Outils.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdint.h>
#include <fcntl.h>
#include <time.h>
#include <string.h>
#include "outils.h"
#include <dirent.h>
#include <errno.h>
#include <stdbool.h>
#include <math.h>

struct dateModif{
  int day;
  int month;
  int year;
  int hour;
  int minute;
  int second;
};

/*fonction pour concatener des chainde ce char*/
void addToString(char * dst, char * src)
{
  int i;
  int j = 0;
  for(i = 0; i < strlen(src) - 1; i++){

    dst[i+strlen(dst)] = src[i];
  }
  src[i] = '\0';

}

/*Prend en parametre un fichier et renvoie sa taille*/

int size(char * file)
{
  int fd = open(file, O_RDWR);
  int i = lseek(fd, 0, SEEK_END);
  close(fd);
  return i;
}

/*fonction qui renvoie les droits d'un fichier*/
int permissionOfFile(char * file)
{
  struct stat buf;
  stat(file, &buf);
  printf("octal : %o\n", buf.st_mode & 0777);
  int i = buf.st_mode & 0777;
  return i;
}
```

```c
/*Enregistre dans *date , la date de derniere modification */
void modifiedDate(char * file, char * date)
{
  struct stat b;
  char t[100];
  if(!stat(file, &b)){
    strftime(t, 100, "%d/%m/%Y %H:%M:%S", localtime( &b.st_mtime));
  }
  strcpy(date, t);
}

/*fonction qui rajoute un ficher à la fin d'un autre fichier dst*/

void addTo(char * file, char * dst)
{
  int fd1 = open(file, O_RDONLY);
  int fd2 = open(dst, O_WRONLY | O_CREAT, 0666);
  char c;
  lseek(fd2, 0, SEEK_END);
  while(read(fd1, &c, 1) != 0)
    write(fd2, &c, 1);
  close(fd1);
  close(fd2);
}

/*fonction pour concatener de fichier dans un troisieme ficher dst*/

void append(char * file1, char * file2, char * dst)
{
  int fd1 = open(file1, O_RDONLY);
  int fd2 = open(file2, O_RDONLY);
  int fd3 = open(dst, O_WRONLY | O_CREAT, 0666);
  char c;
  while(read(fd1, &c, 1) != 0)
    write(fd3, &c, 1);
  while(read(fd2, &c, 1) != 0)
    write(fd3, &c, 1);
  close(fd1);
  close(fd2);
  close(fd3);
}

/*fonction qui rajoute le header du ficher src dans le fichier dst*/
void addHeader(char * src, char * dst)
{
  int fdSrc = open(src, O_RDONLY);
  int fdDst = open(dst, O_WRONLY | O_CREAT, 0666);
  char fileDate[30];
  int fileSize = size(src);
  int i = 0;
  char c;
  lseek(fdDst, 0, SEEK_END);
  /*écrire le nom du ficher dans l'archive*/
  c = '+';
  for(i = 0; i < 3; i++)
    write(fdDst, &c, 1);
  write(fdDst, src, strlen(src));

  /*rajouter des étoiles apres le nom*/
  c = '$';
  for(i = 0; i < 3; i++)
    write(fdDst, &c, 1);
```

```
  /*écrire la taille du fichier dans l'archive*/
  write(fdDst, &fileSize, sizeof(int));
 /*rajouter des étoiles apres le nom*/
  c = ';';
  for(i = 0; i < 3; i++)
    write(fdDst, &c, 1);

  /*écrire la date de derniere modification dans l'archive */
  modifiedDate(src, fileDate);
  i = 0;
  write(fdDst, fileDate, strlen(fileDate));
  c = '*';
  for(i = 0; i < 3; i++)
    write(fdDst, &c, 1);
  close(fdDst);
  close(fdSrc);
}

/*crée une archive à partir de 2 fichiers*/

void makeArchive(char * file1, char * file2, char * archive)
{
  addHeader(file1, archive);
  addTo(file1, archive);
  addHeader(file2, archive);
  addTo(file2, archive);
}

/*fonction qui renvoie la position d'un fichier dans l'archive*/
int positionFile(char * file, char * archive)
{
  int count = 0;
  int fd = open(archive, O_RDONLY);
  char c;
  int i = 0;
  int counter = 0;
  int notFound = 0;

  /*parcourir jusqu'à la fin du fichier*/

  while(read(fd, &c, 1) > 0){
    counter++;
    if(c == '+'){  /*'+':l'entete du nom du fichier*/
      counter += 2; /*le nom du fichier est precedé par 2 étoiles*/
      lseek(fd, counter, SEEK_SET);/*avancer le le curseur du fichier jusqu'au nom*/

      /*boucle pour parcourir le nom du fichier*/
      while(!notFound){ // tant que notFound est false
            read(fd, &c, 1);
            counter++;
            /*si la lettre lu dans "file" diferente du caractere lu dans le fichier*/
            if(file[i] != c)
              notFound = 1;
            if(c  == '$'){
              if(i == strlen(file)){
                return (counter - strlen(file) - 4);
              }
            }
            i++;
      }
      i = 0;
      notFound = 0;
    }
```

```
  }
  return -1;
}

/*Verifie si le nom passé en parametre correspond à un répértoire*/
int ifFolder(char * directory)
{
  DIR *rep;
  struct dirent *file;
  rep = opendir(directory);
  if(rep == NULL)
    return -1;
  file = readdir(rep);
  file = readdir(rep);
  char * filex = "fichier/file2";
  int fd = open(filex, O_WRONLY);
  printf("nom du fichier : %s\n", file->d_name);
  if(fd == -1)
    printf("erreur ouverture fichier\n");
  printf("ouverture reussi, valeur de fd : %d\n", fd);
  char c = '+';
  write(fd,&c,1);

  close(fd);
  return 0;

}

/*Compte le nombre de fichier dans un répértoire*/
int filesCounter(char * directory)
{
  int file_count = 0;
  DIR * dirp;
  struct dirent * file;
  dirp = opendir(directory);
  while ((file = readdir(dirp)) != NULL) {
    if ((file->d_type == DT_REG || file->d_type == DT_DIR) && file->d_type != DT_CHR) {
        file_count++;
    }
  }
  closedir(dirp);
  return file_count - 2;
}

/*Renvoie un tableau de chaine de caractere qui contient les noms des fichiers d'un repertoire*/
char ** filesInFolder(char * directory)
{
  DIR *rep;
  struct dirent *file;
  int numberFiles = filesCounter(directory);
  char ** files = malloc(numberFiles*sizeof(char *));
  char buffer[20];
  rep = opendir(directory);
  int i = 0;
  while ((file = readdir(rep))){
    sprintf(buffer,"%s",file->d_name);
    if(strcmp(buffer, ".") != 0 && strcmp(buffer, "..") != 0){
      files[i] = malloc(strlen(file->d_name) * sizeof(char));
      strcpy(files[i], file->d_name);
      i++;
    }
  }
```

```
  return files;
}

/*Renvoie le nom du prochain fichier dans l'archive et place le curseur juste après le nom*/

char * getNextFileName(int fd)
{
  char c;
  char * name = malloc(25 * sizeof(char));
  int i = 0;
  int found = 0;
  while(read(fd, &c, 1) > 0 && found == 0){
    if(c == '+'){
      found = 1;
      do{
            read(fd, &c, 1);
      }while(c == '+');
      do{
            name[i] = c;
            i++;
      }while(read(fd,&c,1) && c != '$');
    }
  }
  if(found == 1){
    name[i] = '\0';
    return name;
  }
  return NULL;
}

/*Renvoie la taille du prochain fichier trouvé dans l'archive, et place le curseur juste apres la taille*/

int getNextFileSize(int fd)
{
  char c;
  bool found = false;
  int size;
  while(read(fd, &c, 1) > 0 && found == false){
    if(c == '$'){
      do{
            read(fd, &c, 1);
      }while(c == '$');
      found = true;
      lseek(fd, -1, SEEK_CUR);
      read(fd, &size, sizeof(int));

    }
  }
  if(found == false)
    return -1;
  return size;
}


/*li le nombre d'octets n, et les transforme en entier*/
int readAndCast(int fd, int n)
{
  int i, j, tmp;
  char c;
  int res = 0;
  int puissance;
  for(i = 1; i <= n; i++){
    puissance = 1;
```

```
    read(fd, &c, 1);
    tmp = c - '0';
    for(j = 0; j < (n-i);j++)
      puissance = puissance * 10;
    res += puissance*tmp;

  }
  return res;
}


/*Renvoie une structure date, qui contient la date de derniere modification d'un fichier dans une archive, elle ne prend pas un nom de fichier en parametre, elle prend le premier fichier trouvé dans l'archive*/
dateModif lastModifedArchive(int fd)
{
  char c;
  dateModif date = malloc(sizeof(struct dateModif));
  bool found = false;
  int tmp;

  while(read(fd, &c, 1) > 0 && found == false){
    if(c == ';'){
      found = true;
      do{
            read(fd, &c, 1);
      }while(c ==';');
      /* lecture de du jour de modification*/
      lseek(fd, -1, SEEK_CUR);
      date->day = readAndCast(fd, 2);
      read(fd, &c, 1); //lecture des '/'
      /*lecture du mois de modification*/
      date->month = readAndCast(fd, 2);
      read(fd, &c, 1);//lecture des '/'
      /*lecture de l'année de modification*/
      date->year = readAndCast(fd, 4);
      read(fd,&c,1);
      /*lecture de l'heure de modification */
      date->hour = readAndCast(fd, 2);
      read(fd,&c,1);
      /*lecture de la minute de modification */
      date->minute = readAndCast(fd, 2);
      read(fd,&c,1);
      /*lecture de la seconde de modification */
      date->second = readAndCast(fd, 2);
      return date;
    }
  }
}

/*renvoie le contenu du premier fichier trouvé dans une archive*/
char * readNextFile(int fd, int size)
{
  char * file = malloc(size * sizeof(int));
  char c;
  int i;
  do{
    read(fd, &c, 1);
  }while(c == '*');
  file[0] = c;
  for(i = 1; i < size; i++){
    read(fd, &c, 1);
    file[i] = c;
  }
```

```
  file[i] = '\0';
  return file;
}


/*Supprime un fichier d'une archive, renvoie -1 si le fichier n'existe pas dans l'archive, et 1 sinon*/
int deleteFileArchive(char * archive, char * file)
{
  int fdA = open(archive, O_RDWR);
  int filePosition; /*la position du fichier qu'on veut supprimer*/
  int fdTmp = open("tmpFile", O_RDWR | O_CREAT , 0666);
  char c;
  char * f;
  char * buff; /*la chaine qui suit le fichier qu'on veut supprimer*/
  int taille, tmp, i;
  int counter = 0;
  int rd;
  int fileAfter = 0;
  int nil = 0;int found = 0;
  i = 0;
  do {
    f = getNextFileName(fdA);
    if(f != NULL)
      if(strcmp(f, file) == 0)
            found = 1;
      else
            nil = 1;
  }while(nil == 0 && found == 0);
  printf("nil : %d, foud : %d", nil, found);
  if(nil == 0 && found == 1)
  {
    printf("entrée condition\n");
    do{
      read(fdA, &c, 1);
      lseek(fdA,-2, SEEK_CUR);
    }while(c != '+');
    filePosition = lseek(fdA, -1, SEEK_CUR);
    lseek(fdA, 3, SEEK_CUR);
    while(read(fdA, &c, 1) > 0){
      if(c == '+')
            break;
    }
    if(c ==  '+'){
      tmp = lseek(fdA, 0, SEEK_CUR);
      taille = lseek(fdA, 0, SEEK_END) - tmp;
      lseek(fdA, (tmp - 1), SEEK_SET);
      while(read(fdA, &c ,1) > 0){
            write(fdTmp, &c, 1);
      }
      lseek(fdA, filePosition, SEEK_SET);
      lseek(fdTmp, 0, SEEK_SET);
      while(read(fdTmp, &c, 1) > 0){
            write(fdA, &c, 1);
      }
      truncate(archive, (filePosition + taille));
    }
    else
      truncate(archive, filePosition);
    close(fdTmp);
    unlink("tmpFile");
    close(fdA);
    return 0;
  }
```

```
  close(fdTmp);
  unlink("tmpFile");
  close(fdA);
  return -1;
}
```