**Project1 Report**

**Introduction:** In this report, I will present a succinct summary of the thinking pattern and implementation method of Project1. To make it clear, the whole report is logically divided in two parts, for PFM and RPFM manipulations respectively.

**Paged File System:**

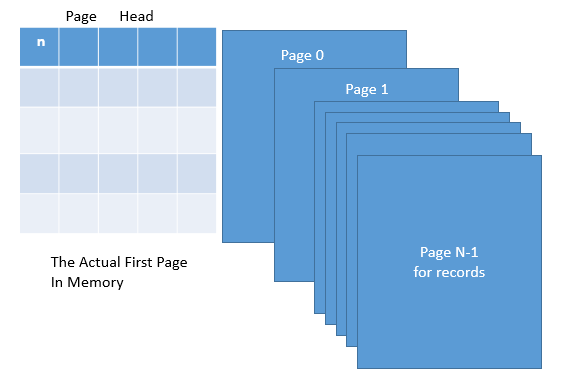


Figure.1 The structure of a file in memory

**Description of Thinking:**

For the purpose of efficiently manage our file pages, by textbook we know that we are supposed to apply “Linked List of Pages” or “Directory of Pages” instead of choosing “Heap Files”, though at the project1 we do not need to worry much about this.

In the level of file management, we only need to the functions like file creation, destroy, opening and close so that at this moment besides storing the number of pages reserved for record, it is not necessary to initialize the Page Head, like storing the pointer to a certain page or the pointer to the next page head once all the storage of first page head is occupied. Hence, the actual second page (in terms of address, it tightly follows page head) would be called Page 0, and the following would be name Page 1, Page 2, … Page N-1 if we only have totally N pages of record.

In the future project, we will achieve the full function of “Directory of Pages”

**Implementation Details:**

About File Manipulations: What we need to notice is that whether the file is already there, if so, the program is supposed to return a value other than 0 (in this case we set it -1, which seems to be common rules). There are a piles of ways to check, like \_access method, simply using the return value of *fopen*. Here I just imitated the way coming from our test code using ***struct stat***.

At this moment, we also need to set a flag for how many handles are currently working on a certain file, and we would benefit from this when dealing with other methods like destroying a file, which is not allowed when this file is opening, hence we check whether point p has a value other than NULL.

About Handle Manipulations: (1) To get the number of pages, we can simply get it from the first data of head page by the movement of file pointer. (2) Based on the rules of naming page, the address offset would be actually page size multiplied by sum of n and 1. For the same reason, we need to pay attention to that when dealing with reading page. (3) To build up robustness of codes, we ought to consider some strange test cases as shown in codes.

**Record Based File System**

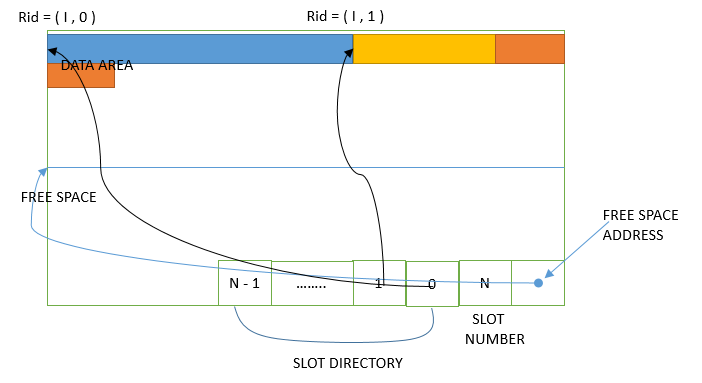


Figure. 2 The structure of page I

**Description of Thinking:**

As we are not allowed to store “ abcdefghi” in a varchar(80), we are absolutely required to apply the page framework for variable-length records. In this case, I choose to implement the most fundamental one like Figure 2. We store the free space address in the bottom of each page so that we can easily find out where the record can be inserted. At the same time, we need a slot directory to memorize the basic descriptions (including length, entry address etc.) of a certain slot and we could reserve their data from the bottom up. To update the slot directory, we will also need the total numbers of slot which is also stored at the bottom of our pages. **\*Notice:** the naming order is slightly different from the textbook for the purpose of concurring with the naming rules of pages.

When we need to insert a record to the file, we first calculate the length of the record and look up for an appropriate place of a certain page. We scan the pages file from the first page to the last page seeking this place. What we should notice is that when considering whether the record can be placed in that page, the extra memory cost of slot directory matters.

If there are no pages in file can store the record, we append a new page and insert the record to that page.

As the slot directory include the address of its corresponding slot, it would be quite easy to read the record data from the slot.

**Implementation Details:**

About File Manipulation with RBFM: According to the requirement, we need to create, destroy, open and close file by invoking the methods defined in PFM, and this could be easily achieved by call an instance. Besides, we should also notice there are slightly different between the type of arguments defined in the RBFM and PFM, so the method of ***c\_str*** would be called for.

About Record Manipulations with RBFM:(1). To get the length of record, we could use “switch” method to calculate the contribution coming from ***int***, float and ***TypeVarChar*** data. However, as the size of int is equal to float and there is always a length indicator before a data of ***TypeVarChar***, so we can slightly modified the original codes as displayed.(2) In the ***findPage***method we should not ignore the extra storage for an object of ***recordDirectory***. Insert method is not so difficult to implement after carefully consider and update the offset for free space, the number of total record and the address for inserting the directory of a slot.

With sufficient carefulness, we could implement the insert and read method. An extra thing worth noticing is storage leak which could occur when we do not free the buffer in time. In the test 10, we are supposed to invoke print record many times, which means we have to invoke the output method quite often. This, with any doubt, can consume quite a lot of time. The basic idea to avoid so is simply use a string variable to storage as much output data as possible before indeed printing it. Using sprintf method can easily translates the int variable to string variable when it is totally not the case for the convert from float variable to string variable because of the precision limitation (an extra 0001 will come). In this case, we just simply output all we have and restart the process.