哈尔滨工业大学计算机学院 2016 夏季学期《C++程序设计》大作业指导书

实验要求：

1. 自拟二进制归档文件（小文件集合）存储协议，归档文件至少包含两个区域，文件头（用于记录归档文件的基本属性和内部所归档小文件的索引等信息），数据区域（用于存储实际小文件数据，可以自己制定存储方案）。
2. 编写归档文件管理器，实现归档文件的初始化、打开、关闭操作。
3. 实现归档文件管理器对归档文件的操作功能：
   a) 添加文件进入归档文件
   b) 从归档文件中提取所需文件（根据文件名）
   c) 从归档文件中删除指定文件
   d) 列出归档文件中现已归档的文件

实验步骤及参考：

根据实验要求可以知道，我们在做这个实验之前首先需要设计二进制归档文件的存储协议。存储协议可以分为两个部分。其一是归档文件保存在磁盘时的文件协议。其二是文件在内存中时的保存协议。

在设计存储协议之前，我们需要简单设计一下我们归档文件的大概结构。对于一个归档文件，我们参考现在已经比较成熟的文件格式的设计，我们不难想象，大约需要存在两个主要的部分，第一个部分为文件头，第二个部分为实际的文件数据存储区域。

文件头主要包括两个大部分：第一个部分是文件的元数据，主要包括文件标识（MARK），文件大小，归档文件最大的容纳量（主要针对定长的归档文件设计，可以方便的规定目录大小，如果是变长设计，可以存储现在已经填充的数据量或者也可以自行设计）以及文件的校验码。

第二个部分就是我们的索引列表或者说是目录区，主要记录的是现在已经存放的文件的基础信息。包括文件名、文件大小、文件实际数据在数据区的存储位置（或相对于整个文件的偏移量），文件校验码，以及文件标识。对于文件标识主要是出于效率考虑，用它可以标识这一块存储空间的状态，是否被占用，或者是已经被删除，对于大规模归档文件，我们无需每删除一个文件就将整体后面的数据前移。这样会浪费大量的时间进行文件操作。因此我们可以只对该区域进行一个标识，当下次存储文件时，如果有比该区域小的文件可以直接覆盖存储。这样可以达到平衡时间和效率的问题。如果感兴趣的同学还可以自行写一个整理函数，对空间碎片进行整理以节约空间使用。

至此，我们的文件头基本结构设计完成了。

接着我们进行数据区的设计。

数据区的设计也可以是多种多样的。其中最简单的存储方式就是顺序存储，因为之前有设计文件偏移量，顺序存储也不会造成文件错乱。我在这里就以最简单的顺序存储为例。当然，对于学有余力的同学。还可以进行一些拓展。比如将数据区划分为 4k 大小的区块，然后文件安装区块存储。这也是一种设计方案，感兴趣的同学可以自己探究一下。

细心的同学肯定不难发现。我们虽然设计完了文件头部的基本结构，但是还有一些细节我们没有处理，我们现在只是知道我们的头文件中需要包含些什么数据，但是每个数据所占用的大小和文件的存储方式都还没有进行一个细节的设计。

现在我们已经知道了我们的每一个部分所需要包含的数据，那么接下来我们就可以对我们的程序所包含的类进行一个简单的设计。

在进行面向对象设计的时候，我们可以遵从从小到大的原则，也可以是从大到小。这个取决于个人习惯。

我个人喜欢从大。那么我们首先考虑，我们整个归档文件这个类里，主要包含文件头和数据区两个部分，数据区实际不用读入内存，我们可以用一个文件指针来解决，那么文件头的部分我们就可以使用一个文件头对象来解决。

这样的话我们就可以首先写出我们的最基本的 fileSet 类。它只有两个必须的成员，一个是指向我们实际归档文件的文件指针。另一个是一个指向文件头类的指针。除了这两个必须的成员外，其余的部分大家可以根据自己的需求来设计。比如我们经常需要在读文件的时候用到文件的头部长度，而且这个量是根据归档文件的容量来确定的，它不会被改变。所以我们就可以使用一个整形成员 headerLength 来存储这个变量。

现在我们就可以根据我们的思考写出如下的类原型：

```
1  class FileSet
2  {
3  public:
4    FileSet();
5    ~FileSet();
6
7    SetHeader* getHeader();
8    void setHeader(SetHeader * newHeader);
9
10   FILE* getFilePoint();
11   void setFilePoint(FILE* newFilePoint);
12
13   int getHeaderLength();
14   void setHeaderLength(int newHeaderLength);
15
16 private:
17   SetHeader * header;
18   int headerLength;
19   FILE* fp;
20 };
```

那么接下来我们可以来设计文件头（SetHeader）类，根据刚才的考虑，文件头包含两个部分，一个是元数据，是一些基本属性的数据。还有一个就是文件索引列表。

对于元数据，我们可以逐一思考，首先对于文件标识（MARK）我们可以用一个特定的

串或者整数来确定，那么我们可以用用一个字符数组来标识，比如在这个实例中，我们把 MARK 规定为一个 4 字节的固定的短字符串 “HIT-”。接下来考虑我们要存储这个归档文件的大小。文件大小是一个非负数据，我们可以用无符号数来存储。如果用 unsigned int 来记录文件大小大约只能记录总长度不超过 2^32-1 字节的文件，也就是总大小 4G 多，这个在实际使用中显然是不够用的。所以我们可以使用__int64 类型的数据来记录整个归档文件大小。接下来我们需要记录我们的归档文件的容量大小。出于方便考虑我们可以使用一个 int 类型的数字来记录这个值，但是在这里也有一些小的窍门告诉大家，我们也可以只用一个 byte 来记录容量大小，只不过此时这个 byte 存储的是一个标志，比如它记录的值 n 代表 2^(10n)或者可以用单个字母 m 表示百万……诸如此类，大家可以自行设计。我在这个实例中使用_int64 来表示这个数据，需要占用 8bytes 空间。最后我们需要一个空间来存储我们的校验码。一般校验码我们可以后很多算法。比如简单的 SHA1 还有现在很常用的 MD5 等。计算刚才已经有的数据，可以知道 4+8+4 刚好 16bytes。这样的话我们的校验码只要 16byte 也就是 128bits 就可以了。在示例中我们使用 SHA1 校验算法。有些同学可能会问 SHA1 校验算法最后得出的是一个 20bytes 的校验串，我们怎么塞到一个 16bytes 的空间里呢？其实非常简单。我们只要截取前 16byte 就可以了。最后我们还需要在这个类中加入一个用于盛放文件标签的列表。我在这里使用了一个指针列表指针。在这里的处理上线性表是最简单的处理方法。易于编程实现，但是它的空间占用和搜索用时都不是最佳的。对于数据结构和算法比较熟悉的同学，可以使用其他更优的数据结构去替换它。比如使用链表可以方便拓展，但是不能提高检索效率。如果使用 B+树可以兼顾效率和空间利用，便于拓展，但是编程实现较为困难。

基于如上设计，我们可以写出这个类的原型了：

```
22  class SetHeader
23  {
24  public:
25      SetHeader();
26      ~SetHeader();
27
28      char* getSetMark();
29      void setSetMark(char* newMark);
30
31      __int64 getFileSize();
32      void setFileSize(__int64 newFileSize);
33
34      int getMaxFileNumber();
35      void setMaxFileNumber(int newSetMaxNumber);
36
37      char* getChecksum();
38      void setChecksum(char* newChecksum);
39
40      FileTag** getFileTagsList();
41      void setFileTagsList(FileTag** newFileTags);
42
43  private:
44      char setMark[4];
45      __int64 fileSize;
46      int maxFileNumber;
47      char checksum[16];
48      FileTag *(*fileTagsList);
49  };
```

完成了文件头的设计，我们接下来该进行文件标签类（FileTag）的设计，这个标签主要用于作为节点存在于文件索引目录中，可以简单的使用结构体或者类。为了功能强大我们使用类进行数据封装。

对于文件标签我们需要保存的就是我们要归档的小文件的元数据。首先我们需要保存文件名，文件名是一个典型的字符串数据，但是我们在存储的时候需要给它规定一个合理的大小，这个大小不能过于浪费空间，但是还要保证能装得下足够长的文件名，进阶需求就是需要保存文件目录，我们可以通过在文件名中加入“/”来模拟路径。这个作为进阶要求，学有余力的同学可以自行探究。经过查资料我们可以发现不论是在 windows 平台还是在 Linux 平台下，文件名都不会超过 255bytes。所以我们在处理文件名的时候只需要开辟一块 256bytes 的空间即可。接下来我们需要一个区域来保存文件的标识。这个可以使用一个 byte 来解决，也可以通过一个整形数据来存放。在示例中我选择了用一个 int 来存放，这样可以保证后续的可拓展性。接下来我们还需要一个 int 来保存我们这个小文件的大小。由于我们做的是小文件的归档，这里使用 int 类型足够保存我们的文件大小。最后我们还需要保存我们文件的索引值，也就是在数据区中的存储位置，我们可以用首字节相对于文件首字节的偏

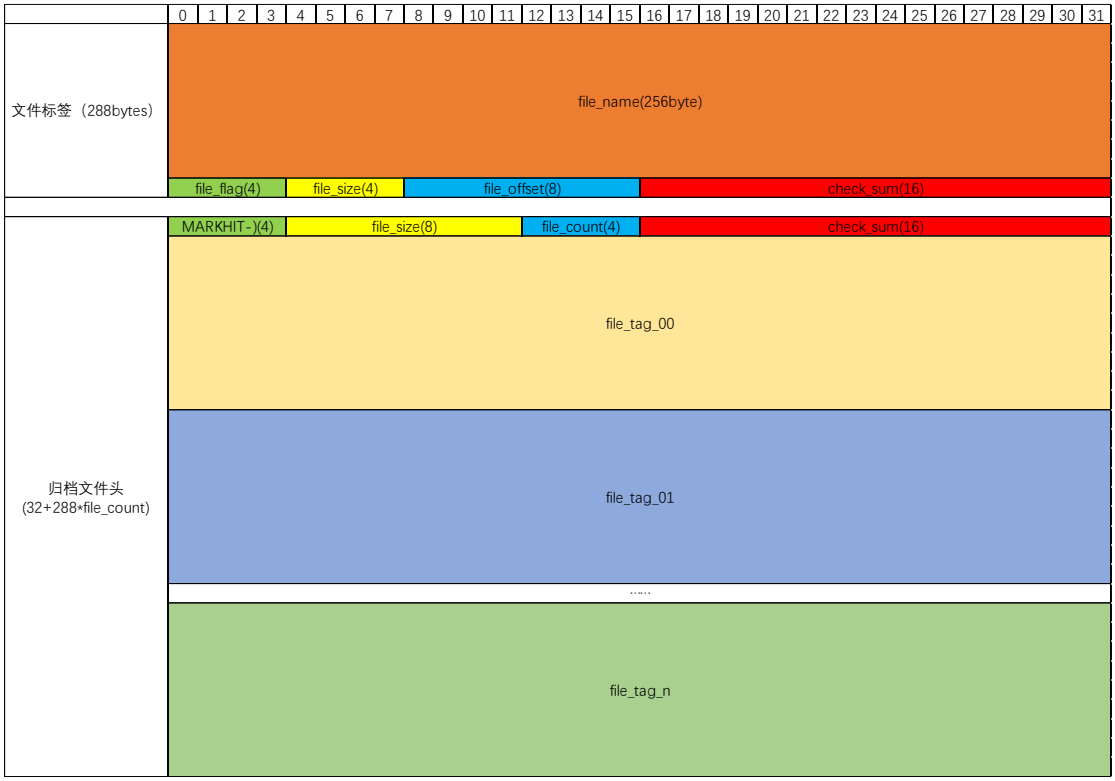
移量量表示。由于我们源文件大小可能超过 4G，所以此处我们应该与源文件大小相匹配，使用一个__int64 的数据来表示我们的数据偏移量大小。经过计算可以发现我们使用的空间为 4+4+8 刚好也是 16bytes。所以我们的校验码依然可以使用 16bytes。

根据以上设计，我们可以设计出如下类原型：

```
51  class FileTag
52  {
53  public:
54    FileTag();
55    ~FileTag();
56
57    char* getFileName();
58    void setFileName(char* newFileName);
59
60    int getFileFlag();
61    void setFileFlag(int newFileFlag);
62
63    int getFileSize();
64    void setFileSize(int newFileSize);
65
66    __int64 getFileOffset();
67    void setFileOffset(__int64 newFileOffset
68
69    char* getChecksum();
70    void setChecksum(char* newChecksum);
71
72    char* object2bytes();
73
74  private:
75    char fileName[256];
76    int fileFlag;
77    int fileSize;
78    __int64 fileOffset;
79    char checksum[16];
80  };
```

至此，我们几个常用类的存储数据存储问题基本设计完了，接下来我们考虑的就是数据持久化的问题，即我们该如何把这些数据写入一个文件，而且可以很轻松的从文件中取出数据构建出我们的类。那么我们就需要根据现有的设计来设计文件存储协议。

这个工作我们可以使用 Excel 来完成。



其中归档文件头中的每一个 file_tag 就是上面的一个文件标签。

至此，整个实验最重要的一个部分已经完成了，就是我们的数据存储结构设计。

接下来我们要做的就是编写持久化方法，也就是如何把一个对象变成我们如上规定的字符序的一个串。其实也就是实现一个所谓的 object2bytes 方法。相对应的还有需要我们考虑该如何实现一个 bytes2object 方法。

其实这里没有什么难度，需要的就是大家认真和仔细。尤其是在数值的处理上。我们最好是自行规定数据的存储方式。大端序存储或小端序存储。这样就不会因为平台原因而造成数据不统一的困惑了。我们需要实现 int2bytes 和 long2bytes 函数，对应的 bytes2int 和 bytes2long。在这里就不给大家展示具体的代码了。还请大家自行探索。

在这里我以 SetHeader 类的 object2bytes 来作为示例给大家展示如何实现一个序列化方法：

```
char * SetHeader::object2bytes()
{
  char* result = new char[32 + maxFileNumber * 288];
  memset(result, 0, 32 + maxFileNumber * 288);

  memcpy(result, setMark, 4);
  memcpy(result + 4, long2bytes(fileSize), 8);
  memcpy(result + 12, int2bytes(maxFileNumber), 4);
  memcpy(result + 16, checksum, 16);

  for (int i = 0; i < maxFileNumber; i++)
  {
    memcpy(result + 32 + i * 288, fileTagsList[i]->object2bytes(),
  }
  return result;
}
```

可以看出其实非常简单，我们只要开辟一块大小足够的空间然后按照既定的顺序把我们的数据写入即可。对于内部的对象成员，我们只要调用其 object2bytes 方法即可。

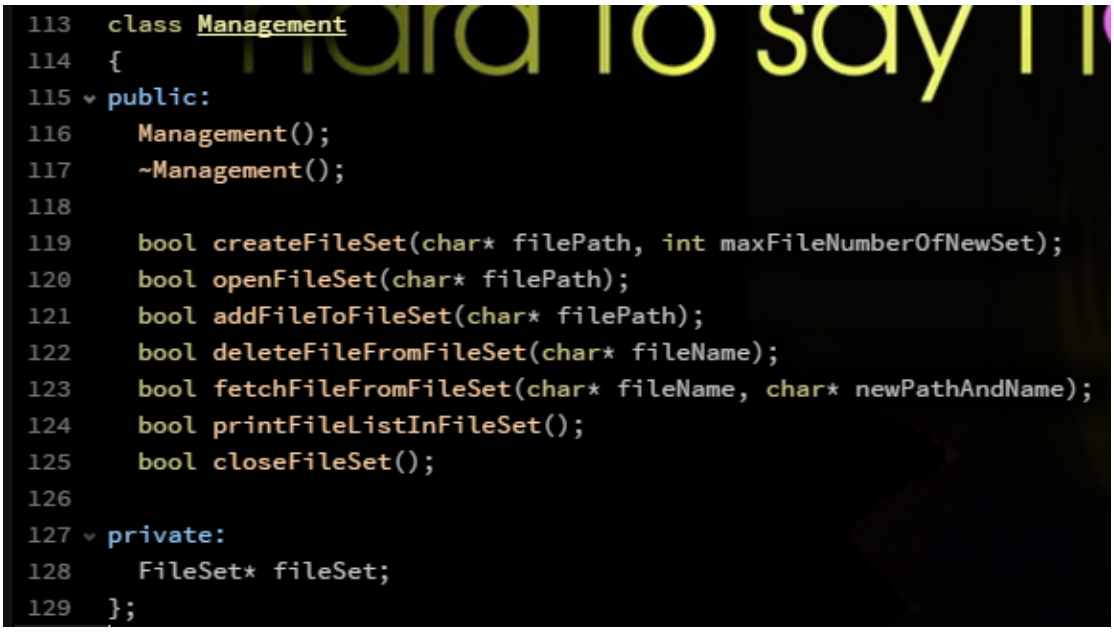
对于反序列化工作，我们可以使用构造函数还完成：

```
 99  SetHeader::SetHeader(char * bytes)
100 {
101     memcpy(setMark, bytes, 4);
102     fileSize = bytes2long(strsub(bytes, 4, 8));
103     maxFileNumber = bytes2int(strsub(bytes, 12, 4));
104     memcpy(checksum, bytes + 16, 16);
105
106     fileTagsList = new FileTag*[maxFileNumber];
107     for (int i = 0; i < maxFileNumber; i++)
108     {
109       fileTagsList[i] = new FileTag(strsub(bytes, 32 + 288 * i, 288));
110     }
111 }
```

有了这些示例，我相信大家可以很快完成其他几个类的序列化和反序列化工作了。

在完成这些基础的工作之后，我们就可以来实现我们这个项目所要求的具体功能了。根据项目的需求，我们可以知道，我们需要实现的不是一个归档文件，而是一个归档文件的管理器，这个管理器具有对创建、打开、关闭归档文件的功能。它也可以调用归档文件类的方法来实现向归档文件中添加、删除、取出数据。简单的说我们就需要实现这样的功能。

那么我们就可以根据刚才设计的管理器功能来实现这个管理器：



```
113  class Management
114  {
115  public:
116      Management();
117      ~Management();
118
119      bool createFileSet(char* filePath, int maxFileNumberOfNewSet);
120      bool openFileSet(char* filePath);
121      bool addFileToFileSet(char* filePath);
122      bool deleteFileFromFileSet(char* fileName);
123      bool fetchFileFromFileSet(char* fileName, char* newPathAndName);
124      bool printFileListInFileSet();
125      bool closeFileSet();
126
127  private:
128      FileSet* fileSet;
129  };
```

接下来我们需要做的事情就是逐一实现里面的功能。

创建归档文件：

这个功能非常简单，我们只要根据指定的路径和设定的最大容量来调用 FileSet 类的构造函数就可以了。现在你可能要问了，我们刚才写过这个构造函数吗？显然是没有的。这就是我们现在所需要实现的。

对于这个构造函数，没有什么特别要注意的地方，主要就是在文件操作上有一些小的技巧，比如我在做这个的时候，我选择一开始就把 FileSet 对象的 SetHeader 对象也初始化好，这样的话似乎又需要我们多写一个构造函数了。没错……不仅仅是这样，在构造 SetHeader 的时候，我们可能也需要将里面 FileTag 列表里的每一个 FileTag 对象也初始化好。你可能不经要问，我现在还没有往里面写文件，那么我为什么要花大把的时间来做这个看似没有意义的初始化所有 FileTag 的工作呢？原因很简单，这样做是为了归一，为了在关闭文件集合写入文件的时候操作统一。我们可以想象，对于已经保存数据的部分我们可以直接调用 FileTag 的 object2bytes 方法来将其变成一个 bytes 串方便了向文件中保存。但是如果没有保存数据的部分，就是一个空的指针。在编程的时候我们就不得不去进行一些额外的判断和处理，这样的话无疑增加了代码的复杂度，同时也提升了出现 bug 的可能性。所以我们可以通过牺牲一点初始化时的效率来保证我们程序的正确性和使用时的方便性。

在这里的细节处理上，我选择给空的 FileTag 对象填充 0 数据，并将其 flag 标记为 0，表示为未使用的文件标签。建立完之后将整个头部信息写入文件，把文件指针留在文件末尾为以后插入小文件做准备。至于文件头的更新操作我们可以放在关闭文件集合时进行。根据这样的思路，我们就可以很容易的写出来三个对应的构造函数，同样在这里我不展示代码了，把这个工作留给大家进行探究。

打开归档文件：

这个工作同样非常简单，我们只要使用文件操作打开相应的文件，并且读出文件头调用相对应的 bytes2object 方法，即可轻松的完成整个 FileSet 的打开操作。在这里同样有一些细节需要大家注意。在打开文件后我们无需把整个文件读入内存。我们不妨回过头看看我们之前设计的归档文件的元数据部分，它是整个文件的最开始。占用了 32 个字节，通过这 32

个字节，我们可以进行以下工作：首先就是通过头 4 个字节来确定，我们将要打开的这个文件是不是我们的程序所需要的文件。然后根据最后 16bytes 的校验码来确认文件的完整性。这里有一个小小的细节，就是我们在计算校验码的时候是不包含元数据的这 32bytes 的，所以在进行校验检验的时候我们一定也不能忘了这个细节，否则在验证的时候就无法得出正确的结果了。这 32bytes 中同样也包含另外一个十分重要的信息，就是我们这个文件集合最多容纳多少个小文件，获取到这个信息我们就可以算出文件头的大小，也同时就可以知道我们该给这些索引目录在内存中开辟多少空间了。

在确定完这些事情之后我们可以通过我们通过读取文件，取得整个文件头，然后调用 bytes2object 方法完成归档文件的打开操作了。

添加文件到归档文件：

核心功能但是难度较为简单。首先我们需要在文件标签列表中找到一个可用的文件标签，可用包括两种情况，第一种情况相对较为简单，就是当文件标签的标识符为 0 的时候，它是一个未使用的标签。此时我们可以把文件名、大小、文件校验码直接写入标签，并把归档文件的文件指针移到末尾，记录此时指针位置作为该文件的偏移值，然后读取待归档的小文件，写入归档文件的数据区。并将标签的标识符设置为 1。第二种情况相对复杂一些，就是当标识符为 2 的时候表示该文件已经被删除，空间可被重复利用。那么我们只需要查看该标签内存储的大小信息，如果该标签原来存储的大小信息大于等于我们现在要存储的文件，那么我们就可以将归档文件的指针移到原标签指向的位置，开始写入数据，并将新文件的文件名、大小、校验码填入文件标签，将标志位设置为 1。如果大小小于现在要存的文件，则跳过，查找下一个可用的文件标签。

从归档文件中移除文件：

这个功能是最简单的了，根据用户提供的文件名，在归档文件中找到对应的文件标签，把标志位设置为 2 即可，表示该空间可以被重复利用。我们无需从数据区将这段数据抹去或用该部分后面的数据来填充这块空白，因为这样会造成大量的文件操作严重影响运行效率。

从归档文件中取回数据：

这个功能也比较简单，直接从文件标签里找到需要取回的文件，然后根据 offset 值从数据区取回数据并且再次进行校验，看与我们之前保存的校验值是否一致，如果一致就可以将文件写入用户指定的位置。注意在此处实现的时候，针对可拓展性，有一个小小的技巧。就是我们可以给 FileSet 类实现一个 fetch 方法接收一个文件名返回该文件的全部字节。然后给 management 类实现一个 fetch 方法接收文件名和保存路径，这样的话在后期使用中，如果需要将小文件取出另存，我们就调用 management 类的 fetch 方法。如果需要直接传递给其他程序使用，那么我们可以直接调用 Fileset 类的 fetch 方法。

输出文件列表：

遍历文件标签列表，如果标签标识位为 1，那么我们就可以输出该标签中的文件名、文件大小和对应的偏移量。

关闭归档文件：

关闭归档文件功能也很简单，需要我们做的就是对 SetHeader 中的 FileTag 列表进行序列化，然后对生成的数据计算校验和，计算结束后保存在 checksum 变量中。然后调用 SetHeader 的 objects2bytes 方法，得到序列化串，将归档文件的文件指针移动到文件开始

的位置，写入刚才的序列化串。关闭文件。

整个功能差不多就是这些，接下来就需要同学们自己去实现一个主函数来进行测试，也可以使用我们提供的主函数进行测试。这个项目其实还有很多可以进行功能拓展的拓展点，接下来我和大家分享一些可以进行拓展的方向，供学有余力的同学进行探究：

1. 内存中文件索引的优化，这个在之前有所提过，我们使用的是最简单的定长对象数组，在这里建议对数据结构和算法比较熟悉的同学可以使用更高级的数据结构来优化查询的效率和提高索引的空间利用率。
2. 在文件存储方面，我们现在使用的是按字节存储，这种存储方式是不利于我们更好的利用空间的，我们可以把数据区划分为小的数据块然后在文件标签中就不需要再记录具体的文件偏移量了，我们就只需要记录文件所在的块号即可，这样设计是有利于我们硬件缓存的使用，同时也有助于提高空间的利用率。对于这个的实现，我们可以在数据块最后的位置写入下一个块的块号。这样当文件大于数据块大小的时候我们可以方便的找到下一个块的位置，在文件标签中我们只需要记录首块的块号即可。
3. 从安全角度出发，大家不难看出，我们现在所采取的策略保存文件，文件是连续保存在数据区的，这样其实是非常危险的，因为我们可以直接从数据区拿到源文件本身，那么从安全角度讲，我们可以采取乱序存储或者对数据区进行加密等手段。在这里只作为一个提示，深入的部分可以供大家自己探索。

写在最后：

这份实验指导书到了这里也就写到了最后，全篇近 7000 字。对于这份大作业，郑老师和我在设计的时候本着从实际出发做些真正有用的东西的原则，为大家定制了这个作业要求，也是希望大家通过这个作业可以得到一些真正的工程项目的训练。

特别感谢郑老师和冬冬学姐在我完成这份指导书时给我的支持。

都说授人以鱼不如授人以渔，在完成大作业的题目设计之后我用了大约一周的时间来完成整个项目以及这份指导书，虽然我写了这份作业完整的参考代码，但是我并不想在这个指导书中提供给大家，从另一个角度，我很详细的写了我在完成这个项目时的心路历程，包括了我完成整个项目时候的设计思路和实现过程。这个不是一个标准的参考答案，只是我个人做项目的一些心得。我希望通过这份指导书以及这个项目，可以让大家学习到我们一般做项目的流程和遇到一个看似很难的项目时，怎么去下手，怎么去思考，学会项目的拆分。

最后就是希望大家都能完成这个作业。愉快的暑假在向大家招手^_^

段艺

2016-7-13