```
#define TRUE 1
#define FALSE 0
```

3. Donnez la définition de la fonction de prototype `void *monalloc(unsigned int taille)` qui prend en entrée la taille de l'espace mémoire désiré par l'utilisateur et qui retourne un pointeur sur le début de l'espace mémoire disponible (juste après l'entête) du bloc correspondant.

   L'algorithme utilisé pour ce faire est le suivant :

   (a) Pointer le premier bloc libre de taille suffisante — supérieure à t + taille — dans la liste des blocs. Ce bloc est le bloc courant. Retourner NULL si ce n'est pas possible ;

   (b) Si l'espace libre du bloc courant est inférieur à $2t$ + taille, aller à l'étape (d) ;

   (c) Sinon, le bloc courant doit être scindé en deux blocs : le premier est le nouveau bloc courant et sert à satisfaire la requête ; le second est ajouté à la liste des blocs juste derrière le bloc courant. Poursuivre par l'étape suivante ;

   (d) Retourner un pointeur sur l'espace libre positionné juste après l'entête du bloc courant.

4. Donnez la définition de la fonction de prototype `void monfree(void *ptr)` qui prend en entrée un pointeur et libère le bloc correspondant. On se fixe comme contrainte que pour tout pointeur p sur une cellule de la liste des blocs libre, la relation suivante

   `p->next > p`

   soit vérifiée.

5. Donnez la définition de la fonction de prototype `void garbagecollect(void)` qui fusionne les blocs voisins pouvant l'être.

6. Comment s'assurer que le pointeur `ptr` pointe bien après une entête ?

7. Que risque-t-il de se passer si un bloc de n octets de mémoire est alloué et que l'on manipule ce bloc sans précaution comme dans l'allocation de D suivante :

```
unsigned int i, n = 48 ;
char *D = (char *) monalloc(n) ;
for(i=0;i<65000;i++)
*(D+i) = 1 ;
```