TD – MALLOC – Partie 2 – B – Liste chainée

1 – Commentez et expliquez le code ci-dessous

```
#include <unistd.h>
#include <stdlib.h>

// ?? – 1
#define ENTETE_SIZE (ALIGN(sizeof(bloc_entete)))

// ?? – 2
typedef struct bloc_entete{
	size_t taille;
	unsigned short libre :1;
} bloc_entete;


// ?? – 3
bloc_entete* ADR_PREMIER_BLOC=0;

// ?? – 4
typedef struct bloc_entete
{	size_t taille;
	struct bloc_entete* suivant_ptr ;
	struct bloc_entete* precedent_ptr ;
} bloc_entete ;

// ? – 5
bloc_entete* ADR_PREMIER_BLOC_LIBRE=0;


void* myalloc(size_t t){

	// ?? – 6
size_t blk_size = ALIGN(t+ENTETE_SIZE);

	// ?? – 7
	if(ADR_PREMIER_BLOC_LIBRE == 0){
		void* bloc_total=(void*)sbrk(blk_size);
		void* bloc_utilisateur=(void*)bloc_total+ENTETE_SIZE

		bloc_entete* entete = (bloc_entete*)bloc_total;
		entete->taille=blk_size;
		entete->suivant_ptr=0;

		entete->precedent_ptr=NULL;
		ADR_PREMIER_BLOC = (bloc_entete*)entete;
	}
	// ?? - 8
	else{  if((bloc_entete*)ADR_PREMIER_BLOC_LIBRE->suivant_ptr==0){
			void* bloc_total=(void*)sbrk(blk_size);
			void* bloc_utilisateur=(void*)bloc_total+ENTETE_SIZE

			bloc_entete* entete = (bloc_entete*)bloc_total;
			entete->taille=blk_size;
			entete->suivant_ptr=0;
			entete->precedent_ptr=0;
			ADR_PREMIER_BLOC = (bloc_entete*)entete;
		}
```

```
		// ?? - 9
		else{	bloc_entete* current = (bloc_entete*)ADR_PREMIER_BLOC_LIBRE->suivant_ptr;
			int alloc=0;

			while( current!=0 && alloc==0){
				if(current->taille >= blk_size){
					if(current->suivant!=0){
						current->suivant_ptr->precedent_ptr = (bloc_entete*)current->precedent_ptr;
					}
					if(current->precedent_ptr!=0 || current->precedent_ptr!=NULL){
						current->precedent_ptr->suivant_ptr = (bloc_entete*)current->suivant_ptr
					}
					alloc=1;
					void* bloc_utilisateur=(void*)current+ENTETE_SIZE;
				}
				else{	current = (bloc_entete*)current->suivant_ptr;
				}
			}
			// ?? - 10
			if(alloc==0){
				void* bloc_total=(void*)sbrk(blk_size);
				void* bloc_utilisateur=(void*)bloc_total+ENTETE_SIZE

				bloc_entete* entete = (bloc_entete*) bloc_total;
				entete->taille=blk_size;
				entete->suivant_ptr=0;
				entete->precedent_ptr= (bloc_entete*)current->precedent_ptr;
			}
		}
	}

	return bloc_utilisateur;
}

void myfree3(void* ptr){
	// ?? - 11
	bloc_entete* entete_courante =(bloc_entete*)ptr-ENTETE_SIZE;
	if(entete_courante->precedent_ptr!=0 || entete_courante->precedent_ptr!=NULL){
		entete_courante->precedent_ptr->suivant_ptr=(bloc_entete*)entete_courante;
	}
	// ?? - 12
	if(entete_courante->suivant_ptr!=0){
		entete_courante->suivant_ptr->precedent_ptr=(bloc_entete*)entete_courante;
	}

}

void main(int argc, char* argv[]){
// Partie a compléter
}
```

Commentaires	
1	On définit la taille alignée de l'entête.
2	On créée une structure définissant l'entête, composé d'une taille et d'un "booléen" libre.

3	Initialement, bloc_entete sera à l'adresse ADR_PREMIER_BLOC. On initialise cette variable à 0 pour savoir qu'elle n'a pas encore été réellement donnée. Par la suite, on lui donnera la vraie adresse du premier bloc libre.
4	On définit une nouvelle structure ayant : une taille, l'adresse de la prochaine entête et l'adresse de l'entête précédente. C'est le principe de la liste chaînée.
5	On créée une adresse de type entete_bloc nommée ADR_PREMIER_BLOC_LIBRE. On l'initialise à zero pour, par la suite, savoir que la variable n'a pas été utilisée et ainsi, pour pouvoir lui mettre une vraie adresse.
6	blk_size est une variable de taille correspondant à l'alignement de la taille voulue (t en paramètre de myalloc) + la taille de l'entête.
7	Si l'adresse est à 0, cela veut dire qu'elle n'a pas vraiment été initialisée. Dans ce cas : On met l'adresse de départ dans bloc_total et on augmente la taille du brk de blk_size. bloc_utilisateur prend la vraie adresse à laquelle il pourra insérer des données (donc adresse de dapart + taille entete). On créée une nouvelle entête qu'on place à l'adresse à l'adresse bloc_total, avec la taille correspondante, l'adresse precedente à NULL et l'adresse suivante à 0. Puis, on initialise ADR_PREMIER_BLOC à la vraie adresse de la premiere entête.
8	On va dans ce else lorsque ADR_PREMIER_BLOC_LIBRE a déjà été initialisé. Dans ce cas, si l'adresse du suivant de l'entête à ADR_PREMIER_BLOC_LIBRE est 0, alors, cela signifie qu'il n'a pas encore de suivant. Dans ce cas, on devra créer une nouvelle entête à cet endroit. Pour ce faire, on utilise le même procédé que précédemment.
9	On accède à ce else lorsque ADR_PREMIER_BLOC_LIBRE et son suivant, on été initialisés à des vraies adresses (pas 0). A ce moment là, on va parcourir les entêtes les unes après les autres, à partir de la première libre, jusqu'à en obtenir une dont la taille disponible est supérieure à celle de blk_size. Lorsqu'on trouve, on modifie l'entete de cette adresse avec le prededent et suivant correspondants (si pas de suivant, alors 0), et on place le bloc utilisateur à la bonne adresse (qu'on va retourner par la suite).
10	Si alloc == 0, cela signifie qu'on a pas trouvé de bloc libre assez grand pour accueillir notre bloc. Dans ce cas, on augmente le brk et on ajoute à la fin le bloc qu'on veut avec son entête.
11	L'entête courante, devient l'adresse de départ, de l'entête, du bloc qu'on souhaite libérer. Si nécessaire, on modifier le précédent pour la bonne adresse.
12	Et si nécessaire, on modifie le suivant, pour que la chaine n'aie pas de trou.

2 – Complétez le main

```
void main(int argc, char* argv[]){
        int *p = myalloc(sizeof(int) * 4);
        *p = 5;

        int t[10];

        int *q = myalloc(sizeof(int) * 5;

        myfree(p);

}
```