بسم الله الرحمن الرحيم

University of Khartoum

Faculty of Mathematical Sciences and Informatics

File Management and Organization (C2043)

‘

File Management and Organization Assignment

(Report)

Content:

Written by:

Name:	**Abobaker Ahmed Khidir Hassan**
Index:	**21-304**
Major:	**Computer Science**

Written in: 18th of December, 2024

About My Work

Introduction:

This is a file management system that supports various operations, such as record insertion, deletion, updating, searching, and indexing using both fixed-length and variable-length records.

Tools:

For this assignment I used C++ programming language using Dev-C++ as the environment.

Note: Make sure to use ISO C++11 to debugging this project.

Overview:

I divided this project contains 5 source files in addition to the main and this report.

- **record_structures(fixed-variable).cpp**

  Contains two classes represent the record structures (fixed length and variable length). The main differences between them are the name field where all other fields (id,score,dateOfBirth) are the same, and in the variable length there is an addition attribute that is the recordLength wheres it is not needed it the fixed length.

- **FixedLengthFileManager.cpp**

  It manages the files those store fixed length records.

  It contains the creation function that create a file to store fixed-length records and the insertion, deletion, updating, and searching for records on that file.

- **VariableLengthFileManager.cpp**

  It manages the files those store variable length records.

  Contains the same of the above file but for the variable-length records.

- **AvailableListManager.cpp**

  It manages the available list in the deletion operation of both types of files, and use it to reclaim the space of the deleted records in files.

- **IndexManager.cpp**

  It manages the indexes of both types of files, and use them to improve the searching performance in both of them.

How Tasks Are Done

Task 1: File Organization

1- It is done in **FixedLengthFileManager.cpp** source file.
   - It contains a function which create a file named “fixed_length_records.txt”, and a function to each mission of this point (add, delete, update, and direct access record).

2- It is done in **VariableLengthFileManager.cpp** source file.
   - It contains a function which create a file named “variable_length_records.txt”, and a function to each mission of this point (add, delete, and update record).
   - The third point of this task (reclaim space for deleted records) is done in **AvailableListManager.cpp** source file. Where it contains a structure like the node where it have (int address, int length, and pointer next) and store the available space of deleted records in a liked list named (available list).
   When a record is deleted, its address and length will be stored at the end of this list. When a new record is added, the insertion function is checks all the available spaces and fit the new record in using First Fit Placement Strategy, and if there is no available place can fit the record it will be appended to the file.

3- It will be done in the next section of this report.

Task 2: Searching

1- It is done in **VariableLengthFileManager.cpp** source file.
   - There is a function there which search for a record using its id, it will check that id with each id in the file sequentially. If the record is found, it will return its byte offset, or it will return -1 if the record is not found.
   - The second point is done in the **main** function because the searching function is used it the deletion and updating functions, and we don’t want to print the searching time with each deletion and updating operation.

2- It is done in **FixedLengthFileManager.cpp** source file.
   - It contains a function that is search for a record using its id using Binary Search Algorithm
   - The file is already sorted because the ids are atomically created by the program.
   - The 2nd and the 3rd points will be in the next section of this report.

Task 3: Indexing

1- It is done in **IndexManager.cpp** source file.
   - There a class named “PrimaryIndex” that contains the insertion and deletion of the entries to the index file using the id as the key.
   - It also contains a function that improve the performance of the binary search for a record.
   - Note: The primary index is about insert an entry for each block in a small file. But I didn’t apply this concept from the begging of this project and I was working as all of this records are in one block. and because of the time and create an entry for each record by the the assumption the blocking factor is 1 record per block, but the RRN concept in this project will be the order of the block in the file rather than the order of the record in the block.

2- It is done in **IndexManager.cpp** source file.
   - There a class named “SecondaryIndex” that contains the insertion and deletion of the entries to the index file using the name as the key.
   - It also contains a function that improve the performance of the sequential search for a record.
   - Note: since the key is variable length, the entry will by variable length too, then searching in the index is still sequentially. And since the key is done by the name, searching will be more useful than searching with the id, and it requite less seeking than searching in the file which decrease the cost.

3- It will be done in the next section of this report.

Comparison Performance

Task 1-3:

Comparison between functions of the fixed-length and variable-length records.

Insertion	**Fixed-Length Records**	**Variable-Length Records**
Execution Time	Faster (appending)	Slower (check the availability list at first)
Disc Accesses	1 disk access	1 disk access (the list in the memory)
Lost Space	Internal Fragmentation	Non

Deletion	**Fixed-Length Records**	**Variable-Length Records**
Execution Time	Constant Time for any record	Slower (sequential search to the target record and and add it to the available list)
Disc Accesses	1 disk access	The order of the record in the file
Lost Space	If we use an available list we can remove all available places	External Fragmentation after reclaim the space

Update	**Fixed-Length Records**	**Variable-Length Records**
Execution Time	Constant Time for any record	Slower (may require reallocation)
Disc Accesses	1 disk access	The order of the record in the file
Lost Space	Internal Fragmentation	External Fragmentation

Task 2 – 2 – b:

Comparison between the binary and sequential search functions for fixed-length.

- **Time and Comparisons:**

Algorithm	**Best Case**	**Average Case**	**Worst Case**
Binary Search	O(1) (Middle Record)	**O(log n)**	**O(log n)**
Sequential Search	O(1) (1st Record)	**O(n)**	**O(n)**

- **Seeking:** Sequential search need more seeking which is more costly.

Task 2 – 2 – c:

The Difference in Performance between Search in Fixed-Length and in Variable-Length Records.

Searching	Fixed-Length Records	Variable-Length Records
Algorithm	Sequentially of Binary	Only Sequentially
Requires	Must be sorted	No sorting is needed
Cost (Time, Comparisons, seeking)	Low in binary	More costly

Task 3-3:

Comparison between Searching in the file and in index (Fixed-Length Records).

Criteria	Search Without Index	Search Using Primary Index
Searching Algorithm	Binary (the data is sorted)	Binary (Entries are sorted)
Time and Comparisons	O(log N) (binary search through sorted records)	O(log N) (binary search through index)
Seeking	More Costly (all the record)	Low (enties are small)
Requires	data must be sorted (more costly)	No sorting is needed

Comparison between searching in the file and in index(Variable-Length Records).

Criteria	Search Without Index	Search Using Secondary Index
Key	Search given the id.	Search given the Name.
Searching Algorithm	Sequential Search	Sequential Search (key is variable length)
Time and Comparisons	O(N) (scan the hall of records fields)	O(N) (scan only entries in the index file)
Seeking	Very costly	Less than the file
Requires	No sorting is needed	No sorting is needed

Challenges and Solutions

I faced many challenges due this project. This is a summary of them and how they became solved.

Challenge 1:

How to transform the theoretical concepts as a real project?

Solution:

Using the compound of programming concepts such as programming fundamentals and OOP concepts to represent the file structure concepts as a code file.

Challenge 2:

When I used some functions (like to_string()), I faced errors that is those functions are not declared!, that was because of my old version of C++.

Solution:

I left VS conde and work using Dev-C++, and I changed the the standard language to ISO C++11 form (Tools -> Compiler Options -> Sittings -> Code Generation -> Language Standard -> ISO C++11).

Challenge 3:

How to automate the id field?

Solution:

By adding another attribute which is lastId to the class that manage the file operations and when adding any record to the file, this attribute will generate an id by default and store it to the file.

Challenge 4:

In the reclaim space in the variable-lengths file, after inserting a new record in an available place, we find that there is a few chars from the old record and this is called External Fragmentation.

Solution:

By editing the deletion function to as:

When deleting a record, it will be marked as deleted (its id in the file will be changed by “00-1”, and try to simulate erase function, the available space will be filled by ‘\0’ (spaces).

Challenge 5:

When I arrived to the indexing task, I find that the primary index requires the blocking concept to create an entry for each block (for the first record in each block), but I didn't apply this concept in any task before, and I was working by the assumption that there is no blocking, or in other words, all records in one block.

Solution:

I find myself in two options:

- To assume that the blocking factor = 1 record per block. Then I will create an entry for each record by the assumption entry for each block.
- To apply blocking concept as non-physical concept. Then the user can chose the blocking factor and an entry will be inserted to the index file with each (bfr) record.

I chose the first one because it is easier to deal with and it will be clearer when debugging the code, and here I want to declare that the RRN is the order of the record in the block, and what I say I was working by the assumption that all records in one block. But after this choice in the indexing step the RRN became doesn't represent that. But we can say it represent the order of the block in the file.

Thank You