Group No :
Ameya Hujare (A20545367)
Deep Pawar (A20545137)
Canyu Chen (A20479758)

Professor: Gerald Balekaki
Institute: Illinois Institute of Technology

CS 525: Advanced Database Organization

Spring 2025 - Assignment 1 - Store Manager

1. INTRODUCTION

The purpose of this assignment is to design a basic storage manager that can read and write fixed-size blocks (pages) from a file on disk to memory and vice versa. Additionally, the storage manager includes features for creating, opening, closing, and deleting files, as well as methods for accessing and modifying data at specific positions or in relation to the current page position.

2. OVERVIEW

The storage manager is implemented based on the interface provided in the storage_mgr.h header file. It follows the principles of modular programming by dividing the functionality into well-defined functions.

Key features:

- **File Management:** Creation, opening, closing, and deletion of files.
- **Page Management:** Reading and writing fixed-size pages.
- **Metadata Handling:** Tracks the total number of pages and current page position.
- **Error Handling:** Uses predefined return codes for debugging and error reporting.
- **Scalability:** Supports dynamic file expansion.

3. CODE STRUCTURE

3.1 Directory Layout

The project files are organized as follows:

```
assign1/
├── README.txt        # This is documentation file
├── dberror.c         # Contains Error handling functions
├── dberror.h         # Contains Header defining error codes and helper functions
├── storage_mgr.c     # Implementation of the storage manager
├── storage_mgr.h     # Contains Header defining the storage manager interface
├── test_assign1_1.c  # Test cases for the storage manager
├── test_helper.h     # Helper functions for tests
├── Makefile          # Build script to compile the program
```

3.2 Header Files

1. **storage_mgr.h**

   It defines the SM_FileHandle and SM_PageHandle structures, and specifies functions for efficient file and page management

2. **dberror.h**

   It defines error codes and provides the printError( ) function for debugging.

3. **test_helper.h**

   It provides macros and utilities for writing and debugging test cases.

4. IMPLEMENTATION DETAILS

4.1 Data Structures

1. **SM_PageHandle**

   It is a Pointer to a memory block representing a page.

2. **SM_FileHandle**

   It represents an open page file.

   Fields contains:

   - fileName: Name of the file.
   - totalNumPages: Total number of pages in the file.
   - curPagePos: Current page position for read/write operations.
   - mgmtInfo: Pointer to additional metadata, such as the file pointer or descriptor.

4.2 File Operations

1. **createPageFile(char *fileName)**

   It creates a new file containing a single page initialized with specific bytes, ensuring the file's size matches PAGE_SIZE.

2. **openPageFile(char *fileName, SM_FileHandle *fHandle)**

   It opens an existing file and initializes the file handle with its associated metadata and if the specified file is not found, the function returns RC_FILE_NOT_FOUND.

3. **closePageFile(SM_FileHandle *fHandle)**

   It closes an open file.

4. **destroyPageFile(char *fileName)**

   It deletes a file from the filesystem.

4.3 Page Operations

1. **writeCurrentBlock(SM_FileHandle fHandle, SM_PageHandle memPage)**

   Writes data to the currently active page.

2. **getBlockPos(SM_FileHandle fHandle)**

   Returns the current page position.

3. **readPreviousBlock(SM_FileHandle fHandle, SM_PageHandle memPage)**

   Reads the page before the current one.

4. **readCurrentBlock(SM_FileHandle fHandle, SM_PageHandle memPage)**

   Reads the current page.

5. **readLastBlock(SM_FileHandle fHandle, SM_PageHandle memPage)**

   Reads the last page of the file.

4.4 Error Handling

All functions return an RC (return code) value. Error codes are defined in dberror.h, including:

- RC_OK - No errors.
- RC_FILE_NOT_FOUND - Missing file.
- RC_READ_NON_EXISTING_PAGE - Attempt to read a non-existing page.

The printError function outputs error messages for debugging.

5. BUILD AND TEST INSTRUCTIONS

5.1 Build Process

The project uses a **Makefile** to automate compilation.

1. Navigate to the **assign1/** directory.
2. Run **make** to build the project.
3. The **test_assign1** executable will be generated.

5.2 Running Tests

1. Execute **./test_assign1** to run the test suite.
2. The test cases are defined in **test_assign1_1.c** and it verifies the functionality of the storage manager.

6. DESIGN CONSIDERATIONS

1. File Metadata

The beginning of the file includes reserved space for storing the total number of pages. Additionally, mgmtInfo is used to hold the file pointer, ensuring quick and efficient access.

2. Error Handling

Done by implementing robust checks for invalid inputs and operations.

3. Memory Management

It ensured that memory for SM_PageHandle is allocated and freed appropriately.

4. Scalability

We used appendEmptyBlock and ensureCapacity to dynamically grow the file size as needed.

7. Key Achievements

- **File Management:** Developed mechanisms for creating new files with an initial empty page, opening existing files while initializing metadata, closing open files, and deleting files when they are no longer needed.
- **Page Operations:** Implemented methods for reading and writing pages, both in absolute and relative terms. This includes accessing the first, last, previous, current, and next blocks, as well as writing to specific blocks.
- **Capacity Management:** Enabled dynamic management of file capacity by allowing the storage manager to append empty blocks and expand file size as required.

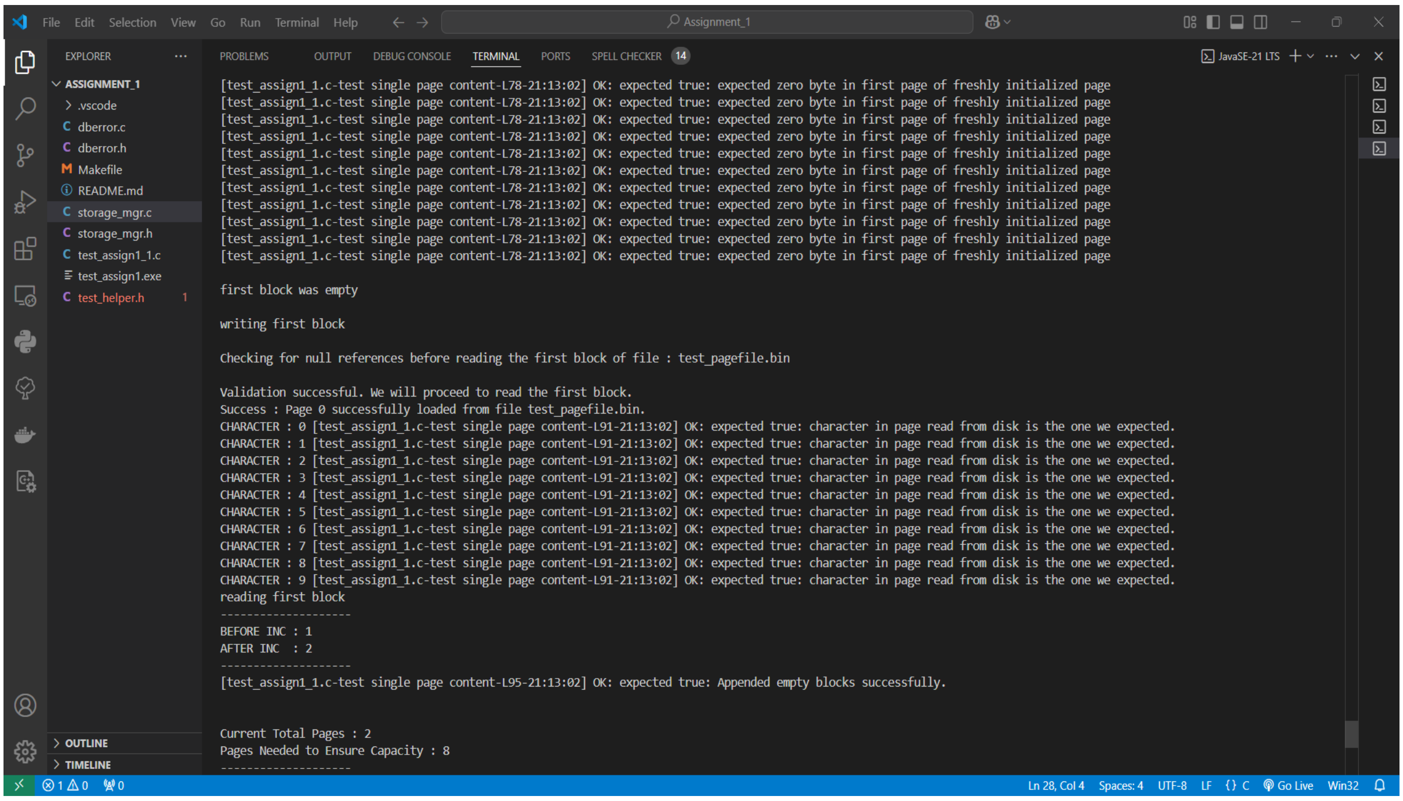
8. Testing and Validation

A comprehensive test suite (test_assign1_1.c) has been created to verify the functionality of the storage manager. These tests cover a range of scenarios, from fundamental file operations to complex page handling tasks. Additional tests are encouraged to further strengthen the system's reliability. Debugging and memory-checking tools have been employed to detect and resolve issues, ensuring a stable and dependable implementation.

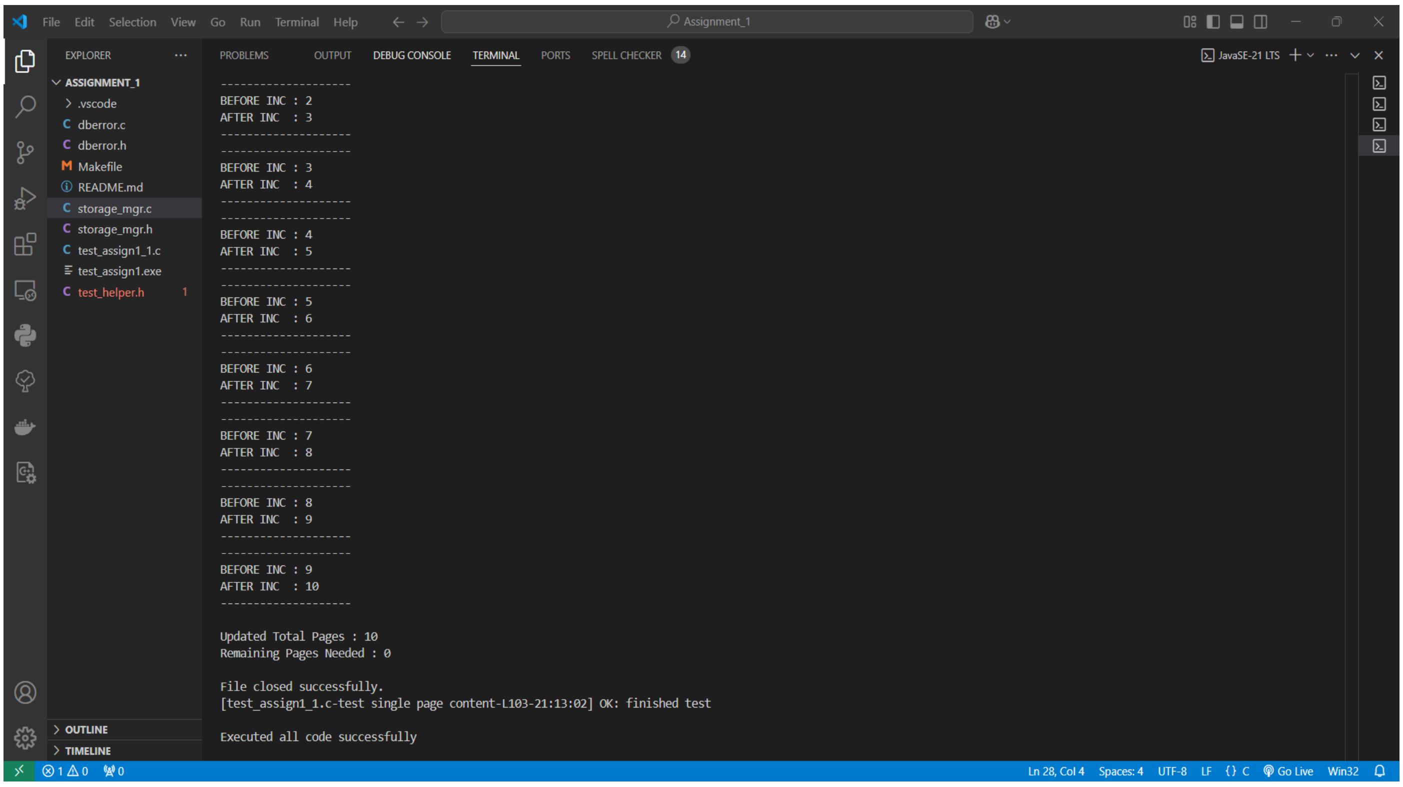
9. CONCLUSION

In this assignment, we have successfully implemented the storage manager which provides a robust and efficient solution for managing files and pages with fixed-size blocks. Its modular design ensures ease of maintenance and flexibility for future upgrades. By incorporating predefined error codes, the implementation enhances debugging efficiency and streamlines error handling. Furthermore, the system's functionality and reliability are thoroughly validated through extensive test cases.

10. OUTPUT



```
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page
[test_assign1_1.c-test single page content-L78-21:13:02] OK: expected true: expected zero byte in first page of freshly initialized page

first block was empty

writing first block

Checking for null references before reading the first block of file : test_pagefile.bin

Validation successful. We will proceed to read the first block.
Success : Page 0 successfully loaded from file test_pagefile.bin.
CHARACTER : 0 [test_assign1_1.c-test single page content-L91-21:13:02] OK: expected true: character in page read from disk is the one we expected.
CHARACTER : 1 [test_assign1_1.c-test single page content-L91-21:13:02] OK: expected true: character in page read from disk is the one we expected.
CHARACTER : 2 [test_assign1_1.c-test single page content-L91-21:13:02] OK: expected true: character in page read from disk is the one we expected.
CHARACTER : 3 [test_assign1_1.c-test single page content-L91-21:13:02] OK: expected true: character in page read from disk is the one we expected.
CHARACTER : 4 [test_assign1_1.c-test single page content-L91-21:13:02] OK: expected true: character in page read from disk is the one we expected.
CHARACTER : 5 [test_assign1_1.c-test single page content-L91-21:13:02] OK: expected true: character in page read from disk is the one we expected.
CHARACTER : 6 [test_assign1_1.c-test single page content-L91-21:13:02] OK: expected true: character in page read from disk is the one we expected.
CHARACTER : 7 [test_assign1_1.c-test single page content-L91-21:13:02] OK: expected true: character in page read from disk is the one we expected.
CHARACTER : 8 [test_assign1_1.c-test single page content-L91-21:13:02] OK: expected true: character in page read from disk is the one we expected.
CHARACTER : 9 [test_assign1_1.c-test single page content-L91-21:13:02] OK: expected true: character in page read from disk is the one we expected.
reading first block
-------------------
BEFORE INC : 1
AFTER INC  : 2
-------------------
[test_assign1_1.c-test single page content-L95-21:13:02] OK: expected true: Appended empty blocks successfully.


Current Total Pages : 2
Pages Needed to Ensure Capacity : 8
-------------------
```



```
-------------------
BEFORE INC : 2
AFTER INC  : 3
-------------------
-------------------
BEFORE INC : 3
AFTER INC  : 4
-------------------
-------------------
BEFORE INC : 4
AFTER INC  : 5
-------------------
-------------------
BEFORE INC : 5
AFTER INC  : 6
-------------------
-------------------
BEFORE INC : 6
AFTER INC  : 7
-------------------
-------------------
BEFORE INC : 7
AFTER INC  : 8
-------------------
-------------------
BEFORE INC : 8
AFTER INC  : 9
-------------------
-------------------
BEFORE INC : 9
AFTER INC  : 10
-------------------

Updated Total Pages : 10
Remaining Pages Needed : 0

File closed successfully.
[test_assign1_1.c-test single page content-L103-21:13:02] OK: finished test

Executed all code successfully
```