**Storage Manager**

Programming Assignment 1

**Team Composition:**

* Penchala Sainath Rachaputi
* Pradeep Kumar Golla Venu Gopal
* Preethi Chowdary Mandadi

**Files List:**

* dberror.h
* dberror.c
* test\_helper.h
* test\_assign1\_1.c
* storage\_mgr.h
* storage\_mgr.c
* MakeFile

**Process to follow for execution:**

Compile:

Open the terminal, then navigate to the contents of the dictionary.

Type:

gcc dberror.c storage\_mgr.c test\_assign1\_1.c -o PAssign1.out

To get the output

Type:

./PAssign.out

Using the MakeFile:

Navigate to the files directory

Type:

Make

To get the output

Type:

./PAssign.out

**Function Descriptions:**

***Manipulating the Page Files***

* **CreatePageFile(char \*fileName):** We Check if the file already exists. If the file exists, we return an error message syaing that the file is already exists. If the file doesn' t exist, then we create the file and allocate the size of one PAGE to it.
* **OpenPageFile(char \*fileName, SM\_FileHandle \*fHandle) :** We first check if the file with the given name exists or no. Then, If it doesn't exist, we return an error message. If the file exists, we check the total number of pages that the file has and store it in page number object. Also, after we open the file, we initiate the structure elements that are needed.
* **ClosePageFile(SM\_FileHandle \*fHandle) :** We close the file and return a success message if success i.e RC\_OK and if the file couldn't be located, we return RC\_FILE\_NOT\_FOUND error message.
* **DestroyPageFile(char \*fileName) :** We first check if the file is present or no, if it exists we remove the file. Upon success, return a success message. Upon failure, return a failure message.

***Writing blocks for the page file***

* **writeBlock(int pageNum, SM\_FileHandle \*fHandle, SM\_PageHandle memPage) :** We first check if the file is present or no and if it exists we get the current Position of the file. Then we write the contents to the file and then close the file.
* **appendEmptyBlock(SM\_FileHandle \*fHandle) :**  We first check if the file is present or no and if it exists wec check for the total number of pages in the file. We add one page and print'\0' in the empty block.
* **writeCurrentBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) :** We first check if the file is present or no and if it exists we write the current block based on absolute position.
* **ensureCapacity(int numberOfPages, SM\_FileHandle \*fHandle) :** We try to locate the specified file and if it doesn't exist we return an error message saying the file doesn't and if the file exists we go on to calculate the total pages the file can accomodate. If the file's memory is not sufficient we calculate the space needed to make sure that the file has enough capacity and allocate the same memory. If the file's memory is sufficient we provide a return message.

***Reading blocks for the page file***

* **readBlock(int pageNum, SM\_FileHandle \*fHandle, SM\_PageHandle memPage) :** We first check if the file is present or no. We do this with the help of the file descriptor in the file handler. If it exists we move the position of the file descriptor to the page requested in page number. We then read the contents of size 4096 bytes and store it to the memory specified in the mempage.
* **getBlockPoss(SM\_FileHandle \*fHandle) :** We first check if the file is present or no. We do this with the help of file descriptor in file handler. If it exists we get the current page position with the help of file handler.
* **readFirstBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) :** We first check if the file is present or no. We do this with the help of file descriptor in file handler. If it exists we move the file descriptor in the file handler to the starting page of the file and read its contents to mempage.
* **readLastBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) :**We first check if the file is present or no. We do this with the help of file descriptor in file handler. If it exists we move the file descriptor in the file handler to the last page of the file and read its contents to mempage.
* **readCurrentBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) :** We first check if the file is present or no. We do this with the help of file descriptor in file handler. If it exists we read its contents to mempage of the current page position in the file handler with the help of file descriptor present in the file handler.
* **readPreviousBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) :** We first check if the file is present or if we are trying to access a non existing page. We do this with the help of file descriptor in file handler and page position in file handler. If it exists we read its contents to mempage of the previuous page position in the file handler with the help of file descriptor present in the file handler.
* **readNextBlock(SM\_FileHandle \*fHandle, SM\_PageHandle memPage) :** We first check if the file is present or if we are trying to access a non existing page. We do this with the help of file descriptor in file handler and page position in file handler. If it exists we read its contents to mempage of the next page position in the file handler with the help of file descriptor present in the file handler.