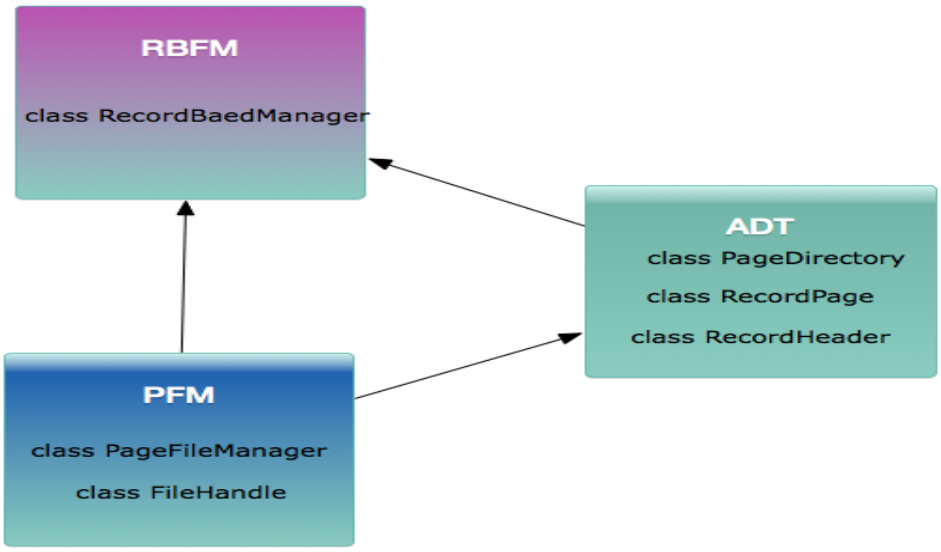
Project 1 Report

1. Overview

In project 1, our goal is to provide record based file manager facilities, such as file access methods, record page organization, which could be used by DBMS's higher layer.

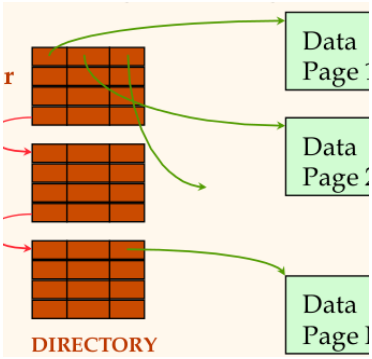
2. Design & Implementation

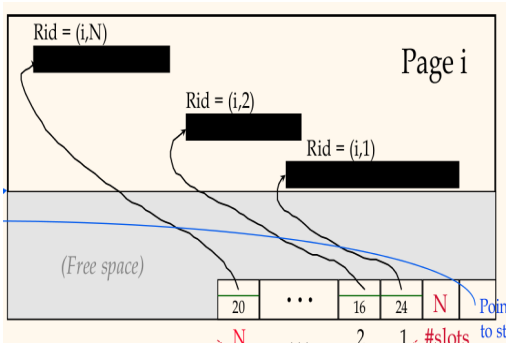
In this project, we have 3 components: PFM(paged file manager), RBFM(record based file manager), ADT(abstract data structure). The arrow from A to B means A uses facilities provided by B.

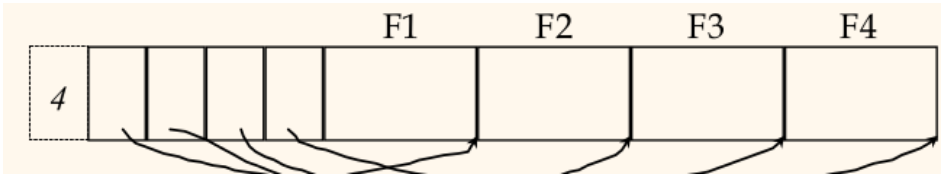


PFM and RBFM is already defined by code base. ADT, standing for abstract data structure, contain three class definitions, which provide setter/getter functions for each of their data fields.

We use following three mechanism to organize record based file:







Their corresponding class is **`PageDirectory, RecordPage, RecordHeader`** respectively. The instance of these classes is associated with outside buffer, which points to the beginning of their own data. **`PageDirectory`** is responsible for locating a **`RecordPage`** with given free space, RecordPage is responsible for returning a given RID RecordHeader. **`RecordHeader`** could be used to retrieve/write record content.

For example, to get the record length at given index position, the code snippet would be like:

```
int readInt(char *data){
   int val;
   memcpy(&val, data, sizeof(int));
   return val;
}
 int RecordPage::get_rcdlen(int index){
   return readInt(data + PAGE_SIZE - UNIT_SIZE * (3 + 2 * index));
}
```

With this design, it could encapsulate the data layout into class getter/setter methods. Then RBFM just need to use their methods to focus on high level logic, like:

```
PageDirectory::set_pgfree_byid(int pgid, int free);
RecordHeader::writeRecord(vector<Attribute> descriptor, void *data);
```

And test cases help me understand the way handling variable length string: first is an integer indicating the length of the string, then the string content.