TD n°4 de langage C : structures, tableaux 2D

Exercice 1 (acquisition)

Soient les déclarations et définitions ci-contre.

La fonction f1 (respectivement f2) comporte t-elle une anomalie ? Si oui laquelle ? Si non, donner un exemple d'appel de cette fonction et de récupération de la valeur de retour.

```c
typedef struct
{
    int x;

    int y;

}xy;
```

```c
xy f1()
{
    xy out;

    out.x=0;

    out.y=0;

    return out;
}
```

```c
xy* f2()
{
    xy out;

    out.x=0;

    out.y=0;

    return &out;
}
```

Exercice 2 (acquisition)

Soient les déclarations et définitions ci-contre.

La fonction g1 (respectivement g2) comporte t-elle une anomalie ? Si oui laquelle ? Si non, donner un exemple d'appel de cette fonction et de récupération de la valeur produite.

```c
void g1(xy in)
{
    in.x=0;

    in.y=0;
}
```

```c
void g2(xy* in)
{
    in->x=0;

    in->y=0;
}
```

Exercice 3 (acquisition)

Soient les déclarations et définitions ci-contre.

Pour chacune des fonctions h1 à h4, répondez aux questions suivantes :

1. La définition de la fonction est elle syntaxiquement correcte ?
2. Si oui, un appel de cette fonction peut-il permettre de modifier le contenu de tt ?
3. Si oui, donnez le code de cet appel.

```c
typedef struct {int tab[5];} eTab;

void h1(eTab t){t[0]=0;}

void h2(eTab t){t.tab[0]=0;}

void h3(eTab* t){t->tab[0]=0;}

eTab h4(eTab t){t.tab[0]=0; return t;}

eTab tt = {{1,1,1,1,1}};
```

Exercice 4 (acquisition)

On définit ci-contre les types point et segment.

Donnez la définition complète d'une fonction `point milieu(segment s)` qui retourne le milieu du segment s.

Même question avec une fonction `point milieu(segment* s)` acceptant en paramètre l'adresse d'un segment.

Même question avec une fonction `void milieu(segment* s, point* m)` qui place le résultat dans la variable pointée par m.

Donnez un exemple d'appel de chacune des trois variantes de la fonction.

```c
typedef struct
{
    double x;
    double y;
}point;
```

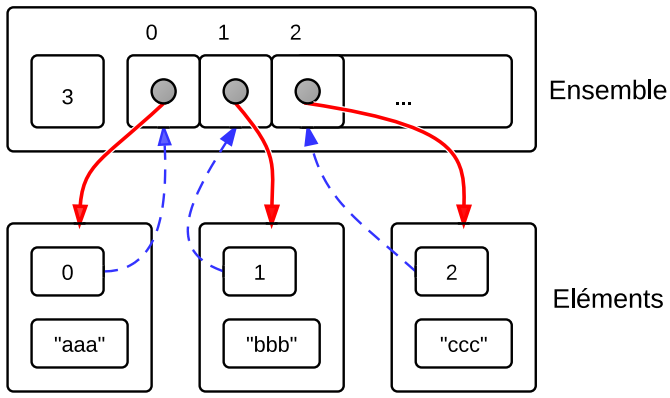
```c
typedef struct
{
    point a;
    point b;
}segment;
```

TD n°4 de langage C : structures, tableaux 2D

Exercice 5 (consolidation)

Le but de l'exercice est la réalisation d'un ensemble dont les éléments peuvent être ajoutés et retirés en temps constant. L'ensemble est représenté par une structure incluant un tableau dont les cellules contiennent les adresses des éléments, ainsi qu'un entier ayant pour valeur le nombre d'éléments stockés. Chaque élément est une structure constituée d'une chaîne de caractères et d'un entier appelé indexe qui contient l'indice auquel l'adresse de l'élément est placée dans le tableau. C'est cet indexe qui permet de retirer n'importe quel élément de l'ensemble en temps constant, indépendamment du nombre d'éléments déjà stockés.

Exemple d'un ensemble contenant trois éléments.




```c
#define MAX 10

#define LMAX 20
```

```c
typedef struct
{
    char str[LMAX];
    int index;
}element;
```

```c
typedef struct
{
    element* tab[MAX];
    int n;
}ensemble;
```

Vous devez réaliser les fonctions suivantes :

1. **`element* creeElem(char* s)`** qui crée un nouvel élément contenant une *copie* de la chaîne s et retourne l'adresse de l'élément créé. Cet élément doit être stocké dans un bloc mémoire de la taille appropriée réservé dans le tas par la fonction malloc.
2. **`void ajouteElem(ensemble* w, element* e)`** qui ajoute l'élément pointé par e dans l'ensemble pointé par w. On suppose que l'élément à ajouté a été préalablement créé avec la fonction creeElem.
3. **`void retireElem(ensemble* w, element* e)`** qui retire de l'ensemble pointé par w l'élément d'adresse e. On suppose que cet élément est initialement présent dans l'ensemble.

Les fonctions d'ajout et de retrait d'éléments doivent s'exécuter en temps constant et donc ne doivent comporter aucune boucle.

Vous devez ensuite réaliser une fonction main() qui crée des éléments contenant les chaînes «aaa», «bbb», «ccc», qui les ajoute à un ensemble initialement vide, puis qui retire l'éléments contenant la chaîne «bbb».

Exercice 6 (consolidation)

Le jeu de la vie est un automate cellulaire qui fait évoluer une grille à deux dimensions dont les cases peuvent prendre deux valeurs possibles, vivante ou morte. L'évolution est définie par une succession de générations. Une cellule est vivante à la génération i si et seulement l'une des deux propriétés suivantes est vérifiée : (1) la cellule était vivante et avait exactement deux voisines vivantes à la génération i-1, ou (2) la cellule avait exactement trois voisines vivantes à la génération i-1.

Définissez un type grille qui représente une grille et réalisez une méthode creeGrille qui crée une grille de n par n cases, où n est donné en paramètre.

Réalisez une fonction qui accepte en paramètre l'adresse d'une grille et qui simule une génération de l'évolution de l'automate sur cette grille.