File Structures

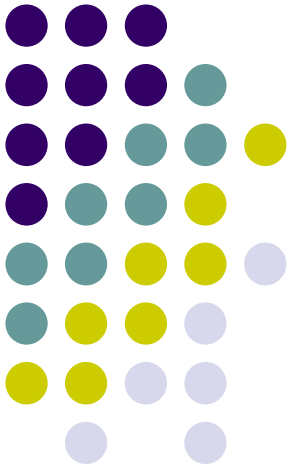
04. C. Fundamental File Structure Concepts

2020. Spring

Instructor: Joonho Kwon

jhkwon@pusan.ac.kr

Data Science Lab @ PNU



Slides are based on the textbook.

datalab

data science laboratory

References

- **Textbook**

- **C++ Programming Langugae**
  - **https://www.learncpp.com/**
    - **Ch11. Inheritance**
    - **Ch12. Virtual Functions**

Outline

- 4.1 Field and Record Organization
- 4.2 Using Classes to Manipulate Buffers
  - Buffer class for delimited fields
  - Buffer class for length-based fields
  - Buffer class for fixed-length fields
- 4.3 Using Inheritance for Record Buffer Classes
- 4.4 Managing Fixed-Length, Fixed-Field Buffers
- 4.5 An Object-Oriented Class for Record Files

So far,

- Three buffer classes
  - A striking similarity: a large percentage of the code is duplicate

- Our next goal
  - **Eliminate almost of duplication through the use of the inheritance**

Inheritance in the C++ stream classes (1/2)

- Learn form the c++ design
  - Inheritance used in stream class
    - At iostream.h and fstream.h

```
class istream: virtual public ios {  . . .
class ostream: virtual public ios {  . . .
class iostream: public istream, public ostream { . . .
class ifstream: public fstreambase, public istream {  . . .
class ofstream: public fstreambase, public ostream { . . .
class fstream: public fstreambase, public iostream { . . .
```

    - the class istream
      - define the read operations, the extraction operators
    - the class ostream
      - define the write operations

Inheritance in the C++ stream classes (2/2)

```
class istream: virtual public ios {  . . .
class ostream: virtual public ios {  . . .
class iostream: public istream, public ostream { . . .
class ifstream: public fstreambase, public istream {  . . .
class ofstream: public fstreambase, public ostream { . . .
class fstream: public fstreambase, public iostream { . . .
```

- two base classes
  - ios and fstreambase
    - Provide common declarations
    - ios: + basic stream operations
    - fstreambase: + access to operating system file operations
- the virtual keyword
  - ensure that class ios included only once in the ancestry of any of these classes

Multiple Inheritance

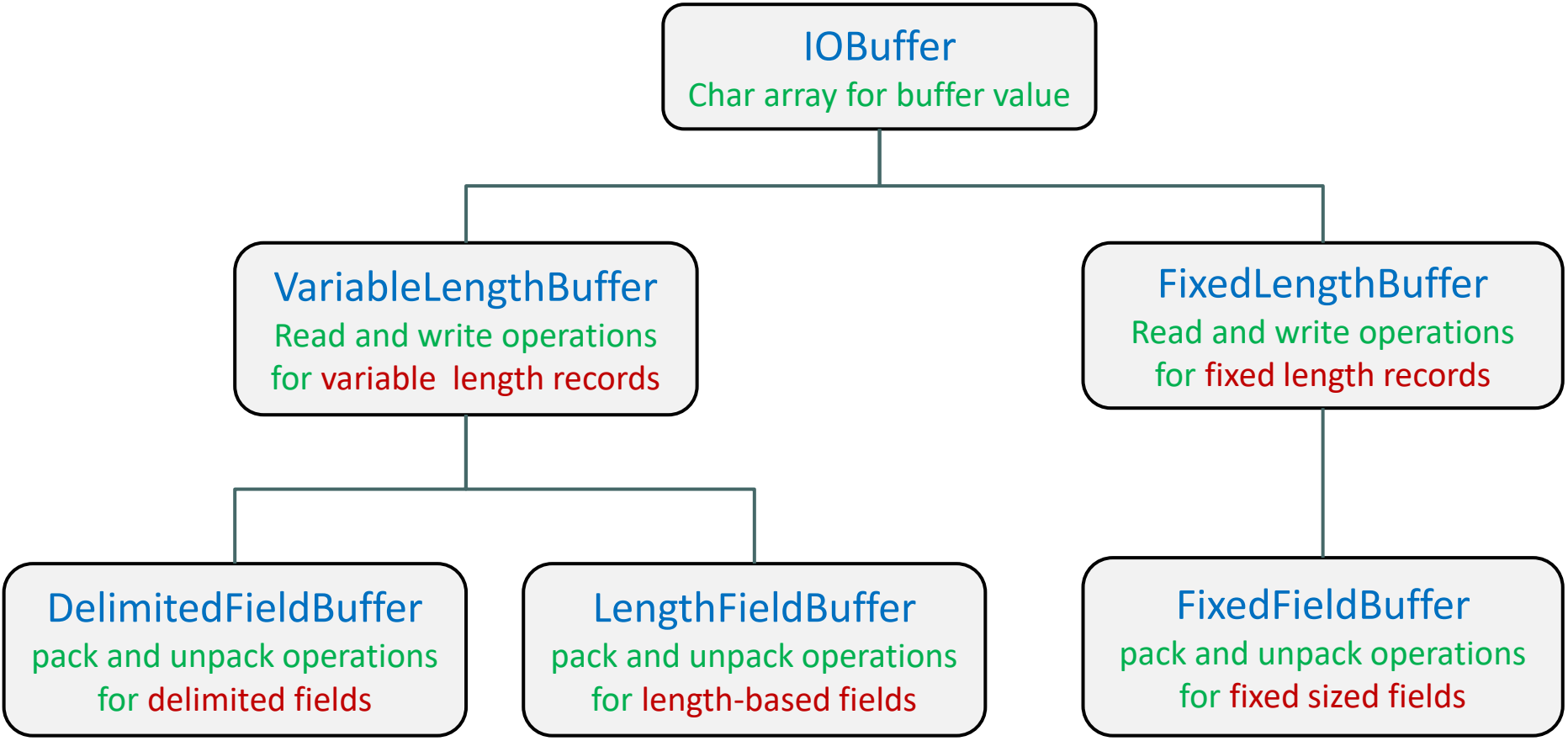
```
class istream: virtual public ios {  . . .
class ostream: virtual public ios {  . . .
class iostream: public istream, public ostream { . . .
class ifstream: public fstreambase, public istream {  . . .
class ofstream: public fstreambase, public ostream { . . .
class fstream: public fstreambase, public iostream { . . .
```

- Benefits
  - Objects of a class are also objects of their base classes
    - Include members and methods of the base classes
    - Operations that work on bases class works on derived class objects
- Example
  - use an istrstream object strbuff to contain a string buffer
    - istrstream is derived from istream
    - strbuff >> p: manipulated by istream operation

```
int ReadVariablePerson (istream & stream, Person & p)
{
        . . .
        istrstream strbuff (buffer);    //create a string stream
        strbuff >> p; //use the istream extraction operator
        return 1;
}
```

Buffer class hierarchy

- Hierarchy for three buffer classes



class IOBuffer (1/2)

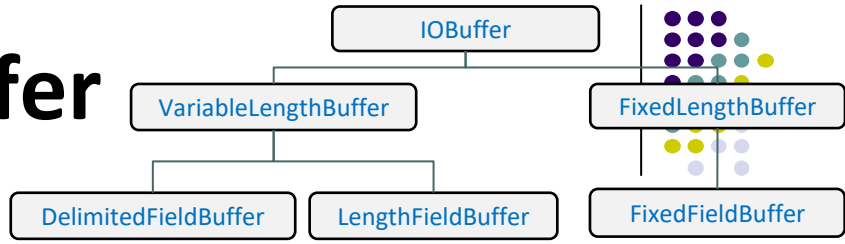
- IOBuffer class (iobuffer.h and iobuffer.cpp)
  - Includes common members and methods of three buffer classes
  - Protected members
    - BufferSize, MaxBytes, NextByte and Buffer
    - Can be used by methods of the class and by methods of derived classes
  - Public methods as virtual
    - Read(), Write(), Pack(), Unpack()
    - Allow each subclass to define its own implementation
    - Declared as a pure virtual with =0
      - IOBuffer does not include an implementation of the method

class IOBuffer (2/2)

- IOBuffer class : an abstract class
  - A class with pure virtual methods
    - No objects can be created
    - Pointer and references of this call can be decared

```
class IOBuffer
{
public:
    IOBuffer (int maxBytes = 1000);
    virtual int Read (istream &) = 0; //read a buffer
    virtual int Write (ostream &) const = 0; //write a buffer
    virtual int Pack(const void * field, int size = -1) = 0;
    virtual int Unpack (void * field,int maxbytes = -1) = 0;
protected:
    char * Buffer; //character array to hold field values
    int BufferrSize; // sum of the sizes of packed fields
    int MaxBytes; //max # of char in the buffer
};
```

class VariableLengthBuffer



- support variable length records
  - Defined in varlen.h
  - Read(), write(): implemented in varlen.cpp

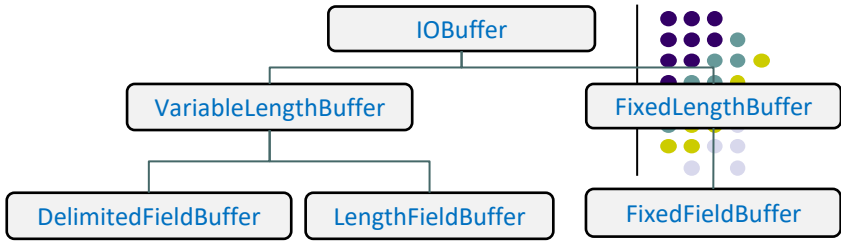
```
class VariableLengthBuffer: public IOBuffer
{
public:
   VariableLengthBuffer(int MaxBytes = 1000);
   int Read(istream &);
   int Write(ostream &) const;
   int SizeOfBuffer() const;
};
```

VariableLengthBuffer::write()

- Write()
  - Test all possible errors
  - Return information to the calling routine via the return value
    - // check tellp() or tellg()?

```
// write the length and buffer into the stream
int VariableLengthBuffer::Write(ostream & stream) const
{
    int recaddr = stream.tellp ();
    unsigned short bufferSize = BufferSize;
    stream.write((char *)&bufferSize, sizeof(bufferSize));
    if (!stream) return -1;
    stream.write(Buffer, BufferSize);
    if (!stream.good()) return -1;
    return recaddr;
}
```

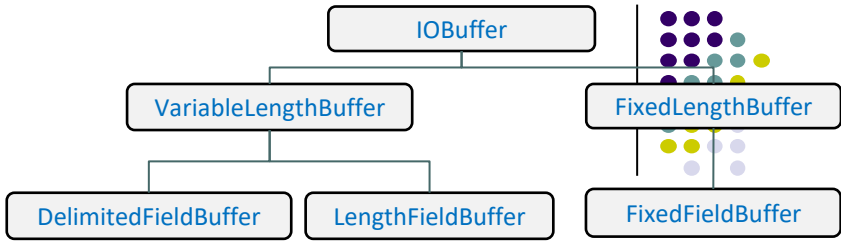
class DelimFieldBuffer



- packing and unpacking delimited fields
  - delim.h and delim.cpp
  - Same codes of the previous section

```
class DelimFieldBuffer: public VariableLengthBuffer
{ public:
    DelimFieldBuffer(char Delim = -1, int maxBytes = 1000);
    int Pack(const void *, int size = -1);
    int Unpack(void *field, int maxBytes = -1);
protected:
    char Delim;
};
```

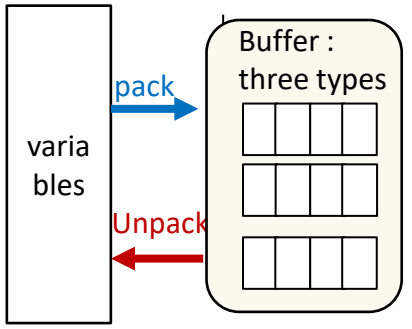
class LengthFieldBuffer



- packing and unpacking legnth fields
  - length.h and length.cpp

```
class LengthFieldBuffer: public VariableLengthBuffer
{
public:
    // construct with fields of specific size
    // construct with a maximum of maxFields
    LengthFieldBuffer (int maxBytes = 1000);
    // copy constructor
    LengthFieldBuffer (const LengthFieldBuffer & buffer)
        : VariableLengthBuffer (buffer) {}
    void Clear (); // clear fields from buffer
    // set the value of the next field of the buffer;
    int Pack (const void * field, int size = -1);
    // extract the value of the next field of the buffer
    int Unpack (void *field, int maxBytes = -1);
    void Print (ostream &) const;
    int Init ();
};
```

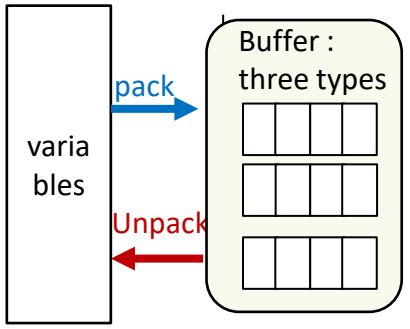
Making objects persistent (1/3)



- An effective strategy
  - Move objects from memory to files
  - Extend the class Person to include pack and unpack operations
    - Ensure that the fields are packed and unpacked

```
int Person::Pack (IOBuffer & Buffer) const
{// pack the fields into a FixedFieldBuffer,
 // return TRUE if all succeed, FALSE o/w
    int numBytes;
    Buffer.Clear ();
    numBytes = Buffer.Pack (LastName);
    if (numBytes == -1) return FALSE;
    numBytes = Buffer.Pack (FirstName);
    if (numBytes == -1) return FALSE;
    // pack the other fields
    ...
    return TRUE;
}
```

Making objects persistent (2/3)



- Person::Unpack(IOBuffer & Buffer)
  - Buffer.unpack( ) : virtual function call
    - The determination of exactly which Unpack method to call is not made during compilation
    - dynamic binding during the run-time

```
int Person::Unpack(IOBuffer & Buffer)
{
    Clear ();
    int numBytes;
    numBytes = Buffer.Unpack(LastName);
    if (numBytes == -1) return FALSE;
    LastName[numBytes] =0;
    numBytes = Buffer.Unpack(FirstName);
    if (numBytes == -1) return FALSE;
    FirstName[numBytes] = 0;
    // unpack the other fields
    ...
    return TRUE;
}
```

Making objects persistent (3/3)

- Dynamic binding example

```
Person MaryAmes;
DelimFieldBuffer Buffer;
MaryAmes.Unpack(Buffer);
```

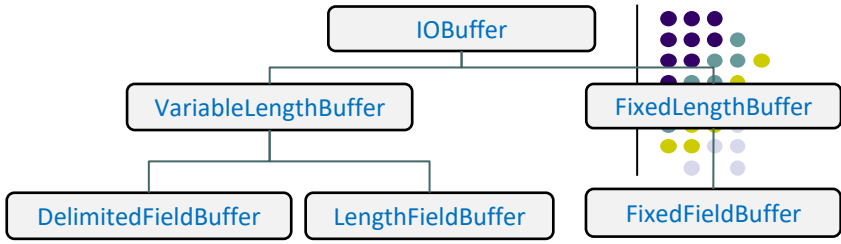
  - At MaryAmes.Unpack()
    - Calls to Buffer.Unpack() use the method DelimFieldBuffer::Unpack()

Outline

- 4.1 Field and Record Organization
- 4.2 Using Classes to Manipulate Buffers
  - Buffer class for delimited fields
  - Buffer class for length-based fields
  - Buffer class for fixed-length fields
- 4.3 Using Inheritance for Record Buffer Classes

- 4.4 Managing Fixed-Length, Fixed-Field Buffers
- 4.5 An Object-Oriented Class for Record Files

Class FixedFieldBuffer



- Support read and write of fixed-length records
  - The write method just writes the fixed-size record
  - The read method must know the record size
  - Each FixedLengthBuffer object has a protected field that records the record size “FieldSize”

```
class FixedFieldBuffer: public FixedLengthBuffer
{
 public:
    …
    // initialize all fields at once
    void Clear (); // clear values from buffer
    int AddField (int fieldSize); // define the next field
    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
};
```

Hold objects of class Person

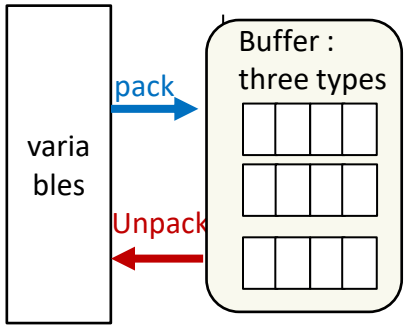
- Person:InitBuffer(): fully initialize the buffer
  - Add the fields one at a time, each with its own size

```
int Person::InitBuffer(FixedFieldBuffer & Buffer)
{
    int result;
    result = Buffer.AddField (10); // LastName [11];
    result = result && Buffer.AddField (10); // FirstName [11];
    result = result && Buffer.AddField (15); // Address [16];
    …
    result = result && Buffer.AddField (9); // ZipCode [10];
    return result;
}
```

- Usage
  - Prepare a buffer for use in reading and writing objects

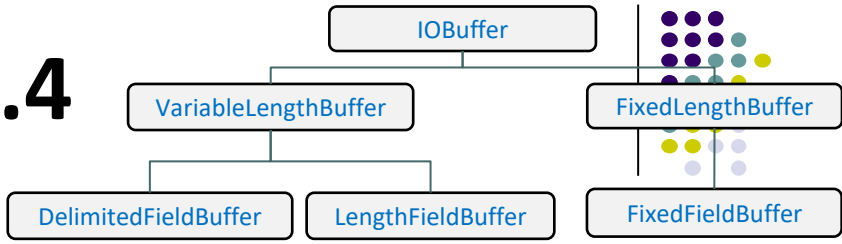
```
FixedFieldBuffer Buffer(6, 61);
MaryAmes.InitBuffer(Buffer);
```

FixedFieldBuffer::UnPack()




```
// extract the value of the next field of the buffer
// return the number of bytes extracted, -1 if error
int FixedFieldBuffer::Unpack (void * field, int maxBytes)
{
    Packing = FALSE;
    if (NextField == NumFields) // buffer is full
        return -1;
    int start = NextByte; // first byte to be unpacked
    int packSize = FieldSize[NextField]; // number bytes to be unpacked
    memcpy(field, &Buffer[start], packSize);
    NextByte += packSize;
    NextField++;
    if (NextField == NumFields) Clear (); // all fields unpacked
    return packSize;
}
```

Test Program of ch4.3/4.4



- Files
  - person.h Iobuffer.h
    - person.cpp iobuffer.cpp
  - varlen.h delim.h length.h
    - Varlen.cpp delim.cpp length.cpp
  - fixlen.h fixfld.h
    - fixlen.cpp fixfld.cpp
  - BufClassTest.cpp (includes main())
    - Functions for testing different Buffer classes
      - InitPerson()
      - testBuffer()
      - testFixedField(); testLength(); testDelim();

InitPerson() at TestProgram

- Global variables

```
Person MaryAmes;
Person AlanMason;
Person JKwon;
```

```
void InitPerson()
{
    cout << "Initializing 3 Persons"<<endl;
    strcpy (MaryAmes.LastName, "Ames");
    strcpy (MaryAmes.FirstName, "Mary");
    strcpy (MaryAmes.Address, "123 Maple");
    strcpy (MaryAmes.City, "Stillwater");
    strcpy (MaryAmes.State, "OK");
    strcpy (MaryAmes.ZipCode, "74075");
    MaryAmes.Print (cout);
    strcpy (AlanMason.LastName, "Mason");
    strcpy (AlanMason.FirstName, "Alan");
    strcpy (AlanMason.Address, "90 Eastgate");
    strcpy (AlanMason.City, "Ada");
    strcpy (AlanMason.State, "OK");
    strcpy (AlanMason.ZipCode, "74820");
    AlanMason.Print (cout);
    strcpy (JKwon.LastName, "Kwon");
    strcpy (JKwon.FirstName, "J");
    strcpy (JKwon.Address, "416");
    strcpy (JKwon.City, "Busan");
    strcpy (JKwon.State, "Bu");
    strcpy (JKwon.ZipCode, "416");
    JKwon.Print (cout);
}
```

testBuffer() at TestProgram

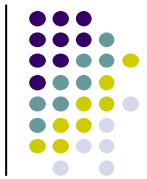
- testBuffer()
  - Write person objects to a file and read them from the file

```
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);
    MaryAmes.Pack (Buff);
    recaddr1 = Buff.Write (TestOut);
    cout << "write at "<<recaddr1<<endl;
    JKwon.Pack(Buff);
    recaddr2 = Buff.Write (TestOut);
    cout << "write at "<<recaddr2<<endl;
    AlanMason.Pack (Buff);
    recaddr3 = Buff.Write (TestOut);
    cout << "write at "<<recaddr3<<endl;
    TestOut.close ();
```

```
    // test reading
    ifstream TestIn (myfile, ios::in);
    Buff.Read(TestIn);
    person.Unpack (Buff);
    person.Print (cout, "First record:");
    Buff.Read(TestIn);
    person.Unpack (Buff);
    person.Print (cout, "Second record:");
    Buff.Read(TestIn);
    person.Unpack (Buff);
    person.Print (cout, "Third record:");
}
```

Three Test Functions



- See how to use three different buffers

```
void testFixedField (){
    cout <<"Testing Fixed Field Buffer"<<endl;
    FixedFieldBuffer Buff (6);
    Person :: InitBuffer (Buff);
    testBuffer (Buff, "fixlen.dat");
}

void testLength (){
    cout << "\nTesting LengthTextBuffer"<<endl;
    LengthFieldBuffer Buff;
    Person :: InitBuffer (Buff);
    testBuffer (Buff, "length.dat");
}

void testDelim () {
    cout << "\nTesting DelimTextBuffer"<<endl;
    DelimFieldBuffer::SetDefaultDelim ('|');
    DelimFieldBuffer Buff;
    Person :: InitBuffer (Buff);
    testBuffer (Buff, "delim.dat");
}
```

Check the outputs and files

- Output messages

```
Testing Fixed Field Buffer
write at 0
write at 61
write at 122
...

Testing LengthTextBuffer
write at 0
write at 48
write at 80
...
Testing DelimTextBuffer
write at 0
write at 42
write at 68
...
```

- Output files
  - fixlen.dat
  - length.dat
  - delim.dat

Outline

- 4.1 Field and Record Organization
- 4.2 Using Classes to Manipulate Buffers
  - Buffer class for delimited fields
  - Buffer class for length-based fields
  - Buffer class for fixed-length fields
- 4.3 Using Inheritance for Record Buffer Classes
- 4.4 Managing Fixed-Length, Fixed-Field Buffers

- 4.5 An Object-Oriented Class for Record Files
  - This will be explained when we learn 5.2.3

Q&A