3.1 Implantation rudimentaire du type pile (LIFO)

Pour commencer, on propose une implantation rudimentaire du type pile : les espaces alloués s'empilent dans le tableau et la désallocation consiste en un dépilement. Ainsi, l'utilisateur doit veiller à ce que les appels de la fonction `monfree` se fassent dans l'ordre inverse exacte de ceux de la fonction `monalloc`.

Ce type d'implantation simple se base sur les définitions :

```
static char *nextfreebyte = buffer ;
```

Le pointeur `nextfreebyte` pointe sur le premier octet non alloué du tableau `buffer`.

Question.

1. Pourquoi le pointeur `nextfreebyte` est-il de classe d'allocation `static`.
2. Implantez les fonctions `monalloc` et `monfree` basées sur le principe ci-dessus.

Remarque. Ce principe ne permet pas une gestion simple de la mémoire dynamique notamment par la contrainte sur les appels. La section suivante propose une implantation plus souple des fonctions `monalloc` et `monfree`.

3.2 Implantation à base de listes doublement chaînées

Nous allons utiliser une structure de donnée auxiliaire permettant de stocker l'ensemble des blocs définis dans l'espace mémoire `buffer`.

Chaque bloc commence par une entête définie par une structure de type `freelist_t` constituée par les champs :

- `isfree` pour savoir si le bloc est libre ou pas ;
- `size` pour la taille en octets de cette zone ;
- `next` pour un pointeur sur l'entête du bloc suivante de la liste ;
- `previous` pour un pointeur sur l'entête du bloc précédent de la liste.

Ainsi, dans un bloc libre de taille totale n octets, ne sont réellement disponibles que $n - t$ octets avec :

```
t = 2*sizeof(unsigned int) + 2*sizeof(freelist_t *) ;
```

On suppose exister une variable globale `freelist_t *Head` qui permet de pointer sur la première entête. De plus, une entête dont le champs `next` (resp. `previous`) pointe sur `NULL` est la dernière (resp. première) de la liste. La liste des blocs est vide si `Head` pointe sur `NULL`.

Question.

1. Donnez la déclaration de la structure de l'entête. Définissez le type `freelist_t` correspondant.
2. Que fait la fonction définie ci-dessous :

```
static void AllocatorInit(void){
  freelist_t *ptr = (freelist_t *) buffer ;
  ptr->isfree = TRUE ;
  ptr->size = BUFFERSIZE ;
  ptr->next = NULL ;
  ptr->previous = NULL ;
  Head = ptr ;
  return ;
}
```

On suppose que les macros suivantes sont définies :