```c
/*
 * Archivo extract.c
 *
 * Descripcion: Archivo fuente con las funciones necesarias para la extraccion
 *  de un archivo .mytar
 *
 * Autores:
 *  Carlos Alejandro Sivira Munoz     15-11377
 *  Cesar Alfonso Rosario Escobar     15-11295
 */


#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <string.h>
#include <pwd.h>
#include <grp.h>
#include "extract.h"
#include "parser.h"



/* fileWriterBounded
 * ----------
 * Escribe de un archivo a otro utilizando los "file descriptors" de ambos.
 * Limita el contenido de escritura a un parametro entero.
 *
 *
 *  fd_source: "file descriptor" del archivo del que se copia.
 *  fd_dest: "file descriptor" del archivo al que se copia.
 *  ins: Estructura que contiene la informacion de las opciones de
 *  mytar
 */
void fileWriterBounded(int fd_source, int fd_dest, int total, mytar_instructions
inst) {

  char *temp_buffer = (char*) malloc( MAX_RW * sizeof(char) + 1 );
  char *buffer = (char*) malloc( MAX_RW * sizeof(char) + 1);
  int read_length, to_write, write_count;
  struct stat st_dest;

  read_length = 1;
  write_count = 0;

  while( (read_length = read(fd_source, buffer, read_length)) != 0 && write_count <
total){

    /*Modifica el string a escribir si se va a desencriptar */
    if (inst.mytar_options[Y]) {
      temp_buffer = (char*)encrypt(buffer, inst.encryption_offset);
      strncpy(buffer,temp_buffer,read_length);

      free(temp_buffer);
    }

```

```
    to_write = 0;
    while(read_length > to_write)
      to_write += write(fd_dest, buffer+to_write, read_length - to_write);

    write_count += read_length;
  }

  free(buffer);
}


/* getField
 * --------------
 * Devuelve un string que representa un campo de cabecera
 * de archivo .mytar
 *
 *  fd: "file descriptor" del archivo .mytar.
 *  field_length: Tamano del campo de cabecera
 *
 * retorna: "string" correspondiente a un campo de cabecera.
 */
char *getField(int fd, int field_length) {
  char *name;
  int just_read;

  name = (char*) malloc ( field_length + 1) ;

  just_read = read(fd, name, field_length);

  /* se anade terminador de string para no recibir basura adicional */
  name[field_length] = '\0';

  lseek(fd, 1, SEEK_CUR);
  return name;
}


/* getFieldSize
 * --------------
 * Calcula el tamano de un campo numerico de cabecera, buscando la siguiente
 * aparicion del caracter "•" que actua como un separador entre los campos.
 *
 *
 *  fd: "file descriptor" del .mytar
 *
 * retorna: Un entero que representa el tamano del campo.
 */
int getFieldSize(int fd) {
  char as;
  int acum;
  int testing;

  as = '0';
  acum = 0;

  while(as != STUFF_TOKEN) {
    acum += read(fd, &as, 1);
  }

  testing = lseek(fd, -acum, SEEK_CUR);
```

```
  acum -=1;

  return acum;
}


/* putField
 * --------------
 *  Esta funcion utiliza la separacion en campos de la cabecera del archivo
 *  .mytar para adquirir campos numericos y devolverlos como tipo long.
 *
 *
 *    fd: file descriptor del .mytar
 *
 *  retorna: un long asociado a un campo numerico de cabecera.
 */
long putField(int fd) {
  char *this_field;
  int size;
  long changed_field;

  size = getFieldSize(fd);
  this_field = getField(fd, size);
  changed_field = atol(this_field);
  free(this_field);

  return changed_field;
}


/* setModeAndOwn
 * --------------
 * Para cualquier archivo, se encarga de modificar su permisos (bits modales)
 * asi como su dueno y grupo, utilizando ung gid y un uid.
 *
 *  attr: Estructura de donde obtiene lo que modifica
 */
void setModeAndOwn(f_att attr) {
  int catcher;

  catcher = chmod(attr.name, attr.mode);
  if (catcher == -1) {
    printf("Error cambiando los permisos de %s\n",attr.name);
    perror("chmod");
  }

  catcher = chown(attr.name, attr.uid, attr.gid);
  if (catcher == -1) {
    printf("Error cambiando al dueno de  %s\n",attr.name);
    perror("chown");
  }
}


/* myLs
 * --------------
 * Imprime un listado similar al del comando ls -l de los archivos
 * presentes en el .mytar
 *
 *
```

```
 *  attr: Estructura que contiene los atributos del archivo.
 *  type: Entero que representa el tipo de archivo. 1=regular
 *    2=directorio, 3=Link simbolico
 */
void myLs(f_att attr, int type ) {

  char mode_formatted[9];
  struct passwd* pwd;
  struct group* grp;
  int i, move;
  long permissions;

  pwd = getpwuid(attr.uid);
  grp = getgrgid(attr.gid);
  permissions = attr.mode & 0777;
  move = 1;

  /* Verifico los bits de permisos para saber cuales
   * estan encendidos, y modifico mode_format en funcion
   * de eso */
  for (i=8; i>-1; i--) {
    if( (attr.mode & move) == move) {

      if (i%3 == 2)
        mode_formatted[i] = 'x';
      else if (i%3 == 1)
        mode_formatted[i] = 'w';
      else
        mode_formatted[i] = 'r';
    }
    else
      mode_formatted[i] = '-';
    move <<= 1;
  }


  if (type == 1) {
    printf("-%9s %5s %5s %4ld %s\n", mode_formatted, pwd->pw_name,
              grp->gr_name,  attr.size, attr.name);
  }
  else if (type == 2) {
    printf("d%9s %5s %5s %4s %s\n",mode_formatted, pwd->pw_name,
               grp->gr_name, "",  attr.name);
  }
  else {
    printf("l%9s %5s %5s %4ld %s -> %-10s\n",mode_formatted,
              pwd->pw_name, grp->gr_name,  attr.size,
              attr.name, attr.link_ptr);
  }

}
/* createFile
 * --------------
 * Crea un archivo de alguno de los tipos considerados (regulares, directorios,
 * links simbolicos) y actualiza sus atributos.
 *
 *  fd: "file descriptor" del archivo .mytar
 *  offset: posicion actual sobre el archivo .mytar
 *  attr: estructura que contiene los atributos a actualizar
 *  ins: Estructura que contiene la informacion de las opciones de
```

```
 *  mytar
 *
 * Retorna la posicion actual del apuntador
 */
int createFile(int fd, long offset, f_att attr, mytar_instructions inst) {
  int new_fd;
  int catcher;
  int return_v;
  struct stat test_state;

  return_v = offset;

  /* El archivo es regular */
  if( (attr.mode & __S_IFMT) == __S_IFREG) {
    return_v +=  attr.size;

    if (inst.mytar_options[T]) {
      myLs(attr, 1);
      lseek(fd, attr.size, SEEK_CUR);

    }
    else {
      new_fd = open(attr.name, CREATION_MODE);
      if (new_fd == -1) {
        lseek(fd, attr.size , SEEK_CUR);
        fprintf(stderr,"Error creando archivo %s\n", attr.name);
        perror("open");
      }
      else {

        setModeAndOwn(attr);
        fileWriterBounded(fd, new_fd, attr.size, inst);
        lseek(fd, -1, SEEK_CUR);


      close(new_fd);
      }
    }
  }

  /* El archivo es un directorio */
  else if( (attr.mode & __S_IFMT) == __S_IFDIR) {


    if (inst.mytar_options[T]) {
      myLs(attr, 2) ;
    }
    else {

      if( stat(attr.name, &test_state) == -1 ) {

        if( mkdir(attr.name, attr.mode) == -1) {
          fprintf(stderr,"Error creando directorio%s\n",
              attr.name);
          perror("mkdir");
        }
        else
          setModeAndOwn(attr);
      }
    }
```

```
299
300
301   }
302
303   /* El archivo es un link simbolico */
304   else if( (attr.mode & __S_IFMT) == __S_IFLNK) {
305
306     /*Verifica si es necesario ignorar el archivo*/
307     if (!inst.mytar_options[N]) {
308
309       if (inst.mytar_options[T]) {
310         myLs(attr, 3);
311       }
312       else {
313         new_fd = symlink(attr.name, attr.link_ptr);
314         if (new_fd == -1) {
315           fprintf(stderr,"Error creando link %s\n", attr.name);
316           perror("symlink");
317         }
318         else {
319
320           printf("nombre link %s nombre apuntador %s\n",attr.name, attr.link_ptr);
321           setModeAndOwn(attr);
322           free(attr.link_ptr);
323         }
324       }
325     }
326
327   }
328
329   /* Verifica si el modo verboso esta activo */
330   if (inst.mytar_options[V]){
331     verboseMode(inst, attr.name);
332   }
333
334   return return_v;
335 }
336
337
338 /* gatherFields
339  * --------------
340  *  Esta funcion junta los campos de cabecera (tanto numericos como no
341  *  numericos) con el objeto de reunir los atributos necesarios para
342  *  crear el archivo empaquetado. Esto ultimo lo hace con una llamada a
343  *  createFile().
344  *
345  *  Los campos estan ordenados de la forma:
346  *    modo # uid # gid [ # size] # name_size # name [# link_pointer] #
347  *
348  *  En donde size y link pointer son atributos que solo se extraen de
349  *  directorios y links simbolicos respectivamente.
350  *
351  *
352  *    fd = "file descriptor" del .mytar
353  *  ins: Estructura que contiene la informacion de las opciones de
354  *  mytar
355  *
356  *  retorna: el offset actual del archivo
357  */
358 int gatherFields(int fd, mytar_instructions inst) {
```

```
359   int new_fd;
360   long name_size, current_offset, previous_offset;
361   f_att attr;
362
363
364   attr.mode = putField(fd);
365   attr.uid = putField(fd);
366   attr.gid = putField(fd);
367
368   /* Si no es un directorio, guardo el tamano del archivo */
369   if ( (attr.mode & __S_IFMT) != __S_IFDIR )
370     attr.size = putField(fd);
371
372   name_size = putField(fd);
373
374   attr.name = (char*) malloc(name_size + 1);
375   read(fd, attr.name, name_size );
376   attr.name[name_size] = '\0';
377
378   /* Para los links simbolicos, extraigo el apuntador */
379   if( (attr.mode & __S_IFMT) == __S_IFLNK) {
380     current_offset = lseek(fd, 1, SEEK_CUR);
381
382     attr.link_ptr = (char*) malloc(attr.size + 1);
383     read(fd, attr.link_ptr, attr.size);
384
385     attr.link_ptr[attr.size] = '\0';
386   }
387
388   previous_offset = lseek(fd, 1, SEEK_CUR);
389
390   /* Creo el tipo de archivo y asigno sus atributos */
391
392   current_offset = createFile(fd, previous_offset, attr, inst);
393
394   free(attr.name);
395
396   return max(previous_offset, current_offset);
397 }
398
399
400
401 /* extractMyTar
402  * --------------
403  * Recibe un archivo .mytar y se encarga de extraer su contenido.
404  *
405  *  mt_name: Nombre del archivo .mytar a procesar
406  *  ins: Estructura que contiene la informacion de las opciones de
407  *  mytar
408  */
409 int extractMyTar(char **mt_name, mytar_instructions inst) {
410
411   int fd_s, dir_status;
412   long stop, pointer;
413   struct stat mytar_state;
414
415
416   if ( (fd_s = open(mt_name[0],O_RDONLY)) == -1)   {
417     perror("open");
418     return -1;
```

```
  }

  if ( stat(mt_name[0], &mytar_state) == -1)  {
    perror("stat");
    return -1;
  }


  pointer =0 ;
  stop = mytar_state.st_size;

  if ( inst.mytar_options[O] ) {

    if( chdir(inst.output_directory) == -1) {
      fprintf(stderr,"Error en directorio de salida\n");
      perror("chdir");

      return -1;
    }
  }

  if (inst.mytar_options[T]){
    fprintf(stdout,".mytar total size: %ld\n",stop);
  }

  while(pointer != stop) {
    pointer = gatherFields(fd_s, inst);
  }

  close(fd_s);

  return 0;
}
```