DBMS Models and Implementation
Instructor: Abhishek Santra
Project 1: B+ Tree Implementation

Made available on: **1/23/2025**
Complete Project Due on: **2/18/2025 (11:59 PM)**
Submit on: **Canvas (1 zipped folder containing all the files/sub-folders)**
uta.instructure.com
Weight: **15% of total**
Total Points: **100**

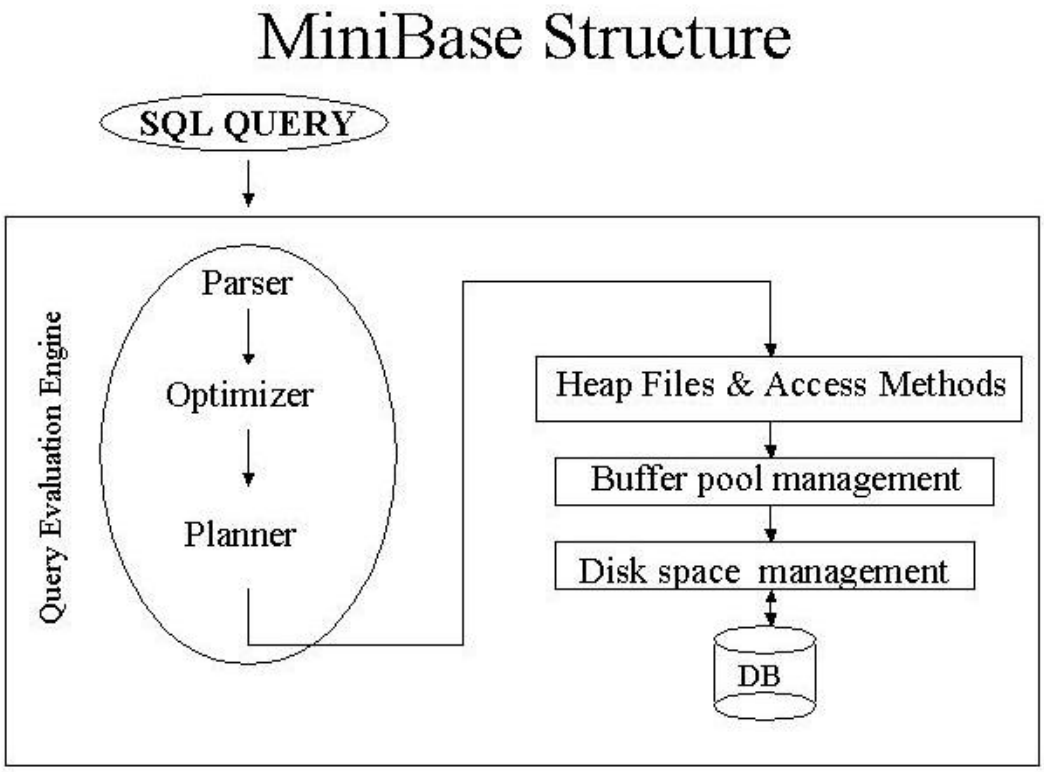
This project implements parts of the index file organization for the database management system MINIBASE. You will use some methods of Buffer Manager Class which are provided to you. The test methods indicate the sequence of calls made on the provided layers as well as on the methods that you will be implementing. In addition to what you implement, this project is intended to show typical implementation of DBMS modules.

I. Problem Statement:

In this project, you will implement a B+ tree in which leaf level pages contain entries of the form <key, rid of a data record> (Alternative 2 for data entries, as discussed in the textbook on page 276.) Multiple values of the same key are stored as separate <key, data ptr> on the leaf page. When you delete, you always delete the <key, data ptr> value. When you insert, you insert the current <key, data ptr> as the last one. You shall implement the full insert and a simple deletion algorithm. Your insert routine must be capable of dealing with overflows (at any level of the tree) by splitting pages using the algorithm discussed in the class/book. *You will not consider re-distribution*. For this project, you shall implement a naïve delete (simply remove the record without performing any merging or redistribution). Before jumping into this project, make sure you thoroughly understand pages 344 to 363 of the textbook (3rd edition) for the algorithms.

You are given, in addition to other packages and classes, *`HFPage`* and *`BTSortedPage`*. *`BTSortedPage`* is derived from *`HFPage`*, and it augments the `insertRecord` method of *`HFPage`* by storing records on the *`HFPage`* in sorted order by a specified <key> value. The key value must be included as the initial part of each record to enable easy comparison of the key value of a new record with the key values of existing records on a page. The documentation available as part of the java code (i.e., javadoc) should be enough to understand what operation each function performs (please see the javadoc URL for detailed descriptions).

You need to use two page-level classes, *BTIndexPage* and *BTLeafPage*, both of which extend *BTSortedPage*. These page classes are used to build the B+ tree index. The methods you will need to use such as create, destroy, open, & close a B+ tree index, and to open scans that return all data entries (from the leaf pages) which satisfy some range selection on the keys that are provided.



Overview of the Minibase along with all the layers

Use the following links for additional information on the project and Minibase, respectively:

1) https://itlab.uta.edu/courses/cse5331And4331/javadocs/index.html
2) https://itlab.uta.edu/courses/cse5331And4331/mini_doc-2.0/minibase.html
3) https://www.eecs.yorku.ca/course_archive/2013-14/W/4411/proj/javadoc/ **(for Btree package)**

II. Design Overview:

1. A key point to remember

- You should note that key values are passed to functions using `KeyClass` objects (an abstract class). The contents of a key should be interpreted using the `AttrType` variable. The key can be either a string (`attrString`) or an integer (`attrInteger`), as per the definition of `AttrType` in `global` package. If the key is a string, its value is stored in a `StringKey` class which extends the `KeyClass`. Likewise, if the key is an integer, its value is stored in an `IntegerKey` class which also extends the `KeyClass`.

- Based on the above structure, keys can be of type `AttrType` and if they are not either `attrString` or `attrInteger`, you need to return an error message .

- The `BTSortedPage` class, which augments the `insertRecord` method of `HFPage` by storing records on a page in sorted order according to a specified `key` value, assumes that the key value is included as the initial part of each record, to enable easy comparison of the key value of a new record with the key values of existing records on a page.
  (Note. For this project, the key will be a positive integer (attrInteger) for simplicity)

2. Methods that need to be implemented in BTreeFile.java

- `insert`, `_insert` and `NaïveDelete` methods belonging to the BTreeFile class need to be implemented for this project. BTreeFile is a derived class of the IndexFile class, which means a BTreeFile is a kind of IndexFile.

The methods to be implemented are described under the Btree package of the given java documentation. Specifically, you will be responsible to provide code for the following methods:

- **Insert() method**
  See the book for the algorithm. Start with validating the key [in our case it is integer (`attrInteger`)]. If the tree is empty, create first page as newrootPage (of type BTLeafPage) which will be a leaf page and set its page id to the headerpageId (already initialized in `BTreeFile() constructor`) and set its next page and previous page pointer to *INVALID_PAGE;* insert the <key,rid> using the `insertRecord()` method then unpin the page as it is dirty (it is already pinned when you get from the buffer) and update the header page using `updateHeader()` else make a call to `_insert()` `[newRootEntry=_insert(key,rid,headerPage.get_rootId());]` method to insert the record <key, data> and set the pointers. If a page overflows (i.e., no space for the new entry), you should split the page. You may have to insert additional entries of the form <key, id of child page> into the higher-level index pages as part of a split. Note that this could recursively go all the way up to the root, possibly resulting in a split of the root node of the B+ tree.

  - **_insert() method**
    Check the `pageType`. If it is an INDEX page call `_insert()` recursively to insert and split if necessary. If it a LEAF page again call `_insert()` recursively to insert and handle split.

    The `keyCompare()` method is used by `_insert()` to compare the key.

    ```
    int keyCompare(KeyClass key1, KeyClass key2)
    ```

    This method returns an integer. key1>key2 returns a positive value, key1<key2 will return a negative value and 0 if equal (key1 == key2).

NOTE: As discussed in the class, **condition for traversal can be chosen, but needs to be consistent for an implementation.** Clearly state in the report, **which traversal choice you went ahead with in your implementation**,

Choice1) the right pointer needs to be traversed if key1 value is greater than or equal to key2; else traverse the left pointer. OR,

Choice2) the right pointer needs to be traversed if key1 value is greater than key2; else traverse the left pointer

- **Delete method**

The Delete method simply **removes the entry <key, data ptr> from the appropriate BTLeafPage**, *if it exists*. You do not need to implement redistribution or page merging when the number of entries falls below threshold. **All duplicate values have to be deleted**. This method is given and is set to call NaiveDelete() method by default.

- NaiveDelete() method

In NaïveDelete() you need to remove the data entry <key, data ptr> from the leaf page of the index without any merging or redistribution.

The method required to search is already given. It returns the leaf page at the left most part of the tree and then search for the key to be deleted as leaf pages are organized as a doubly link list. You can see the search algorithm in the book for more clarity. *You need to be careful if the search key does not exist at the leaf level*

```
BTLeafPage findRunStart(key, curRid);
```

If the search key does not exist at the leaf level, then your code must handle it gracefully by giving proper message on the screen.

3. Classes that are given and you will be using

B+ Tree page level classes (Figure 1)

- **HFPage:** This is the base class, you can look at `heap` package to get more details.
- **BTSortedPage**: This class is derived from the class HFPage. Its only function is to maintain records on a HFPage in a sorted order. Only the slot directory is re-arranged. The data records remain in the same positions on the page.
- **BTIndexPage**: This class is derived from BTSortedPage. It inserts records of the type < key, pageNo> on the BTSortedPage. The records are sorted by the key.

- **BTLeafPage**: This class is derived from BTSortedPage. It inserts records of the type <key, dataRid> on the BTSortedPage. *dataRid* is the rid of the data record. The records are sorted by the key. Further, leaf pages must be maintained in a doubly-linked list.

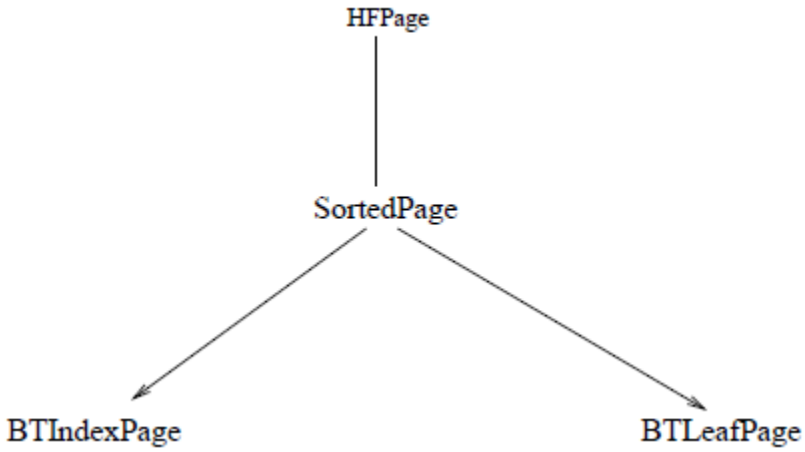


Figure 1: The classes used for the B+ Tree pages

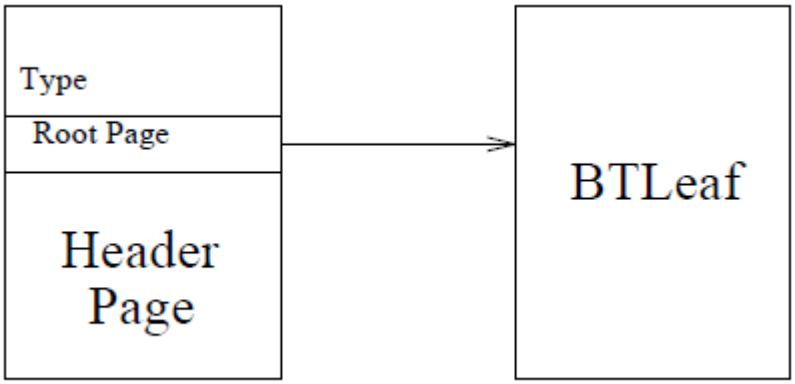


Figure 2: B+ tree with one leaf page

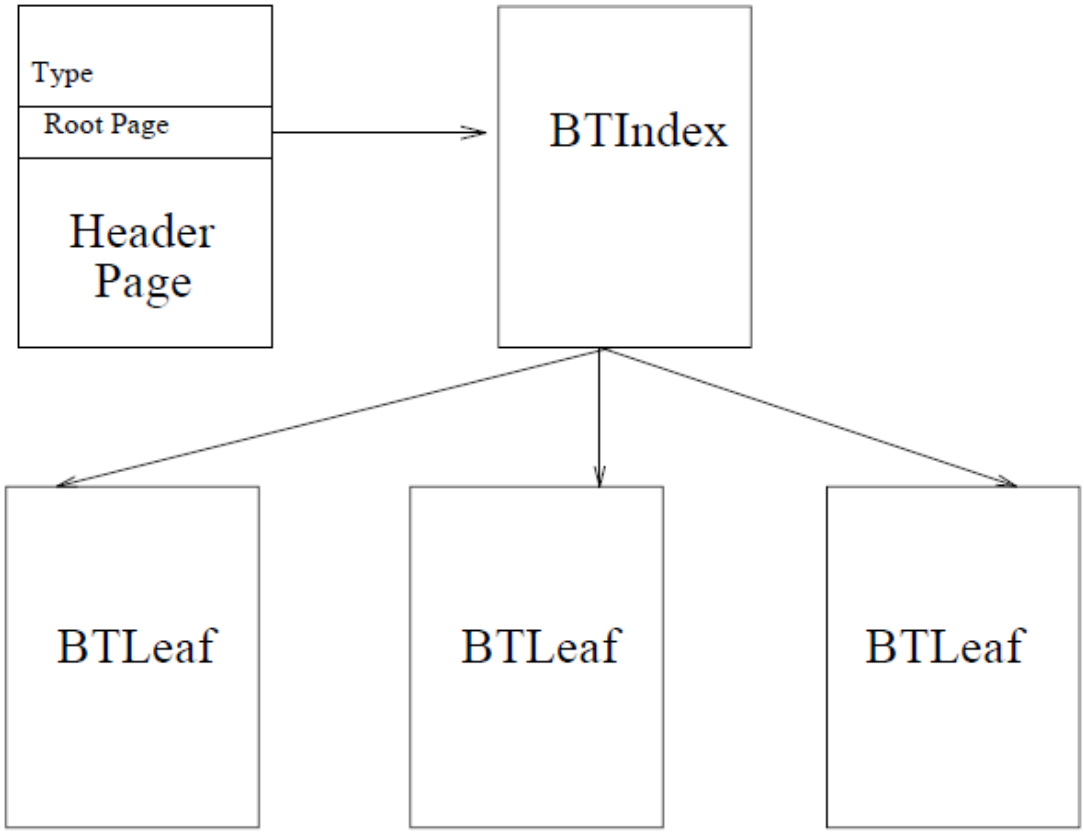


Figure 3: B+ tree with more than one leaf page

Figure 2 shows what a BTreeFile with only one BTLeafPage looks like; the single leaf page is also the root. Note that there is *no* BTIndexPage in this case. Figure 3 shows a tree with a few BTLeafPages, and this can easily be extended to contain multiple levels of BTIndexPages as well.

Note that the java documentation provided to you for methods in `BTreeFile`, `BTIndexPage`, and `BTLeafPage`, do not specify whether they throw any exceptions; or rather, what exceptions that they throw. You should follow the error protocol defined as before to implement your methods with reasonable exceptions thrown.

III. Some useful hints and tips:

- Remove the UnsupportedOperationException lines as you implement your code.
  (i.e. these are simply place holders so that the skeleton code would compile)

- Follow the example of good Java programming style set by the skeleton code.
  (i.e. you will be docked if your code isn't well-commented and/or formatted)

- Study the javadocs given in https://www.eecs.yorku.ca/course_archive/2013-14/W/4411/proj/javadoc/. They have most of the methods that will be needed by you for

this project. Carefully understand the parameters expected and return object type of each method from the javadoc. For example,

- Use the proper constructor while creating a leaf or index page
- To fetch the id of the current page: **getCurPage()**
- Remember to pin and unpin index and leaf pages in the correct order using the methods **pinPage(...), unpinPage(...)** - signature defined in BTreeFile.java
- To set the next and prev page you have the methods: **setNextPage(PageId pageNo)** and **setPrevPage(PageId pageNo)**
- To update the header: **updateHeader(PageId newRoot)** - signature/description present inBTreeFile.java
- To check amount of space available in an index/leaf page: **available_space()**
- To insert an entry into an index page: **insertKey(...)**
- To insert an entry into a leaf page: **insertRecord(...)**
- Useful methods that can be used to redistribute records in times of leaf/index split: **getFirst(...), deleteSortedRecord(...), insertRecord(...), insertKey(...), BT.keyCompare(KeyClass k1, KeyClass k2)**
- To fetch the type of the page: **getType()**
- Different types of pages defined under Constant Field Values: **NodeType.BTHEAD, NodeType.LEAF, NodeType.INDEX**

- Some useful methods whose Javadoc is not available
  - **BT.getKeyDataLength(KeyClass key, NodeType.INDEX)**: returns the number of bytes required by an entry made into an index page
  - **BT.getKeyDataLength(KeyClass key, NodeType.LEAF)**: returns the number of bytes required by an entry made into a leaf page
  - **PageId getPageNoByKey(KeyClass key)** - an instance method of the BTIndexPage class that returns the page id of the page at the next level where the key will be directed. Useful for traversing the tree.

IV. What you are asked to do:

1. Complete the implementation of the `BTreeFile.java` skeleton class given to you
2. Implement the methods - `insert()`, `_insert()` and `NaïveDelete()`
3. Compile and run using the test input file (**btree_project_spring_25/src/tests/test-insert-file.txt**) given to you. We may add/extend the test cases for evaluation. We encourage you to test your code for different insertion and deletion sequences.

V. Getting Started

- **All the codes will be evaluated on omega.uta.edu – the University's UNIX server.** Check the following link for more information. https://uta.service-now.com/kb_view.do?sysparm_article=KB0010329
- Download the **skeleton code** from Canvas. Unzip the folder **btree_project_spring_25** into your local directory.
- A sample Makefile for you to compile your project is given in both the directories btree_project_spring_25/src/btree and btree_project_spring_25/src/tests. You will have to slightly modify them by following the comments. The changes are mainly to set the correct paths based on where you copy them on your computer/omega. You can also design your own Makefile. Whatever you do, please make sure that you use the classpath provided in the original Makefile.
- Steps of execution using the makefile on cygwin, omega, linux or unix terminal
  - In **btree_project_spring_25/src/btree** folder type "**make**" to compile the BtreeFile.java
  - In **btree_project_spring_25/src/tests** folder type "**make**" to compile then type "**make bttest**" to run the code.
  - Type "make clean" from any of the folder to remove the *.class* files.
- The java documentations of some classes can be found in the javadoc. For the btree package use the link mentioned above in section I. Read the classes and methods descriptions carefully, for it will help you understand the program structure.
- You can find other useful utilities in the java documentation.
- If you want to use an IDE (e.g., Eclipse) for doing the project, you need to figure out how to create a project with the given files. *We have provided the Eclipse configuration steps on canvas.*

VI. Error Handling

This version of Minibase makes use of Java's built-in runtime exceptions. In general, only two exceptions are necessary:

- **IllegalArgumentException**
  Throw this when a method argument is invalid, for example if the user attempts to select a record that doesn't exist.
- **IllegalStateException**
  Throw this if the database reaches an undefined state, for example if the user attempts to get the next record after completing a scan.

Required error checking is already included in the javadoc comments (i.e. "@throws"). Note that some methods throw exceptions simply because they call other methods that throw them (i.e. BufMgr.newPage() -> BufMgr.pinPage() -> etc).

VII. Project Report

Please include (at least) the following sections in a **REPORT {pdf format}** file that you will turn in with your code:

- **Overall Status**
  Give a *brief* overview of how you implemented the major components. Clearly state the traversal alternative chosen, while inserting. If you were unable to finish any portion of the project, please give details about what is completed and your understanding of what is not. (This information is useful when determining partial credit.)
- **File Descriptions**
  List any new files you have created and *briefly* explain their major functions and/or data structures. If you have performed additional testing using new files, please summarize them.
- **Division of Labor**
  Describe how you divided the work, i.e., which team member did what. Please also include how much time each of you spent on this project.
- **Logical errors and how you handled them**
  List at least 2 logical errors you encountered during the implementation of the project. Pick those that challenged you. This will provide us some insights into how we can improve the description and forewarn students for future assignments.

VIII. What to Submit

- After you are satisfied that your code does exactly what the project requires, you may turn it in for grading. Please submit your **project report** and **the BTree package in a single zipped folder**. We will ignore source code in any other directories.
- All the above files should be placed in a single zipped folder named as
  '**4331-5331_Project1_Spring25_team_<teamNo>**'
  Only one zipped folder should be uploaded using canvas.
- You can submit your zip file multiple times. The **latest one (based on timestamp) will be used for grading**. So, be careful in what you turn in and when!
- **Only one person per team should turn in the zip file.**
- [IMPORTANT] To discourage late submissions, a **penalty of 25% per day (no partial penalty)** will be imposed. This means that no submission will be accepted if it is delayed by more than 3 days. We certainly do not want this delay to hurt your next project!

IX. Coding Style:

Be sure to observe the following standard Java naming conventions and style. These will be used across all projects for this course; hence it is necessary that you understand and follow them correctly. You can look this up on the web. Remember the following:

a. Class names begin with an upper-case letter, as do any subsequent words in the class name.
b. Method names begin with a lower-case letter, and any subsequent words in the method name begin with an upper-case letter.
c. Class, instance and local variables begin with a lower-case letter, and any subsequent words in the name of that variable begin with an upper-case letter.
d. No hardwiring of constants. Constants should be declared using all upper case identifiers with _ as separators.
e. All user prompts (if any) must be clear and understandable
f. Give meaningful names for classes, methods, and variables even if they seem to be long. The point is that the names should be easy to understand for a new person looking at your code
g. Your program is properly indented to make it understandable. Proper matching of if … then … else and other control structures is important and should be easily understandable
h. Do not put multiple statements in a single line

In addition, ensure that your code is properly documented in terms of comments and other forms of documentation for generating meaningful javadoc.

X. Grading Scheme for the Complete Project:

The project will be graded based on **a mandatory demo that will be scheduled after the submission deadline**. The grading rubric is as follows (out of 100 points):

1.	Correctness of the `insert()` code:	**30**
2.	Correctness of the `NaiveDelete()` method:	**20**
3.	Reporting and fixing at least 2 logical errors in the program:	**10**
4.	Documentation (Javadoc) and commenting of the code: (Including coding style)	**10**
5.	Report:	**10**
6.	Q/A performance during demo	**20**

For evaluation, your java program must be executable (without any modification by us) from the Linux command prompt or Cygwin command prompt (on **UTA's Omega server**). So, please test it out before submitting it. However, the source code files can be created and/or edited on any editor that produces an ASCII text file. As mentioned in the class, an IDE is not necessary for this and subsequent projects. If you decide to use it, please learn it on your own and make sure your code compiles and executes with appropriate package information.

The timestamp of the code executed should be the same as the submission timestamp. Else, penalty will be applied as indicated above.

FYI, when your code runs correctly for the initial run, the output from BTTest.java should look like:

```
Replacer: Clock


Running  tests....

 ***************** The file name is: AAA0  **********
------------------------- MENU ------------------

[0]   Print the B+ Tree Structure
[1]   Print All Leaf Pages
[2]   Choose a Page (Index/Leaf) to Print

           --- Positive Integer Key (for choices [3]-[5]) ---

[3]   Insert a Record
[4]   Delete a Record (Naive Delete)
[5]   Delete some records (Naive Delete)

[6]  Quit!
Hi, make your choice :3
Initially 100 values have been inserted from 401 to 500.
------------------------- MENU ------------------

[0]   Print the B+ Tree Structure
[1]   Print All Leaf Pages
[2]   Choose a Page (Index/Leaf) to Print

           --- Positive Integer Key (for choices [3]-[5]) ---

[3]   Insert a Record
[4]   Delete a Record (Naive Delete)
[5]   Delete some records (Naive Delete)

[6]  Quit!
Hi, make your choice :0



---------------The B+ Tree Structure---------------
1     5
2            3
2            4
2            6
--------------- End ---------------


------------------------- MENU ------------------

[0]   Print the B+ Tree Structure
[1]   Print All Leaf Pages
[2]   Choose a Page (Index/Leaf) to Print

           --- Positive Integer Key (for choices [3]-[5]) ---
```

```
[3]   Insert a Record
[4]   Delete a Record (Naive Delete)
[5]   Delete some records (Naive Delete)

[6]  Quit!
Hi, make your choice :2
Input the page number:
5

**************To Print an Index Page ********
Current Page ID: 5
Left Link      : 3
0 (key, pageId):   (432,  4 )
1 (key, pageId):   (463,  6 )
************** END ********

------------------------- MENU ------------------

[0]   Print the B+ Tree Structure
[1]   Print All Leaf Pages
[2]   Choose a Page (Index/Leaf) to Print

           --- Positive Integer Key (for choices [3]-[5]) ---

[3]   Insert a Record
[4]   Delete a Record (Naive Delete)
[5]   Delete some records (Naive Delete)

[6]  Quit!
Hi, make your choice :2
Input the page number:
3

**************To Print an Leaf Page ********
Current Page ID: 3
Left Link      : -1
Right Link     : 4
0 (key, [pageNo, slotNo]):   (401,  [ 401 401 ] )
1 (key, [pageNo, slotNo]):   (402,  [ 402 402 ] )
2 (key, [pageNo, slotNo]):   (403,  [ 403 403 ] )
3 (key, [pageNo, slotNo]):   (404,  [ 404 404 ] )
4 (key, [pageNo, slotNo]):   (405,  [ 405 405 ] )
5 (key, [pageNo, slotNo]):   (406,  [ 406 406 ] )
6 (key, [pageNo, slotNo]):   (407,  [ 407 407 ] )
7 (key, [pageNo, slotNo]):   (408,  [ 408 408 ] )
8 (key, [pageNo, slotNo]):   (409,  [ 409 409 ] )
9 (key, [pageNo, slotNo]):   (410,  [ 410 410 ] )
10 (key, [pageNo, slotNo]):   (411,  [ 411 411 ] )
11 (key, [pageNo, slotNo]):   (412,  [ 412 412 ] )
12 (key, [pageNo, slotNo]):   (413,  [ 413 413 ] )
13 (key, [pageNo, slotNo]):   (414,  [ 414 414 ] )
14 (key, [pageNo, slotNo]):   (415,  [ 415 415 ] )
15 (key, [pageNo, slotNo]):   (416,  [ 416 416 ] )
16 (key, [pageNo, slotNo]):   (417,  [ 417 417 ] )
17 (key, [pageNo, slotNo]):   (418,  [ 418 418 ] )
18 (key, [pageNo, slotNo]):   (419,  [ 419 419 ] )
```

```
19 (key, [pageNo, slotNo]):   (420,  [ 420 420 ] )
20 (key, [pageNo, slotNo]):   (421,  [ 421 421 ] )
21 (key, [pageNo, slotNo]):   (422,  [ 422 422 ] )
22 (key, [pageNo, slotNo]):   (423,  [ 423 423 ] )
23 (key, [pageNo, slotNo]):   (424,  [ 424 424 ] )
24 (key, [pageNo, slotNo]):   (425,  [ 425 425 ] )
25 (key, [pageNo, slotNo]):   (426,  [ 426 426 ] )
26 (key, [pageNo, slotNo]):   (427,  [ 427 427 ] )
27 (key, [pageNo, slotNo]):   (428,  [ 428 428 ] )
28 (key, [pageNo, slotNo]):   (429,  [ 429 429 ] )
29 (key, [pageNo, slotNo]):   (430,  [ 430 430 ] )
30 (key, [pageNo, slotNo]):   (431,  [ 431 431 ] )
************** END ********

------------------------- MENU ------------------

[0]   Print the B+ Tree Structure
[1]   Print All Leaf Pages
[2]   Choose a Page (Index/Leaf) to Print

           --- Positive Integer Key (for choices [3]-[5]) ---

[3]   Insert a Record
[4]   Delete a Record (Naive Delete)
[5]   Delete some records (Naive Delete)

[6]  Quit!
Hi, make your choice :3


 -----------------------------
1. Insert Single Value
2. Insert Multiple Values
3. Insert Values from a File
Make your choice(4 to exit) :
```