4TIN303U - PROGRAMMATION C
Dynamic allocation 2

▸ **Exercice 1.** Dans le programme suivant, dessinez l'état de la mémoire au fil de l'exécution.

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

struct student
{
    int st_number;
    int age;
    char* name;
};

typedef struct student Student;

Student* makeStudent(int number, int age, char* name)
{
    if(name == NULL)
    {
        fprintf(stderr, "Null pointer given as an argument \n");
        exit(EXIT_FAILURE);
    }

    Student* res = malloc(sizeof(Student));
    if(res == NULL)
    {
        fprintf(stderr, "Memory allocation failed \n");
        exit(EXIT_FAILURE);
    }

    res->st_number = number;
    res->age = age;

    int length = 0;
    while(name[length]!='\0')
    {
        length++;
    }
    res->name = malloc(sizeof(char)*(length+1));
    if(res->name == NULL)
    {
        fprintf(stderr, "Memory allocation failed \n");
        exit(EXIT_FAILURE);
    }
    for(int i = 0; i < length+1; i++)
    {
        res->name[i] = name[i];
    }
    return res;
}

int main (void)
{
    Student* st1 = makeStudent(11111111, 21, "Dupont");
    Student* st2 = st1;
    st2->st_number = 22222222;
    Student st3 = *st1;
    st3.st_number = 33333333;
    st3.name[0] = 'B';
```

```
    printf("%d, %s\n %d, %s\n %d, %s\n", st1->st_number, st1->name, st2->st_number, st2->name, st3.st_number, st3.name);
    free (st1->name);
    free (st1);
    free (st2);

    return EXIT_SUCCESS;
}
```

▸ **Exercice 2.** Soit la structure suivante :

```
struct point {
    float x;
    float y;
};
```

1. Écrivez (si vous ne l'avez pas déjà fait) une fonction qui alloue dynamiquement un tableau de taille *length* (passé en paramètre) pouvant contenir des éléments de type `struct point`. Votre fonction doit retourner l'adresse de la première case du tableau.
2. Écrire une fonction permettant de libérer ce tableau.
3. Écrire une fonction remplissant l'élément d'indice `index` du tableau avec des coordonnées x et y.
4. Écrire une fonction permettant d'afficher le contenu du tableau.
5. Écrire une fonction permettant de tester le programme sur des coordonnées de points passées en paramètre de la ligne de commande. Ainsi votre programme pourra par exemple avec la commande `./myProgram 1 2.5 3 4` afficher :

   ```
   Point 1 :  (1.000000,2.500000)
   Point 2 :  (3.000000,4.000000)
   ```

   Votre programme devra quitter en indiquant l'usage sur l'erreur standard en cas de mauvais nombre d'arguments.

▸ **Exercice 3.** ********* Dans le programme suivant, dessinez l'état mémoire au fil de l'exécution.

Ce code ne termine pas. Pourquoi ?

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

struct maze
{
    struct maze* right;
    struct maze* left;
    bool isExit;
};

typedef struct maze Maze;

Maze* makeMaze(bool ex)
{
    Maze* res = malloc(sizeof(Maze));
    if(res == NULL)
    {
        fprintf(stderr, "Memory allocation failed \n");
        exit(EXIT_FAILURE);
```

```
    }
    res->isExit = ex;
    res->right = NULL;
    res->left = NULL;
    return res;
}

void addLeft(Maze* origin, Maze* toBeLeft)
{
    while(origin->left != NULL)
    {
        origin = origin->left;
    }
    origin->left = toBeLeft;
    toBeLeft->right = origin;
}

void freeMaze(Maze* myMaze)
{
    if(myMaze == NULL)
        return;
    freeMaze(myMaze->right);
    freeMaze(myMaze->left);
    free(myMaze);
}

int main(void)
{
    Maze* m1 = makeMaze(false);
    Maze* m2 = makeMaze(false);
    Maze* m3 = makeMaze(false);

    m1->right = m2;
    addLeft(m1,makeMaze(false));
    addLeft(m2,makeMaze(true));
    addLeft(m1,m3);
    addLeft(m1,m2);
    addLeft(m1,m3);

    freeMaze(m1);
    return EXIT_SUCCESS;
}
```