```
#include "File_IO.h"
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <unistd.h>

#define RAISE 100;

void addCharToHistogram(Histogram *h, unsigned int index, char c);
void addCursorToHistogram(Histogram *h, unsigned int index, unsigned int cursor)
;

Histogram* readFileFromTo(FILE *datei, const unsigned int from, const unsigned i
nt to, Histogram *h, unsigned int *size);
size_t readMemory(char *ptrStart/*, char *buffer*/, size_t maxBytes);

/**
 * FÃ¼gt ein Charakter dem Histogram an einer Position hinzu.
 *Â @param h Zeiger auf das Histogram.
 * @param pos Index welches Histogram geÃ¤ndert werden soll.
 * @param c Charakter das dem Histogram hinzugefÃ¼gt werden soll.
 */
void addCharToHistogram(Histogram *h, unsigned int index, char c) {
    // kleinbuchstaben
    if (c >= 'a' && c <= 'z') {
        h[index].letter[c - 97 + 26]++;
    }

    // groÃM-^_buchstaben
    if (c >= 'A' && c <= 'Z') {
        h[index].letter[c - 65]++;
    }
}

/**
 * FÃ¼gt die Cursor Position in der Datei dem Histogram hinzu.
 * @param h
 * @param index
 * @param cursor
 */
void addCursorToHistogram(Histogram *h, unsigned int index, unsigned int cursor)
 {
    //h[index].cursor = (int*) malloc (sizeof(int));
    h[index].cursor = cursor;
    //printf("h[%d].cursor = %d\n", index, h[index].cursor);
}

/**
 * Initialisiert ein Histogram.
 */
void init(Histogram* h, unsigned int index) {
    h[index].cursor = -1;
    short i;
    for (i = 0; i < 52; i++) {
        h[index].letter[i] = 0;
    }
}

/**
 * Liest den Inhalt einer Datei ein innerhalb eines bestimmten Bereiches.
 * @param datei Pointer auf die Datei die gelesen werden soll.
 * @param from Position an der das erste Zeichen gelesen wird.
 * @param to Position an der das letzte Zeichen gelesen wird.
 * @return int The total Number of Histogramme
*/
Histogram* readFileFromTo(FILE *datei, const unsigned int from, const unsigned i
nt to, Histogram *h, unsigned int *size) {
    //printf("call readFileFromTo()\n");
    fseek(datei, from, SEEK_SET);

    *size = RAISE;
    //printf("size = %d\n", *size);

    h = (Histogram*) malloc (sizeof(Histogram) * (*size));
    unsigned int index = 0; // ZÃ¤hle die Histogramme hoch, nach jedem Zeilenend
e.
    char c;
    init(h, index);
    //h[index].letter = (char) malloc (sizeof(char) * 52);

    // Erste Cursor position ist der Wert von From!
    addCursorToHistogram(h, index, from);

    bool changeStateOfHistogram = FALSE;

    do { // Lese Zeichen bis \n entdeckt wird

        c = fgetc(datei); // Lese Zeichen
        if (isalpha(c) != 0) {
            //if (index == 100)
            //printf("Lese Zeichen: %c\n", c);
            // Wenn kein Zeilenumbruch dann FÃ¼ge Zeichen dem Histogram hinzu.
            // Jede Zeile ist ein eigenes Histogram
            addCharToHistogram(h, index, c);
            if (changeStateOfHistogram == FALSE) {
                changeStateOfHistogram = TRUE;
            }
            continue;
        }

        if (c == '\n') {
            //if (index == 100)
            //printHistogramStruct(h, 100);
            // Wenn c ein Zeilenumbruch ist
            if (changeStateOfHistogram == TRUE) {
                index++;
                changeStateOfHistogram = FALSE;
                if (index >= *size) {
                    *size += RAISE; // ErhÃ¶he den Speicher um x
                    h = realloc(h, (*size)*sizeof(Histogram));
                }
                init(h, index);
            }

            // FÃ¼ge Cursor dem Histogram hinzu
            addCursorToHistogram(h, index, ftell(datei));
        }

    } while (( (unsigned) ftell(datei) <= to && c != -1));

    //printf("index = %d\n", index);
    //printf("size = %d\n", *size);
```

```
    if ( (index+1) <= (*size) ) {
        *size = index;
        //printf("Reduziere mit realloc() size = %d\n", index+1);
        h = realloc(h, (index)*sizeof(Histogram)); // Reduziere ggf. zu viel all
okierten Speicher.
    }



    return h;
}

/**
 * Liest den Inhalt einer Datei ein, abhÃ¤ngig von dem aktuellen Rang des Prozes
ses und der Gesamtzahl der Prozesse.
 * @param myrank Der Rang dieses Prozesses.
 * @param numRank Die Gesammtzahl der Prozesse.
*/
Histogram* readFile(const char* filename, const int myRank, const int numRank, H
istogram *h, unsigned int *size) {
    FILE *datei;

   /* Bitte Pfad und Dateinamen anpassen */
   datei = fopen(filename, "r");
   if(NULL == datei) {
      printf("Konnte Datei %s nicht Ã¶ffnen!\n", filename);
      return EXIT_SUCCESS;
   }
    //fseek(datei, 0L, SEEK_SET); // Cursor an den Anfang setzen.
    //printf("Anfang in Datei: %d\n",ftell(datei));

    /* Bestimme die LÃ¤nge der Datei */
    fseek(datei, 0L, SEEK_END); // Setze den Cursor ans Ende der Datei
    //printf("Ende in Datei: %d \n",ftell(datei));

    unsigned int length = ftell(datei);

    //printf("Size of File: %d\n",length);

    /* Teile die LÃ¤nge der Datei durch die Anzahl der Prozesse */
    unsigned int devided = length / numRank;

    /* Berechne From */
    unsigned int from = myRank * devided;

    /* Berechne To */
    unsigned int to = ((myRank+1) * devided);

    /* Wenn das nicht der erste Prozess ist */
    if (myRank != 0) {
        // Bestimme die korrekte From position.
        fseek(datei, from, SEEK_SET);

        // Lese Zeichen bis Zeilenende
        char c;
        while( (c = fgetc(datei)) != '\n')
         ;

        // after line break has found, the next char is the first char in next l
ine
        // new from position is from here
```



```
        from = ftell(datei);
    }

    // 6) if this is not the last prozess
    // determine real to position
    if (myRank != (numRank-1)) {
        fseek(datei, to, SEEK_SET);

        // read chars while line break has found
        char c;
        while( (c = fgetc(datei)) != '\n')
         ;

        // after line break has found, the next char is the first char in next l
ine
        // go back to before the line break.
        to = ftell(datei);
        to--;
    } else {
        fseek(datei, 0, SEEK_END);
        to = ftell(datei);
    }
    /*
    printf("Prozess: %d von %d \n", myRank, numRank);
    printf("From: %d \n", from);
    printf("TO: %d \n", to);
    */
    h = readFileFromTo(datei, from, to, h, size);
    //printf("%d Elements in HistogramArray.\n",*size);

    fclose(datei);

    return h;
}


int writeFile(const char *filename_out, const char *filename_in, Histogram **h,
unsigned int *size) {
    FILE *out;
    FILE *in;

    out = fopen(filename_out, "wt");
    in = fopen(filename_in, "r");

  if(NULL == out) {
    printf("Konnte Datei %s nicht Ã¶ffnen!\n", filename_out);
    return EXIT_FAILURE;
  }

    if (NULL == in) {
    printf("Konnte Datei %s nicht oeffnen!\n", filename_in);
        return EXIT_FAILURE;
    }

  char zeile[126];
  unsigned int i;
  int cursor;
    for (i = 0; i < *size; i++) {
        // Hole cursor wo das original wort steht:
        cursor = (*h[i]).cursor;

    // Setze Cursor an position
```

```
        fseek(in, cursor, SEEK_SET);
        // lese zeichen bis Zeilenende
        fgets(zeile, 126, in);

        fputs(zeile, out);
    }
    fclose(out);
    fclose(in);

    return 0;
}

int writeFileFromMemory(const char *filename_out, const char *fOrigin, Histogram
 **h, unsigned int *size) {
  FILE *datei_left = fopen("sortMe_left.txt", "r");

  FILE *datei_right = fopen("sortMe_right.txt", "r");

  // original datei
  FILE *origin = fopen(fOrigin, "r");

  fseek(datei_left, 0L, SEEK_END);

    // Bestimme die Laenge der linken Datei in Bytes
    unsigned int length_left = ftell(datei_left);

    fseek(datei_left, length_left, SEEK_SET); // setze den cursor wieder an den
anfang

    // Bestimme die LÃ¤nge der original Datei
    fseek(origin, 0L, SEEK_END);
    unsigned int length_origin = ftell(origin);

    // Berechne die realitve LÃ¤nge von der rechten Datei
    unsigned int length_right = length_origin - length_left;

    //unsigned int left_start = 0;
    //unsigned int left_end = length_left;

    //unsigned int right_start = left_end+1;
    //unsigned int right_end = length_origin;

    /*
    char c;
    // gehe nun bis nÃ¤chstes newline gefunden wurde
    while( (c = fgetc(datei)) != '\n') {
      ;
    }

    length_left = ftell(datei)-1;
    fclose(datei);
    */
    //unsigned length_right = length_left+1;
    /*

        // after line break has found, the next char is the first char in next l
ine
        // go back to before the line break.
        to = ftell(datei);

    fd = open (filename, O_RDWR, S_IRUSR | S_IWUSR);
```



```
  file_in = mmap (0, length, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);*/
  printf("length_left: %d\n", length_left);
  printf("length_right: %d\n", length_right);

  unsigned int i;
  unsigned int cursor;
  bool isLeftMapped = FALSE;
  bool isRightMapped = FALSE;

  size_t num = 0;
  char zeile[127];

  int f_left = open ("sortMe_left.txt", O_RDWR, S_IRUSR | S_IWUSR);
  int f_right = open ("sortMe_right.txt", O_RDWR, S_IRUSR | S_IWUSR);

  char* file_memory_left;
  char* file_memory_right;
  char* file_memory;

  fclose(datei_left);
  fclose(datei_right);
  fclose(origin);

  //size_t page_size = (size_t) sysconf (_SC_PAGESIZE);

  // TEST
  /*
  file_memory = mmap (0, length_left, PROT_READ | PROT_WRITE, MAP_SHARED, f_left
, 0);
  int unmap_result = munmap (file_memory, length_left);
  //close(f_left);
  file_memory = mmap (0, length_right, PROT_READ | PROT_WRITE, MAP_SHARED, f_rig
ht, 0);
  unmap_result = munmap (file_memory, length_right);
  file_memory = mmap (0, length_left, PROT_READ | PROT_WRITE, MAP_SHARED, f_left
, 0);
  */
  FILE *out = fopen(filename_out, "wt");


  char c = '\n';
  for (i = 0; i < *size; i++) {
        // Hole cursor wo das original wort steht:
        cursor = (*h[i]).cursor;
        //printf("cursor: %d ",cursor);

        if (cursor <= length_left) {
          //printf(" left\n");
          if (isRightMapped == TRUE) {
            // unmapp right
            munmap(file_memory, length_right);
            isRightMapped = FALSE;
          }

          if (isLeftMapped == FALSE) {
            // mappe linke seite
            file_memory = mmap (0, length_left, PROT_READ | PROT_WRITE, MAP_SHAR
ED, f_left, 0);
            isLeftMapped = TRUE;
          }
        } else {
          if (isLeftMapped == TRUE) {
```

```
            // unmapp left
            munmap (file_memory, length_left);
            isLeftMapped = FALSE;
          }

          if (isRightMapped == FALSE) {
            // mappe rechte seite
            file_memory = mmap (0, length_right, PROT_READ | PROT_WRITE, MAP_SHA
RED, f_right, 0);
            cursor = length_origin - cursor;
            isRightMapped = TRUE;
          }
        }

        num = readMemory(file_memory+cursor, 127);
        // Der Cursor muss fÃ¼r die rechte seite ja auch auf die 1GB File passen
, daher muss der Cursor angepasst werden
        fwrite (file_memory+cursor, 1 , num , out );
        fwrite (&c, 1, 1, out);
    }

    fclose(out);
    close(f_left);
    close(f_right);

    if (isLeftMapped == TRUE) {
      // unmapp left
        munmap (file_memory, length_left);
  }

  if (isRightMapped == TRUE) {
    // unmapp right
    munmap (file_memory, length_right);
    }

  return 0;
}
/*
void mmapTry(const char* filename) {

    FILE *datei;
   // Bitte Pfad und Dateinamen anpassen
   datei = fopen(filename, "r");

  void* file_in;
  void* file_out;
  int fd, fo;

  fseek(datei, 0L, SEEK_END); // Setze den Cursor ans Ende der Datei
    //printf("Ende in Datei: %d \n",ftell(datei));

    unsigned int length = ftell(datei);
    fclose(datei);

    fd = open (filename, O_RDWR, S_IRUSR | S_IWUSR);
    fo = open ("out.txt", O_RDWR, S_IRUSR | S_IWUSR);

  file_in = mmap (0, length, PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
  file_out = mmap (0, length, PROT_READ, MAP_SHARED, fo, 0);

  close (fd);
  close (fo);
```



```
  char *zeile = (char*) malloc (sizeof(char)*126);
  int integer;
  //sscanf (file_memory+4, "%d", &integer);
  //printf ("value: %d\n", integer);
  //sscanf (file_memory+0, "%s", zeile);
  sscanf(file_in, "%[^\n]", zeile);
  printf ("value: %s\n", zeile);

  char *c = (char*) malloc (sizeof(char));
  int index = 0;

  do {
    // Lese Zeichen bis \n erkannt wurde
    sscanf(file_memory+index, "%c", c);
    if (*c != '\n') {
      printf("%c",*c);
    }
    index++;
  } while (*c != '\n');

  //printf("\n%c\n",file_memory[1]);

  munmap (file_in, length);
}*/
/*
int sumAllLetters(unsigned int index, Histogram **h) {
  int i;
  int sum_letter;
  int total_sum = 0;

  for (i = 0; i < 26; i++) {
    sum_letter = getSumOfLetterCaseInsensitive(i, index, h);
    total_sum += sum_letter;
  }

  return total_sum;
}
*/


size_t readMemory(char *ptrStart/*, char *buffer*/, size_t maxBytes) {
   size_t bytesRead = 0;

   while (bytesRead < maxBytes - 1 && ptrStart[bytesRead] != '\n') {
       //buffer[bytesRead] = ptrStart[bytesRead];
       bytesRead++;
   }
   //buffer[bytesRead] = '\n';
   //buffer[bytesRead + 1] = '\0';

   return bytesRead;
}
```