```
#include "record.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <arpa/inet.h>

#include <stdint.h>


struct header

{

    unsigned int TYPE : 15;

    unsigned int F : 1;

    unsigned int LENGTH : 16;

};

struct record

{

    struct header * HEADER;

    char * PAYLOAD;

    uint32_t UUID;

};

int record_init(struct record *r)

{

    r->HEADER = (struct header *) malloc (sizeof(struct header));

    if(r->HEADER==NULL)

    {

        return 1;

    }
    r->HEADER->LENGTH=0;
    r->HEADER->F=0;
    r->HEADER->TYPE=0;
    return 0;
}

void record_free(struct record *r)
{
    if(r->HEADER->LENGTH!=0)
        free(r->PAYLOAD);
}

/**
 * Renvoie le type d'un enregistrement
 * @pre: r != NULL
 */
int record_get_type(const struct record *r)
{
    return r->HEADER->TYPE;
}

/**
 * Définit le type d'un enregistrement
 * @pre: r != NULL
 * @post: record_get_type(r) == type
 */
```

```
void record_set_type(struct record *r, int type)

{

    r->HEADER->TYPE = type;

}


/**

 * Renvoie la taille du payload de l'enregistrement (dans l'endianness native
de la machine!)

 * @pre: r!= NULL

 */

int record_get_length(const struct record *r)

{

    return r->HEADER->LENGTH;

}


/**

 * Définit le payload de l'enregistrement, en copiant n octets

 * du buffer. Si le buffer est NULL (ou de taille 0), supprime

 * le payload

 * @pre: r != NULL && buf != NULL && n > 0

 * @post: record_get_length(r) == n

             && record_get_payload(<buf2>, n) == n

 *        && memcmp(buf, <buf2>, n) == 0

 * @return: -1 en cas d'erreur, 0 sinon

 */

int record_set_payload(struct record *r,

                 const char * buf, int n)
```

```
{
    if(r == NULL || n < 0)
    {
        return -1;
    }
    if(buf == NULL || n == 0)
    {
        if(r->PAYLOAD==NULL)
        {
            r->HEADER->LENGTH = 0;
            free(r->PAYLOAD);
            return 0;
        }

    }
    r->PAYLOAD = (char *)malloc(n*sizeof(char));
    if(r->PAYLOAD==NULL)
    {
        return -1;
    }
    r->HEADER->LENGTH = n;
    memcpy(r->PAYLOAD,buf,n);
    return 0;
}

/**
```

```
 * Copie jusqu'à n octets du payload dans un buffer
 * pré-alloué de taille n
 * @pre: r != NULL && buf != NULL && n > 0
 * @return: n', le nombre d'octets copiés dans le buffer
 * @post: n' <= n && n' <= record_get_length(r)
 */
int record_get_payload(const struct record *r,
                char *buf, int n)
{
    if(n<=r->HEADER->LENGTH)
    {
        memcpy(buf,r->PAYLOAD,n);
        return n;
    }
    else
    {
        memcpy(buf,r->PAYLOAD,r->HEADER->LENGTH);
        return r->HEADER->LENGTH;
    }
}

/**
 * Teste si l'enregistrement possède un footer
 * @pre: r != NULL
 * @return: 1 si l'enregistrement a un footer, 0 sinon
 */
```

```
int record_has_footer(const struct record *r)

{

    return r->HEADER->F;

}


/**

 * Supprime le footer d'un enregistrement

 * @pre: r != NULL

 * @post: record_has_footer(r) == 0

 */

void record_delete_footer(struct record *r)

{

    r->UUID=0;

    r->HEADER->F=0;

}


/**

 * Définit l'uuid d'un enregistrement

 * @pre: r != NULL

 * @post: record_has_footer(r) &&

          record_get_uuid(r, &<uuid2>) => uuid2 == uuid

 */

void record_set_uuid(struct record *r, unsigned int uuid)

{

    r->UUID=uuid;

    r->HEADER->F=1;
```

```
}


/**
 * Extrait l'uuid d'un enregistrement
 * @pre: r != NULL
 * @post: (record_has_footer(r) && uuid != 0) ||
 *       (!record_has_footer(r) && uuid == 0)
 */
unsigned int record_get_uuid(const struct record *r)
{
    return r->UUID;
}
/**
 * Ecrit un enregistrement dans un fichier
 * @pre: r != NULL && f != NULL
 * @return: n', le nombre d'octets écrits dans le fichier.
           -1 en cas d'erreur
 */
 int record_write(const struct record *r, FILE *f)
 {
    int no =0;
    struct header * tfl = (struct header *) malloc(sizeof(struct header));
    tfl->TYPE = (r->HEADER)->TYPE;
    tfl->F = (r->HEADER)->F;
    tfl->LENGTH = htons((r->HEADER)->LENGTH);
    if(fwrite(tfl,sizeof(struct header),1,f)!=1)
```

```
{
    return -1;
}
no = no+sizeof(struct header);
fprintf(stderr,"n0 : 1 : %d\n",no);
if(r->HEADER->LENGTH>0 && r->PAYLOAD!=NULL)
{
   no=no+r->HEADER->LENGTH;
    fprintf(stderr,"n0 : 2 : %d\n",no);
    if(fwrite(r->PAYLOAD,r->HEADER->LENGTH,1,f)!=1)
    {
    return -1;
    }
}
if((r->HEADER->F)==1)
{


    no=no+sizeof(uint32_t);
    fprintf(stderr,"n0 : 3 : %d\n",no);
    if(fwrite(&(r->UUID),sizeof(uint32_t),1,f)!=1)
    {
        return -1;
    }
}
fprintf(stderr,"NO : final %d\n",no);
return no;
```

```
 }
/**
 * Lit un enregistrement depuis un fichier
 * @pre: r != NULL && f != NULL
 * @return: n', le nombre d'octets luts dans le fichier.
          -1 en cas d'erreur
 */
 int record_read(struct record *r, FILE *f)
 {
     int n =0;
     struct header * TFL = (struct header *) malloc(sizeof(struct header));
     if(fread(TFL,sizeof(struct header),1,f)!=1)
     {
         return -1;
     }
     n = n+sizeof(struct header);
     r->HEADER->LENGTH = ntohs(TFL->LENGTH);
     r->HEADER->TYPE = TFL->TYPE;
     r->HEADER->F = TFL->F;
     if(r->HEADER->LENGTH!=0)
     {
         r->PAYLOAD = (char *) malloc(r->HEADER->LENGTH*sizeof(char));
         if(r->PAYLOAD==NULL)
         {
             return -1;
         }
```

```
        if(fread(r->PAYLOAD,r->HEADER->LENGTH,1,f)!=1)
        {
            return -1;
        }
        n = n+r->HEADER->LENGTH;
    }
    if(r->HEADER->F==1)
    {
        uint32_t * uuid = (uint32_t *) malloc(sizeof(uint32_t));
        if(fread(uuid,sizeof(uint32_t),1,f)!=1)
        {
            return -1;
        }
        n = n + sizeof(uint32_t);
        r->UUID = *uuid;
    }
    return n;
}
```