External Merge Sort Program Documentation

main.cpp:

get_num_blocks
Calculates the number of blocks required for processing entities with given buffers.

Parameter	Description
num_of_entities	Total number of entities to process.
num_of_buffers	Number of available buffers.

Return: number of blocks needed.

get_num_passes
Calculates the number of passes needed for external merge sort.

Parameter	Description
num_of_records	Total number of records.
num_of_buffers	Number of available buffers.

Return: number of passes needed.

pass0
Pass 0 of the external merge sort algorithm. It breaks down the records from the input file into blocks, sorts each block individually, and returns the sizes of the blocks generated.

Parameter	Description
in_file	Total number of records.
out_file	Number of available buffers.
amt_of_mem	The amount of memory available for sorting.
&num_of_buffers	Number of available buffers for sorting.
&num_of_records	Total number of records in the input file.

Return: a vector containing the sizes of the blocks generated.

merge_blocks
Merges a specified number of sorted blocks into one sorted block. It calculates the index of the start and end records for the blocks to be merged, then performs a merge sort operation on them.

Parameter	Description
sorter	A reference to the FileSorter object used for sorting.
block_sizes	Vector containing the sizes of individual blocks.
start_block	Index of the first block to merge.
num_of_blocks_to_merge	Number of blocks to merge.

Returns: the size of the resulting merged block.

pass
Represents a pass (1, 2, ... n) of the external merge sort algorithm. In each pass, it merges subsets of sorted blocks into larger sorted blocks. The number of blocks in a subset to be merged depends on the number of available buffers.

Parameter	Description
in_file	Total number of records.
out_file	Number of available buffers.
amt_of_mem	The amount of memory available for sorting.
num_of_buffers	Number of available buffers for sorting.

Return: a vector containing the sizes of the merged blocks.

fileSorter.h:

TwoPassMergeSort
Sorts records in the file from record index 'i' to 'j'. It reads records into a vector buffer, sorts them either in ascending or descending order based on the sorting order, and then writes the sorted records to the output file.

Parameter	Description
i	The starting index of the range of records to be sorted.
j	The ending index of the range of records to be sorted.

Return: an integer indicating the success of the sorting operation (1 for success).

TwoPassMergeSort
Merges records within the specified range by reading them into a buffer, which employs a priority queue. The buffer continuously pops the smallest or largest record (depending on the sorting

GetBufferSize
Computes the number of buffers available using the amount of memory allocated to the sorter. The amount of memory available is provided in megabytes (MB), hence it's converted to bytes by multiplying by 1024*1024. Then, it divides the available memory in bytes by the size of each record multiplied by 2 (assuming each buffer holds twice the size of a record) to determine the number of buffers available.

Return: The number of buffers available based on the allocated memory and the record size.

GetNumRecords
Returns the total number of records in the file.

perror
prints a descriptive error message based on the error code
Error Codes:
- -1: "No disk space left."

order) and writes it to the output file until all records within the range are merged.

Parameter	Description
start_block	The index of the starting block.
block_sizes	Vector containing the sizes of individual blocks.
num_of_blocks_to_merge	Number of blocks to merge.
start_record	The index of the starting record.
end_record	The index of the ending record.

Return: an integer indicating the success of the sorting operation (1 for success).

- -2: "File IO error."
- -3: "Buffer is full."
- -4: "Sorting failed."

Parameter	Description
x	The error code indicates the type of error.

record.h:

&operator=
Assignment operator for Record objects.

Parameter	Description
&other	The Record object to copy data from.

Return: a reference to the current Record object after assignment.

&operator[]
Accesses the record data at the specified index.

Parameter	Description
index	The index of the data element to access.

Return: a reference to the data element at the specified index.

***data()**
Returns a pointer to the raw data of the Record object.

operator==
Checks if two records are equal by comparing their keys.

Parameter	Description
r1	The first Record object to compare.
r2	The second Record object to compare.

Return: true if the records are equal, otherwise false.

operator<
Compare two records lexicographically.

Parameter	Description
r1	The first Record object to compare.
r2	The second Record object to compare.

Return: true if the key of the first record is lexicographically less than the key of the second record, otherwise false.

operator>
Compare two records lexicographically.

Parameter	Description
r1	The first Record object to compare.
r2	The second Record object to compare.

Return: true if the key of the first record is lexicographically greater than the key of the second record, otherwise false.

buffer.h:

push
Pushes a record onto the buffer. If the sorting order is ascending (1), min-heap is used. Otherwise max-heap is used

Parameter	Description
val	The record to be pushed.

Return: true if the record was successfully pushed, false if the buffer is full.

top
Returns the top element of the buffer, which can be either the smallest or largest record depending on the sorting order specified.

pop
Removes the top element from the buffer.

empty
Checks if the buffer is empty.
Return: true if the buffer is empty, otherwise false.