tester si un nombre entier est pair en utilisant les opérateurs bit-à-bit ?

```
nombre = 4;
printf("%d est %spair\n", nombre, (nombre & 1)? "im": "");
printf("%d est %spair\n", nombre, ((~nombre)& 1)? "": "im");
```

Exercice 3 – Écrivons un peu de code

On souhaite écrire un programme qui tire n nombres au hasard, compte et affiche combien sont pairs. Par exemple :

```
./prog 1548
761 nombres pairs parmi 1548
```

1. Écrire une fonction `tab_nb_pairs()` qui prend en paramètre un tableau de nombres entiers (et peut-être d'autre(s) paramètre(s)) et renvoie combien de nombres du tableau sont pairs. On n'utilisera pas d'opérateurs bit-à-bit pour tester la parité des nombres.

```
int tab_nb_pairs(int* tab, int n)
{
    int nb_pairs = 0;

    for (int i = 0; i < n; i++)
    {
        if (tab[i] % 2 == 0)
        {
            nb_pairs++;
        }
    }

    return nb_pairs;
}
```

2. Écrire **tout** le code nécessaire pour avoir un tel programme : instructions de préprocesseur, fonction `main()`, *etc.* La fonction `main()` tire aléatoirement n nombres (n est un paramètre du programme) à l'aide de la fonction (existante) `rand()`, elle les stocke dans un tableau, fait appel à la fonction `tab_nb_pairs()` et affiche le résultat. La définition de la fonction `tab_nb_pairs()` sera omise, seul son emplacement dans le code source sera signifié par un commentaire. On considérera que le programme est toujours bien utilisé

(nombre de paramètres, ...) et que les fonctions réussissent toujours, ce n'est donc pas la peine de vérifier le nombre de paramètres du programme ou les valeurs de retour des fonctions.

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

// tab_nb_pairs() est définie ici

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

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

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

    printf("%d nombres pairs parmi %d\n", tab_nb_pairs(tab, n), n);

    free(tab);

    return EXIT_SUCCESS;
}
```

Exercice 4 – Une interface pour gérer des listes de lecture musicales

Dans cet exercice, il est question d'implémenter une interface pour gérer des listes de lecture[1]. Un exemple d'utilisation de cette interface dans un programme peut donner l'affichage suivant :

```
  Let it be (4:04)
> Allumer le feu (4:19)
  Pour que tu m'aimes encore (4:11)
Durée totale: 12:34
```

L'interface utilisera les deux structures suivantes :

```c
struct musique_s {
    char* titre;
    int duree; // en secondes
};

struct playlist_s {
    int nb_musiques; // nombre de musiques dans la playlist
    int capacite; // capacité de la playlist
    struct musique_s* musiques; // tableau de musiques
    int musique_courante; // indice dans la tableau de la musique en cours
        ↪ de lecture
};
```

1. Écrire la fonction qui initialise une playlist qui peut contenir au maximum `taille` musiques. À l'initialisation, `musique_courante` vaut `-1`. 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 playlist_s* nouvelle_playlist(int taille)
{
    struct playlist_s* p = malloc(sizeof(struct playlist_s));
    p->nb_musiques = 0;
```

[1] Que Shakespeare appelerait des *playlists*.

```
    p->capacite = taille;
    p->musique_courante = -1;
    p->musiques = malloc(taille * sizeof(struct musique_s));

    return p;
}
```

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

```
void liberer_playlist(struct playlist_s* playlist)
{
    free(playlist->musiques);
    free(playlist);
}
```

3. Écrire la fonction qui permet d'ajouter une musique à la playlist, à la suite de toutes les musiques déjà présentes. Si la playlist est déjà pleine, la fonction renverra `0`, sinon elle renverra `1`.

```
int ajouter_musique(struct playlist_s* playlist, char* titre, int duree)
{
    if (playlist->nb_musiques == playlist->capacite)
    {
        return 0;
    }

    playlist->musiques[playlist->nb_musiques].titre = titre;
    playlist->musiques[playlist->nb_musiques].duree = duree;
    playlist->nb_musiques++;

    return 1;
}
```

4. Écrire la fonction qui permet de connaître la durée totale (en secondes) de la playlist.

```
int duree_playlist(struct playlist_s* playlist)
{
    int duree = 0;
    for (int i = 0; i < playlist->nb_musiques; i++)
    {
        duree += playlist->musiques[i].duree;
    }

    return duree;
}
```

5. Écrire la fonction qui permet de passer à la musique suivante. Cette fonction met à jour l'attribut `musique_courante` de la playlist et renvoie un pointeur vers la nouvelle musique courante. S'il n'y a pas de prochaine musique (la musique courante est la dernière musique de la playlist), `NULL` est renvoyé.

```
struct musique_s* musique_suivante(struct playlist_s* playlist)
{
    if (playlist->musique_courante == playlist->nb_musiques-1)
    {
        return NULL;
    }

    playlist->musique_courante++;

    return &playlist->musiques[playlist->musique_courante];
}
```

6. Écrire la fonction qui décompose un nombre de secondes en minutes et secondes. On considérera qu'il est toujours possible de déréférencer `minutes` et `secondes`.

```c
void convertir_secondes(int total_secondes, int* minutes, int* secondes)
{
    *minutes = total_secondes / 60;
    *secondes = total_secondes % 60;
}
```

7. Écrire la fonction qui affiche une playlist sous la forme de l'exemple donné au début de cet exercice (sans la durée totale). Le caractère `>` devant le titre d'une musique indique qu'il s'agit de la musique courante ; attention à l'alignement des titres des autres musiques. Les nombres entre parenthèses indiquent la durée de chaque musique.

```c
void afficher_playlist(struct playlist_s* playlist)
{
    for (int i = 0; i < playlist->nb_musiques; i++)
    {
        int minutes, secondes;
        convertir_secondes(playlist->musiques[i].duree, &minutes, &
            ↪ secondes);

        printf("%s %s (%d:%02d)\n",
            i == playlist->musique_courante ? ">" : " ",
            playlist->musiques[i].titre,
            minutes,
            secondes
        );
    }
}
```

8. (Bonus) Écrire la fonction qui mélange aléatoirement l'ordre des musiques dans une playlist. On appliquera l'algorithme de Fisher-Yates[2], qui pour mélanger un tableau `a` de `n` éléments (indicés de 0 à $n-1$) est le suivant :

   Pour `i` allant de `n-1` à `1` faire
   `j` ← nombre aléatoire tel que $0 \leq j \leq i$
   échanger `a[j]` et `a[i]`
   Fin Pour

```c
void melanger_playlist(struct playlist_s* playlist)
{
    for (int i = playlist->nb_musiques-1; i > 0; i--)
    {
        int j = rand() % (i+1);
        struct musique_s tmp = playlist->musiques[j];
        playlist->musiques[j] = playlist->musiques[i];
        playlist->musiques[i] = tmp;
    }
}
```

[2] https://en.wikipedia.org/wiki/Fisher-Yates_shuffle