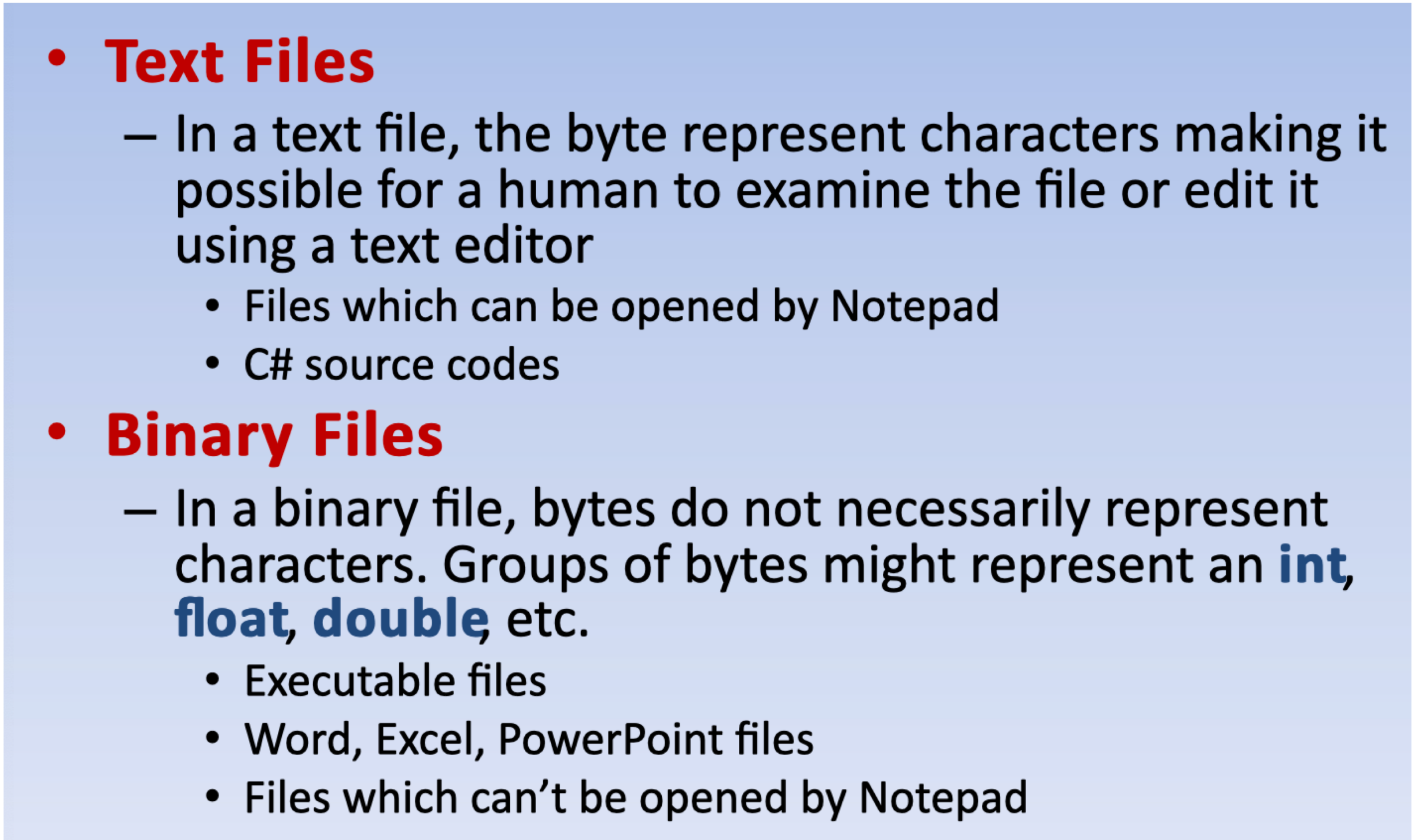
National University of Computer & Emerging Sciences

Lecture 3
File Handling

Dr.Kifayat Alizai

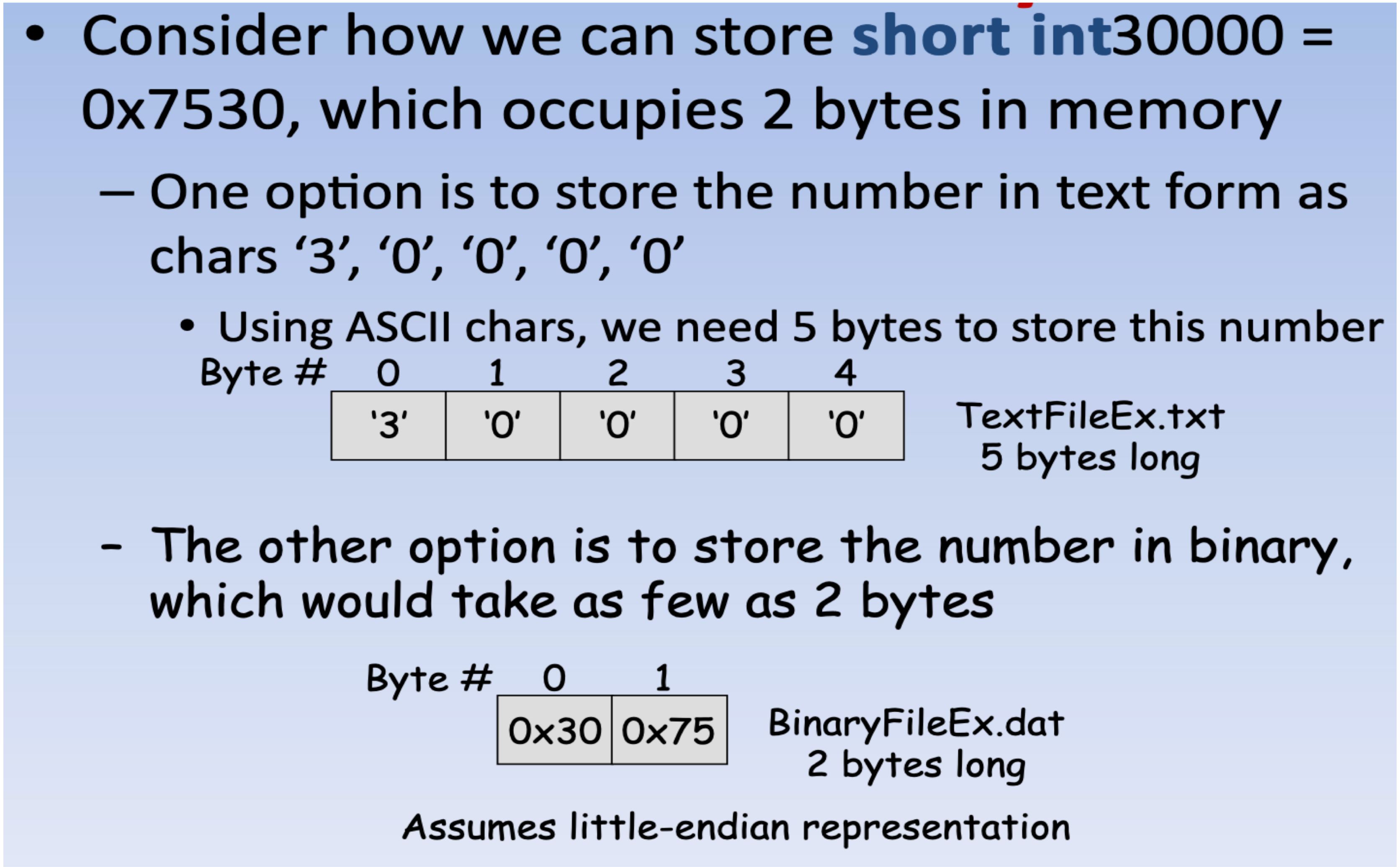
File Types



- **Text Files**
  - In a text file, the byte represent characters making it possible for a human to examine the file or edit it using a text editor
    - Files which can be opened by Notepad
    - C# source codes
- **Binary Files**
  - In a binary file, bytes do not necessarily represent characters. Groups of bytes might represent an **int**, **float**, **double**, etc.
    - Executable files
    - Word, Excel, PowerPoint files
    - Files which can’t be opened by Notepad

Reference: https://slideplayer.com/slide/8846511/

Text File vs Binary File



- Consider how we can store **short int**30000 = 0x7530, which occupies 2 bytes in memory
  - One option is to store the number in text form as chars '3', '0', '0', '0', '0'
    - Using ASCII chars, we need 5 bytes to store this number

Byte #	0	1	2	3	4
	'3'	'0'	'0'	'0'	'0'

TextFileEx.txt
5 bytes long

  - The other option is to store the number in binary, which would take as few as 2 bytes

Byte #	0	1
	0x30	0x75

BinaryFileEx.dat
2 bytes long

Assumes little-endian representation

Reference: https://slideplayer.com/slide/8846511/

Text File vs Binary File

- Why distinguish between text and binary files?
  - The reason is some operating systems, e.g., Windows stores text files and binary files in different ways
  - Text files are divided into lines, so there must be some special way o mark the end of each line
  - Binary files are easy to use by programs and text files are easy to understand for humans

Reference: https://slideplayer.com/slide/8846511/

Binary Files

- Binary file contains unformatted, non-ASCII data
- If you're storing a large amount of numerical data it's more efficient to use binary I/O, in which numbers are stored as they are in the computer's RAM memory, rather than as strings of characters.
  - text version requires 8 bytes for "12345678",
  - 6.02314e13 needs 10 bytes
  - Whereas, value 12345678 as int needs 4 bytes

Binary Files

- Use *ios::binary* argument in the second parameter to write() and read() when working with binary data.

- Indicate by using `binary` flag on open:

```
inFile.open("nums.dat", ios::in | ios::binary);
```

Example Codes

- **Purpose**: Reading a Binary File

  Code: 1-read binary file.cpp

- **Purpose**: Creating a copy of an image from an existing image (Binary File)

  Code: 2-read write binary file.cpp

- **Purpose**: Creating a new image from an existing image (Binary File)

  Code: 3-modify binary file.cpp

Binary Files

- Use `read` and `write` instead of `<<, >>`

```
char ch;
// read in a letter from file
inFile.read(&ch, sizeof(ch));
```

address of where to put the data being read in. The `read` function expects to read `char`s

how many bytes to read from the file

```
// send a character to a file
outFile.write(&ch, sizeof(ch));
```

Binary Files

- To `read`, `write` non-character data, must use a **typecast operator** to treat the address of the data as a character address
- reinterpret_cast operator
  - makes it possible for a buffer of certain type, like int, to look to the read() and write() functions like a buffer of type char.

Binary Files

- To `read`, `write` non-character data, must use a **typecast operator** to treat the address of the data as a character address

```
int num;
// read in a binary number from a file
inFile.read(reinterpret_cast<char *>&num,
                                   sizeof(num));
```

treat the address of `num` as the address of a `char`

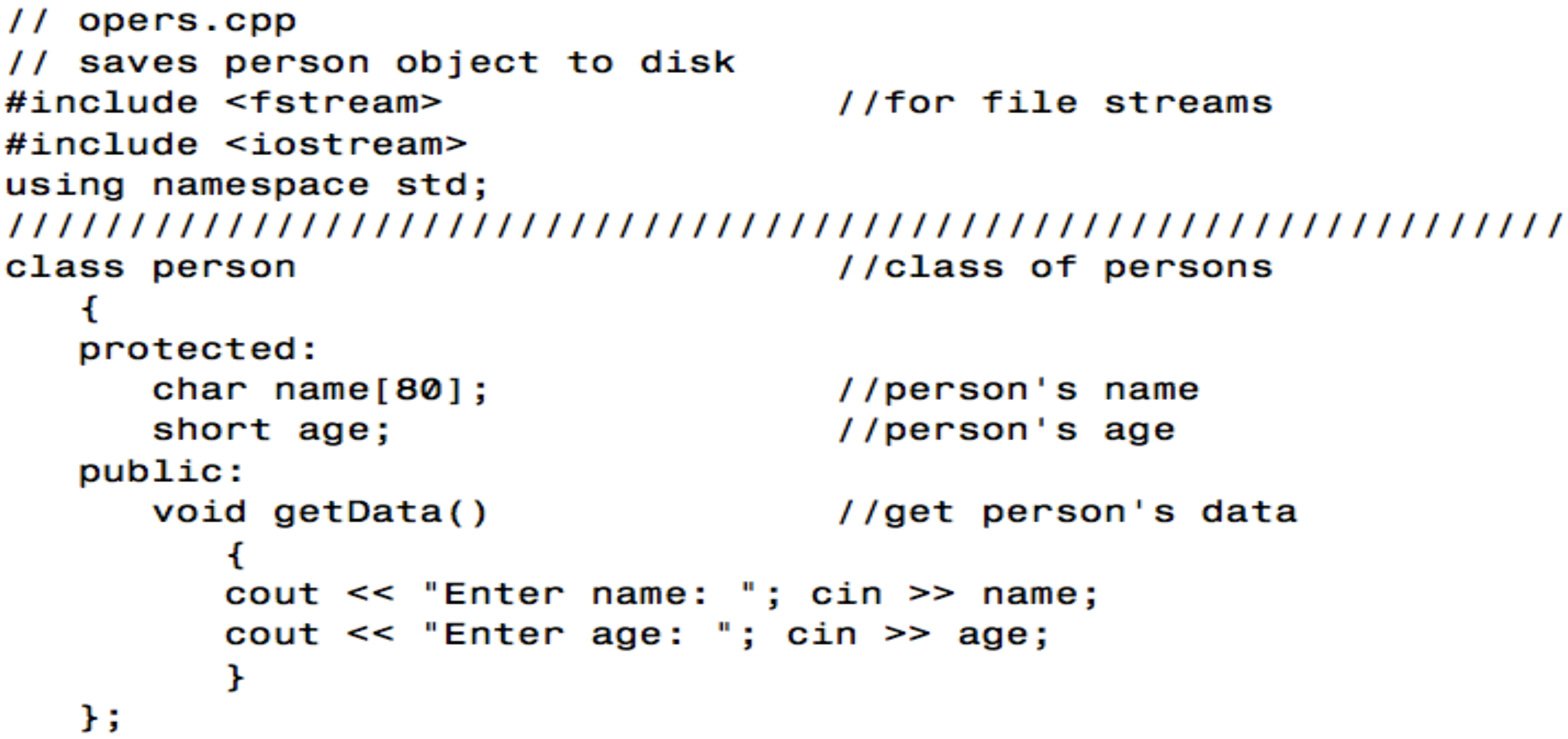
```
// send a binary value to a file
outf.write(reinterpret_cast<char *>&num,
                                 sizeof(num));
```

Object I/O

- When writing and reading an object, we generally use binary mode.

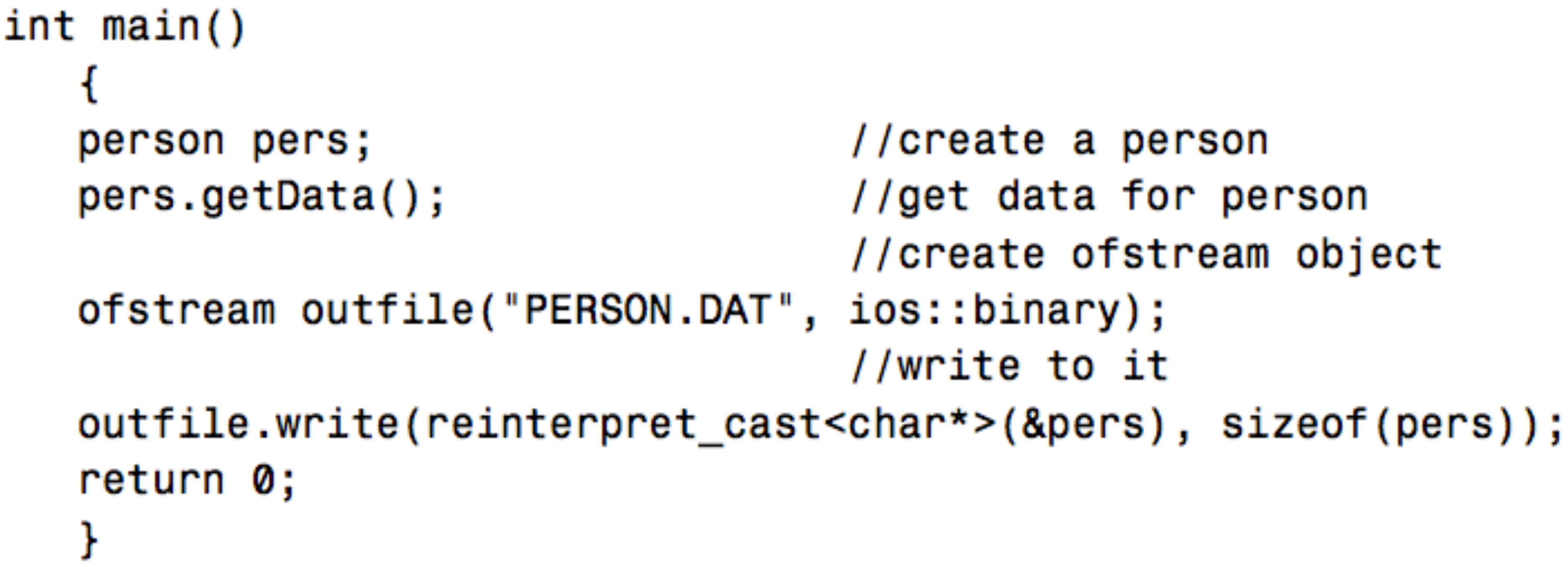
Writing an Object on Disk

Code: 4-writeObj.cpp

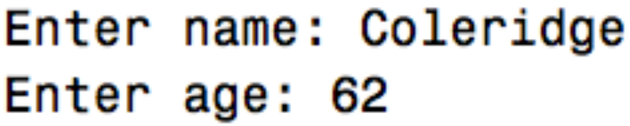


```
// opers.cpp
// saves person object to disk
#include <fstream>                //for file streams
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
class person                      //class of persons
   {
   protected:
      char name[80];              //person's name
      short age;                  //person's age
   public:
      void getData()              //get person's data
         {
         cout << "Enter name: "; cin >> name;
         cout << "Enter age: "; cin >> age;
         }
   };
```

Lecture 3: File Handling



```
int main()
   {
   person pers;                      //create a person
   pers.getData();                   //get data for person
                                     //create ofstream object
   ofstream outfile("PERSON.DAT", ios::binary);
                                     //write to it
   outfile.write(reinterpret_cast<char*>(&pers), sizeof(pers));
   return 0;
   }
```



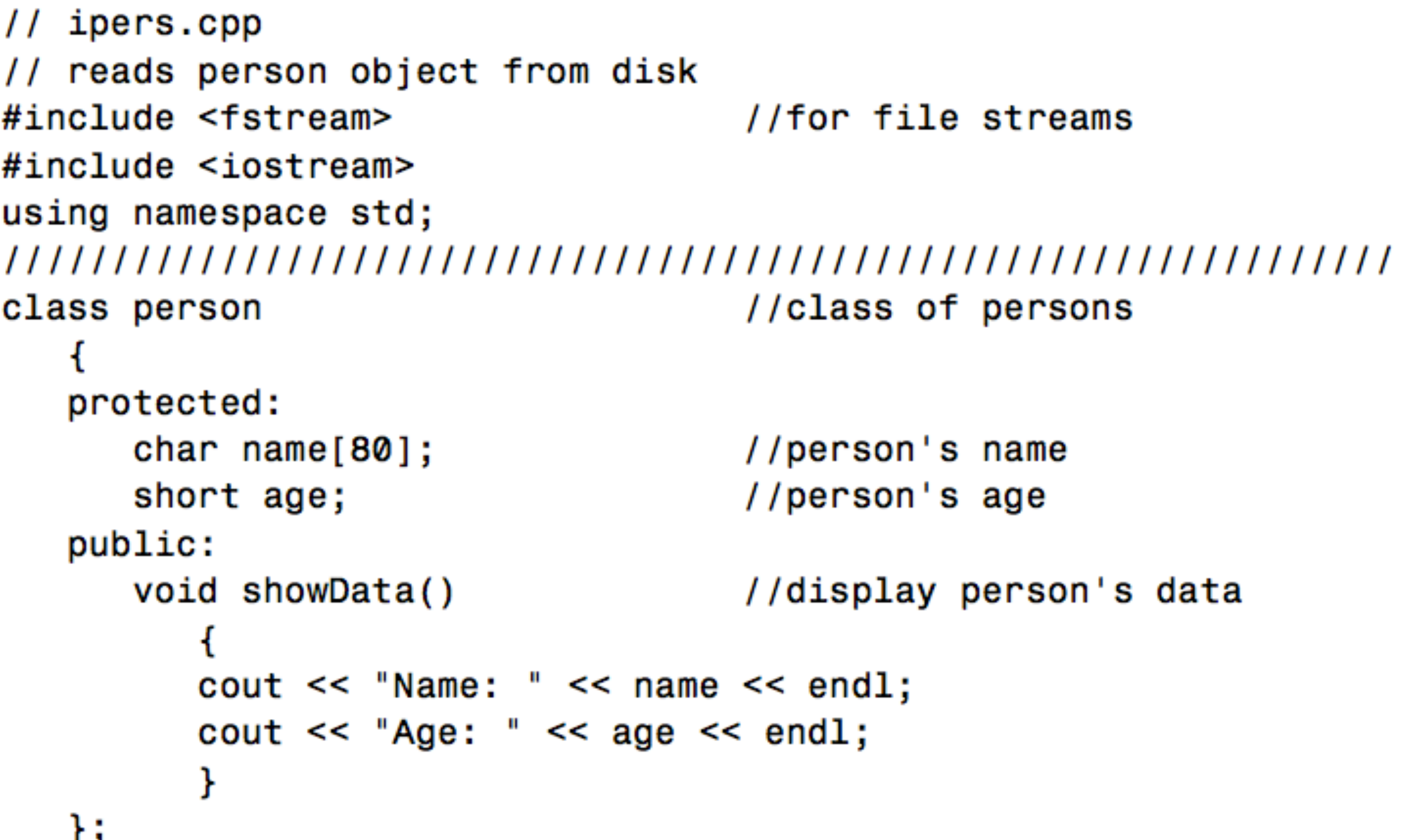
```
Enter name: Coleridge
Enter age: 62
```

We can also do operator overloading and use Friend Function to write an Object

Code: 4-writeObj-FriendFunction.cpp

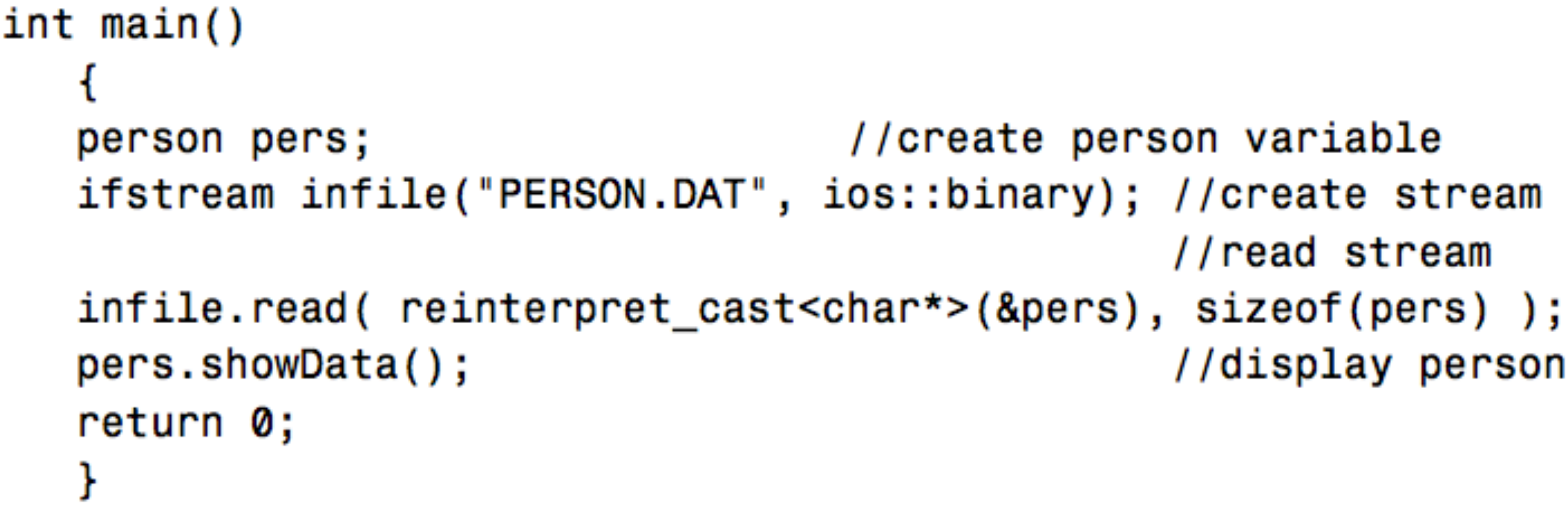
Reading an Object from Disk

Code: 5-readObj.cpp



```
// ipers.cpp
// reads person object from disk
#include <fstream>                //for file streams
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////////
class person                      //class of persons
   {
   protected:
      char name[80];              //person's name
      short age;                  //person's age
   public:
      void showData()             //display person's data
         {
         cout << "Name: " << name << endl;
         cout << "Age: " << age << endl;
         }
   };
```

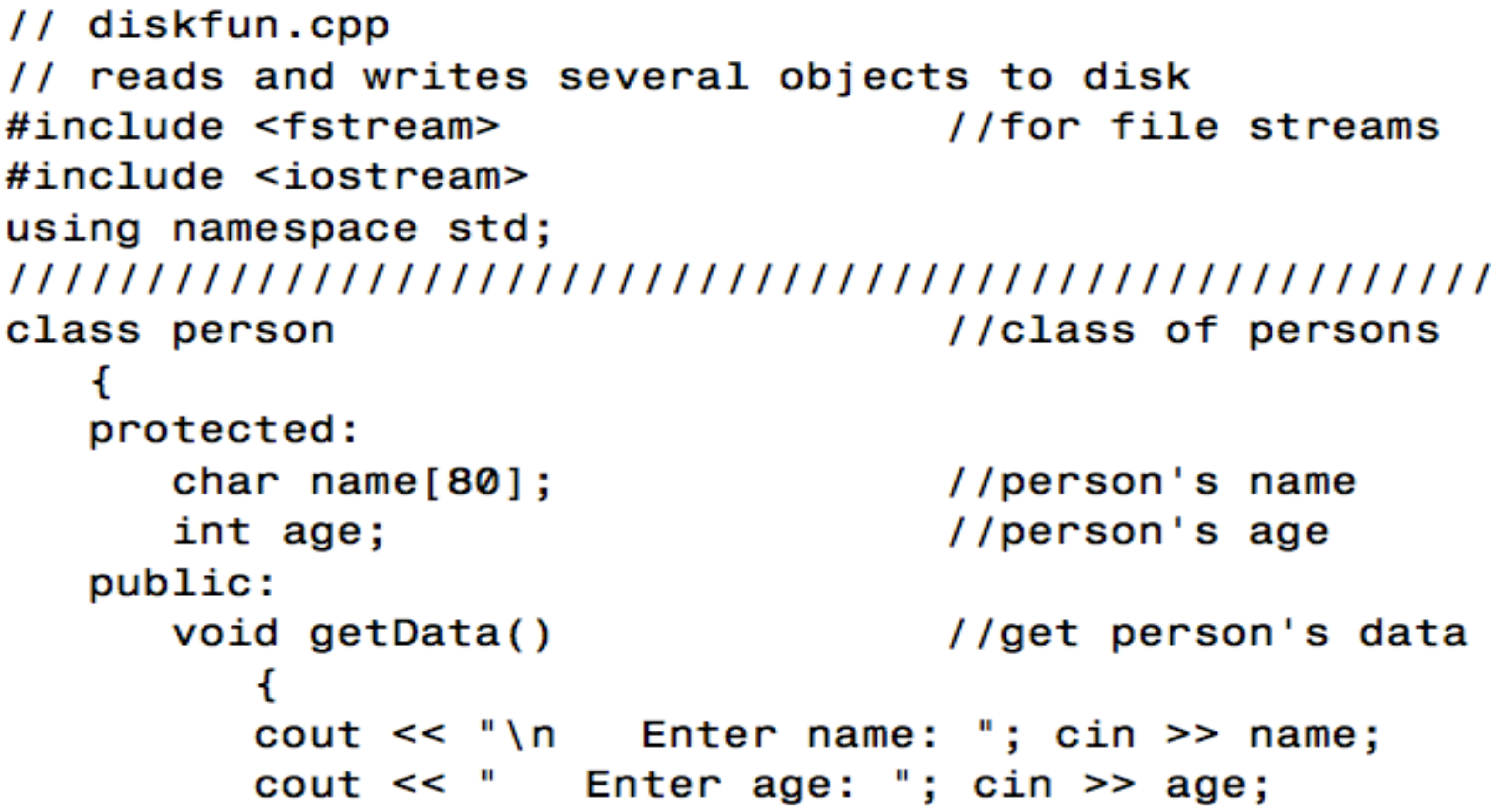
Lecture 3: File Handling



```
int main()
   {
   person pers;                               //create person variable
   ifstream infile("PERSON.DAT", ios::binary); //create stream
                                               //read stream
   infile.read( reinterpret_cast<char*>(&pers), sizeof(pers) );
   pers.showData();                            //display person
   return 0;
   }
```

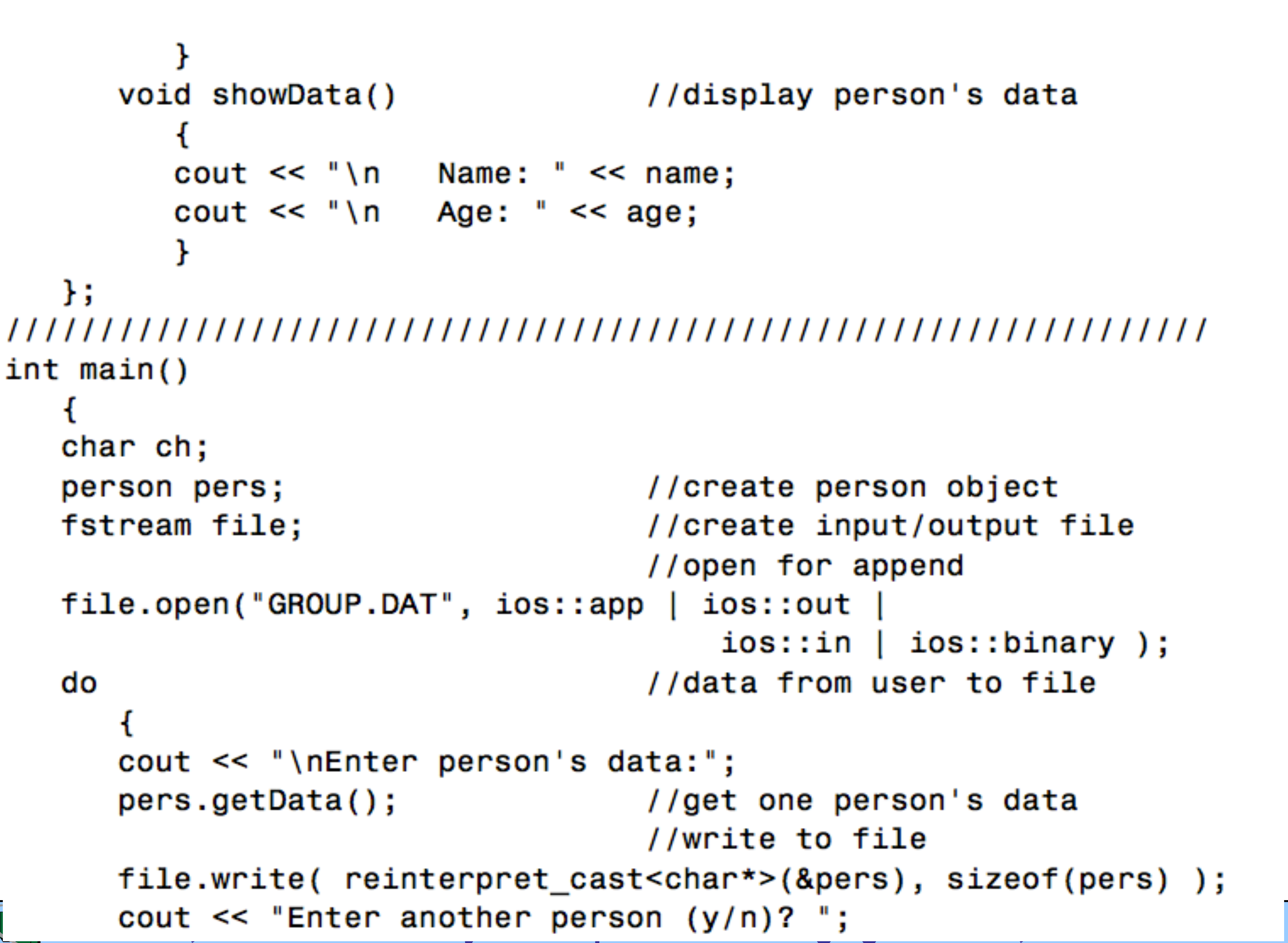
I/O with Multiple Objects

Code: 6-multipleObject-IO.cpp



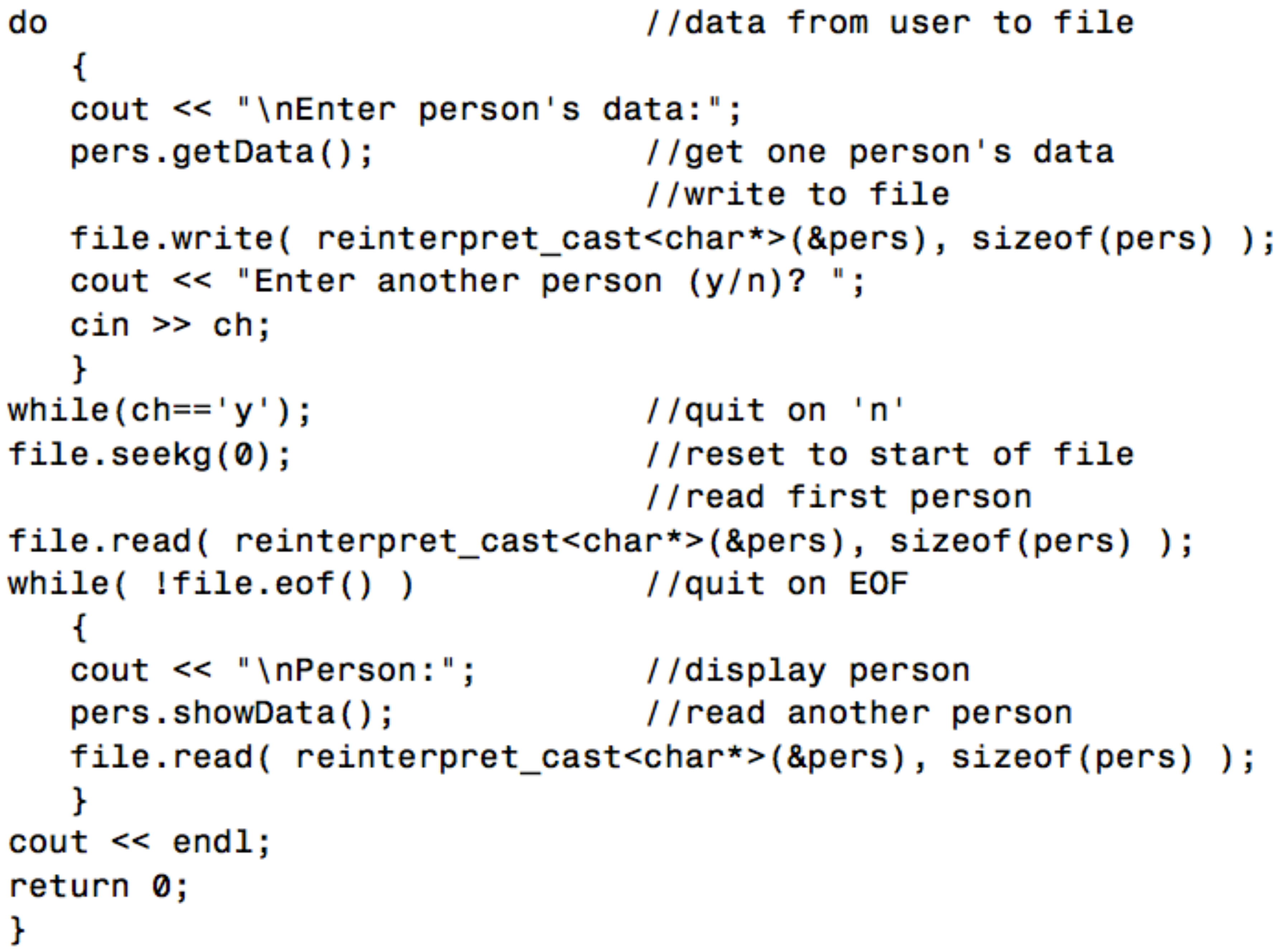
```
// diskfun.cpp
// reads and writes several objects to disk
#include <fstream>                //for file streams
#include <iostream>
using namespace std;
//////////////////////////////////////////////////////
class person                      //class of persons
   {
   protected:
      char name[80];              //person's name
      int age;                    //person's age
   public:
      void getData()              //get person's data
         {
         cout << "\n   Enter name: "; cin >> name;
         cout << "   Enter age: "; cin >> age;
```

Lecture 3: File Handling



```
         }
      void showData()             //display person's data
         {
         cout << "\n   Name: " << name;
         cout << "\n   Age: " << age;
         }
   };
////////////////////////////////////////////////////////////////
int main()
   {
   char ch;
   person pers;                   //create person object
   fstream file;                  //create input/output file
                                  //open for append
   file.open("GROUP.DAT", ios::app | ios::out |
                                      ios::in | ios::binary );
   do                             //data from user to file
      {
      cout << "\nEnter person's data:";
      pers.getData();             //get one person's data
                                  //write to file
      file.write( reinterpret_cast<char*>(&pers), sizeof(pers) );
      cout << "Enter another person (y/n)? ";
```

Lecture 3: File Handling



```
do                             //data from user to file
   {
   cout << "\nEnter person's data:";
   pers.getData();             //get one person's data
                               //write to file
   file.write( reinterpret_cast<char*>(&pers), sizeof(pers) );
   cout << "Enter another person (y/n)? ";
   cin >> ch;
   }
while(ch=='y');                //quit on 'n'
file.seekg(0);                 //reset to start of file
                               //read first person
file.read( reinterpret_cast<char*>(&pers), sizeof(pers) );
while( !file.eof() )           //quit on EOF
   {
   cout << "\nPerson:";        //display person
   pers.showData();            //read another person
   file.read( reinterpret_cast<char*>(&pers), sizeof(pers) );
   }
cout << endl;
return 0;
}
```

Creating Records with Structures

- Can write structures to, read structures from files

- To work with structures and files,
  - use `ios::binary` file flag upon open
  - use `read`, `write` member functions

Creating Records with Structures

```
struct TestScore
{
  int studentId;
  double score;
  char grade;
};
TestScore oneTest;
...
// write out oneTest to a file
gradeFile.write(reinterpret_cast<char *>
  (&oneTest), sizeof(oneTest));
```