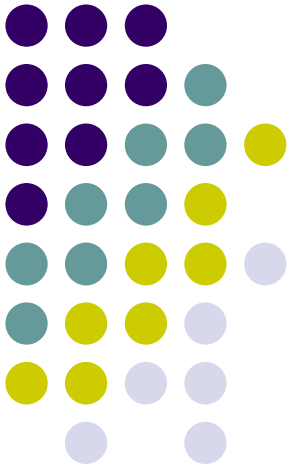
File Structures

04. A. 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

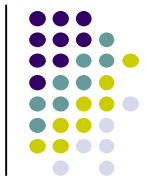
Outline

- 4.1 Field and Record Organization
- 4.2 Using Classes to Manipulate Buffers
- 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

Introduction (1/2)

- File structure
  - Make data persistent
    - one program can create data in memory and store in a file
    - another can read the file and re-create the data in its memory
- Field and Record
  - Field: the basic unit of data
    - A single data value
  - Record
    - A list of different fields
  - Array
    - Many copies of a singe field

Introduction (2/2)



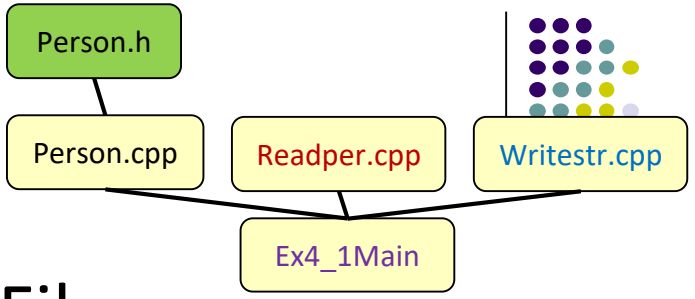
- Object vs. Record
  - Object: Data residing in memory
  - Record: Data in File
- Members
  - Object's fields

Stream

- A stream in c++
  - an abstraction that represents a device on which input and ouput operations are performed
  - basically be represented as a source or destination of characters of indefinite length
  - http://www.cplusplus.com/reference/iolibrary/

- A stream file in our textbook
  - Simple representation for representing objects as a records in file
  - a file organized as a stream of bytes
    - a stream of bytes contain no added information

Ex1 (1/8)



- Store objects of class Person in File
  - As a stream of bytes

Person.h

```
class Person
{
public:
    // data members;
    char LastName[11];
    char FirstName[11];
    char Address[16];
    char City[16];
    char State[3];
    char ZipCode[10];
    // method
    Person();       // default constructor
};
```

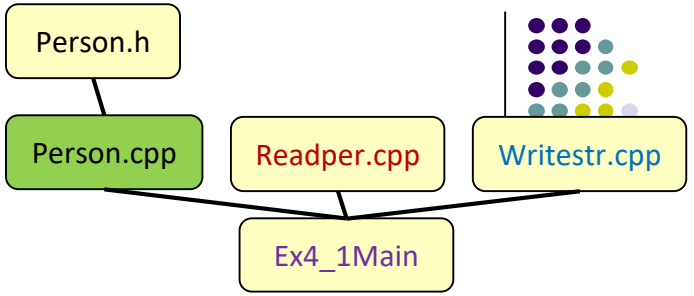
Object1

```
Mary Ames
123 Maple
Stillwater, OK 74075
```

Object2

```
Alan Mason
90 Eastgate
Ada, OK 74820
```

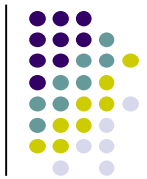
Ex1 (2/8)



- Person.cpp

```
#include "Person.h"

// Constructor, set each field to the empry string
Person::Person()
{
    LastName[0]=0;
    FirstName[0]=0;
    Address[0]=0;
    City[0]=0;
    State[0]=0;
    ZipCode[0]=0;
}
```

Ex1 (3/8)



- Execute program

```
$ ./ex4_1
Enter the file name: aaa.txt
Enter last name, or <cr> to end: Ames
Enter first name: Mary
Enter address: 123 Maple
Enter city: Stillwater
Enter state: OK
Enter zip code: 74075
Enter last name, or <cr> to end: Mason
Enter first name: Alan
Enter address: 90 Eastgate
Enter city: Ada
Enter state: OK
Enter zip code: 74820
Enter last name, or <cr> to end:
```

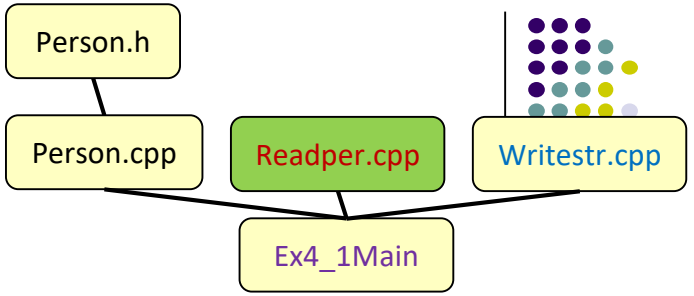
Object1

```
Mary Ames
123 Maple
Stillwater, OK 74075
```

Object2

```
Alan Mason
90 Eastgate
Ada, OK 74820
```

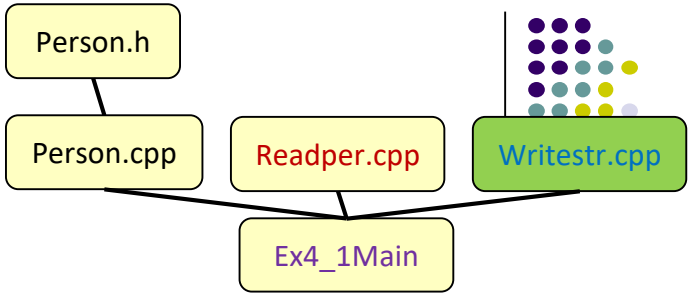
Ex1 (4/8)



- Readper.cpp

```
#include <iostream>
#include <cstring>
#include "Person.h"

using namespace std;

istream& operator >> (istream& stream, Person& p)
{
    // read fileds from input
    cout << "Enter last name, or <cr> to end: " << flush;
    stream.getline(p.LastName, 30);
    if (strlen(p.LastName)==0) return stream;
    cout<<"Enter first name: " << flush;  stream.getline(p.FirstName,30);
    cout<<"Enter address: " << flush;  stream.getline(p.Address,30);
    cout<<"Enter city: " << flush;    stream.getline(p.City,30);
    cout<<"Enter state: " << flush;   stream.getline(p.State,15);
    cout<<"Enter zip code: " << flush; stream.getline(p.ZipCode,10);
    return stream;
}
```

Ex1 (5/8)



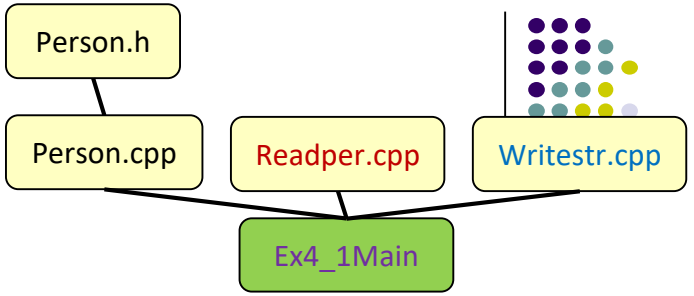
- Writestr.cpp

```
#include <iostream>
#include "Person.h"

using namespace std;

ostream& operator<<(ostream& stream, Person& p)
{
    // insert fields into file
    stream << p.LastName << p.FirstName << p.Address
        << p.City << p.State << p.ZipCode;
    return stream;
}
```

a stream of bytes contain no added information

```
$ cat aaa.txt
AmesMary123 MapleStillwaterOK74075MasonAlan90 EastgateAdaOK74820
```

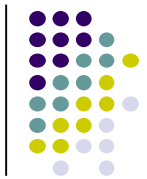
Ex1 (6/8)



- ex4_1Main.cpp (1/2)

```
#include <fstream>
#include <cstring>
#include <iostream>
#include "Person.h"

using namespace std;

//forward declaration, defined in Readper.cpp
istream& operator >> (istream& stream, Person& p);
// forward declaration, defined in Writestr.cpp
ostream& operator<<(ostream& stream, Person& p);

int main()
{
    char filename[20];
    Person p;
    cout << "Enter the file name: " << flush;
    cin.getline(filename, 19);
    // Note, filestream is used
    ofstream stream(filename, ios::out);
```

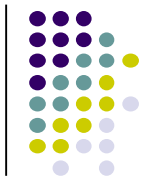
Ex1 (7/8)



- ex4_1Main.cpp (2/2)

```
    if(stream.fail())
    {
        cout << "File open failed!" << endl;
        return -1;
    }
    while(1)
    {
        cin >> p;        // read fields of person
        if (strlen(p.LastName) ==0) break;
        // write person to output stream
        stream << p;     // write fields of person
    }

    return 0;
}
```

Ex1 (8/8)



- Makefile

```
CFLAGS= -Wall
OBJS1 = Person.o Readper.o Writestr.o ex4_1Main.o
all: ex4_1
%.o: %.cpp
    g++ -c -o $@ $(CFLAGS) $<
ex4_1: $(OBJS1)
    g++ -o ex4_1 $(OBJS1)
clean:
    -rm -rf ex4_1 $(OBJS1)
```

- What is the meaning of –rm?

Problem of Ex1

- Input

Object1

```
Mary Ames
123 Maple
Stillwater, OK 74075
```

Object2

```
Alan Mason
90 Eastgate
Ada, OK 74820
```

- Output

```
AmesMary123 MapleStillwaterOK74075MasonAlan90 EastgateAdaOK74820
```

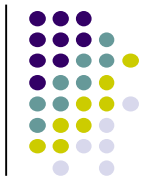
- Problem
  - Lost the integrity of the fundamental organization units of our inputs
  - Mismatch between Fields and files

Field organization

- Field
  - The smallest logically meaningful unit of information in a file (not physical)
  - a field is a conceptual tool

- Field structures (4 methods)
  - adding structure to files to maintain the identity of fields
    - Fix the length of fields
    - Begin each field with a length indicator
    - Separate the fields with delimiters
    - Use a “Keyword = value” expression

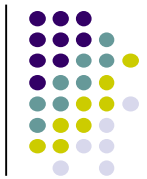
Field Structures (1/3)



- 1. Fix the length of fields
  - force the fields into a predictable length (i.e., fixed-length fields)
  - Adv
    - Simple arithmetic is sufficient to recover the data
  - Disadvantages
    - makes the file much larger
    - Data can be too long to fit into the allocated spaces
      - Inappropriate for data such as names and address

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

Field Structures (2/3)



- 2. Begin each field with a length indicator
  - store the **field length** just ahead of the field
  - one byte for up to 256-byte field

```
04Ames04John09123 Maple10Stillwater02OK0574075
05Mason04Alan1190 Eastgate03Ada02OK0574820
```

- 3. Separate the fields with delimiters
  - use some special character or sequence of characters that will not appear within a field as a delimiter
  - **delimiter character**
    - white-space characters (e.g., blank, new line, tab) or special characters (e.g., vertical bar character)

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

Field Structures (3/3)

- 4. Use a "Keyword=Value" expression to identify fields
  - adv.
    - self-describing structure
    - (i.e., a field provides information about itself)
  - disadv.
    - waste a lot of space for the keywords

```
last=Ames|first=John|address=123 Maple|city=Stillwater|
state=OK|zip=74075|
```

Reading a Stream of Fields

- Operator overloading
  - reads the stream of bytes back in, breaking the stream into fields and storing it as a Person object

```
cin >> p;                    // Person p;
```

  - At Readper.cpp

```
istream& operator >> (istream& stream, Person& p)
{
    // read fileds from input
    cout << "Enter last name, or <cr> to end: " << flush;
    stream.getline(p.LastName, 30);
    if (strlen(p.LastName)==0) return stream;
    cout<<"Enter first name: " << flush;  stream.getline(p.FirstName,30);
    cout<<"Enter address: " << flush;  stream.getline(p.Address,30);
    cout<<"Enter city: " << flush;    stream.getline(p.City,30);
    cout<<"Enter state: " << flush;   stream.getline(p.State,15);
    cout<<"Enter zip code: " << flush; stream.getline(p.ZipCode,10);
    return stream;
}
```

Record organization

- Record
  - a set of fields that belong together when the file is viewed in terms of a higher level of organization
- Record Structures (5 methods)
  - Make records a predictable number of bytes (Fixed-length records)
  - Make records a predictable number of fields
  - Begin each record with a length indicator
  - Use an index to keep track of addresses
  - Place a delimiter at the end of each record

Record Structures (1/4)

- 1. Make records a predictable number of bytes (fixed-length records)
  - each record contains the same number of bytes
  - the most commonly used methods
  - not imply that the size or number of fields
  - Case1) fixed length records with fixed length fields

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

  - Case2) frequently used to hold variable numbers of variable-length fields

Ames|John|123 Maple|Stillwater|OK|74075| ← Unused space →
Mason|Alan|90 Eastgate|Ada|OK|74820| ← Unused space →

Record Structures (2/4)

- 2. Make records a predictable number of fields
  - each record contains a fixed number of fields

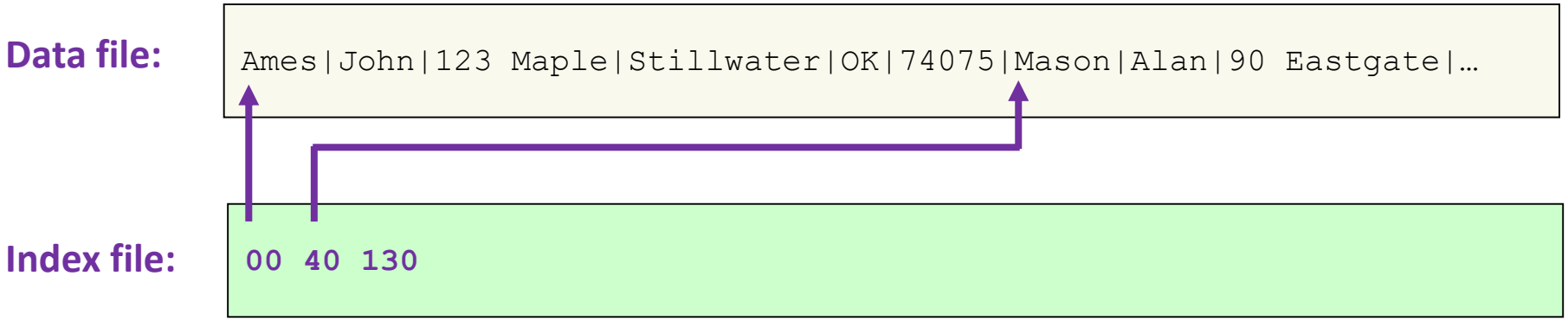
```
Ames|John|123 Maple|Stillwater|OK|74075|Mason|Alan|90 Eastgate|…
```

- 3. Begin each record with a length indicator
  - each record contains a field indicating how many bytes there are in the record
  - commonly used for handling variable-length records

```
40Ames|John|123 Maple|Stillwater|OK|74075|36Mason|Alan|90 Eastgate|…
```

Record Structures (3/4)

- 4. Use an index to keep track of addresses
  - use an index to keep a byte offset for each record in the original file
  - two-file mechanism



Record Structures (4/4)

- 5. Place a delimiter at the end of each record
  - use end-of-line character (e.g., #) as a record delimiter

```
Ames|John|123 Maple|Stillwater|OK|74075|# Mason|Alan|90 Eastgate|…
```

Ex2: Record with delimiters (1/7)

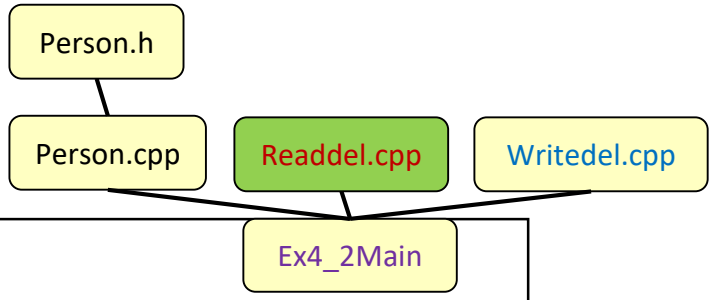
- Execute program

```
$ ./ex4_2
<<<<<<< Records with delimiters >>>>>>>>>
Enter the output file name:del1.txt
Provide the input as the following format:
Last|First|Addr|City|State|Zip|
Kwon|Joonho|PNU|Busan|Bu|4613714|
Provide the input as the following format:
Last|First|Addr|City|State|Zip|
Park|Chanho|Brodwat 222|L.A.|CA|96311|
Provide the input as the following format:
Last|First|Addr|City|State|Zip|
Son|Heungmin|Westminster|London|L|137847|
Provide the input as the following format:
Last|First|Addr|City|State|Zip|
||||||
<<<<<<< Read delimitered records from a file >>>>>>>>>
Kwon|Joonho|PNU|Busan|Bu|4613714|
Park|Chanho|Brodwat 222|L.A.|CA|96311|
Son|Heungmin|Westminster|London|L|137847|
```

```
$ cat del1.txt
Kwon|Joonho|PNU|Busan|Bu|4613714|Park|Cha
nho|Brodwat 222|L.A.|CA|96311|Son|Heungmin
|Westminster|London|L|137847|
```

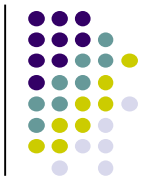
Ex2: Record with delimiters (2/7)

- Readdel.cpp

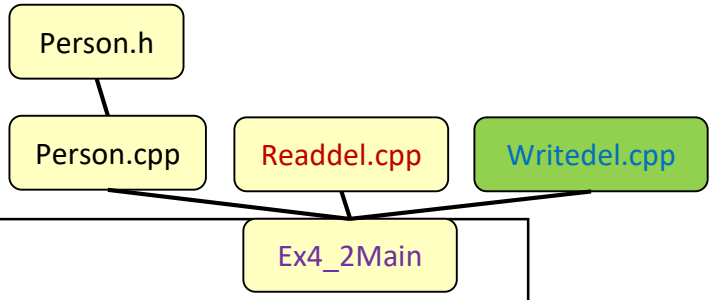



```
#include <iostream>
#include <cstring>
#include "Person.h"

using namespace std;

istream & operator >> (istream & stream, Person & p)
{
    // read fields from file
    char delim;

    stream.getline(p.LastName, 30,'|');
    //cout << "Size of Last: " << strlen(p.LastName) << flush;
    if (strlen(p.LastName)==0) return stream;
    stream.getline(p.FirstName,30,'|');
    stream.getline(p.Address,30,'|');
    stream.getline(p.City, 30,'|');
    stream.getline(p.State,15,'|');
    stream.getline(p.ZipCode,10,'|');
    return stream;
}
```

Ex2: Record with delimiters (3/7)



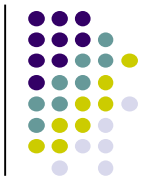
- Writedel.cpp



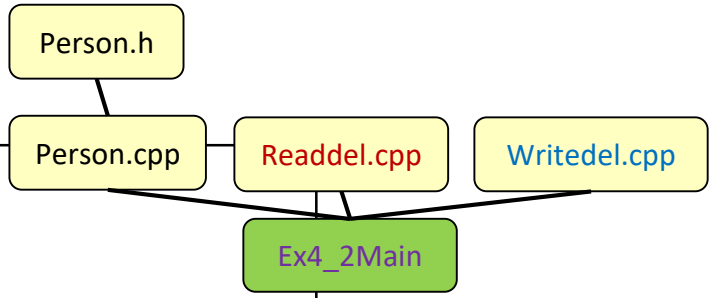

```
#include <iostream>
#include "Person.h"

using namespace std;

ostream & operator << (ostream & stream, Person & p)
{
    // insert fields into file
    stream << p.LastName <<'|'<< p.FirstName <<'|'
        << p.Address << '|' << p.City << '|'
        << p.State <<'|' << p.ZipCode << '|';
    return stream;
}
```

```
$ cat del1.txt
Kwon|Joonho|PNU|Busan|Bu|4613714|Park|Cha
nho|Brodwat 222|L.A.|CA|96311|Son|Heungmin
|Westminster|London|L|137847|
```

Ex2: Record with delimiters (4/7)

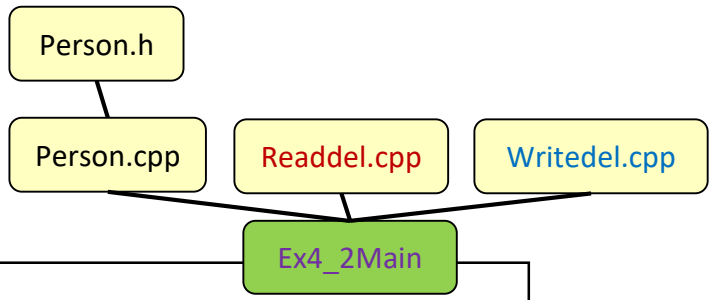


- Ex4_2main.cpp



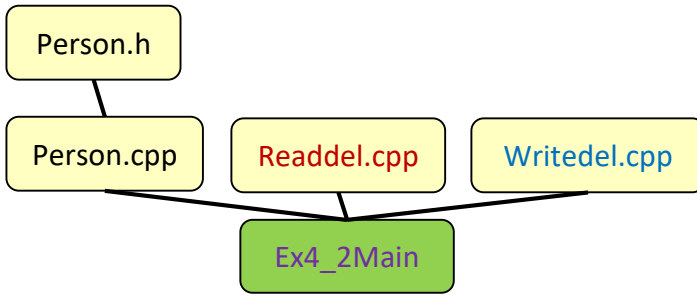

```
#include <fstream>
#include <cstring>
#include <iostream>
#include<limits>  // for numeric_limits
#include "Person.h"
using namespace std;
//forward declaration, defined in Readdel.cpp
istream& operator >> (istream& stream, Person& p);
// forward declaration, defined in Writedel.cpp
ostream & operator << (ostream & stream, Person & p);

int main()
{
    cout<<"<<<<<<< Records with delimiters >>>>>>>>>"<<endl;
    char filename [20];
    Person p;
    cout << "Enter the output file name:"<<flush;
    cin.getline(filename, 19);
    ofstream outfile (filename, ios::out);
    if (outfile.fail()) {
        cout << "File open failed!" <<endl;
        return 0;
    }
```

Ex2: Record with delimiters (5/7)



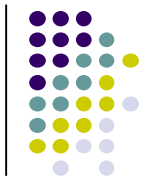

```
while (1) {
    // read fields of person
    cout << "Provide the input as the following format:" << endl;
    cout << "Last|First|Addr|City|State|Zip|"<<endl;
    cin >> p;
    // discards the input buffer
    cin.ignore(numeric_limits<streamsize>::max(),'\n');
    cout << "Strlen of LastName: " << strlen(p.LastName) << endl;

    if (strlen(p.LastName)==0) break;
    // write person to file
    // cout << p << flush;
    // cout<< endl;
    outfile << p;
}
outfile.close();
```

Ex2: Record with delimiters (6/7)




```
    cout<<"<<<<<<< Read delimitered records from a file >>>>>>>>>"<<endl;
    ifstream infile(filename, ios::in);

    while (1) {
        // read fields of person from a file
        infile >> p;
        if (strlen(p.LastName)==0) break;
        // display Person records to stdout
        cout << p << endl;
    }
    infile.close();

    return 0;
}
```

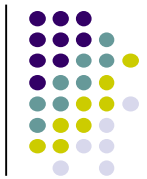
Ex2: Record with delimiters (7/7)



- makefile

```
CFLAGS= -Wall
OBJS1 = Person.o Readper.o Writestr.o ex4_1Main.o
OBJS2 = Person.o Readdel.o Writedel.o ex4_2Main.o
TARGET = ex4_1 ex4_2
all: $(TARGET)
%.o: %.cpp
    g++ -c -o $@ $(CFLAGS) $<
ex4_1: $(OBJS1)
    g++ -o $@ $(OBJS1)
ex4_2: $(OBJS2)
    g++ -o $@ $(OBJS2)
clean:
    -rm -rf ex4_1 ex4_2 $(OBJS1) $(OBJS2)
```

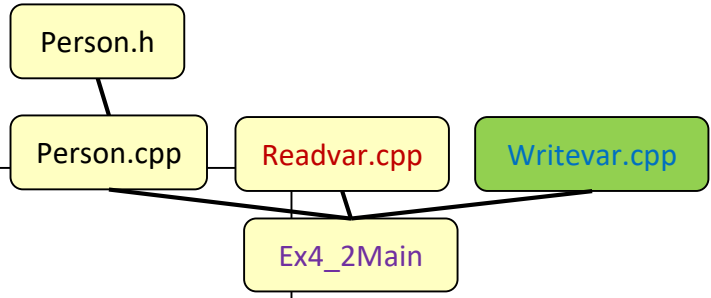
A record with a length (1/3)



- A record structure with a length indicator
  - preserve the variability in the length of records
    - use a record-length field at the beginning of the record
    - => the sum of the lengths of the fields in each record

A record with a length (2/3)

- Writing the variable-length records to the file
  - accumulate the entire contents of a record in a buffer before writing it out
    - the buffer is a character array to place the fields and field delimiters
  - form of the record-length field
    - => binary integer or ASCII characters

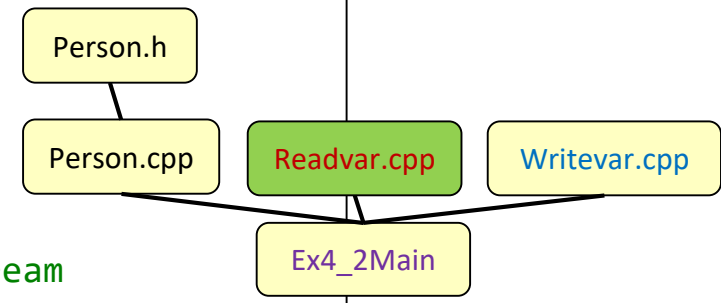



```
const int MaxBufferSize = 200;
int WritePerson (ostream & stream, Person & p)
{
        char buffer [MaxBufferSize];
        strcpy(buffer, p.LastName); strcat(buffer, "|");
        // omitted
        strcat(buffer, p.State); strcat(buffer, "|");
        strcat(buffer, p.ZipCode); strcat(buffer, "|");
        short length = strlen(buffer);
        // write length
        stream.write ((char *)&length, sizeof(length));
        stream.write ((char *)&buffer, length);
}
```

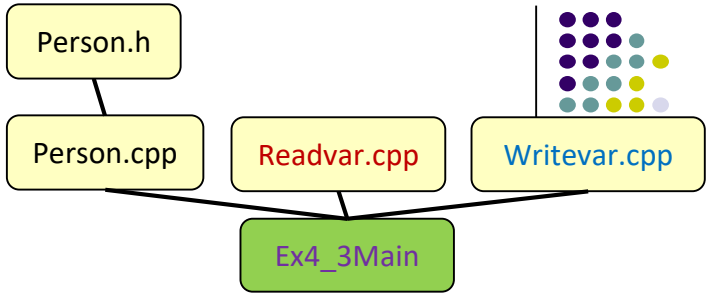
A record with a length (3/3)

- Reading the variable-length records from the file
  - read the variable-length records preceded by record-length fields
  - read the length of a record
  - move the characters of the record into a buffer
  - break the record into fields: unpack( )

```
int ReadVariablePerson(istream& stream, Person& p)
{
    // read a variable sized record from stream  and store it in p
    short length;
    stream.read((char *) &length, sizeof(length));
    // create buffer space
    char* buffer = new char[length+1];
    stream.read(buffer,length);
    buffer[length]=0;    // null terminate
    istrstream strbuff(buffer); // create a string stream
    strbuff >> p;    // use  istream extraction operator
    return 1;
}
```



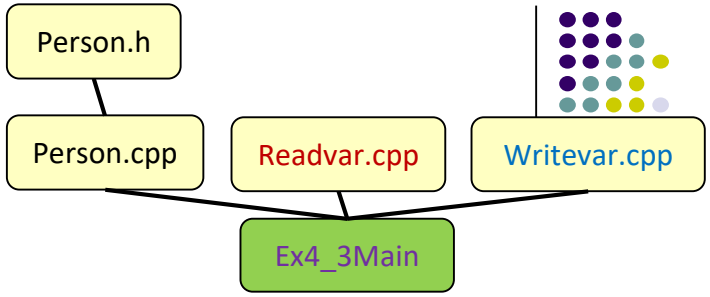
Ex3 (1/3)



- Ex4_3Main.cpp
  - Setting a file for write, later this file will be read

```
int main()
{
    cout<<"<<<<<<< Variable Length >>>>>>>>>"<<endl;
    char filename [20];
    Person p;
    cout << "Enter the output file name:"<<flush;
    cin.getline(filename, 19);
    ofstream outfile (filename, ios::out);
    if (outfile.fail()) {
        cout << "File open failed!" <<endl;
        return 0;
    }
```

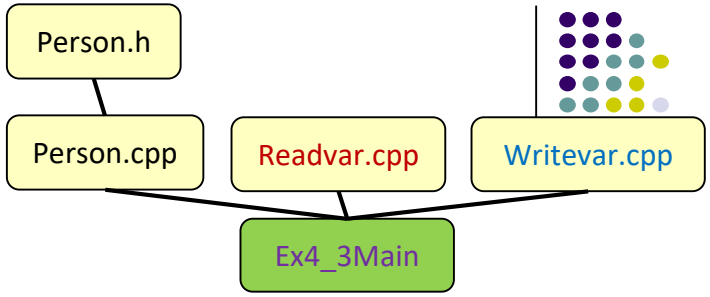
Ex3 (2/3)



- Ex4_3Main.cpp
  - Test for WriteVariablePerson()
    - Instead of using outfile << p, we use the function call `WriteVariablePerson(outfile, p);`

```
while (1) {
    // read fields of person
    cout << "Provide the input as the following format:" << endl;
    cout << "Last|First|Addr|City|State|Zip|"<<endl;
    cin >> p;
    // discards the input buffer
    cin.ignore(numeric_limits<streamsize>::max(),'\n');
    cout << "Strlen of LastName: " << strlen(p.LastName) << endl;

    if (strlen(p.LastName)==0) break;
    // write person to file
    // cout << p << flush;
    // cout<< endl;
    WriteVariablePerson(outfile, p);
}
outfile.close();
```

Ex3 (3/3)



- Ex4_3Main.cpp
  - Test for ReadVariablePerson()
    - Instead of using infile >> p, we use the function call `ReadVariablePerson(infile, p);`

```
    cout<<"<<<<<<< Read records with a length from a file >>>>>>>>>"<<endl;
    ifstream infile(filename, ios::in);

    while (1) {
        // read fields of person from a file
        ReadVariablePerson(infile, p);
        if (strlen(p.LastName)==0) break;
        // display Person records to stdout
        cout << p << endl;
    }
    infile.close();

    return 0;
}
```

Q&A