BufferManager模块：

邬凡

Buffer Manager负责缓冲区的管理，主要功能有：

> 1.根据需要，读取指定的数据到系统缓冲区或将缓冲区中的数据写出到文件
> 2.实现缓冲区的替换算法，当缓冲区满时选择合适的页进行替换
> 3.记录缓冲区中各页的状态，如是否被修改过等
> 4.提供缓冲区页的pin功能，及锁定缓冲区的页，不允许替换出去

为提高磁盘I/O操作的效率，缓冲区与文件系统交互的单位是块，块的大小为文件系统与磁盘交互单位的整数倍，定为4KB。

BufferManager由两个类构成，外部接口如下：

```
enum class PageType {
    UndefinedPage,
    RecordPage,
    IndexPage,
    RecordCatalogPage,
    IndexCatalogPage
};

class Page {
    ......
    string          tableName;
    string          attributeName;
    PageType        pageType;
    PageIndexType   pageIndex;
    int             fileHandle;
    char            pageData[PAGESIZE];
};
```

> **Page**类为其他组件与Buffer进行数据交换的媒介，其他组件通过设定page中的tableName，attributeName，pageType，PageIndex传递给BufferManager，BufferManager通过page中的信息从硬盘或者从缓冲区获取数据或者写入数据。需要写入和读取的数据都在char数组pageData中。

```
class BufferManager {
public:
    ......
     bool readPage       (Page &page);
     bool writePage      (Page &page);
     bool allocatePage   (Page &page);
     bool deallocatePage (Page &page);
    ......

     void closeAllFiles();

     void pinPage(Page &page);
     void unpinPage(Page &page);

     void clearCache();

     static map<string, PageIndexType> tableFileHandles;
     static map<pair<string, string>, PageIndexType>
indexFileHandles;
     static map<string, PageIndexType> tableCatalogFileHandles;
     static map<pair<string, string>, PageIndexType>
indexCalalogFileHandles;

     static const string recordFilesDirectory;
     static const string indexFilesDirectory;
     static const string recordCatalogFilesDirectory;
     static const string indexCatalogFilesDirectory;

     static Page cachePages[CACHECAPACITY];
     static bool pined[CACHECAPACITY];
     static bool isDirty[CACHECAPACITY];
     static int lruCounter[CACHECAPACITY];
};
```

> 对于文件的操作使用了POSIX。为方便起见，所有的文件打开操作全部由BufferManager自动提供。当用户需要写入特定文件的时候BufferManager会自动检查文件的存在或是否打开并获取文件句柄。文件句柄使用了map进行存储，其索引值为相应的表名索引名。缓存使用了一个Page数组进行内存缓存，为了实现LRU使用了一个与缓冲区大小相同的计数器来记录缓冲块没有被用的时间，为了实现替换时写回使用了一个与缓冲区大小相同的bool数组记录缓冲区是否被写过，为了实现pin使用了一个bool数组来记录pin过的缓冲区。

下面介绍读写的具体实现

readPage()
首先判断出缓冲区内是否有需要的页，如果有，读取缓冲区，改变计数器，将除了当前读取的缓冲页的其他缓冲页的计数器全部加1，返回，如果没有,检查当前文件是否被打开，没打开通过系统调用获取文件句柄，打开了从map中获取文件句柄，通过文件句柄用系统调用直接读取文件的页，读取完了以后寻找当前缓冲区中计数器最大且没有pin的页，进行替换，如果替换的页dirty需要写回。
writePage()
将read中的操作改变成write即可。
allocatePage(), deallocatePage()
在一个文件内部回收的页由0号页作为头使用页内头四个字节作为指针进行连接。allocatePage()会分配给参数Page页号作为新的页，分配时如果回收链表有页，则从链表获取页，如果链表为空即0号页指向-1，计算文件大小/PAGESIZE的到文件尾部页号。删除页将页插入链表中即可。

测试方法：

通过调用allocate和deallocate输出页号与预期相同。
一开始不经过缓冲区跑整个系统成功后加入缓冲区发现功能正常。
由于时间有限没有在Buffer上另外做过多测试。