```c
/*----------------------------------------------------------------------
--*/
/* actions.c fichier contenant la déclaration des */
/* différentes fonctions utilisées */
/*----------------------------------------------------------------------
--*/


#include "actions.h"


/*----------------------------------------------------------------------
--*/
/*CreateAction Insérer une nouvelle action à la suite de l'agenda */
/* */
/*En entrée: liste_actions Structure de type actions_t */
/* chaine_actions Chaine de caractères */
/* */
/*En sortie: la fonction retourne une nouvelle liste chainée contenan
/* une nouvelle action si la plage horaire n'est pas déjà occupée */
/*----------------------------------------------------------------------
--*/

void CreateAction (actions_t * liste_actions, char * chaine_actions)
{
  int i;
  char jour_heure[3];
  char nom_action[10];
  actions_t ** ptrAction;

  for (i=0; i<13; i++)
  {/*on divise la case chaine_actions avec le jour et l'heure d
    côté */
    if (i<3)
    {/*et le nom de l'action de l'autre*/
      jour_heure[i] = chaine_actions[i];
    }
    else
    {
      nom_action[i-3] = chaine_actions[i];
    }
  }

  ptrAction = Recherche( liste_actions, jour_heure); /*ptrAction représ
    la case précédente avant laquelle */
                /*il faut insérer la nouvelle action*/
  if (* ptrAction != NULL)
  {
    if (jour_heure != (* ptrAction)->suiv->jour_heure)
    {
      AlloueAction( * ptrAction, jour_heure, nom_action);/*on alloue un
        nouvel élément*/
    }
    else
```

```
    {
        printf("La plage horaire est deja occupee, vous ne pouvez pas ajou
          d'autres elements");
    }
  }
  else
  {
    AllocueAction( * ptrAction, jour_heure, nom_action);
  }


 


 


 
/*-------------------------------------------------------------------
--*/
/*AllocueAction    Allouer de l'espace mémoire à la suite du pointeur */
/*        Insérer le nouvel élément dans cet espace mémoire*/
/*                                                          */
/*En entrée:    ptrAction Pointeur sur l'adresse de l'élément précédent
/*      jour    Entier représentant le jour et l'heure  */
/*      nom    Chaine de caractères représentant le  */
/*      nom de l'action          */
/*                                                          */
/*En sortie:    la fonction retourne une nouvelle liste chainée contenan
/*    un nouvel espace mémoire et une nouvelle action      */
/*-------------------------------------------------------------------
--*/
 
void AllocueAction( actions_t * ptrAction, char jour [3], char action [10
{
  int       i;
  actions_t   *nouv;
 
  nouv = (actions_t *) malloc(sizeof(actions_t));
  nouv->suiv = ptrAction->suiv;
  ptrAction = nouv;
 
 
 
  for (i=0; i<3; i++)
  {       /*on divise la case chaine_actions avec le jour et l'heure d
    côté */       /*et le nom de l'action de l'autre*/
    nouv->jour_heure[i] = jour[i];
  }
  for (i=0; i<10; i++)
  {       /*on divise la case chaine_actions avec le jour et l'heure d
    côté */       /*et le nom de l'action de l'autre*/
    nouv->nom_action[i] = action[i];
  }
}
 
 
```

```c


/*-------------------------------------------------------------------
--*/
/*Recherche  Rechercher une valeur dans une liste chaînée    */
/*                                                           */
/*En entrée:    tete_liste  Structure de type actions_t     */
/*    val  Entier représentant le jour et l'heure           */
/*                                                           */
/*En sortie:    la fonction retourne l'adresse de l'élément précédent
/*    où il faut insérer la nouvelle valeur ou supprimer    */
/*    une valeur existante                                   */
/*-------------------------------------------------------------------
--*/

actions_t ** Recherche ( actions_t * tete_liste, char val [3])
{
  int     i;
  actions_t  ** cour = &tete_liste;

  for (i=0; i<3; i++)
  {
    while ( ( (*cour) != NULL ) && ( (*cour)->suiv->jour_heure[i] != val
      && ( (*cour)->suiv->jour_heure[i] < val[i]) )
    {
      *cour = (*cour) -> suiv;
    }
  }

  return cour;
}




/*-------------------------------------------------------------------
--*/
/*Sauvegarde    fonction qui sauvegarde la SDD sous la forme d'un fichie
/*                                                           */
/*En entrée:    fichier_sauvegarde  Fichier créer dans semaine.c  */
/*    liste_actions  Structure de type actions_t*/
/*    semaine    Chaine de caractères    */
/*                                                           */
/*En sortie:    Renvoit un fichier contenant une sauvegarde de la SDD*/
/*-------------------------------------------------------------------
--*/

void Sauvegarde ( FILE * fichier_sauvegarde, actions_t * liste_actions,
* semaine)
{
  int  i;
  actions_t  * cour = liste_actions;
```

```
	char	  action[19];
	
	while ( cour != NULL)
	{
		for (i=0;i<19;i++)
		{
			if (i<6)
			{
		action[i]= semaine[i];
			}
			else if (i<9)
			{
		  action[i]=cour->jour_heure[i-6];
			}
			else
			{
		action[i]=cour->nom_action[i-9];
			}
		}
		
		fputs (action, fichier_sauvegarde);
		cour = cour->suiv;
	}
	
}
	
	
	
	
	
/*-------------------------------------------------------------------------
--*/
/*CreateListeFromActions  Fonction qui cherche si une action est présent
/*		dans l'agenda		*/
/*		*/
/*En entrée:	liste_actions	Structure de type actions_t */
/*		rechAction	Chaine de caractères	*/
/*		*/
/*En sortie:	Renvoit, si l'action est présente, les jours qui */
/*	contiennent une ou plusieurs fois cette action sous forme de */
/*	tableau de caractères		*/
/*-------------------------------------------------------------------------
--*/
	
char * CreateListeFromActions ( actions_t * liste_actions, char * rechAc
{
	char	* liste_jour;
	int	i = 0;
	actions_t	* cour = liste_actions;
	
	liste_jour = "";
	
	while ( cour != NULL)
	{
		if (cour->nom_action == rechAction)
```

```
    {
      if ( i == 0 )
      {
    » liste_jour[i] = cour->jour_heure[0];
    » i = i+1;
      }
      else
      {
    » if ( liste_jour[i-1] != cour->jour_heure[0])
    » ··{
    » ····liste_jour[i] = cour->jour_heure[0];
    » ····i = i+1;
    » ··}
      }

    ····}
    ····cour = cour->suiv;
    ··}
    ··return liste_jour;
    }
    
    
    
    
    
    /*-------------------------------------------------------------------
    --*/
    /*SupprimeAction» Fonction qui supprime une action en connaissant  »*/
    /*» » » la date et l'heure» » » » » */
    /*» » » » » » » » » » */
    /*En entrée:» » liste_actions»Structure de type actions_t»» */
    /*» » » val»» Entier représentant le jour et l'heure» */
    /*» » » » » » » » » » */
    /*En sortie:» » la fonction retourne une nouvelle liste chainée ne »*/
    /*» » contenant plus l'action à supprimer si elle existe» » */
    /*-------------------------------------------------------------------
    --*/
    
    void SupprimeAction (actions_t * liste_actions, char val [3])
    {
    ··int»» i;
    ··actions_t » * cour = liste_actions;
    ··actions_t » ** ptrAction;
    ··
    ··ptrAction = Recherche (cour, val);
    ··
    ··for (i=0; i<3 ; i++)
    ··{
    ····if ((* ptrAction) != NULL)
    ····{
      ····if (val[i] == (*ptrAction)->suiv->jour_heure[i])
      ····{
    » LibererAction( * ptrAction);» /*on supprime un élément*/
      ····}
      ····else
```

```
254         ......{¤¬
255         » printf("La plage horaire ne contient aucun élément, il n'y a pas d'act
              supprimer");....¤¬
256         ......}¤¬
257         ....}¤¬
258         ....else¤¬
259         ....{¤¬
260         ......printf("La plage horaire ne contient aucun élément, il n'y a pas
              d'action à supprimer");¤¬
261         ....}¤¬
262         ..}¤¬
263         }¤¬
264         ¤¬
265         ¤¬
266         ¤¬
267         ¤¬
```