Proposition de corrigé

❑ Question 1

Ecriture de la fonction d'initialisation

```c
maillon_t *init(void)
{
    maillon_t *first = malloc(sizeof(maillon_t));
    maillon_t *last = malloc(sizeof(maillon_t));
    first->donnee = INT_MIN;
    first->suivant = last;
    last->donnee = INT_MAX;
    last->suivant = NULL;
    return first;
}
```

❑ Question 2

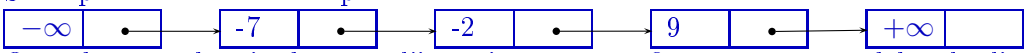
Ecriture de la fonction de localisation

```c
maillon_t *localise(maillon_t *t, int v)
{
    maillon_t *res = t;
    while (res->suivant->donnee<v)
    {
        res = res->suivant;
    }
    return res;
}
```

❑ Question 3

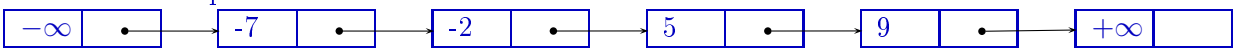
Jeu de test pour l'insertion

L'énoncé indique que les valeurs à insérer sont toujours *strictement* comprises entre INT_MIN et INT_MAX. Si on part de la liste de départ donnée dans l'énoncé :



On a donc pas besoin de tester l'insertion en toute fin ou en en tout debut les listes, les seuls cas à tester sont :

- L'insertion d'un entier n qui ne figure pas encore dans la liste (valeur de retour `true`). Par, exemple, l'insertion de 5 qui doit donner :



- L'insertion d'un entier déjà présent dans la liste (valeur de retour `false`). Par exemple, l'insertion de 9 qui ne modifie pas la liste et renvoie `false`.

❑ Question 4

Message d'erreur du compilateur et première correction

La fonction `localise` prend en paramètre un pointeur sur un struct `maillon_t`, ici on passe l'adresse d'un objet de ce type car on utilise l'opérateur `&` qui renvoie l'adresse du maillon, ce qui cause l'erreur. On propose la première correction suivante :

```c
    maillon_t *p = localise(t,v);
```

❑ Question 5

Correction de la fonction d'insertion

La fonction `insere` ne prend pas en compte le cas où la valeur est déjà présente dans la liste. On peut aussi éventuellement testé si `malloc` échoue.

```
bool insere(maillon_t *t, int v)
{
    maillon_t *p = localise(t,v);
    maillon_t *n = malloc(sizeof(maillon_t));
    if (p->suivant->donnee!=v)
    {
        n->suivant = p->suivant;
        n->donnee = v;
        p->suivant = n;
        return true;
    }
    else
    { return false;}
}
```

❑ Question 6

Fonction de suppression d'un maillon

```
bool supprime(maillon_t *t,int v)
{
    maillon_t *emp = localise(t,v);
    if (emp->suivant->donnee==v)
    {
        emp->suivant = emp->suivant->suivant;
        return true;
    }
    else
    { return false;}
}
```

❑ Question 7

Complexité des fonctions insere et supprime

Dans les deux cas, on fait appel à tt localise qui parcourt dans le pire des cas la totalité des éléments de la liste. Les autres opérations s'effectuent en temps constant. La complexité est donc en $O(n)$ où n désigne le nombre d'éléments de la liste.

❑ Question 8

Modèle mémoire

Les différentes régions sont :

- La zone des données qui contient les données dont la taille est connue à la compilation (notamment les variables globales).
- La pile qui contient les paramètres d'appels à une fonction ainsi que les variables locales à celle ci
- Le tas qui contient les variables alloués dynamiquement par le programme via `malloc`.

Dans le cas de ce programme, la valeur entière 717, sera stocké dans la zone de données (car `v` est une variable globale initialisée à cette valeur), dans la pile lors de l'appel à `insere(t, v)`, car les paramètres sont passés par valeur et donc copié. Et enfin dans le tas car l'insertion dans la liste crée un maillon contenant la valeur de V donc 717.