File Structures

05. A. Managing Files of Records

2020. Spring

Instructor: Joonho Kwon

jhkwon@pusan.ac.kr

Data Science Lab @ PNU

datalab

data science laboratory

Slides are based on the textbook.

References

- **Textbook**

Objectives

- Extend the file structure concepts of Chapter 4:
  - Search keys and canonical forms
  - Sequential search and Direct access
  - Files access and file organization

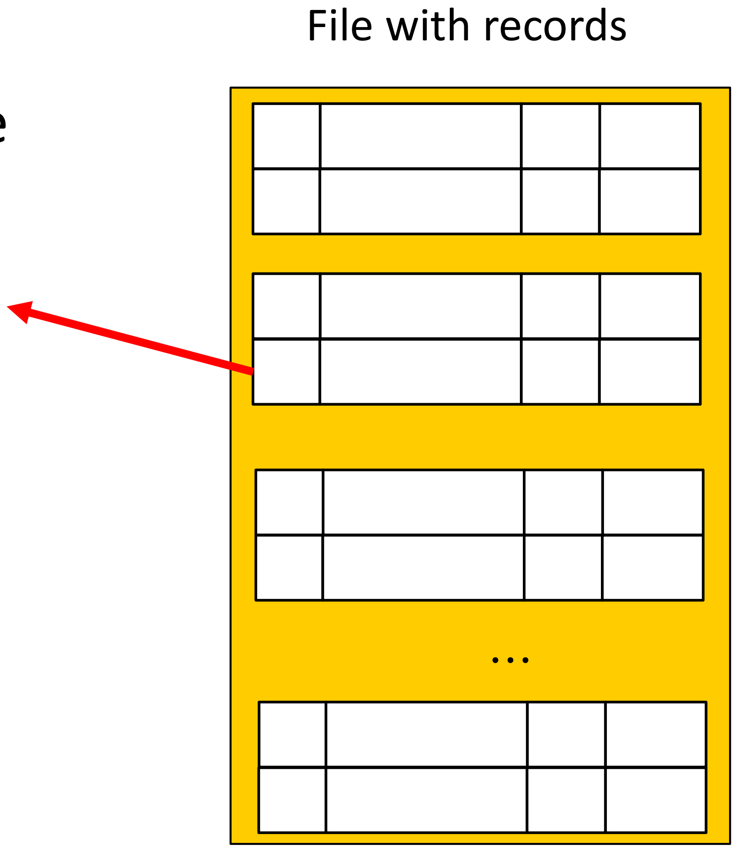
Outline

- 5.1 Record Access
- 5.2 More about Record Structures
- 5.3 Encapsulating Record I/O Ops in a Single Class
- 5.4 File Access and File Organization
- 5.5 Beyond Record Structures

- Skipped
  - 5.6 Portability and Standardization

Records access

- Goal
  - Retrieving just one specific record rather than reading all the way through file
- Key
  - used to identify the record based on the record's contents
  - is another fundamental conceptual tool



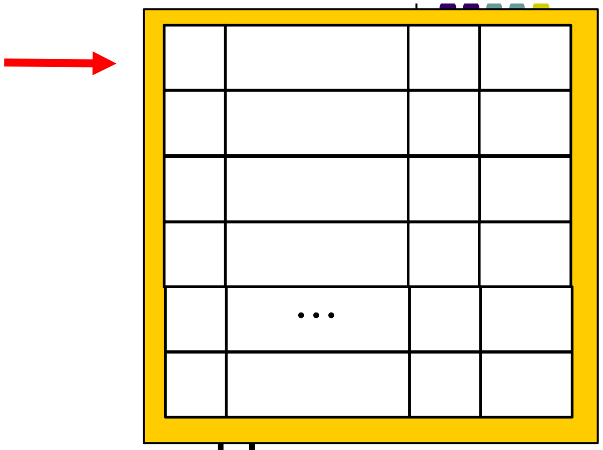
Record Key (1/2)

- Canonical form : a standard form of a key
  - e.g. Ames or ames or AMES (need conversion)
- Distinct keys
  - uniquely identify a single record
- Primary keys
  - Primary keys should be dataless (not updatable)
  - Primary keys should be unchanging
  - Example
    - a table of contents of books
    - Social-security-number: good primary key
      - but, 999-99-9999 for all non-registered aliens

Record Key (2/2)

- secondary keys
  - do not uniquely identify a record
  - Example
    - The city field in our name and address file
    - A name is perfectly fine secondary key and an important secondary key in a retrieval system

A Sequential Search


File with records

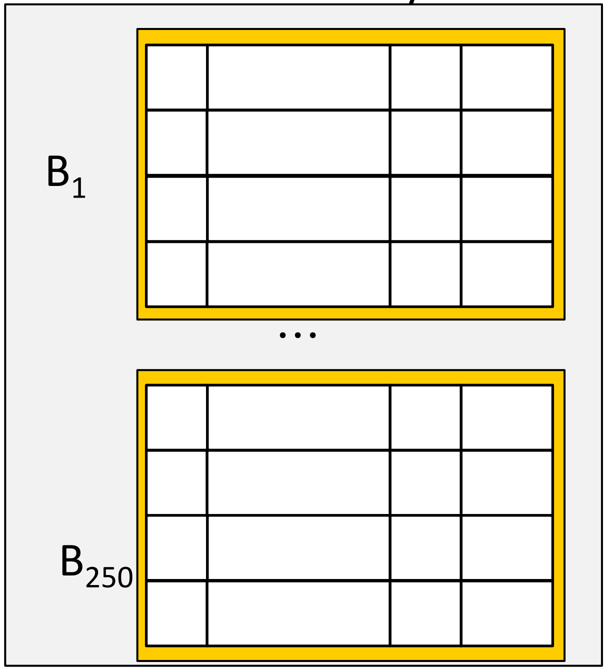
- Sequential search
  - a program that reads through the file, record by record, looking for a record with a particular key
- Evaluating performance of SS
  - Use the number of comparison
    - So small, compared with cost of a disk access
  - Instead, count low-level Read calls
    - Assume each Read() call requires a seek
  - Searching a record in a file with n records
    - $\frac{n}{2}\mathrm{READ}()$ calls

Improving sequential search performance

- record blocking
  - the major cost associated with a disk access
  - reading in a block of several records all at once and then processing that block of records in RAM

A File of 4,000 records

Each record: 512 bytes h



Unlocked sequential search with 512 byte size buffer

=> average 2,000 READ() calls

Blocked sequential search

- by blocking BF = 16 recs/block, 8K size buffer

=> Average 1/2*(400/16)= 250/2=125 READ() calls

Tools for Sequential Search

- UNIX tools for sequential processing
  - cat, wc, grep
- When sequential search is useful
  - Searching patterns in ASCII files
  - Processing files with few records
  - Searching records with a certain secondary key value

Direct Access

- Direct access
  - Obtain a record with a single seek
  - O(1) operation
- Codes for Direct access

```
int IOBuffer::DRead(istream & stream, int recref)
// read specified record
{
    stream.seekg(recref, ios::beg);
    if (stream.tellg() != recref) return -1;
    return Read(stream);
}

int IOBuffer::DWrite(ostream & stream, int recref) const
// write specified record
{
    stream.seekp(recref, ios::beg);
    if (stream.tellp() != recref) return -1;
    return Write(stream);
}
```

recref: the record reference = the byte address of the record

Extension to IOBuffer class

- Includes direct read and write methods for the base class IOBuffer
  - Using byte address of the record as the record reference
  - can add new derived classes with their own different Read and Write methods
    - inheritance and O-O design!

```
class IOBuffer
{
public:
    …
    // sequential read and write operations
    virtual int Read (istream &) = 0; //read a buffer
    virtual int Write (ostream &) const = 0; //write a buffer
    // these are the direct access read and write operations
    virtual int DRead (istream &, int recref); // read
    virtual int DWrite (ostream &, int recref) const; //
protected:
    char * Buffer; //character array to hold field values
    …
};
```

How we know the beginning of the record ?

- Method1
  - Information about record location information is stored in a separate index file
- Method2
  - Assume we have Relative Record Number (RRN)
    - a file is a sequence of records
  - RRN of a record gives its position relative to the beginning of the file

Implementation

- in C++, the application program does the calculation
  - Byte offset = $n \times r$ (n:RRN value, r: fixed record length)
  - use the seekg() and seekp() command to jump to the byte that begins the record
  - can extend the class FixedLengthBuffer with its own methods DRead and DWrite
    - interpret the recref argument as RRN instead of byte address
    - Dread() and DWrite() are defined as virtual in class IOBuffer

Outline

- 5.1 Record Access
- 5.2 More about Record Structures
- 5.3 Encapsulating Record I/O Ops in a Single Class
- 5.4 File Access and File Organization
- 5.5 Beyond Record Structures

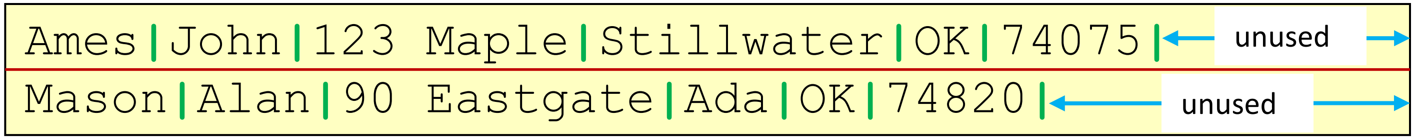
- Skipped
  - 5.6 Portability and Standardization

Choosing a record structure and record length

- The RRN method
  - gives us direct access to a record having a fixed length
  - two general approaches to field structure within a fixed-length record
    - (a) fixed-length records with fixed-length fields

```
Ames    John    123 Maple     Stillwater    OK74075
Mason   Alan    90 Eastgate   Ada           OK74820
```

    - (b) fixed-length records with variable-length fields



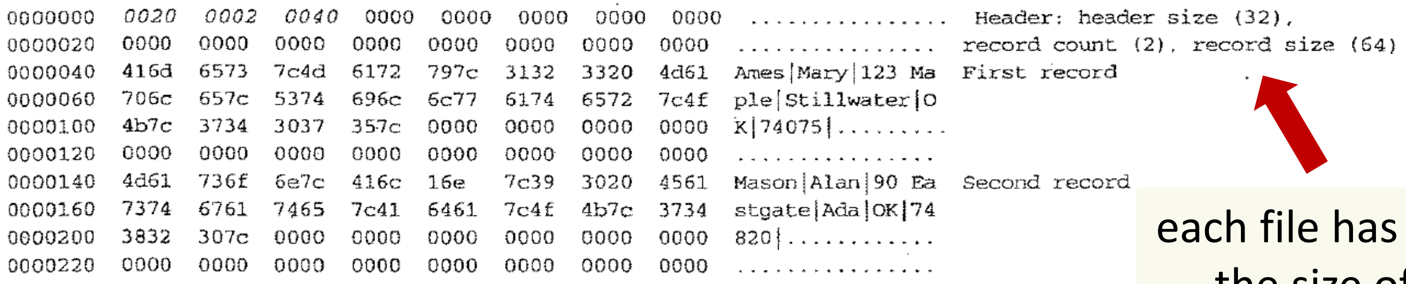
```
Ames|John|123 Maple|Stillwater|OK|74075|
Mason|Alan|90 Eastgate|Ada|OK|74820|
```

Header information (1/2)

- how to distinguish the real-data portion of the record from the unused-space portion
  - place a record-length count at the beginning of the record
  - use a special delimiter at the end of the record
  - count fields
- no single right way to implement this file structure
  - seek the solution that is most appropriate for our needs and situation
  - O-O design for customizing

Header information (2/2)

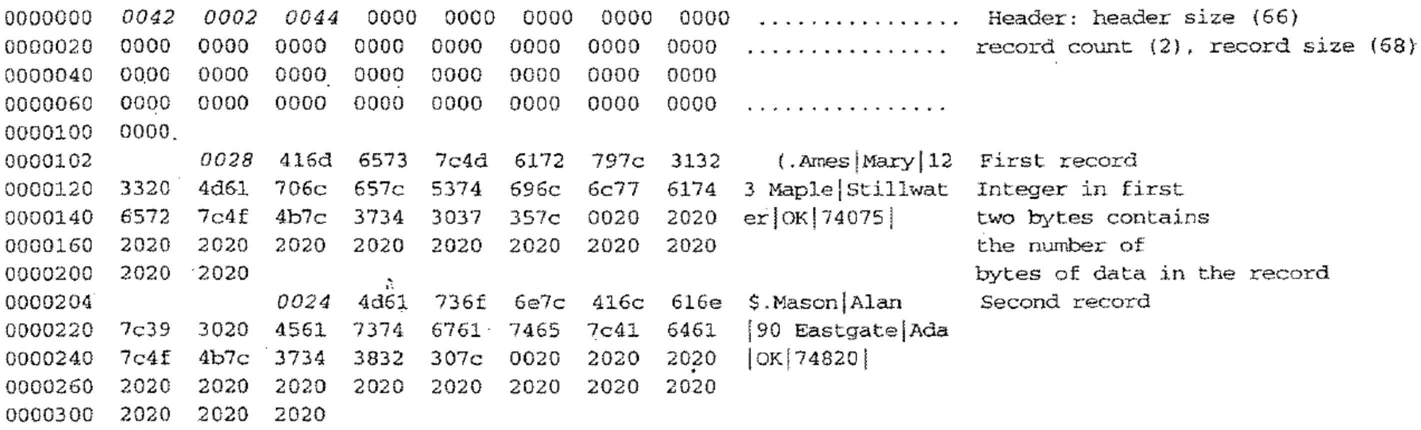
- Hex dump output
  - (a) fixed-length fields



```
0000000  0020  0002  0040  0000  0000  0000  0000  0000  ................  Header: header size (32),
0000020  0000  0000  0000  0000  0000  0000  0000  0000  ................  record count (2), record size (64)
0000040  416d  6573  7c4d  6172  797c  3132  3320  4d61  Ames|Mary|123 Ma  First record
0000060  706c  657c  5374  696c  6c77  6174  6572  7c4f  ple|Stillwater|O
0000100  4b7c  3734  3037  357c  0000  0000  0000  0000  K|74075|.........
0000120  0000  0000  0000  0000  0000  0000  0000  0000  ................
0000140  4d61  736f  6e7c  416c  16e   7c39  3020  4561  Mason|Alan|90 Ea  Second record
0000160  7374  6761  7465  7c41  6461  7c4f  4b7c  3734  stgate|Ada|OK|74
0000200  3832  307c  0000  0000  0000  0000  0000  0000  820|............
0000220  0000  0000  0000  0000  0000  0000  0000  0000  ................
```

  - (b) variable-length fields

each file has a header record
- the size of the header
- the number of records
- the size of each record



```
0000000  0042  0002  0044  0000  0000  0000  0000  0000  ................  Header: header size (66)
0000020  0000  0000  0000  0000  0000  0000  0000  0000  ................  record count (2), record size (68)
0000040  0000  0000  0000  0000  0000  0000  0000  0000
0000060  0000  0000  0000  0000  0000  0000  0000  0000  ................
0000100  0000.
0000102        0028  416d  6573  7c4d  6172  797c  3132    (.Ames|Mary|12  First record
0000120  3320  4d61  706c  657c  5374  696c  6c77  6174  3 Maple|Stillwat  Integer in first
0000140  6572  7c4f  4b7c  3734  3037  357c  0020  2020  er|OK|74075|      two bytes contains
0000160  2020  2020  2020  2020  2020  2020  2020  2020                    the number of
0000200  2020  2020                                                        bytes of data in the record
0000204              0024  4d61  736f  6e7c  416c  616e  $.Mason|Alan      Second record
0000220  7c39  3020  4561  7374  6761  7465  7c41  6461  |90 Eastgate|Ada
0000240  7c4f  4b7c  3734  3832  307c  0020  2020  2020  |OK|74820|
0000260  2020  2020  2020  2020  2020  2020  2020  2020
0000300  2020  2020  2020
```

Header records (1/2)

- Header records
  - placed at the beginning of the file to hold information about the file
    - to keep a count of the number of records in the file
    - the length of the data records
    - the date and time of the file's most recent update
  - help make a file a self-describing object
    - make the file-access software able to deal with more variation in file structures
  - a different structure between the header record and the data records

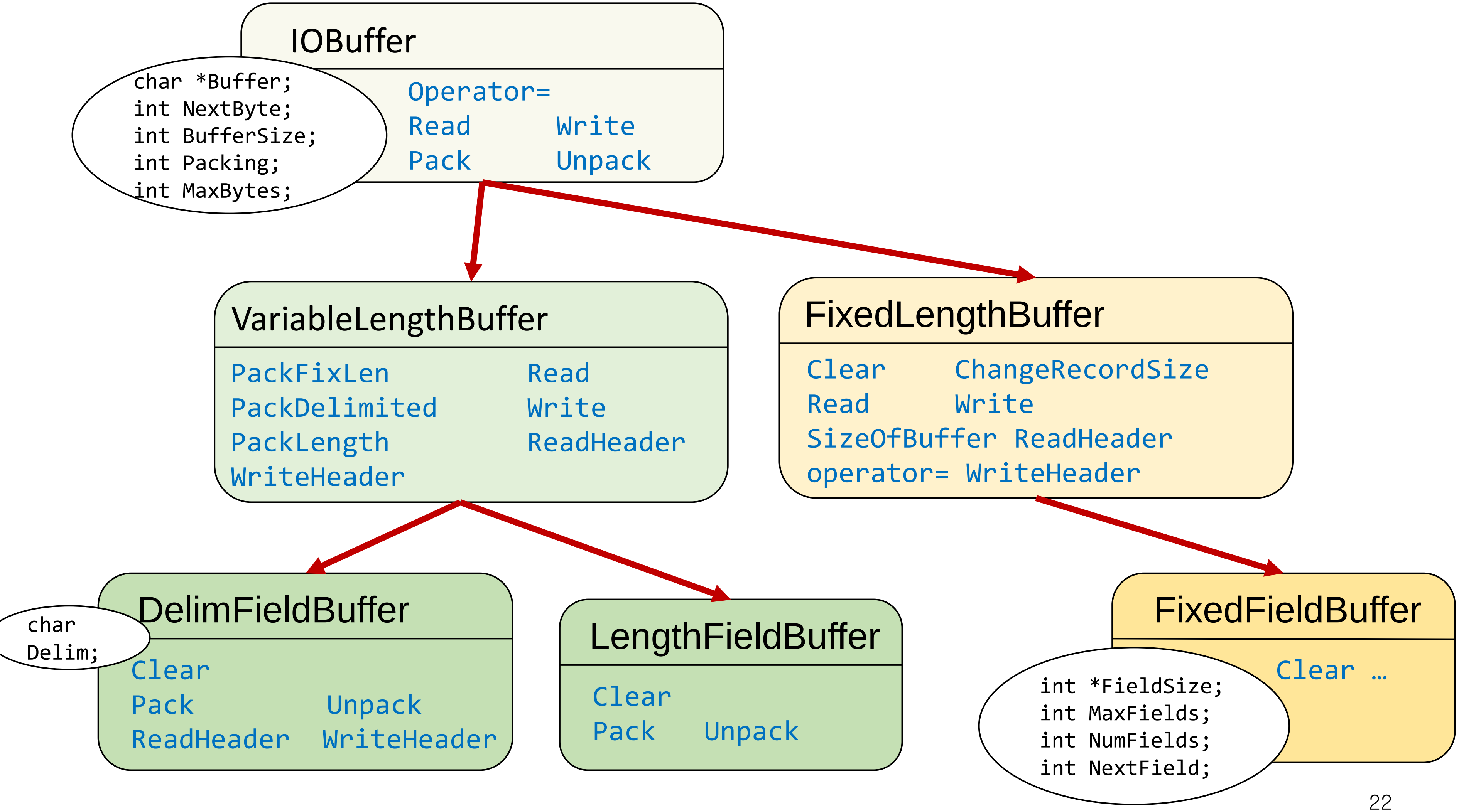
Header records (2/2)

- header records are widely used, important file design tool
  - the construction of tree-structured indexes for files
    - header records are placed at the beginning of the index → header represents the RRN of the record that is the root of the index
    - to be used for B-tree

Adding headers to buffer classes

- principle
  - each file contains a header that incorporates information about the type of objects stored in the file
- extend the class IOBuffer to include the following methods
  - the WriteHeader () method
    - add a header to a file
    - return # of bytes in the header
  - the ReadHeader () method
    - read the header and check for consistency

class hierarchy for headers



IOBuffer

```
char *Buffer;
int NextByte;
int BufferSize;
int Packing;
int MaxBytes;
```

Operator=
Read Write
Pack Unpack

VariableLengthBuffer

PackFixLen Read
PackDelimited Write
PackLength ReadHeader
WriteHeader

FixedLengthBuffer

Clear ChangeRecordSize
Read Write
SizeOfBuffer ReadHeader
operator= WriteHeader

DelimFieldBuffer

```
char
Delim;
```

Clear
Pack Unpack
ReadHeader WriteHeader

LengthFieldBuffer

Clear
Pack Unpack

FixedFieldBuffer

```
int *FieldSize;
int MaxFields;
int NumFields;
int NextField;
```

Clear …

class IOBuffer

```
class IOBuffer
{
public:
    IOBuffer (int maxBytes = 1000);

    virtual int Pack(const void * field, int size = -1) = 0;
    virtual int Unpack (void * field,int maxbytes = -1) = 0;

    // sequential read and write operations
    virtual int Read (istream &) = 0; //read a buffer
    virtual int Write (ostream &) const = 0; //write a buffer

    // these are the direct access read and write operations
    virtual int DRead (istream &, int recref); // read specified record
    virtual int DWrite (ostream &, int recref) const; // write specified record

    // these header operations return the number of bytes in the header
    virtual int ReadHeader (istream &); // write a buffer to the stream
    virtual int WriteHeader (ostream &) const; // write a buffer to the stream
protected:
    int Initialized; // TRUE if buffer is initialized
    char *Buffer; //character array to hold field values
    int BufferSize; // sum of the sizes of packed fields
    int MaxBytes; //max # of char in the buffer
    int NextByte; // index of next byte to be packed/unpacked
    int Packing;  // TRUE if in packing mode, FALSE, if unpacking
};
```

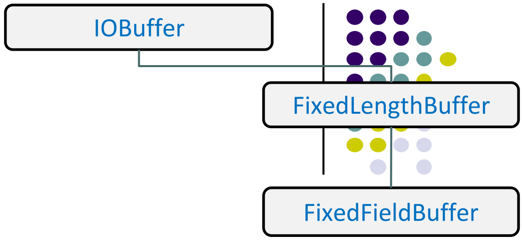
ReadHeader()/WriteHeader()

```
static const char * headerStr = "IOBuffer";
static const int headerSize = strlen(headerStr);

int IOBuffer::ReadHeader (istream & stream)
{
    char str[headerSize+1];
    stream.seekg(0, ios::beg);
    stream.read(str, headerSize);
    if (!stream.good()) return -1;
    if (strncmp(str, headerStr, headerSize)==0) return headerSize;
    else return -1;
}

int IOBuffer::WriteHeader (ostream & stream) const
{
    stream.seekp(0, ios::beg);
    stream.write(headerStr, headerSize);
    if (!stream.good()) return -1;
    return headerSize;
}
```

fixed-length buffer(1/3)



IOBuffer

FixedLengthBuffer

FixedFieldBuffer

- Class

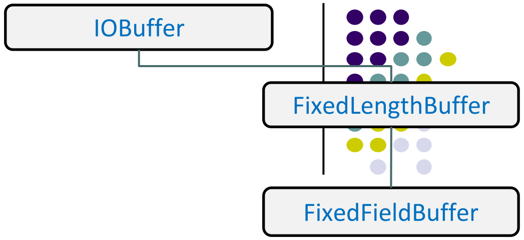
```
class FixedLengthBuffer: public IOBuffer
{
public:
    FixedLengthBuffer (int recordSize = 1000);
    // copy constructor
    FixedLengthBuffer (const FixedLengthBuffer & buffer);

    void Clear (); // clear values from buffer
    int Read (istream &);
    int Write (ostream &) const;

    int ReadHeader (istream &); // read header from stream
    int WriteHeader (ostream &) const; // write a header to the stream

    void Print (ostream &) const;
    int SizeOfBuffer () const; // return size of buffer
protected:
    int Init (int recordSize);
    int ChangeRecordSize (int recordSize);
};
```

fixed-length buffer(2/3)



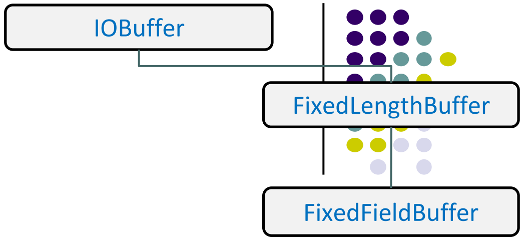
- WriteHeader()
  - write the string IOBuffer at the beginning of the file
  - add the string “Fixed” and the record size

```
static const char * headerStr = "Fixed";
static const int headerStrSize = strlen (headerStr);

int FixedLengthBuffer::WriteHeader (ostream & stream) const
{
    int result;
    if (!Initialized) return -1; // cannot write unitialized buffer
    // write the parent (IOBuffer) header
    result = IOBuffer::WriteHeader (stream);
    if (!result) return -1;
    // write the string "Fixed"
    stream.write(headerStr, headerStrSize);
    if (!stream.good ()) return -1;
    // write the record size
    stream.write((char *)&BufferSize, sizeof(BufferSize));
    if (!stream.good ()) return -1;
    return stream.tellp ();
}
```

```
// A header consists of the
//   IOBUFFER header
//   FIXED             5 bytes
//   record size       2 bytes
```

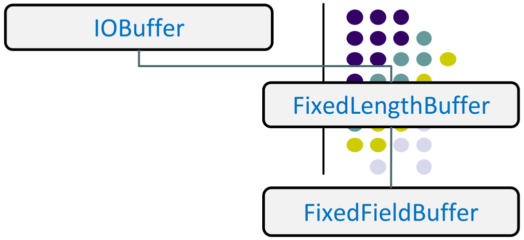
fixed-length buffer(3/3)



- ReadHeader()
  - read the record size from the header
  - check the buffer size

```
int FixedLengthBuffer::ReadHeader (istream & stream)
{
    char str[headerStrSize+1];
    int recordSize;
    int result;
    // read the IOBuffer header
    result = IOBuffer::ReadHeader(stream);
    if (result < 0) return -1;
    // read the string "Fixed"
    stream.read (str, headerStrSize);
    if (!stream.good()) return -1;
    if (strncmp(str, headerStr, headerStrSize) != 0) return -1;
    stream.read ((char*)&recordSize, sizeof(recordSize));
    if (Initialized) // check header for consistency
    {
        if (recordSize != BufferSize) return -1;
    }
    // else initialize the buffer from the header
    ChangeRecordSize(recordSize);
    return stream.tellg();
}
```

Fixed field buffer (1/4)



- Class

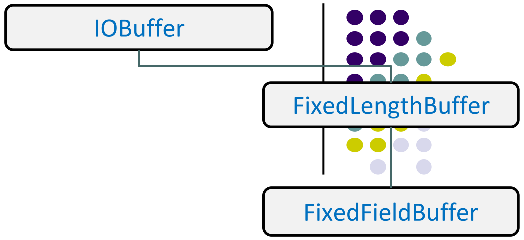
```
class FixedFieldBuffer: public FixedLengthBuffer
{
 public:
    FixedFieldBuffer (int maxFields, int RecordSize = 1000);
    FixedFieldBuffer (int maxFields, int * fieldSize);
    // initialize all fields at once
    void Clear (); // clear values from buffer


    …
    int ReadHeader (istream &); // write a buffer to the stream
    int WriteHeader (ostream &) const; // write a buffer to the stream

    int Pack (const void * field, int size = -1);
    int Unpack (void * field, int maxBytes = -1);
    void Print (ostream &) const;
    int NumberOfFields () const; // return number of defined fields

 protected:
    int * FieldSize; // array to hold field sizes
    int MaxFields; // maximum number of fields
    int NumFields; // actual number of defined fields
    int NextField; // index of next field to be packed/unpacked
};
```

Fixed field buffer (2/4)



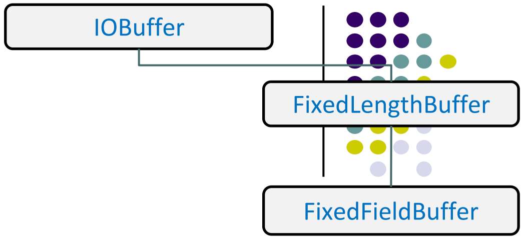
- WriteHeader()

```
static const char * headerStr = "Field";
static const int headerStrSize = strlen (headerStr);

int FixedFieldBuffer :: WriteHeader (ostream & stream) const
{
    int result;
    if (!Initialized) return -1; // cannot write unitialized buffer
    // write the parent (FixedLengthBuffer) header
    result = FixedLengthBuffer::WriteHeader (stream);
    if (!result) return -1;
    // write the header string
    stream.write (headerStr, headerStrSize);
    if (!stream.good()) return -1;
    // write the record description
    //cout << "packing numfields "<<NumFields<<endl;
    stream.write ((char*)&NumFields, sizeof(NumFields));
    for (int i = 0; i < NumFields; i ++)
    {
        //cout << "packing fieldsize "<<FieldSize[i]<<endl;
        stream.write ((char*)&FieldSize[i], sizeof(FieldSize[i]));
    }
    if (!stream) return -1;
    return stream.tellp ();
}
```

```
// A header consists of the
//  FixedLengthBufferheader
//  Field               5 bytes
//  Variable sized record of length fields
//  that describes the file records
//  Header record size  2 bytes
//  number of fields          4 bytes
//  field sizes         4 bytes per field
```

Fixed field buffer (3/4)


IOBuffer

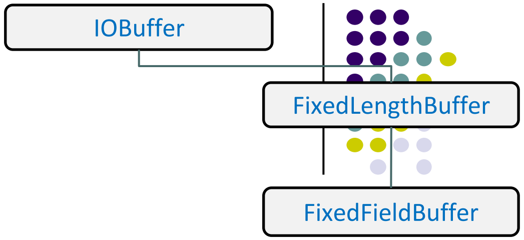
FixedLengthBuffer

FixedFieldBuffer

- ReadHeader()

```
int FixedFieldBuffer::ReadHeader(istream & stream)
{
    char * str = new char[headerStrSize+1];
    int numFields, *fieldSize;
    int result;
    // read the FixedLengthBufferheader
    result = FixedLengthBuffer::ReadHeader(stream);
    if (result < 0) return -1;
    // read the header string
    stream.read (str, headerStrSize);
    if (!stream.good()) return -1;
    if (strncmp (str, headerStr, headerStrSize) != 0) return -1;
    // read the record description
    stream.read ((char*)&numFields, sizeof(numFields));
    if (!stream) return -1; // failed to read numFields
    fieldSize = new int[numFields];
    for (int i = 0; i < numFields; i ++)
    {
        stream.read ((char*)&fieldSize[i], sizeof(fieldSize[i]));
    }
```

Fixed field buffer (4/4)



- ReadHeader() cont’d

```
    if (Initialized) // check header for consistency
    {
        if (numFields != NumFields) return -1;
        for (int j = 0; j < numFields; j ++)
            if (fieldSize[j] != FieldSize[j]) return -1;
        return stream.tellg(); // everything matches
    }
    // else initialize the buffer from the header
    Init (numFields, fieldSize);
    return stream.tellg();
}
```

TestBuffer() in BufFileTest.cpp (1/2)

- Modified TestBuffer() codes from Ch4.3-4.4

```
void testBuffer (IOBuffer & Buff, char * myfile)
{
    Person person;
    int result;
    int recaddr1, recaddr2, recaddr3, recaddr4;

    // Test writing
    //Buff . Print (cout);
    ofstream TestOut (myfile,ios::out);
    result = Buff . WriteHeader (TestOut);
    cout << "write header "<<result<<endl;
    MaryAmes.Pack (Buff);
    //Buff . Print (cout);
    recaddr1 = Buff.Write (TestOut);
    cout << "write at "<<recaddr1<<endl;
    JKwon.Pack(Buff);
    recaddr2 = Buff.Write (TestOut);
    cout << "write at "<<recaddr2<<endl;
    AlanMason.Pack (Buff);
    //Buff.Print (cout);
    recaddr3 = Buff.Write (TestOut);
    cout << "write at "<<recaddr3<<endl;
    TestOut.close ();
```

```
// Test Writing
WriteHeader() is used
```

TestBuffer() in BufFileTest.cpp (22)

```
    // test reading
    ifstream TestIn (myfile, ios::in);
    result = Buff.ReadHeader (TestIn);
    cout <<"read header "<<result<<endl;
    Buff.DRead (TestIn, recaddr3);
    //Buff.Read(TestIn);
    person.Unpack (Buff);
    person.Print (cout, "Third record:");
    Buff.DRead (TestIn, recaddr1);
    //Buff.Read(TestIn);
    person.Unpack (Buff);
    person.Print (cout, "First record:");
    Buff.DRead (TestIn, recaddr2);
    //Buff.Read(TestIn);
    person.Unpack (Buff);
    person.Print (cout, "Second record:");
}
```

// test reading ReadHeader() and Dread() are used

Check the output file

- Check the file using hexdump
  - `$hexdump -C fixlen.dat`

```
00000000  49 4f 42 75 66 66 65 72  46 69 78 65 64 3d 00 00  |IOBufferFixed=..|
00000010  00 46 69 65 6c 64 06 00  00 00 0a 00 00 00 0a 00  |.Field..........|
00000020  00 00 0f 00 00 00 0f 00  00 00 02 00 00 00 09 00  |................|
00000030  00 00 41 6d 65 73 00 00  00 00 00 00 4d 61 72 79  |..Ames......Mary|
00000040  00 00 00 00 00 00 31 32  33 20 4d 61 70 6c 65 00  |......123 Maple.|
00000050  00 00 00 00 00 53 74 69  6c 6c 77 61 74 65 72 00  |.....Stillwater.|
00000060  00 00 00 00 4f 4b 37 34  30 37 35 00 00 00 00 4b  |....OK74075....K|
00000070  77 6f 6e 00 00 00 00 00  00 4a 00 00 00 00 00 00  |won......J......|
00000080  00 00 00 34 31 36 00 00  00 00 00 00 00 00 00 00  |...416..........|
00000090  00 00 42 75 73 61 6e 00  00 00 00 00 00 00 00 00  |..Busan.........|
000000a0  00 42 75 34 31 36 00 00  00 00 00 00 4d 61 73 6f  |.Bu416......Maso|
000000b0  6e 00 00 00 00 00 41 6c  61 6e 00 00 00 00 00 00  |n.....Alan......|
000000c0  39 30 20 45 61 73 74 67  61 74 65 00 00 00 00 41  |90 Eastgate....A|
000000d0  64 61 00 00 00 00 00 00  00 00 00 00 00 00 4f 4b  |da............OK|
000000e0  37 34 38 32 30 00 00 00  00                       |74820....
```

Outline

- 4.5 An Object-Oriented Class for Record Files
  - This will be explained when we learn 5.2.3
- 5.2 More about Record Structures
  - 5.2.3 Adding Headers to C++ Buffer Class

- 5.3 Encapsulating Record I/O Ops in a Single Class
- 5.4 File Access and File Organization
- 5.5 Beyond Record Structures
- Skipped
  - 5.6 Portability and Standardization

An object-oriented class for record files

- We know how to transfer objects to and from files

→ Encapsulate the knowledge in a class that supports all of our file operations

- class BufferFile
  - Supports manipulation of files that are tied to specific buffer types

Class BufferFile at ch4.5/ch5.2

```
class BufferFile
{
   public:
    BufferFile (IOBuffer &); // create with a buffer

    int Open (char * filename, ios_base::openmode MODE); // open an existing file
    int Create (char * filename, ios_base::openmode MODE); // create a new file
    int Close ();
    int Rewind (); // reset to the first data record
    // Input and Output operations
    int Read (int recaddr = -1);
    int Write (int recaddr = -1); // write the current buffer contents
    int Append (); // write the current buffer at the end of file

    // Access to IOBuffer
    IOBuffer & GetBuffer ();

   protected:
    IOBuffer & Buffer;
    fstream File;
    int HeaderSize; // size of header
    int ReadHeader ();
    int WriteHeader ();
};
```

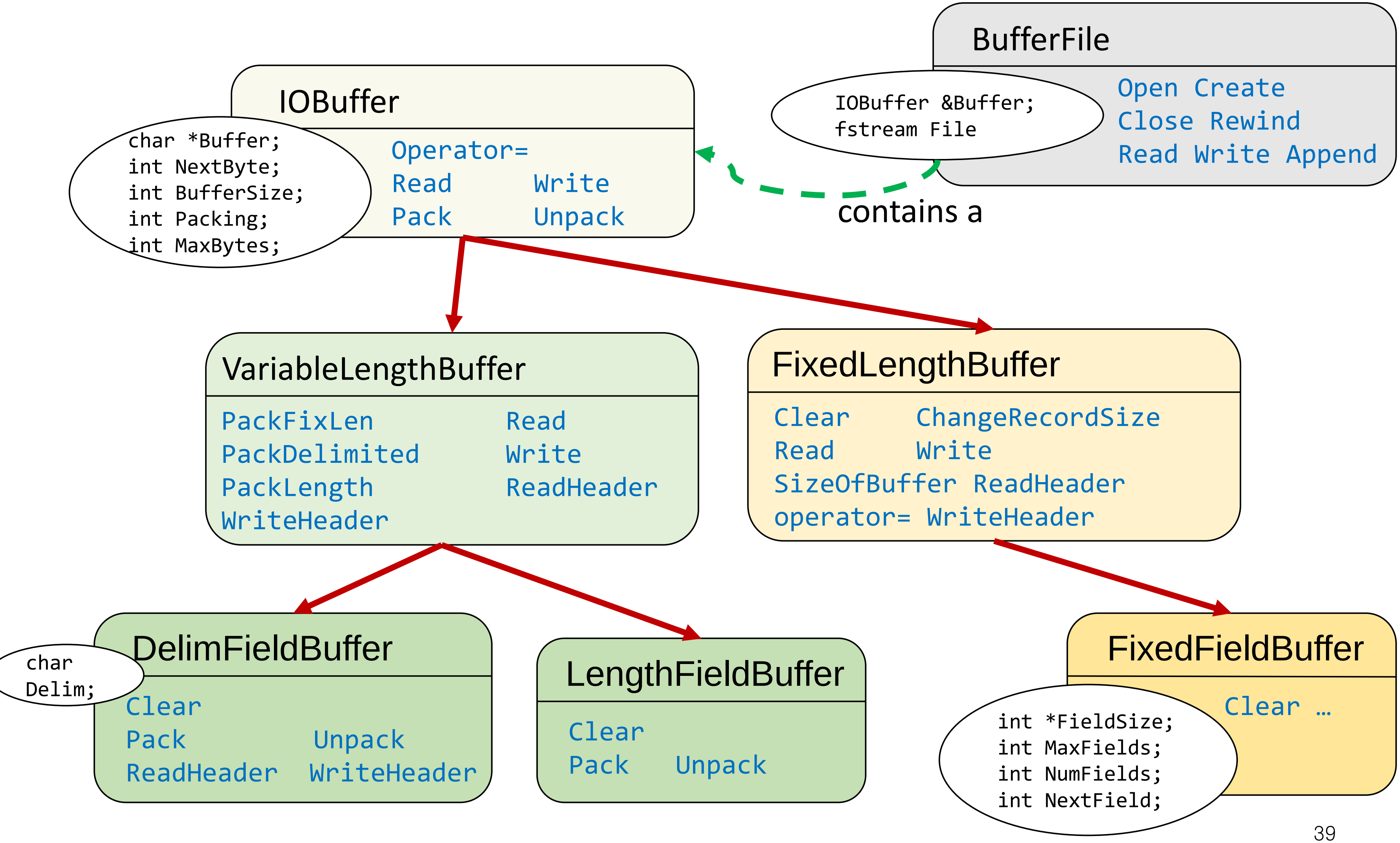
An object of class BufferFile

- can be created and attached to os file
  - Each read or write is performed using the same buffer
    - Each record is guaranteed to be of the same basic type

```
DelimFieldBuffer buffer;
BufferFile file(buffer);
file.Open(myfile);
file.Read();
buffer.Unpack(myobject);
```

- Example
  - A buffer is created
  - Bufferfile object file is attached to it
  - Open and Read methods are called
    - After Read, buffer contains the packed record
  - Buffer.Unpack puts the record into myobject

class hierarchy for BufferFile



BufferFile

IOBuffer &Buffer;
fstream File

Open Create
Close Rewind
Read Write Append

contains a

IOBuffer

char *Buffer;
int NextByte;
int BufferSize;
int Packing;
int MaxBytes;

Operator=
Read Write
Pack Unpack

VariableLengthBuffer

PackFixLen Read
PackDelimited Write
PackLength ReadHeader
WriteHeader

FixedLengthBuffer

Clear ChangeRecordSize
Read Write
SizeOfBuffer ReadHeader
operator= WriteHeader

DelimFieldBuffer

char
Delim;

Clear
Pack Unpack
ReadHeader WriteHeader

LengthFieldBuffer

Clear
Pack Unpack

FixedFieldBuffer

int *FieldSize;
int MaxFields;
int NumFields;
int NextField;

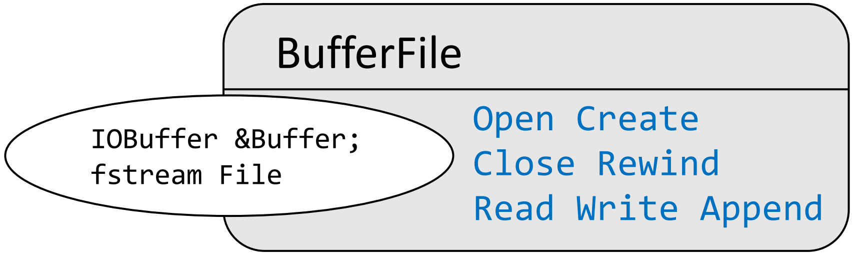
Clear …

Adding Headers to Buffer Classes (1/2)

- IOBuffer class
  - Includes two following methods (slide 23)

```
// these header operations return the number of bytes in the header
virtual int ReadHeader(istream &); // write a buffer to the stream
virtual int WriteHeader(ostream &) const; // write a buffer to the stream
```

Adding Headers to Buffer Classes (2/2)



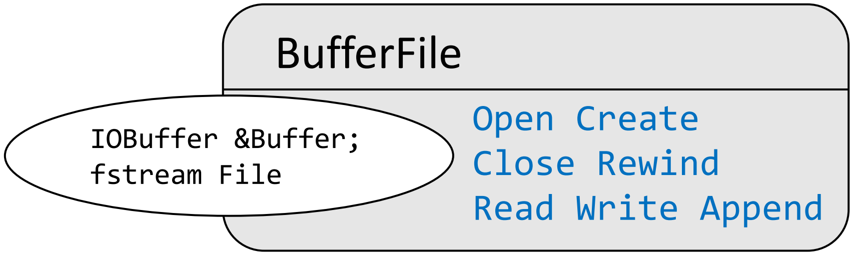
BufferFile

```
IOBuffer &Buffer;
fstream File
```

```
Open Create
Close Rewind
Read Write Append
```

- BufferFile::Create()
  - create a new file and write a header on it.
- BufferFile::Write()
  - write the current buffer contents
- BufferFile::Open()
  - checks for consistency or initializes the buffer
    - call BufferFile::ReadHeader

BufferFile::Create()



BufferFile

```
IOBuffer &Buffer;
fstream File
```

Open Create
Close Rewind
Read Write Append

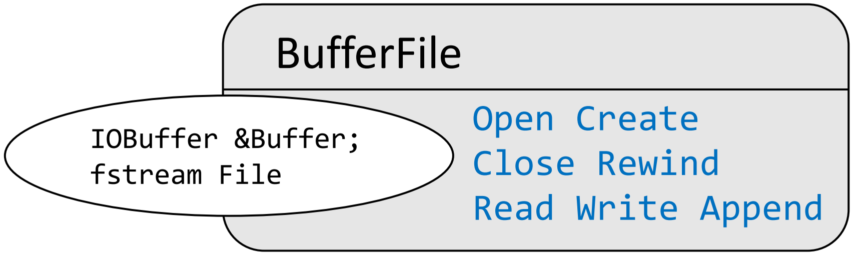
- puts the correct header in every file

```
int BufferFile::WriteHeader ()
{
    return Buffer.WriteHeader(File);
}

// create a new file and write a header on it.
// use ios::nocreate to ensure that no file exists
int BufferFile::Create (char * filename, int mode)
{
    if (!(mode & ios::out)) return FALSE; // must include ios::out
    File.open(filename, mode|ios::in|ios::out|ios::app|ios::binary);
    if (!File.good())
    {
        File.close();
        return FALSE;
    }
    HeaderSize = WriteHeader();
    return HeaderSize != 0;
}
```

Note that a header
Is also written

BufferFile::Open()



BufferFile

IOBuffer &Buffer;
fstream File

Open Create
Close Rewind
Read Write Append

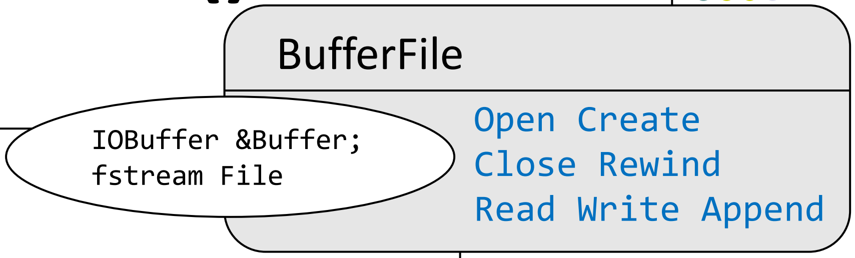
- checks for consistency or initializes the buffer

```
// open an existing file and check the header
// a correct header must be on the file
// use ios::nocreate to ensure that a file exists
int BufferFile::Open(char * filename, int mode)
{
    // these modes are not allowed when opening an existing file
    if (mode&ios::trunc) return FALSE;
    File.open (filename, mode|ios::in|ios::app|ios::binary);

    if (!File.good()) return FALSE;
    File.seekg(0, ios::beg);
    File.seekp(0, ios::beg);
    HeaderSize = ReadHeader();
    if (!HeaderSize) // no header and file opened for output
        return FALSE;
    File.seekp (HeaderSize, ios::beg);
    File.seekg (HeaderSize, ios::beg);
    return File.good();
}
```

Note that a header
Is also read

BufferFile::Read() and Write()



BufferFile

IOBuffer &Buffer;
fstream File

Open Create
Close Rewind
Read Write Append

```
// read a record into the buffer
// return the record address
// return <0 if read failed
// if recaddr == -1, read the next record in the File
// if recaddr != -1, read the record at that address
int BufferFile::Read (int recaddr)
{
    if (recaddr == -1)
        return Buffer.Read (File);
    else
        return Buffer.DRead (File, recaddr);
}

// write the current buffer contents
int BufferFile::Write (int recaddr)
{
    if (recaddr == -1)
        return Buffer.Write(File);
    else
        return Buffer.DWrite(File, recaddr);
}
```

BufferFile::Rewind()

- Rewind()
  - Reposition the get and put file pointers to the beginning of the 1st data record (after the header record)
  - needed because Headersize is a protected member

```
int BufferFile::Rewind()
{
    File.seekg(HeaderSize, ios::beg);
    File.seekp(HeaderSize, ios::beg);
    return 1;
}
```

Another example of BufferFile

- Example from the textbook
  - BufferFile is combined with a fixed-length buffer
  - After the Read, buffer contains the packed record

```
DelimFieldBuffer buffer;
BufferFile file (buffer);
file.Create(“record.dat”, ios::out);//Page 631, F.16
person.Pack(Buffer);
file.Write(); //call Buffer.Write (File) see the page 632
             // then stream.Write(Buffer, BufferSize) 614, F.6
```

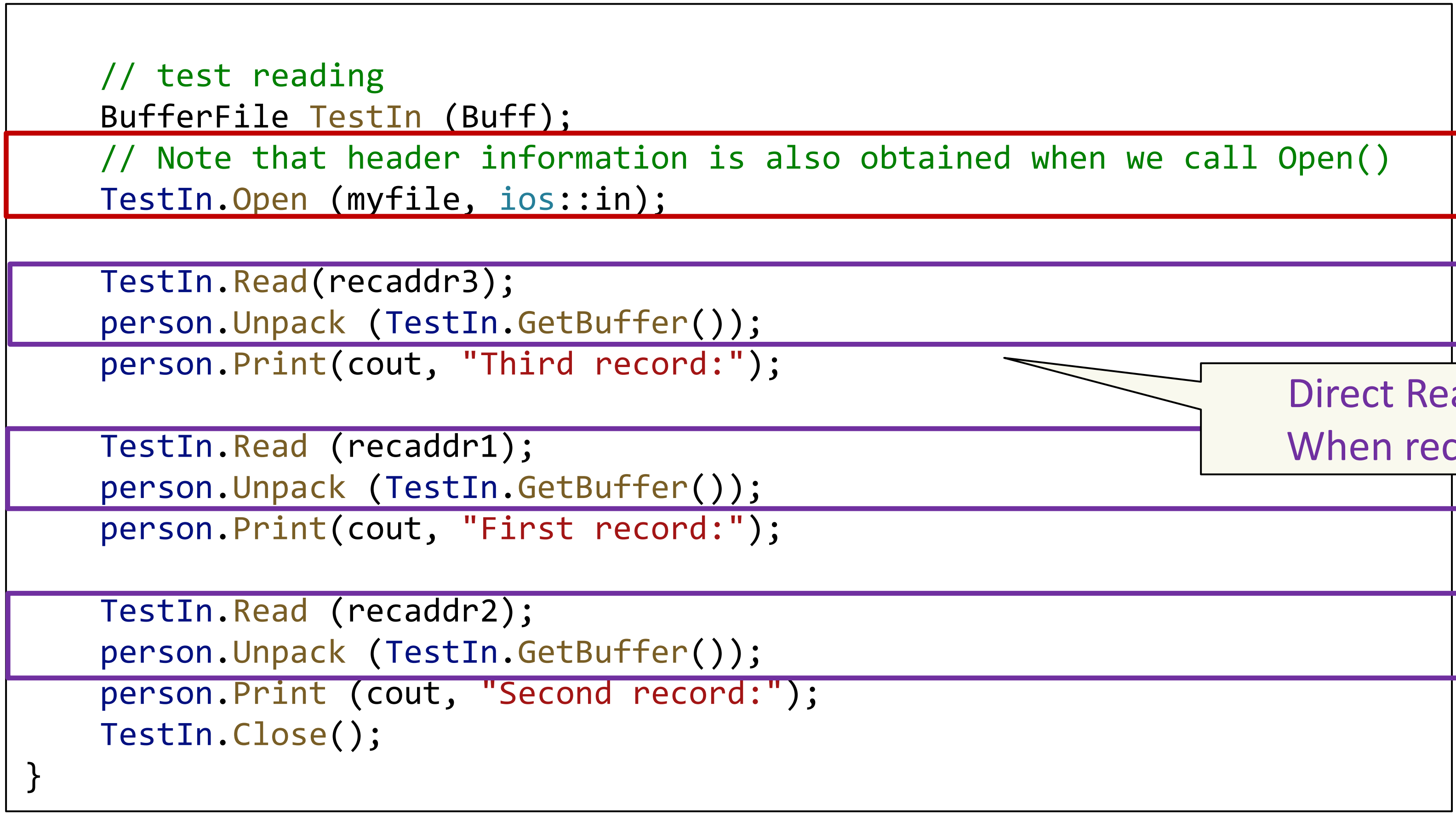
TestBufferFile() in BufFileTest.cpp (1/2)

- Please see differences between TestBuffer() and TestBufferFile()

```
void testBufferFile (IOBuffer & Buff, char * myfile)
{
    Person person;
    int result;
    int recaddr1, recaddr2, recaddr3, recaddr4;

    // Test writing
    BufferFile TestOut (Buff);
    // Note that header information is also written
    result = TestOut.Create (myfile, ios::in|ios::out);
    cout << "create file "<<result<<endl;
    MaryAmes.Pack (TestOut.GetBuffer());
    recaddr1 = TestOut.Write();
    cout << "write at "<<recaddr1<<endl;
    JKwon.Pack (TestOut.GetBuffer());
    recaddr2 = TestOut.Write();
    cout << "write at "<<recaddr2<<endl;
    AlanMason.Pack (TestOut.GetBuffer());
    recaddr3 = TestOut.Write();
    cout << "write at "<<recaddr3<<endl;
    TestOut.Close ();
```

TestBufferFile() in BufFileTest.cpp (2/2)



```
    // test reading
    BufferFile TestIn (Buff);
    // Note that header information is also obtained when we call Open()
    TestIn.Open (myfile, ios::in);

    TestIn.Read(recaddr3);
    person.Unpack (TestIn.GetBuffer());
    person.Print(cout, "Third record:");

    TestIn.Read (recaddr1);
    person.Unpack (TestIn.GetBuffer());
    person.Print(cout, "First record:");

    TestIn.Read (recaddr2);
    person.Unpack (TestIn.GetBuffer());
    person.Print (cout, "Second record:");
    TestIn.Close();
}
```

Direct Read is done When recaddr != -1

Q&A