**Code Structure**

The code is organized in the following manner:

git/

└── assign1/

├── README.txt

├── dberror.c

├── dberror.h

├── storage\_mgr.c

├── storage\_mgr.h

├── test\_assign1\_1.c

├── test\_helper.h

└── Makefile

**Key Components**

* **dberror.h**: Contains error codes used throughout the implementation.
* **storage\_mgr.h**: Defines the interface for the storage manager and the data structures used.
* **storage\_mgr.c**: Implements the functions defined in storage\_mgr.h.
* **test\_assign1\_1.c**: Contains test cases to validate the functionality of the storage manager.
* **test\_helper.h**: Provides utility functions for testing purposes.
* **Makefile**: Used to compile the code and create an executable for running tests.
* **README.txt**: Describes the solution and code structure.

**Implementation Details**

**1. Header File: storage\_mgr.h**

#ifndef STORAGE\_MGR\_H

#define STORAGE\_MGR\_H

#include "dberror.h"

/\* handle data structures \*/

typedef struct SM\_FileHandle {

char \*fileName; // Name of the file

int totalNumPages; // Total number of pages in the file

int curPagePos; // Current page position

void \*mgmtInfo; // Management info (e.g., file pointer)

} SM\_FileHandle;

typedef char \*SM\_PageHandle; // Pointer to memory for storing page data

/\* interface \*/

extern void initStorageManager(void);

extern RC createPageFile(char \*fileName);

extern RC openPageFile(char \*fileName, SM\_FileHandle \*fHandle);

extern RC closePageFile(SM\_FileHandle \*fHandle);

extern RC destroyPageFile(char \*fileName);

extern RC readBlock(int pageNum, SM\_FileHandle \*fHandle, SM\_PageHandle memPage);

extern int getBlockPos(SM\_FileHandle \*fHandle);

extern RC readFirstBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage);

extern RC readPreviousBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage);

extern RC readCurrentBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage);

extern RC readNextBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage);

extern RC readLastBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage);

extern RC writeBlock(int pageNum, SM\_FileHandle \*fHandle, SM\_PageHandle memPage);

extern RC writeCurrentBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage);

extern RC appendEmptyBlock(SM\_FileHandle \*fHandle);

extern RC ensureCapacity(int numberOfPages, SM\_FileHandle \*fHandle);

#endif

**2. Implementation File: storage\_mgr.c**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "storage\_mgr.h"

#include "dberror.h"

#define PAGE\_SIZE 4096 // Define the page size

// Initialize the storage manager

void initStorageManager(void) {

// Initialization logic can be added here if needed

}

// Create a new page file

RC createPageFile(char \*fileName) {

FILE \*file = fopen(fileName, "wb");

if (!file) return RC\_FILE\_NOT\_FOUND;

// Write one empty page filled with '\0' bytes

char \*emptyPage = (char \*)calloc(PAGE\_SIZE, sizeof(char));

fwrite(emptyPage, sizeof(char), PAGE\_SIZE, file);

free(emptyPage);

// Write initial page count (1)

int initialPageCount = 1;

fseek(file, 0, SEEK\_SET);

fwrite(&initialPageCount, sizeof(int), 1, file);

fclose(file);

return RC\_OK;

}

// Open an existing page file

RC openPageFile(char \*fileName, SM\_FileHandle \*fHandle) {

FILE \*file = fopen(fileName, "rb");

if (!file) return RC\_FILE\_NOT\_FOUND;

fHandle->fileName = strdup(fileName);

fread(&(fHandle->totalNumPages), sizeof(int), 1, file);

fHandle->curPagePos = 0;

fHandle->mgmtInfo = file; // Store the file pointer

return RC\_OK;

}

// Close a page file

RC closePageFile(SM\_FileHandle \*fHandle) {

if (fHandle->mgmtInfo) {

fclose((FILE \*)fHandle->mgmtInfo);

free(fHandle->fileName);

fHandle->mgmtInfo = NULL;

}

return RC\_OK;

}

// Destroy a page file

RC destroyPageFile(char \*fileName) {

if (remove(fileName) != 0) return RC\_FILE\_NOT\_FOUND;

return RC\_OK;

}

// Read a block from a page file

RC readBlock(int pageNum, SM\_FileHandle \*fHandle, SM\_PageHandle memPage) {

if (pageNum < 0 || pageNum >= fHandle->totalNumPages) return RC\_READ\_NON\_EXISTING\_PAGE;

FILE \*file = (FILE \*)fHandle->mgmtInfo;

fseek(file, sizeof(int) + pageNum \* PAGE\_SIZE, SEEK\_SET);

fread(memPage, PAGE\_SIZE, 1, file);

fHandle->curPagePos = pageNum;

return RC\_OK;

}

// Get current block position

int getBlockPos(SM\_FileHandle \*fHandle) {

return fHandle->curPagePos;

}

// Read the first block

RC readFirstBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) {

return readBlock(0, fHandle, memPage);

}

// Read the last block

RC readLastBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) {

return readBlock(fHandle->totalNumPages - 1, fHandle, memPage);

}

// Read the previous block

RC readPreviousBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) {

return readBlock(fHandle->curPagePos - 1, fHandle, memPage);

}

// Read the current block

RC readCurrentBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) {

return readBlock(fHandle->curPagePos, fHandle, memPage);

}

// Read the next block

RC readNextBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) {

return readBlock(fHandle->curPagePos + 1, fHandle, memPage);

}

// Write a block to a page file

RC writeBlock(int pageNum, SM\_FileHandle \*fHandle, SM\_PageHandle memPage) {

if (pageNum < 0 || pageNum >= fHandle->totalNumPages) return RC\_WRITE\_FAILED;

FILE \*file = (FILE \*)fHandle->mgmtInfo;

fseek(file, sizeof(int) + pageNum \* PAGE\_SIZE, SEEK\_SET);

fwrite(memPage, PAGE\_SIZE, 1, file);

return RC\_OK;

}

// Write the current block

RC writeCurrentBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) {

return writeBlock(fHandle->curPagePos, fHandle, memPage);

}

// Append an empty block

RC appendEmptyBlock(SM\_FileHandle \*fHandle) {

fHandle->totalNumPages++;

FILE \*file = (FILE \*)fHandle->mgmtInfo;

fseek(file, 0, SEEK\_END); // Move to the end of the file

char \*emptyPage = (char \*)calloc(PAGE\_SIZE, sizeof(char));

fwrite(emptyPage, sizeof(char), PAGE\_SIZE, file);

free(emptyPage);

return RC\_OK;

}

// Ensure capacity for a specific number of pages

RC ensureCapacity(int numberOfPages, SM\_FileHandle \*fHandle) {

while (fHandle->totalNumPages < numberOfPages) {

appendEmptyBlock(fHandle);

}

return RC\_OK;

}

**3. Test File: test\_assign1\_1.c**

#include <stdio.h>

#include <stdlib.h>

#include "storage\_mgr.h"

#include "dberror.h"

void testStorageManager() {

SM\_FileHandle fHandle;

SM\_PageHandle memPage = (SM\_PageHandle)malloc(PAGE\_SIZE);

// Create a new page file

printf("Creating page file...\n");

createPageFile("testfile.bin");

// Open the created page file

printf("Opening page file...\n");

openPageFile("testfile.bin", &fHandle);

// Write to the current block

sprintf(memPage, "Hello, World!");

writeCurrentBlock(&fHandle, memPage);

// Read the current block

readCurrentBlock(&fHandle, memPage);

printf("Read from current block: %s\n", memPage);

// Clean up

closePageFile(&fHandle);

destroyPageFile("testfile.bin");

free(memPage);

}

int main() {

initStorageManager();

testStorageManager();

return 0;

}

**4. README.txt**

# Simple Storage Manager

## Description

This project implements a simple storage manager that allows for managing page files on disk. The storage manager supports basic operations such as creating, opening, and closing page files, as well as reading from and writing to specific pages.

## Code Structure

- \*\*storage\_mgr.h\*\*: Header file defining the storage manager interface and data structures.

- \*\*storage\_mgr.c\*\*: Implementation of the storage manager functions.

- \*\*dberror.h\*\*: Contains error codes used throughout the implementation.

- \*\*test\_assign1\_1.c\*\*: Contains test cases to validate the functionality of the storage manager.

- \*\*test\_helper.h\*\*: Provides utility functions for testing.

- \*\*Makefile\*\*: Used to compile the code and create an executable for running tests.

## Key Features

- Fixed-size pages (PAGE SIZE).

- Support for multiple file operations: create, open, close, destroy.

- Ability to read and write specific blocks/pages of data.

## Usage

To use the storage manager, include `storage\_mgr.h` in your application, initialize the storage manager, and call the desired functions to manipulate page files.

## Future Improvements

- Implement additional error handling and logging.

- Add support for more complex file operations.

**5. Makefile**

CC = gcc

CFLAGS = -Wall -g

OBJS = storage\_mgr.o dberror.o

TARGET = test\_assign1

all: $(TARGET)

$(TARGET): $(OBJS) test\_assign1\_1.o

$(CC) -o $(TARGET) $(OBJS) test\_assign1\_1.o

%.o: %.c

$(CC) $(CFLAGS) -c $<

clean:

rm -f $(TARGET) \*.o

**Testing**

The provided test file (test\_assign1\_1.c) includes a simple test case that creates a page file, writes to it, reads from it, and cleans up afterward. You can extend this with additional test cases to cover more functionalities.

Run the make command to compile the project, then execute the binary to run the tests:

make

./test\_assign1

This implementation provides a foundational structure for a simple storage manager, adhering to the specified interface and requirements. Further enhancements can be made based on specific use cases or performance needs.