1 Stückweises Lesen einer Datei

Implementieren Sie `read_file` im folgenden Programm so, dass der übergebene Dateideskriptor `fd` vollständig gelesen wird, wobei Ihnen nur ein einzelner Integer (`buf`) als Puffer zur Verfügung steht. Erweitern Sie die Funktion um Code, der immer einen einzelnen Integer-Wert liest, bis er am Ende der Datei angekommen ist. Werte ungleich `0` sollen dabei zusammen mit ihrem Offset in der Datei ausgegeben werden.

```c
#include <assert.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

void read_file(int fd) {
  int buf;

  // TODO
  printf("int=%d at offset=%lu\n", buf, 0);
}

int main(void) {
  int fd;

  fd = open("myfile", O_RDWR | O_CREAT, 0600);
  assert(fd != -1);
  assert(pwrite(fd, &fd, sizeof(fd), 1048576) == sizeof(fd));
  read_file(fd);
  assert(pwrite(fd, &fd, sizeof(fd), 1024) == sizeof(fd));
  read_file(fd);
  assert(pwrite(fd, &fd, sizeof(fd), 524288) == sizeof(fd));
  read_file(fd);
  assert(close(fd) == 0);
  assert(unlink("myfile") == 0);

  return 0;
}
```

2 Persistente Datenstrukturen

Das folgende Programm implementiert eine Array-Datenstruktur, die in einem per `malloc` allokierten Speicherbereich verwaltet wird. Passen Sie das Programm so an, dass die Array-Datenstruktur mithilfe von `mmap` in einer Datei `myarray` verwaltet wird. `array_init` soll dabei den vorherigen Zustand aus der Datei wiederherstellen, sofern sie bereits vorhanden ist. Die restlichen Funktionen sollen sich wie bisher verhalten.

```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

struct array_element {
  int age;
  char name[256];
};

struct array {
  off_t size;
  off_t count;
  struct array_element element[];
};

void array_init(struct array *arr, off_t size) {
  arr->size = size;
  arr->count = 0;
}

void array_reset(struct array *arr) {
  arr->count = 0;
}

void array_append(struct array *arr, int age, char *name) {
  if ((sizeof(*arr) + sizeof(struct array_element) * (arr->count
   ↪  + 1)) > arr->size) {
    return;
  }

  arr->element[arr->count].age = age;
  strncpy(arr->element[arr->count].name, name, 256);
  arr->count++;
}

struct array_element *array_get(struct array *arr, off_t index) {
  if (index >= arr->count) {
    return NULL;
```

```
  }

  return &arr->element[index];
}

void array_print(struct array *arr) {
  for (off_t i = 0; i < arr->count; i++) {
    struct array_element *e = array_get(arr, i);
    printf("arr[%lu] = { %d, %s }\n", i, e->age, e->name);
  }
}

int main(void) {
  struct array *arr;

  arr = malloc(1024);

  array_init(arr, 1024);
  array_print(arr);

  array_reset(arr);
  array_append(arr, 18, "Max Mustermann");
  array_append(arr, 21, "Moritz Mustermann");
  array_append(arr, 18, "Petra Mustermann");
  array_append(arr, 21, "Paula Mustermann");
  array_print(arr);

  array_reset(arr);
  array_append(arr, 21, "Paula Mustermann");
  array_print(arr);

  free(arr);

  return 0;
}
```

Material

Makefile und Code-Vorlagen liegen dem beiliegenden Übungsmaterial bei. Sie können alle Binaries automatisch mittels `make` oder gezielt mit `make BINARYNAME` bauen lassen.