C++ Object-Oriented Prog.
Unit 5: Object-Oriented Design

CHAPTER 19: FILE PROCESSING

DR. ERIC CHOU IEEE SENIOR MEMBER

LECTURE 1

Introduction to File Processing

Introduction

Storage of data

- Arrays, variables are temporary
- Files are permanent
  - Magnetic disk, optical disk, tapes

In this chapter

- Create, update, process files
- **Sequential** and **random** access
- **Formatted** and **raw** processing

The Data Hierarchy

From smallest to largest

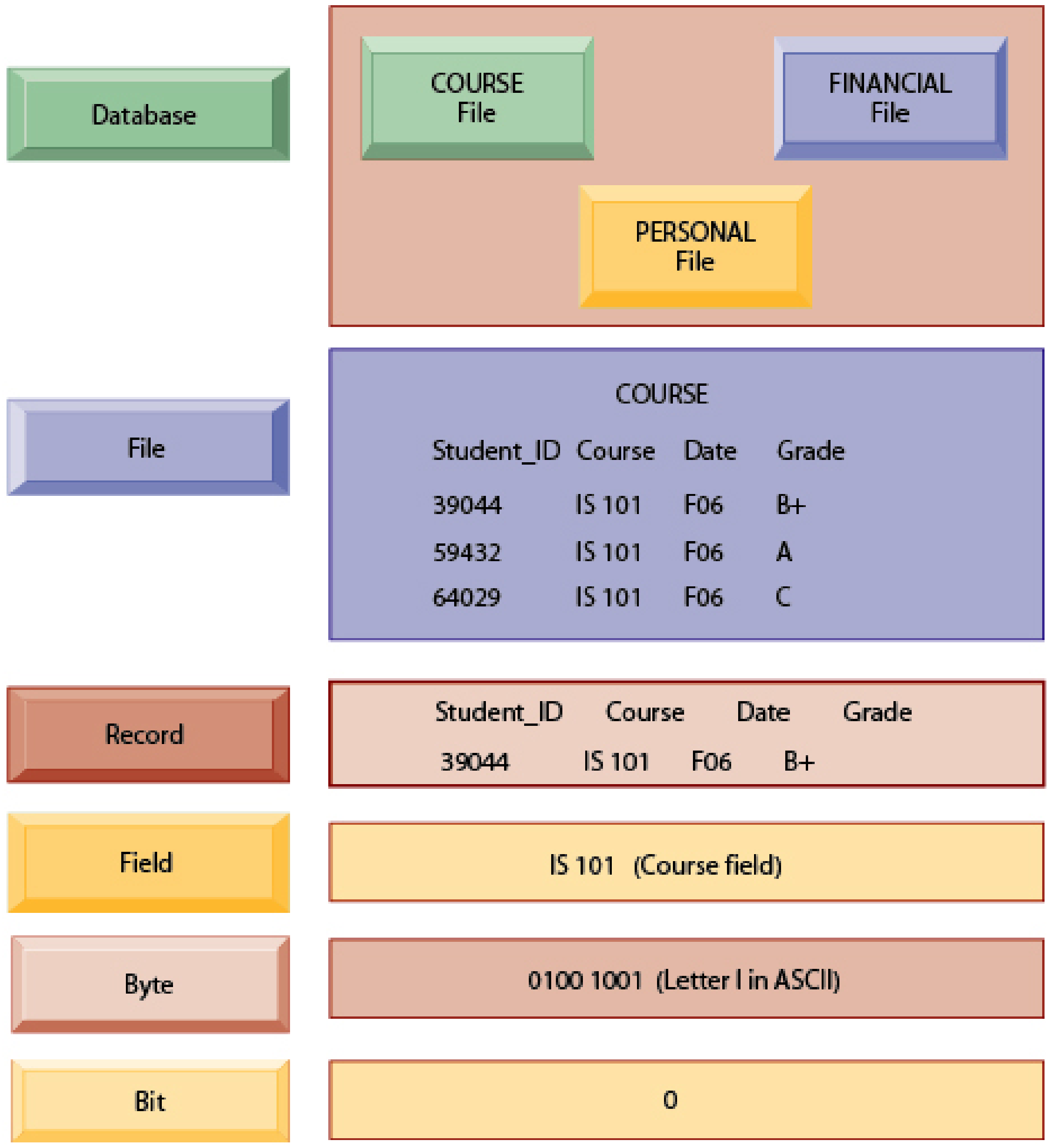
- Bit (binary digit)
  - 1 or 0
  - Everything in computer ultimately represented as bits
  - Cumbersome for humans to use
  - Character set
    - Digits, letters, symbols used to represent data
    - Every character represented by 1's and 0's
- Byte: 8 bits
  - Can store a character (**`char`**)
  - Also Unicode for large character sets (**`wchar_t`**)

The Data Hierarchy

From smallest to largest (continued)

- Field: group of characters with some meaning
  - Your name
- Record: group of related fields
  - `struct` or `class` in C++
  - In payroll system, could be name, SS#, address, wage
  - Each field associated with same employee
  - Record key: field used to uniquely identify record
- File: group of related records
  - Payroll for entire company
  - Sequential file: records stored by key
- Database: group of related files
  - Payroll, accounts-receivable, inventory…

Student Database



COURSE

Student_ID	Course	Date	Grade
39044	IS 101	F06	B+
59432	IS 101	F06	A
64029	IS 101	F06	C

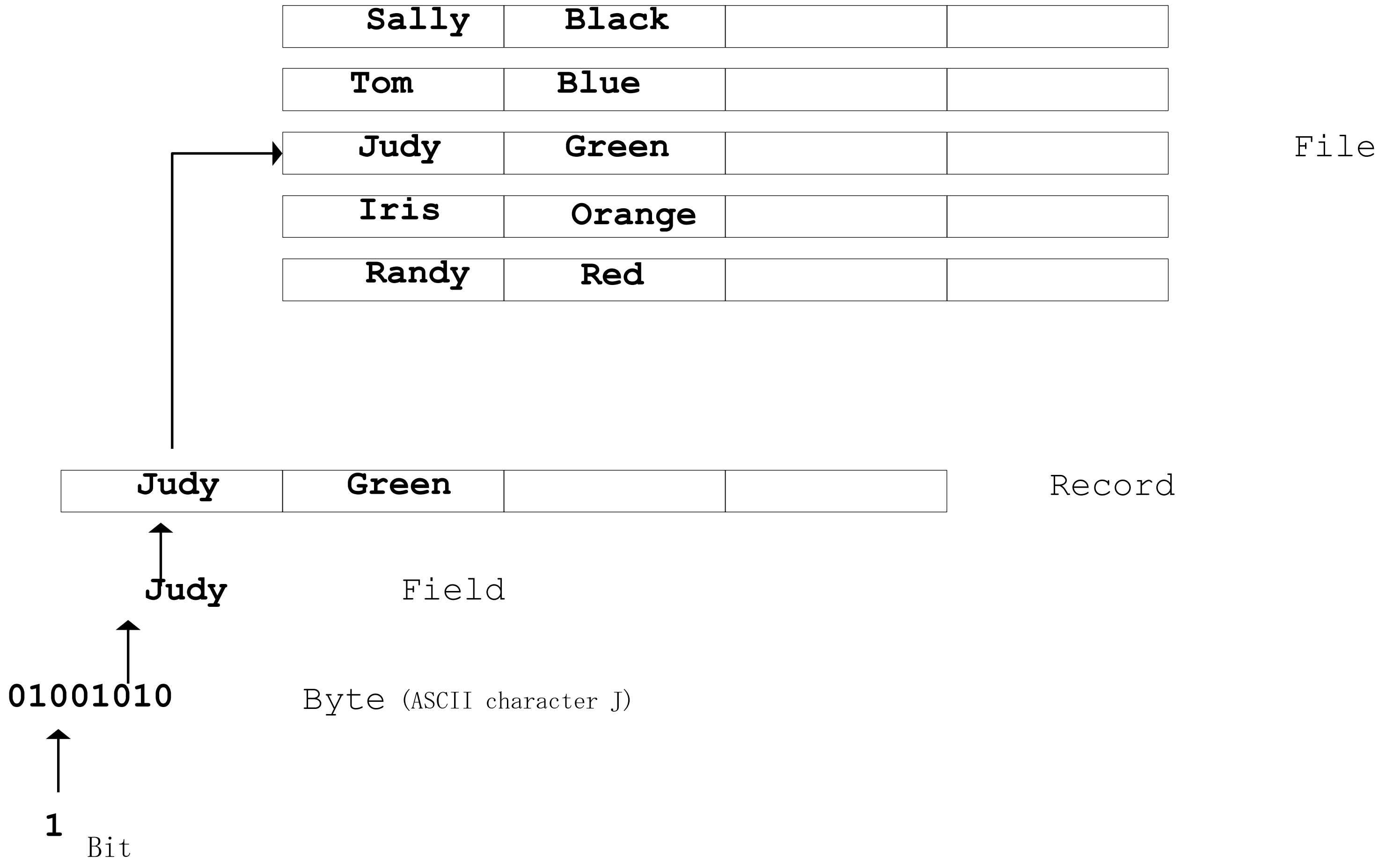
Student_ID	Course	Date	Grade
39044	IS 101	F06	B+

IS 101 (Course field)

0100 1001 (Letter I in ASCII)

0

The Data Hierarchy



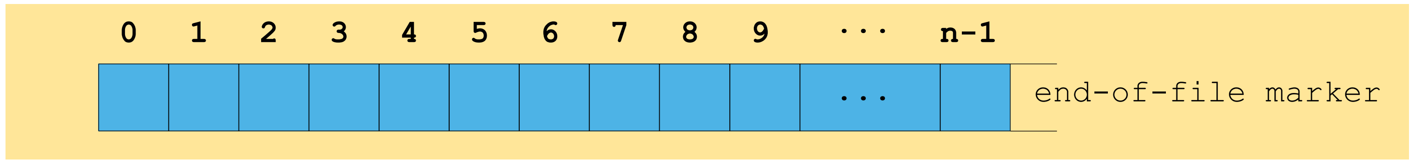
LECTURE 2

File and I/O Streams

Files and Streams

C++ views file as sequence of bytes

- Ends with *end-of-file* marker



When file opened

- Object created, stream associated with it
- **`cin`**, **`cout`**, etc. created when **`<iostream>`** included
  - Communication between program and file/device

Files and Streams

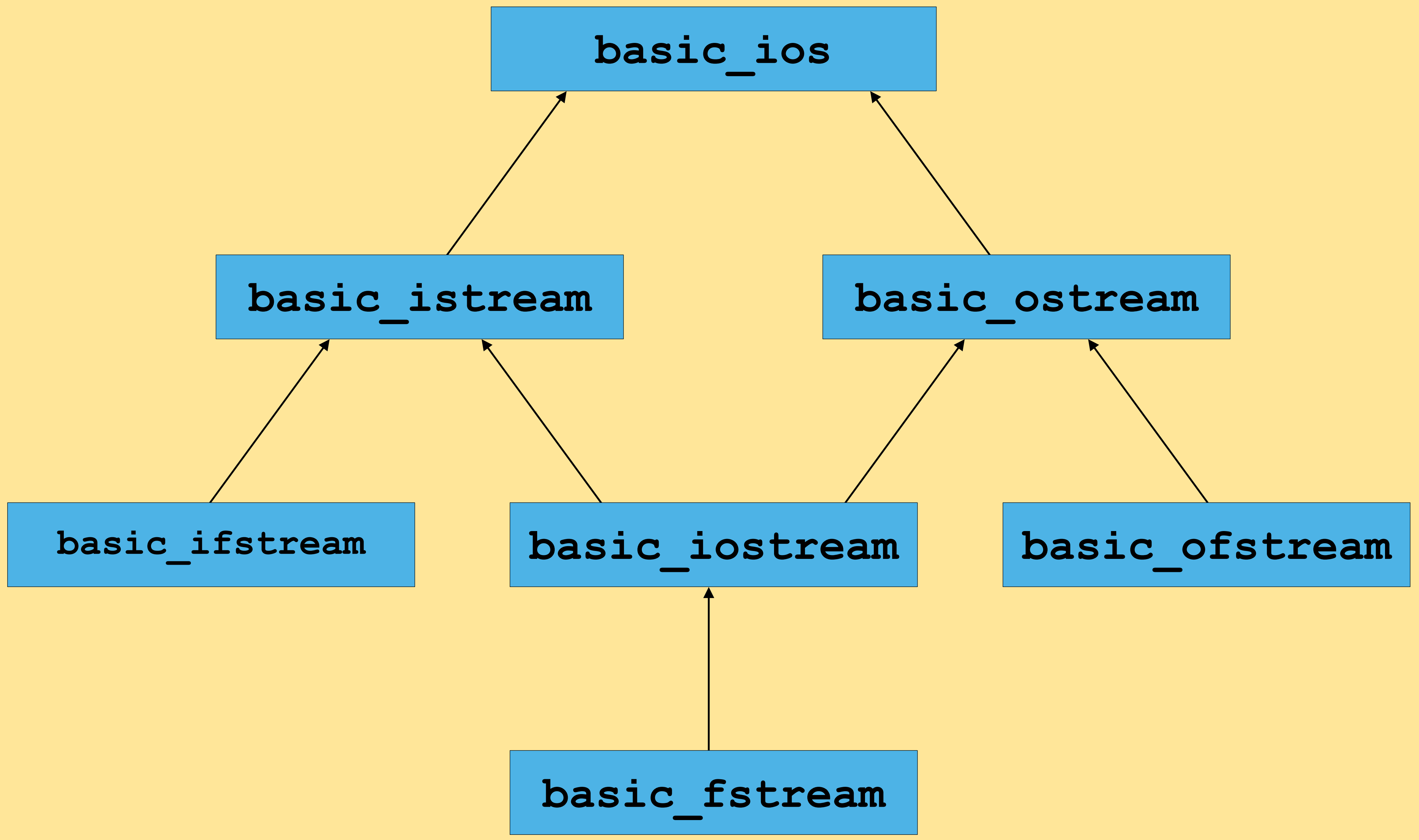
To perform file processing

- Include **`<iostream>`** and **`<fstream>`**
- Class templates
  - **`basic_ifstream`** (input)
  - **`basic_ofstream`** (output)
  - **`basic_fstream`** (I/O)
- **`typedef`**s for specializations that allow **`char`** I/O
  - **`ifstream`** (**`char`** input)
  - **`ofstream`** (**`char`** output)
  - **`fstream`** (**`char`** I/O)

Files and Streams

Opening files

- Create objects from template
- Derive from stream classes
  - Can use stream methods from Ch. 12
  - **`put`**, **`get`**, **`peek`**, etc.


basic_ios
basic_istream
basic_ostream
basic_ifstream
basic_iostream
basic_ofstream
basic_fstream

LECTURE 2

Sequential File Handler (Formatted Output)

Creating a Sequential-Access File

C++ imposes no structure on file

- Concept of "record" must be implemented by programmer

To open file, create objects

- Creates "line of communication" from object to file
- Classes
  - `ifstream` (input only)
  - `ofstream` (output only)
  - `fstream` (I/O)
- Constructors take *file name* and *file-open mode*
  - `ofstream outClientFile( "filename", fileOpenMode );`
- To attach a file later
  - `Ofstream outClientFile;`
  - `outClientFile.open( "filename", fileOpenMode);`

Creating a Sequential-Access File

File-open modes

Mode	Description
ios::app	Write all output to the end of the file.
ios::ate	Open a file for output and move to the end of the file (normally used to append data to a file). Data can be written anywhere in the file.
ios::in	Open a file for input.
ios::out	Open a file for output.
ios::trunc	Discard the file's contents if it exists (this is also the default action for **ios::out**)
ios::binary	Open a file for binary (i.e., non-text) input or output.

- **`ofstream`** opened for output by default
  - **`ofstream outClientFile( "clients.dat", ios::out );`**
  - **`ofstream outClientFile( "clients.dat");`**

Creating a Sequential-Access File

Operations

- Overloaded **`operator!`**
  - **`!outClientFile`**
  - Returns nonzero (true) if **`badbit`** or **`failbit`** set
    - Opened non-existent file for reading, wrong permissions
- Overloaded **`operator void*`**
  - Converts stream object to pointer
  - `0` when when **`failbit`** or **`badbit`** set, otherwise nonzero
    - **`failbit`** set when EOF found
  - **`while ( cin >> myVariable )`**
    - Implicitly converts **`cin`** to pointer
    - Loops until EOF

Creating a Sequential-Access File

Operations

- Writing to file (just like **`cout`**)
  - **`outClientFile << myVariable`**
- Closing file
  - **`outClientFile.close()`**
  - Automatically closed when destructor called

Demo Program: file1.cpp

Go Notepad++!!!

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>  // exit prototype
using namespace std;
using std::ios;
using std::cerr;
using std::ofstream;
int main(){
   int account;
   char name[30];
   double balance;

   // ofstream constructor opens file
   ofstream outClientFile("clients.dat", ios::out );

   // exit program if unable to create file
   if ( !outClientFile ) {  // overloaded ! operator
     cerr << "File could not be opened" << endl;
     exit(1);
   } // end if

   cout << "Enter the account, name, and balance." << endl
         << "Enter end-of-file to end input.\n? ";

   // read account, name and balance from cin, then place in file
   while ( cin >> account >> name >> balance ) {
     outClientFile << account << ' ' << name << ' ' << balance << endl;
     cout << "? ";
   } // end while
   return 0;  // ofstream destructor closes file
} // end main
```

file1.cpp

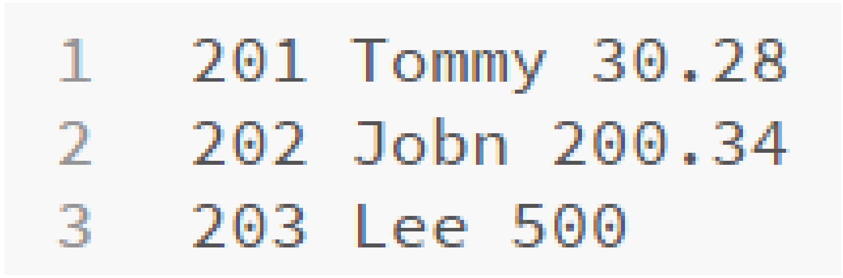
Notice the the header files required for file I/O.

ofstream object created and used to open file **"clients.dat"**. If the file does not exist, it is created.

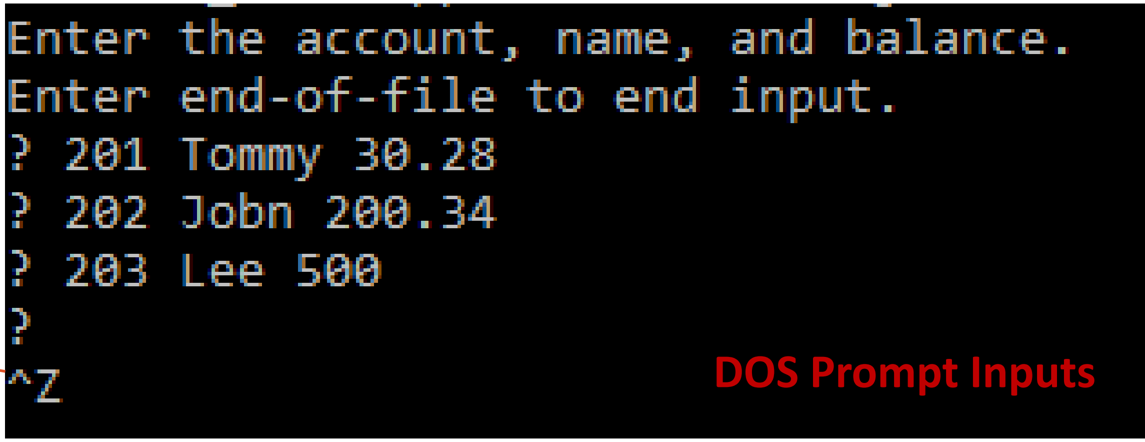
! operator used to test if the file opened properly.

End of File Mark to end Input

clients.dat



```
201 Tommy 30.28
202 Jobn 200.34
203 Lee 500
```



```
Enter the account, name, and balance.
Enter end-of-file to end input.
? 201 Tommy 30.28
? 202 Jobn 200.34
? 203 Lee 500
?
^Z
```

DOS Prompt Inputs

eC Learning Channel

LECTURE 2

Read Data From Sequential File (Formatted Input)

Functions use in File Handling

Function	Operation
open()	To create a file
close()	To close an existing file
get()	Read a single character from a file
put()	write a single character in file.
read()	Read data from file
write()	Write data into file.

Reading Data from a Sequential-Access File

- Reading files
  - `ifstream inClientFile( "filename", ios::in );`
  - Overloaded `!`
    - `!inClientFile` tests if file was opened properly
  - `operator void*` converts to pointer
    - `while (inClientFile >> myVariable)`
    - Stops when EOF found (gets value `0`)

Demo Program: file2.cpp

1. read in data from clients.dat record by record (one record a line).

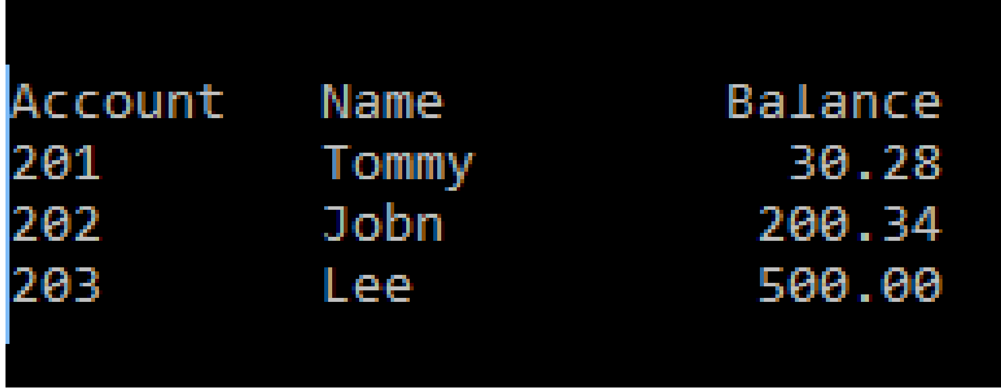
2. write the data to terminal (cout) line by line.

Go Notepad++!!!

```cpp
#include <iostream>
#include <fstream>
 #include <iomanip>
 #include <cstdlib> // exit prototype
using namespace std;
using std::ios;
using std::cerr;
using std::fixed;
using std::showpoint;
void outputLine( int account, const char * const name,  double balance ){
      cout << left << setw( 10 ) << account << setw( 13 ) << name
          << setw( 7 ) << setprecision( 2 ) << right << balance << endl;
} // end function outputLine

int main(){
   int account;
   char name[ 30 ];
   double balance;
   // ifstream constructor opens the file
   ifstream inClientFile( "clients.dat", ios::in );
   // exit program if ifstream could not open file
   if ( !inClientFile ) {
      cerr << "File could not be opened" << endl;
      exit( 1 );
   } // end if

    cout<<left<<setw(10)<<"Account"<<setw(13)<<"Name"<<"Balance"<<endl<<fixed<<showpoint;
    while (inClientFile >> account >> name >> balance) outputLine( account, name, balance );
    return 0; // ifstream destructor closes the file
} // end main
```

Open and test file for input.

Read from file until EOF found.


```
Account    Name          Balance
201        Tommy           30.28
202        Jobn           200.34
203        Lee            500.00
```

LECTURE 2

Read Data From Sequential File (Char)

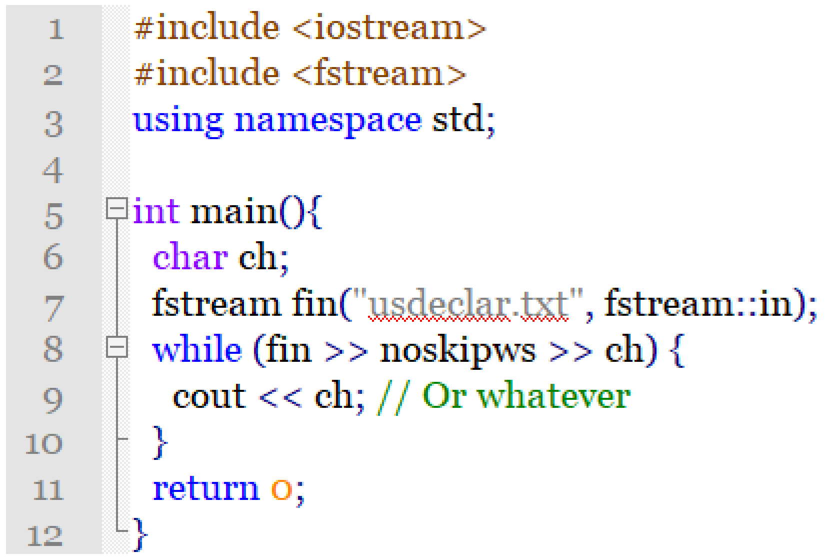
Read a File Character by Character

Demo Program: char1.cpp

1. Using no skip words mode by istream

2. use noskipws mode and stream input to read in character by character.

3. When the input file stream is empty the fin >> statement will get a NULL symbol.



```
#include <iostream>
#include <fstream>
using namespace std;

int main(){
  char ch;
  fstream fin("usdeclar.txt", fstream::in);
  while (fin >> noskipws >> ch) {
    cout << ch; // Or whatever
  }
  return 0;
}
```

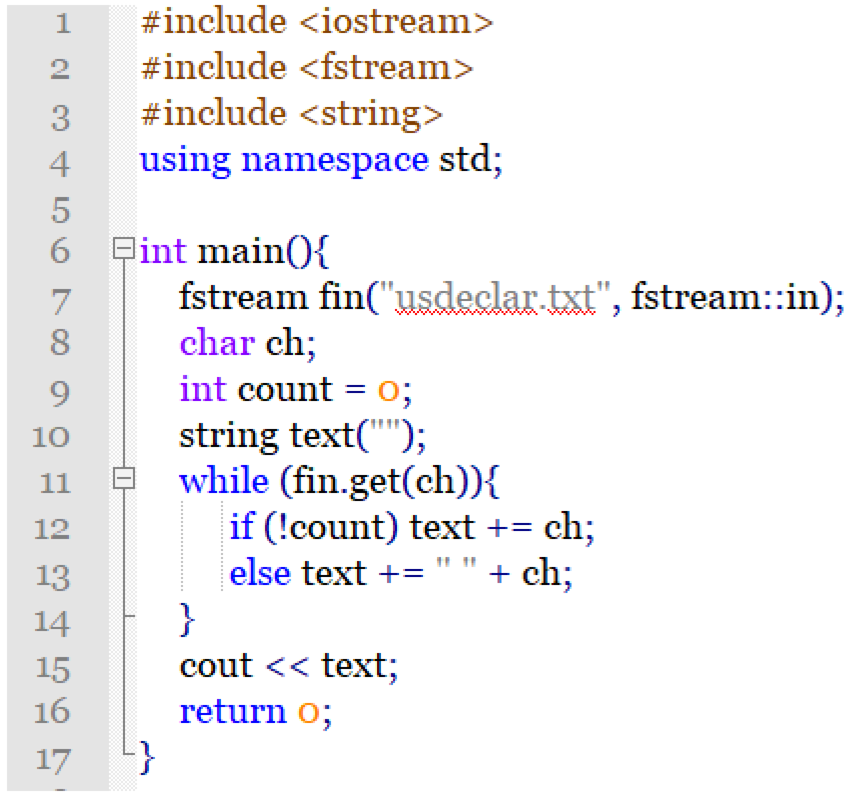
Read a File Character by Character

Demo Program: char2.cpp

1. Using get() instance method for input stream.

2. Obtain a NULL symbol if run out of input characters.

3. Merge all characters to a text string for later processing.



```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main(){
    fstream fin("usdeclar.txt", fstream::in);
    char ch;
    int count = 0;
    string text("");
    while (fin.get(ch)){
        if (!count) text += ch;
        else text += " " + ch;
    }
    cout << text;
    return 0;
}
```

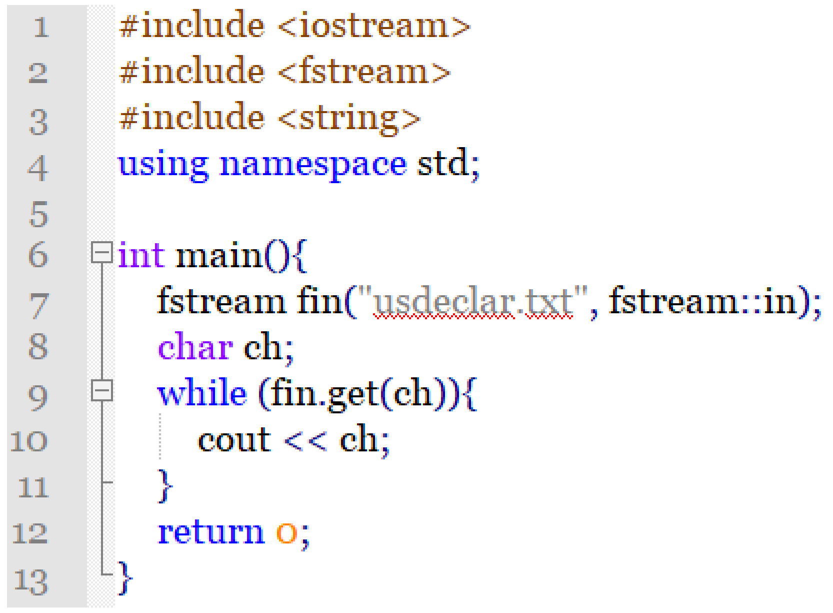
Read a File Character by Character

Demo Program: char3.cpp

1. Using get() instance method for input stream.

2. Obtain a NULL symbol if run out of input characters.

3. Just output the character to console.



```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main(){
    fstream fin("usdeclar.txt", fstream::in);
    char ch;
    while (fin.get(ch)){
        cout << ch;
    }
    return 0;
}
```

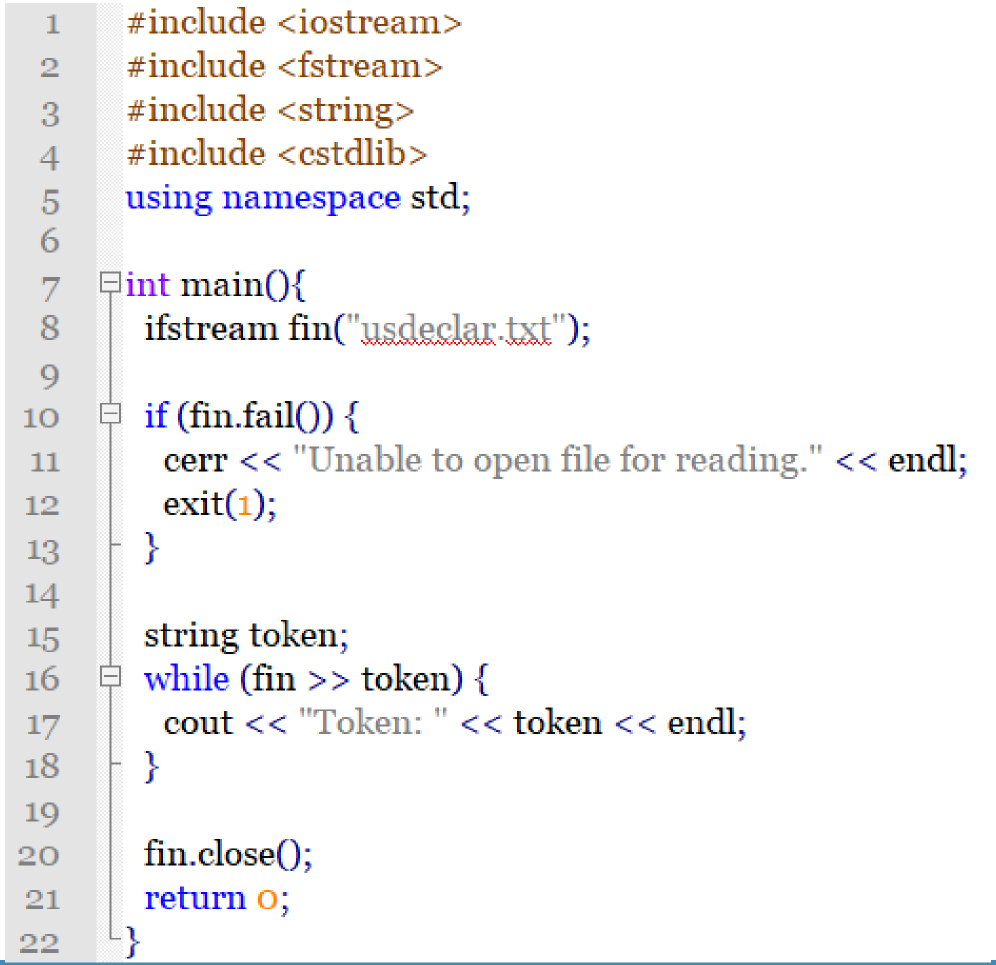
LECTURE 2

Read Data From Sequential File (Token)

Read a File Token by Token

Demo Program: token1.cpp

1. Read in one string at a time.



```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
using namespace std;

int main(){
  ifstream fin("usdeclar.txt");

  if (fin.fail()) {
    cerr << "Unable to open file for reading." << endl;
    exit(1);
  }

  string token;
  while (fin >> token) {
    cout << "Token: " << token << endl;
  }

  fin.close();
  return 0;
}
```

Read a File Token by Token With Processing

Demo Program: token2.cpp

1. Read in one string at a time.
2. remove all the numbers and punctuation marks.
3. convert the token to lower case.
4. trim() out the whitespace character.
5. merge to a long text string.
6. Convert the string a vector of tokens. (Empty strings removed)

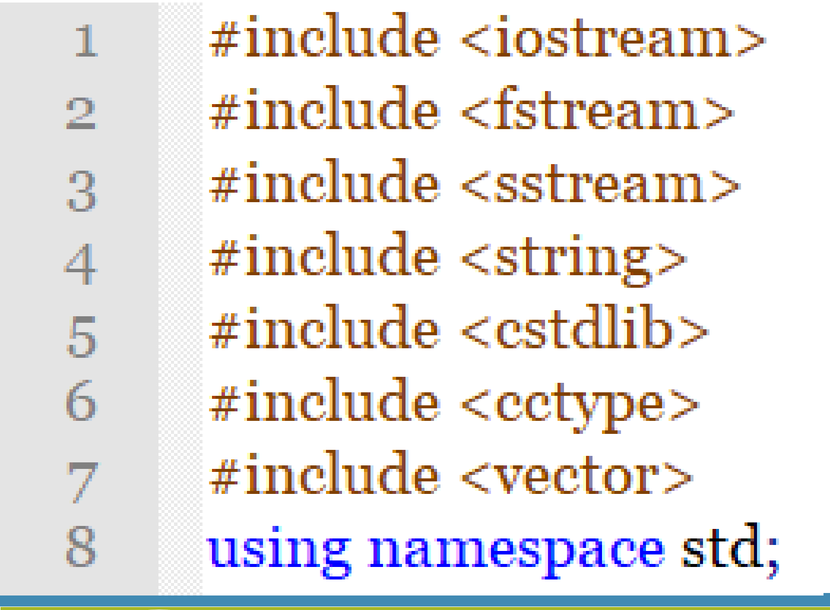
Result: A vector word list of all words in the text document.

(No number, no punctuation marks.)

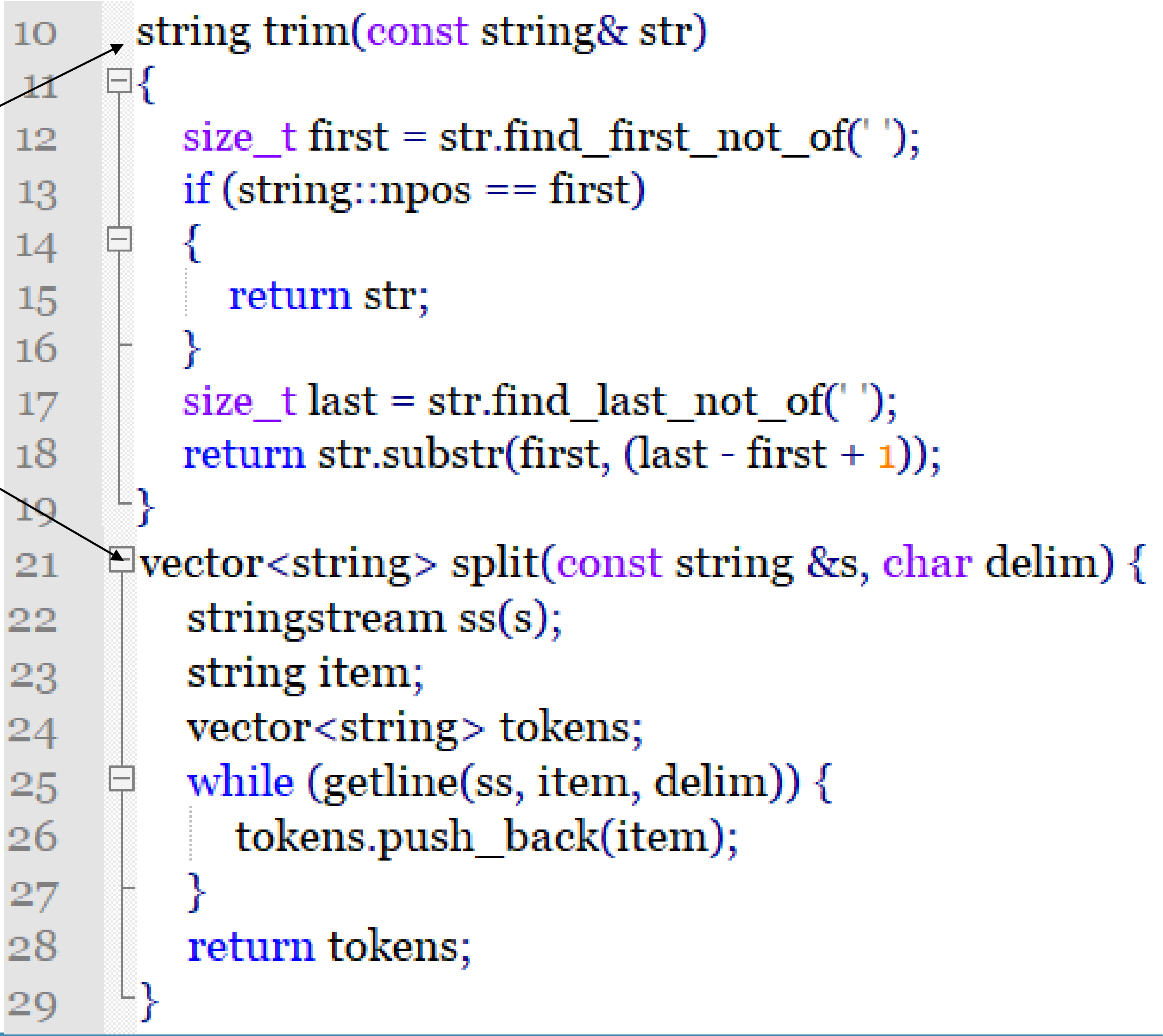
token2.cpp

trim() function: take out all leading and trailing spaces.

Split a string by delimiter into a vector of tokens



```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <cstdlib>
#include <cctype>
#include <vector>
using namespace std;
```

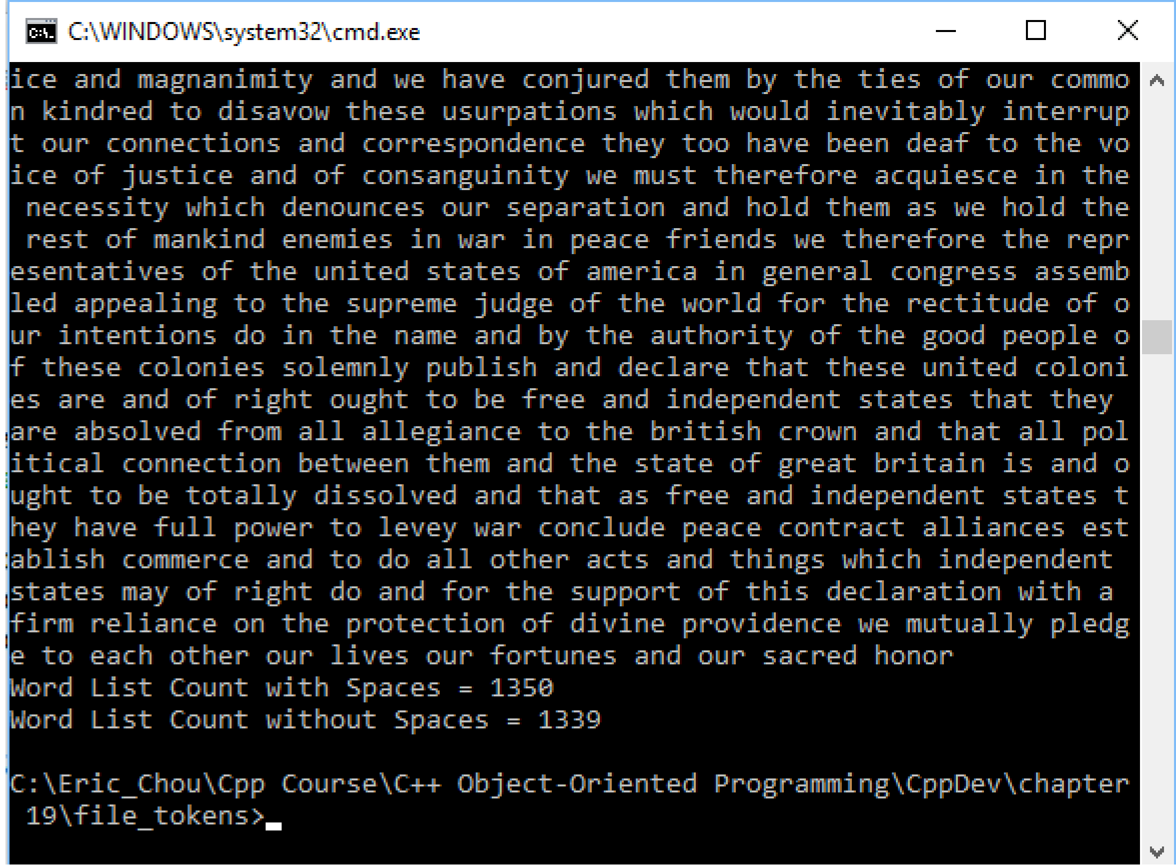


```cpp
string trim(const string& str)
{
    size_t first = str.find_first_not_of(' ');
    if (string::npos == first)
    {
        return str;
    }
    size_t last = str.find_last_not_of(' ');
    return str.substr(first, (last - first + 1));
}
vector<string> split(const string &s, char delim) {
    stringstream ss(s);
    string item;
    vector<string> tokens;
    while (getline(ss, item, delim)) {
        tokens.push_back(item);
    }
    return tokens;
}
```

```cpp
int main(){
  ifstream fin("usdeclar.txt");
  if (fin.fail()) {
    cerr << "Unable to open file for reading." << endl;
    exit(1);
  }
  string text("");
  string token;

  int count = 0;
  while (fin >> token) {
    // remove non-letter, no-space letters.
    string str("");
    for (int i=0; i<token.length(); i++){
      if (!isalpha(token[i]) && token[i] != ' ') str += ' ';
      else str  += tolower(token[i]);
    }
    str = trim(str);
    text += str + " ";
  }

  vector <string> wlist = split(text, ' ');
  vector <string> wlist2;
  for(int i=0; i<wlist.size(); i++) {
     if (wlist[i].length() != 0) { wlist2.push_back(wlist[i]);  cout << wlist[i] << " " ;}
  }
  cout << endl;
  cout << "Word List Count with Spaces = "<< wlist.size()<< endl;
  cout <<  "Word List Count without Spaces = "<< wlist2.size() << endl;

  fin.close();
  return 0;
}
```

Remove all punctuation marks, \t, \n, \b, and etc. Convert the tokens to lower case.

Convert the text string into a vector of clean tokens



```
ice and magnanimity and we have conjured them by the ties of our commo
n kindred to disavow these usurpations which would inevitably interrup
t our connections and correspondence they too have been deaf to the vo
ice of justice and of consanguinity we must therefore acquiesce in the
 necessity which denounces our separation and hold them as we hold the
 rest of mankind enemies in war in peace friends we therefore the repr
esentatives of the united states of america in general congress assemb
led appealing to the supreme judge of the world for the rectitude of o
ur intentions do in the name and by the authority of the good people o
f these colonies solemnly publish and declare that these united coloni
es are and of right ought to be free and independent states that they
are absolved from all allegiance to the british crown and that all pol
itical connection between them and the state of great britain is and o
ught to be totally dissolved and that as free and independent states t
hey have full power to levey war conclude peace contract alliances est
ablish commerce and to do all other acts and things which independent
states may of right do and for the support of this declaration with a
firm reliance on the protection of divine providence we mutually pledg
e to each other our lives our fortunes and our sacred honor
Word List Count with Spaces = 1350
Word List Count without Spaces = 1339

C:\Eric_Chou\Cpp Course\C++ Object-Oriented Programming\CppDev\chapter
 19\file_tokens>
```

LECTURE 2

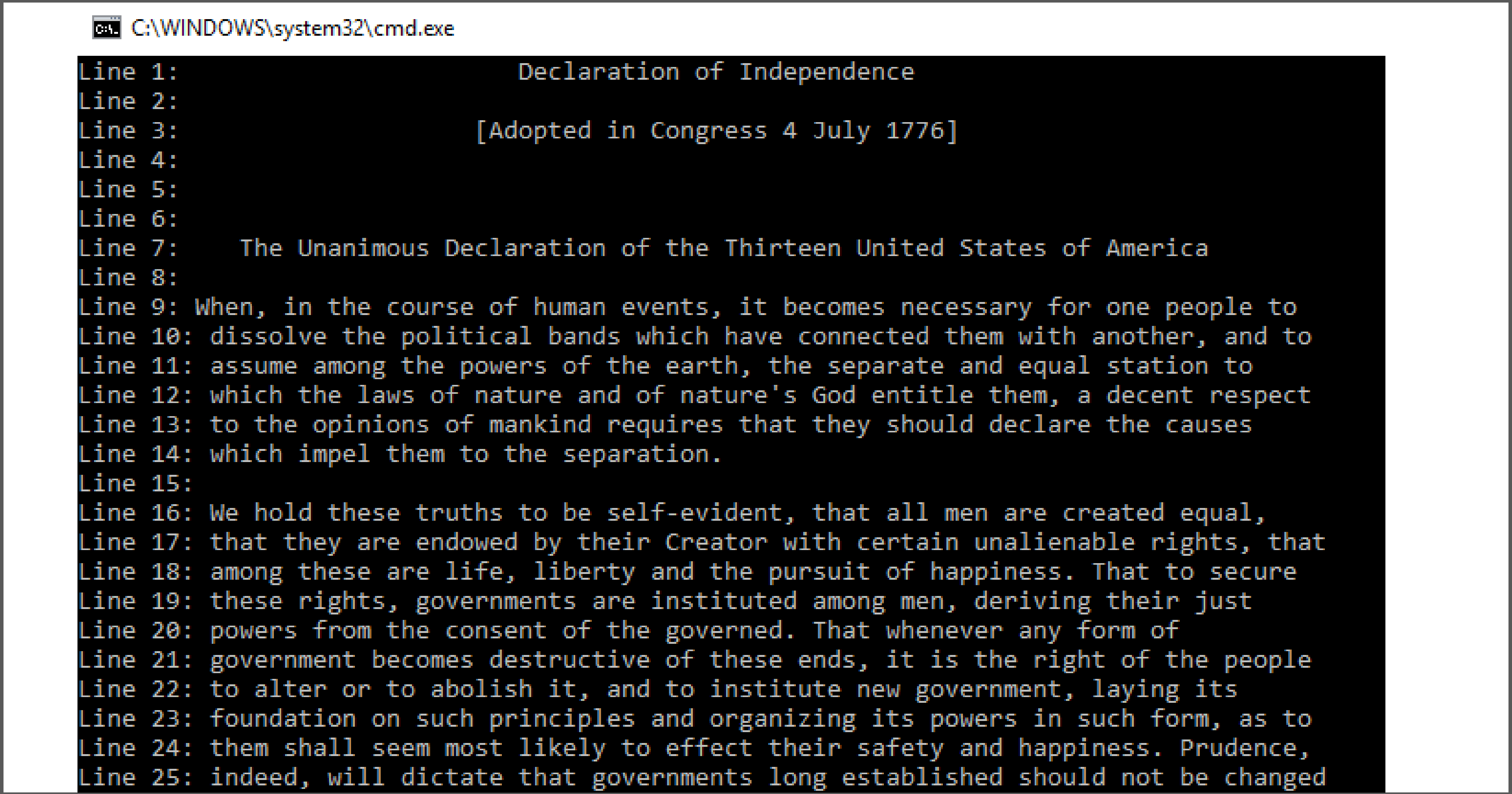
Read Data From Sequential File (Line)

Demo Program: line1.cpp

1. Get a line from the input text file at a time.

Go Notepad++!!!

```cpp
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main(){
    ifstream fin("usdeclar.txt", fstream::in);
    char ch;
    int count = 0;
    string line;
    while (getline(fin, line)){
        cout << "Line " << ++count << ": " << line << endl;
    }
    return 0;
}
```



C:\WINDOWS\system32\cmd.exe

```
Line 1:                       Declaration of Independence
Line 2:
Line 3:                    [Adopted in Congress 4 July 1776]
Line 4:
Line 5:
Line 6:
Line 7:    The Unanimous Declaration of the Thirteen United States of America
Line 8:
Line 9: When, in the course of human events, it becomes necessary for one people to
Line 10: dissolve the political bands which have connected them with another, and to
Line 11: assume among the powers of the earth, the separate and equal station to
Line 12: which the laws of nature and of nature's God entitle them, a decent respect
Line 13: to the opinions of mankind requires that they should declare the causes
Line 14: which impel them to the separation.
Line 15:
Line 16: We hold these truths to be self-evident, that all men are created equal,
Line 17: that they are endowed by their Creator with certain unalienable rights, that
Line 18: among these are life, liberty and the pursuit of happiness. That to secure
Line 19: these rights, governments are instituted among men, deriving their just
Line 20: powers from the consent of the governed. That whenever any form of
Line 21: government becomes destructive of these ends, it is the right of the people
Line 22: to alter or to abolish it, and to institute new government, laying its
Line 23: foundation on such principles and organizing its powers in such form, as to
Line 24: them shall seem most likely to effect their safety and happiness. Prudence,
Line 25: indeed, will dictate that governments long established should not be changed
```

LECTURE 2

Read Data From Sequential File (Block)

Demo Program: block.cpp

1. Read the whole text file into a stringstream by buffer read mode (**rdbuf()**).
2. Convert the stringstream object into string by **str()**.

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
using namespace std;

int main(){
  ifstream fin;
  fin.open("usdeclar.txt");

  stringstream strStream;
  strStream << fin.rdbuf();  //read the file in buffer mode into a stringstream strStream
  string str = strStream.str();//str holds the content of the file

  cout << str << endl;//you can do anything with the string!!!
  return 0;
}
```

Summary

Read Characters from File

```
while (fin.get(ch)){
    cout << ch;
}
```

Read Lines from File

```
string line;
while (getline(fin, line)){
    cout << "Line " << ++count << ": " << line << endl;
}
```

Read Tokens from File

```
string token;
while (fin >> token) {
  cout << "Token: " << token << endl;
}
```

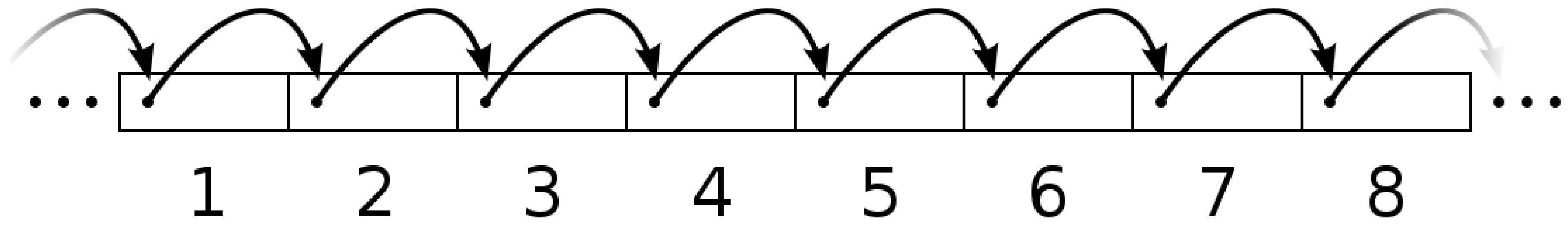
Read a Block from File

```
stringstream strStream;
strStream << fin.rdbuf();
string str = strStream.str();
```

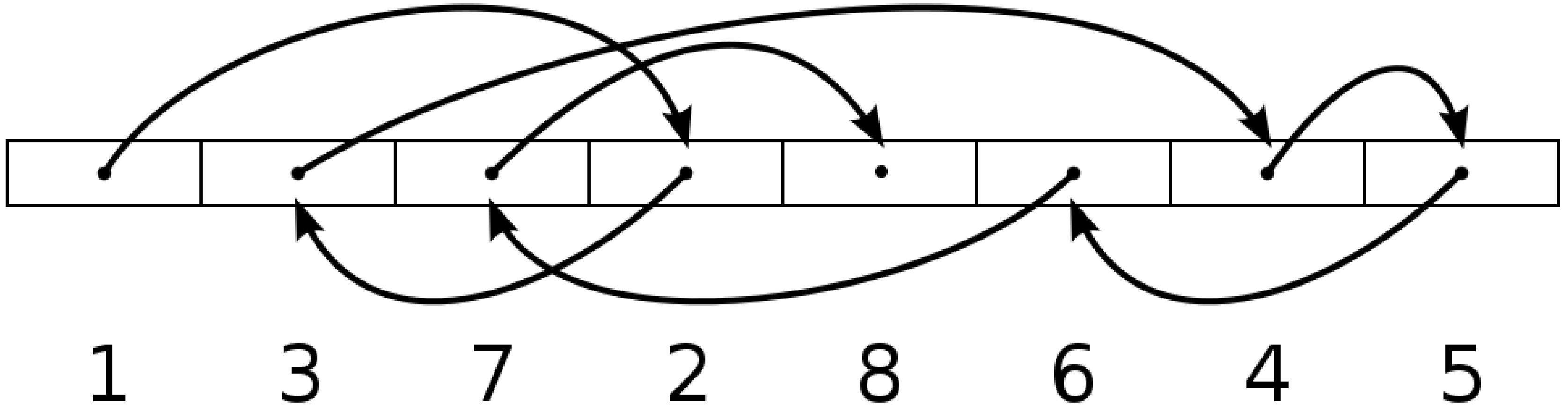
LECTURE 2

Read Data From Sequential File

Sequential access



Random access



Reading Data from a Sequential-Access File

File position pointers

- Number of next byte to read/write
- Functions to reposition pointer
  - `seekg` (seek get for `istream` class)
  - `seekp` (seek put for `ostream` class)
  - Classes have "get" and "put" pointers
- `seekg` and `seekp` take *offset* and *direction*
  - Offset: number of bytes relative to direction
  - Direction (`ios::beg` default)
    - `ios::beg` - relative to beginning of stream
    - `ios::cur` - relative to current position
    - `ios::end` - relative to end

Reading Data from a Sequential-Access File

Examples

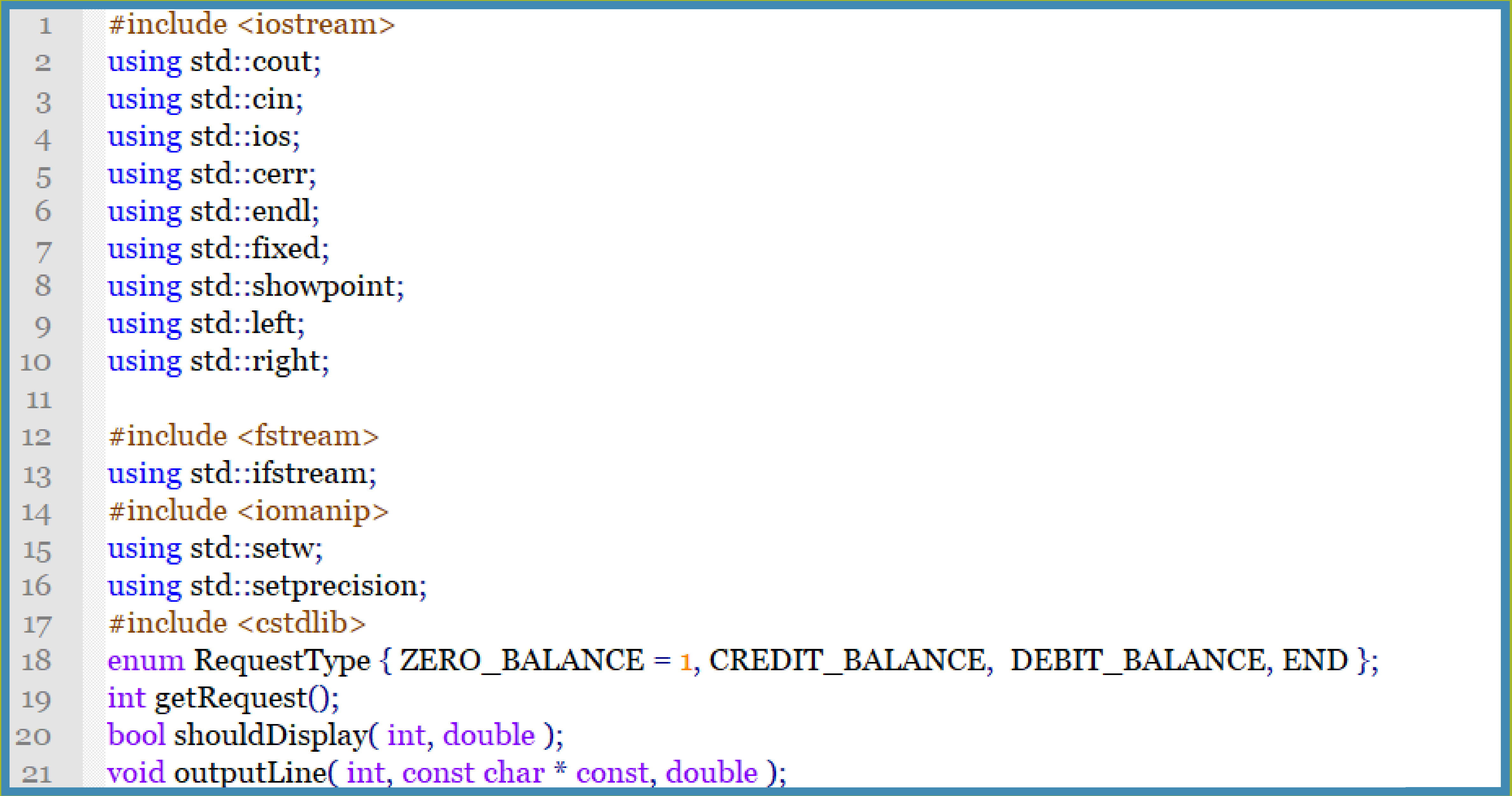
- **`fileObject.seekg(0)`**
  - Goes to front of file (location 0) because **`ios::beg`** is default
- **`fileObject.seekg(n)`**
  - Goes to nth byte from beginning
- **`fileObject.seekg(n, ios::cur)`**
  - Goes n bytes forward
- **`fileObject.seekg(y, ios::end)`**
  - Goes y bytes back from end
- **`fileObject.seekg(0, ios::cur)`**
  - Goes to last byte
- **`seekp`** similar

Reading Data from a Sequential-Access File

- To find pointer location
  - **tellg** and **tellp**
  - **location = fileObject.tellg()**
- Upcoming example
  - Credit manager program
  - List accounts with zero balance, credit, and debit

Sequential File Acces

Function	Description
seekg()	Moves get pointer (input) to a specified location
seekp()	Moves put pointer(output) to a specified location
tellg()	Gives the current position of the get pointer
tellp()	Gives the current position of the put pointer

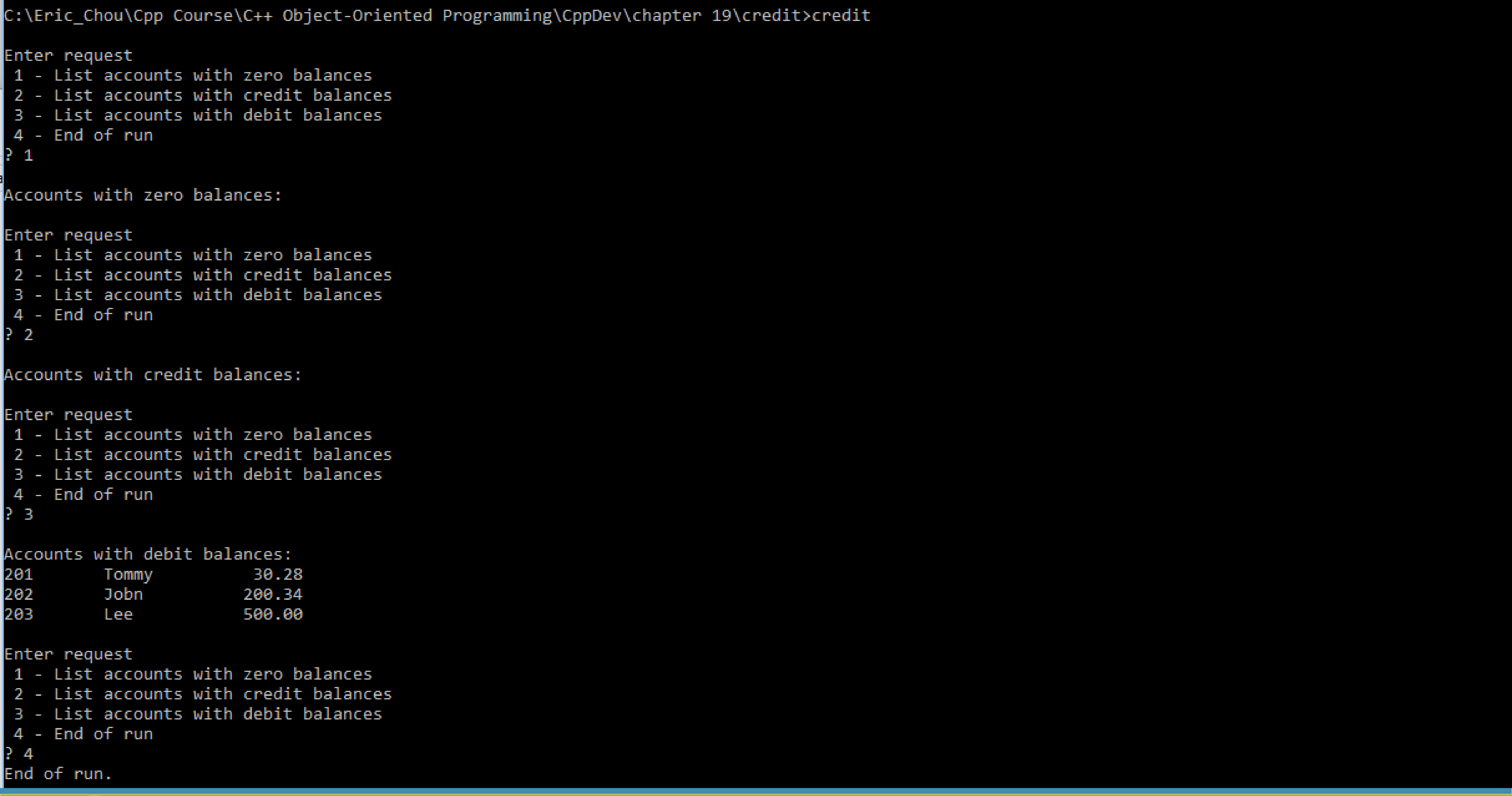


```cpp
#include <iostream>
using std::cout;
using std::cin;
using std::ios;
using std::cerr;
using std::endl;
using std::fixed;
using std::showpoint;
using std::left;
using std::right;

#include <fstream>
using std::ifstream;
#include <iomanip>
using std::setw;
using std::setprecision;
#include <cstdlib>
enum RequestType { ZERO_BALANCE = 1, CREDIT_BALANCE,  DEBIT_BALANCE, END };
int getRequest();
bool shouldDisplay( int, double );
void outputLine( int, const char * const, double );
```

```cpp
int main(){
    // ifstream constructor opens the file
    ifstream inClientFile( "clients.dat", ios::in );
    // exit program if ifstream could not open file
    if ( !inClientFile ) {
      cerr << "File could not be opened" << endl;
      exit(1);
    } // end if
    int request;
    int account;
    char name[ 30 ];
    double balance;
    // get user's request (e.g., zero, credit or debit balance)
    request = getRequest();
    // process user's request
    while ( request != END ) {
      switch ( request ) {
        case ZERO_BALANCE:
          cout << "\nAccounts with zero balances:\n";
          break;
        case CREDIT_BALANCE:
          cout << "\nAccounts with credit balances:\n";
          break;
        case DEBIT_BALANCE:
          cout << "\nAccounts with debit balances:\n";
          break;
      } // end switch
      // read account, name and balance from file
      inClientFile >> account >> name >> balance;
      // display file contents (until eof)
      while (!inClientFile.eof()) {
        // display record
        if ( shouldDisplay( request, balance ) )
          outputLine( account, name, balance );
        // read account, name and balance from file
        inClientFile >> account >> name >> balance;
      } // end inner while
      inClientFile.clear();   // reset eof for next input
      inClientFile.seekg( 0 ); // move to beginning of file
      request = getRequest();  // get additional request from user
    } // end outer while
    cout << "End of run." << endl;
    return 0; // ifstream destructor closes the file
} // end main
```

Use **`clear`** to reset eof. Use **`seekg`** to set file position pointer to beginning of file.

```
// obtain request from user
 int getRequest(){
  int request;
  // display request options
  cout << "\nEnter request" << endl
     << " 1 - List accounts with zero balances" << endl
     << " 2 - List accounts with credit balances" << endl
     << " 3 - List accounts with debit balances" << endl
     << " 4 - End of run" << fixed << showpoint;
  // input user request
  do {
    cout << "\n? ";
    cin >> request;
  } while ( request < ZERO_BALANCE && request > END );
  return request;
} // end function getRequest

// determine whether to display given record
bool shouldDisplay( int type, double balance ){
  // determine whether to display credit balances
  if ( type == CREDIT_BALANCE && balance < 0 )
    return true;
  // determine whether to display debit balances
  if ( type == DEBIT_BALANCE && balance > 0 )
    return true;
  // determine whether to display zero balances
  if ( type == ZERO_BALANCE && balance == 0 )
    return true;
  return false;
} // end function shouldDisplay

// display single record from file
void outputLine( int account, const char * const name,  double balance ){
  cout << left << setw( 10 ) << account << setw( 13 ) << name
     << setw( 7 ) << setprecision( 2 ) << right << balance
     << endl;
} // end function outputLine
```



```
C:\Eric_Chou\Cpp Course\C++ Object-Oriented Programming\CppDev\chapter 19\credit>credit

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 1

Accounts with zero balances:

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 2

Accounts with credit balances:

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 3

Accounts with debit balances:
201       Tommy          30.28
202       Jobn          200.34
203       Lee           500.00

Enter request
 1 - List accounts with zero balances
 2 - List accounts with credit balances
 3 - List accounts with debit balances
 4 - End of run
? 4
End of run.
```

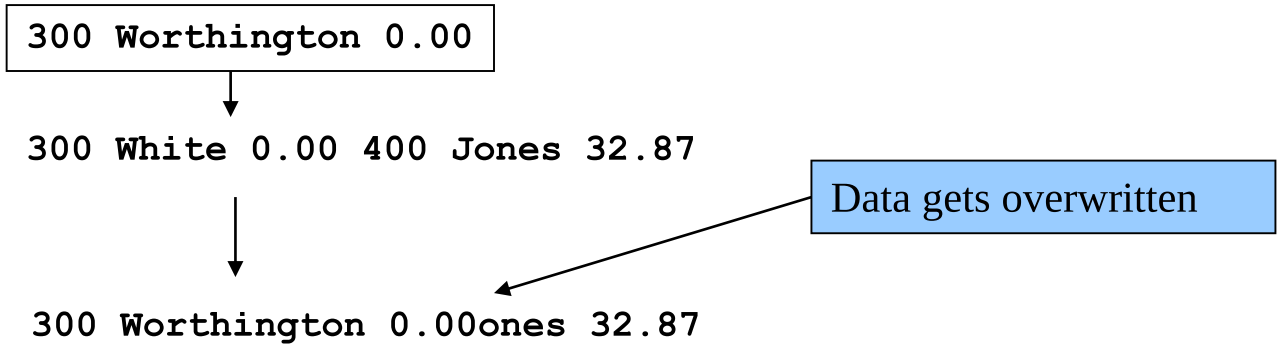
LECTURE 2

Writing Sequential Files

Updating Sequential-Access Files

Updating sequential files

- Risk overwriting other data
- Example: change name "White" to "Worthington"
  - Old data
  - `300 White 0.00 400 Jones 32.87`
  - Insert new data



```
300 Worthington 0.00

300 White 0.00 400 Jones 32.87

300 Worthington 0.00ones 32.87
```

- Formatted text different from internal representation
- Problem can be avoided, but awkward

LECTURE 2

Random Access Files

Random-Access Files

Instant access

- Want to locate record quickly
  - Airline reservations, ATMs
- Sequential files must search through each one

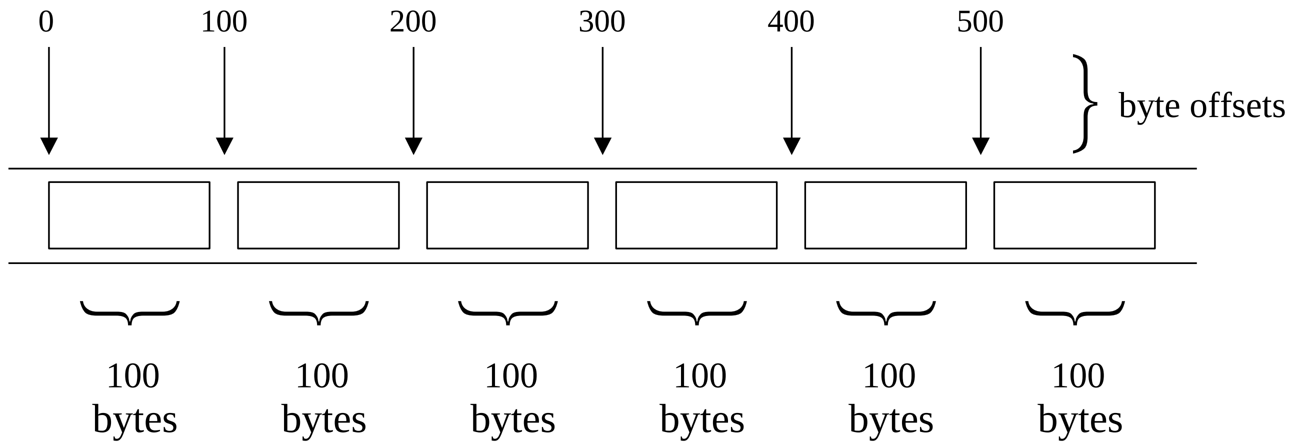
Random-access files are solution

- Instant access
- Insert record without destroying other data
- Update/delete items without changing other data

Random-Access Files

C++ imposes no structure on files

- Programmer must create random-access files
- Simplest way: fixed-length records
  - Calculate position in file from record size and key



Creating a Random-Access File

`"1234567"` (`char *`) vs `1234567` (`int`)

- `char *` takes 8 bytes (1 for each character + null)
- `int` takes fixed number of bytes (perhaps 4)
  - 123 same size in bytes as 1234567

`<<` operator and `write()`

- **`outFile << number`**
  - Outputs **`number`** (**`int`**) as a **`char *`**
  - Variable number of bytes
- **`outFile.write( *const char **, *size* );`**
  - Outputs raw bytes
  - Takes pointer to memory location, number of bytes to write
    - Copies data directly from memory into file
    - Does not convert to **`char *`**

Creating a Random-Access File

Example

```
outFile.write( reinterpret_cast<const char
  *>(&number), sizeof( number ) );
```

- `&number` is an `int *`
  - Convert to `const char *` with `reinterpret_cast`
- `sizeof(number)`
  - Size of `number` (an `int`) in bytes
- `read` function similar (more later)
- Must use `write`/`read` between compatible machines
  - Only when using raw, unformatted data
- Use `ios::binary` for raw writes/reads

Creating a Random-Access File

Usually write entire **`struct`** or object to file

Problem statement

- Credit processing program
- Store at most 100 fixed-length records
- Record
  - Account number (key)
  - First and last name
  - Balance
- Account operations
  - Update, create new, delete, list all accounts in a file

Next: program to create blank 100-record file

Project not finished Yet.

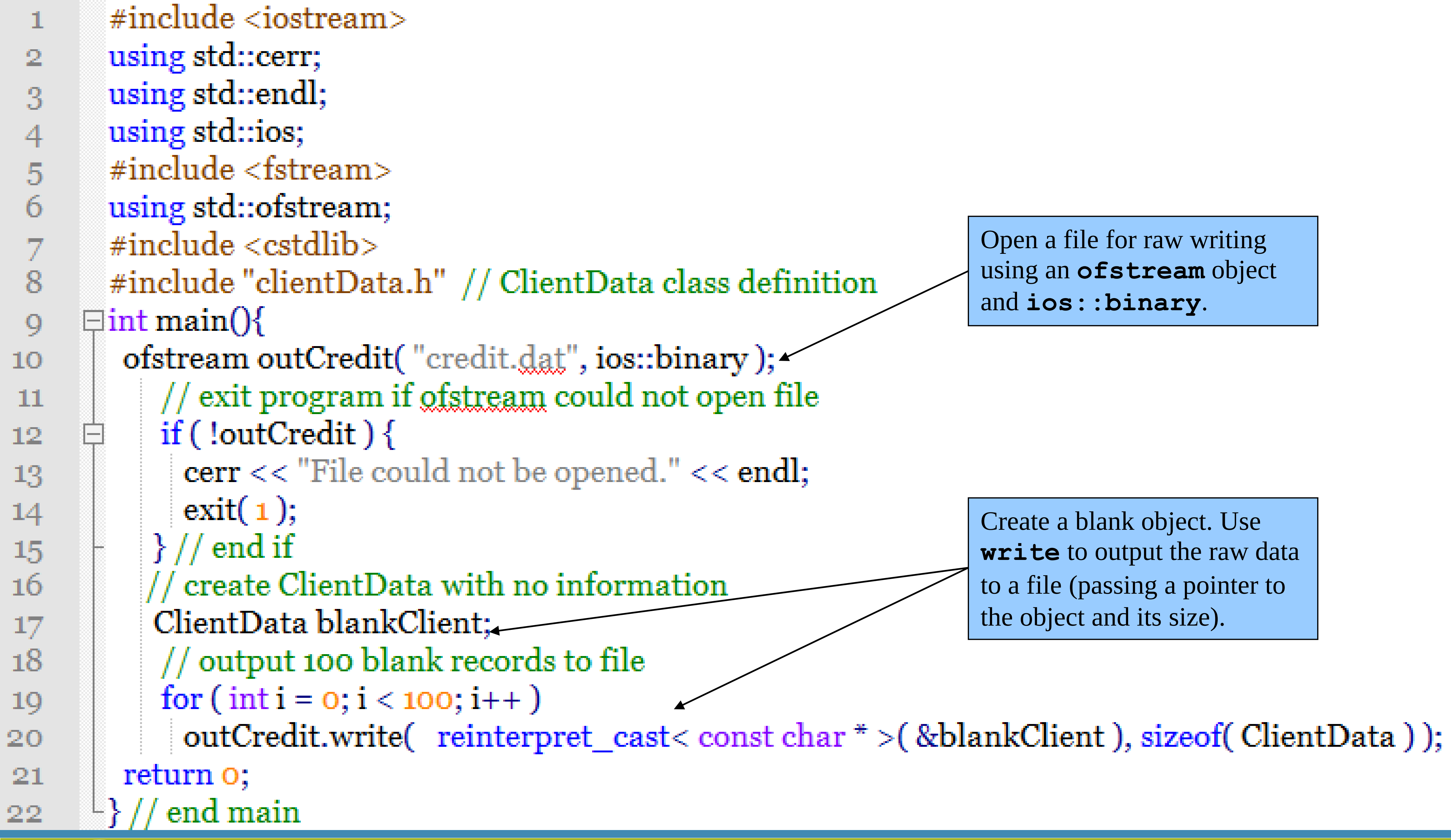
```cpp
#ifndef CLIENTDATA_H
#define CLIENTDATA_H
#include <iostream>
using std::string;
class ClientData {
  public:
    ClientData( int = 0, string = "", string = "", double = 0.0 );  // default ClientData constructor
    void setAccountNumber( int );   // accessor functions for accountNumber
    int getAccountNumber() const;
    void setLastName( string );     // accessor functions for lastName
    string getLastName() const;
    void setFirstName( string );    // accessor functions for firstName
    string getFirstName() const;
    void setBalance( double );      // accessor functions for balance
    double getBalance() const;

  private:
    int accountNumber;
    char lastName[15];
    char firstName[10];
    double balance;
}; // end class ClientData
#endif
```

Class **`ClientData`** stores the information for each person. 100 blank **`ClientData`** objects will be written to a file.

Put limits on the size of the first and last name. **`accountNumber`** (an **`int`**) and **`balance`** (**`double`**) are already of a fixed size.

```cpp
#include <iostream>
#include <cstring>
#include "clientData.h"
using std::string;
// default ClientData constructor
ClientData::ClientData( int accountNumberValue,  string lastNameValue, string firstNameValue,
                        double balanceValue ){
    setAccountNumber( accountNumberValue );
    setLastName( lastNameValue );
    setFirstName( firstNameValue );
    setBalance( balanceValue );
} // end ClientData constructor

// get account-number value
int ClientData::getAccountNumber() const{
    return accountNumber;
} // end function getAccountNumber
// set account-number value
void ClientData::setAccountNumber( int accountNumberValue ){
    accountNumber = accountNumberValue;
} // end function setAccountNumber

// get last-name value
string ClientData::getLastName() const{
    return lastName;

} // end function getLastName

```

```
// set last-name value
void ClientData::setLastName( string lastNameString ){
    // copy at most 15 characters from string to lastName
    const char *lastNameValue = lastNameString.data();
    int length = strlen( lastNameValue );
    length = ( length < 15 ? length : 14 );
    strncpy( lastName, lastNameValue, length );
    // append null character to lastName
    lastName[ length ] = '\0';
} // end function setLastName

// get first-name value
string ClientData::getFirstName() const{
    return firstName;
} // end function getFirstName

// set first-name value
void ClientData::setFirstName( string firstNameString ){
    // copy at most 10 characters from string to firstName
    const char *firstNameValue = firstNameString.data();
    int length = strlen( firstNameValue );
    length = ( length < 10 ? length : 9 );
    strncpy( firstName, firstNameValue, length );
    // append new-line character to firstName
    firstName[ length ] = '\0';
} // end function setFirstName

// get balance value
double ClientData::getBalance() const{
    return balance;
} // end function getBalance

// set balance value
void ClientData::setBalance( double balanceValue ){
    balance = balanceValue;
} // end function setBalance
```



```
#include <iostream>
using std::cerr;
using std::endl;
using std::ios;
#include <fstream>
using std::ofstream;
#include <cstdlib>
#include "clientData.h"  // ClientData class definition
int main(){
 ofstream outCredit( "credit.dat", ios::binary );
    // exit program if ofstream could not open file
    if ( !outCredit ) {
      cerr << "File could not be opened." << endl;
      exit( 1 );
   } // end if
   // create ClientData with no information
   ClientData blankClient;
    // output 100 blank records to file
    for ( int i = 0; i < 100; i++ )
      outCredit.write(  reinterpret_cast< const char * >( &blankClient ), sizeof( ClientData ) );
 return 0;
} // end main
```

Open a file for raw writing using an **`ofstream`** object and **`ios::binary`**.

Create a blank object. Use **`write`** to output the raw data to a file (passing a pointer to the object and its size).

LECTURE 2

Writing Random Access Files

Writing Data Randomly to a Random-Access File

Use **`seekp`** to write to exact location in file

- Where does the first record begin?
  - Byte 0
- The second record?
  - Byte 0 + sizeof(object)
- Any record?
  - (Recordnum - 1) * sizeof(object)

random2.cpp

```cpp
#include <iostream>
using std::cerr;
using std::endl;
using std::cout;
using std::cin;
using std::ios;
#include <iomanip>
using std::setw;
#include <fstream>
using std::ofstream;
#include <cstdlib>
#include "clientData.h"  // ClientData class definition
```

```cpp
int main(){
    int accountNumber;
    char lastName[ 15 ];
    char firstName[ 10 ];
    double balance;
    ofstream outCredit( "credit.dat", ios::binary );

    // exit program if ofstream cannot open file
    if ( !outCredit ) {
      cerr << "File could not be opened." << endl;
      exit( 1 );
    } // end if
    cout << "Enter account number "
       << "(1 to 100, 0 to end input)\n? ";

    // require user to specify account number
    ClientData client;
    cin >> accountNumber;
    client.setAccountNumber( accountNumber );
    // user enters information, which is copied into file
    while ( client.getAccountNumber() > 0 &&
      client.getAccountNumber() <= 100 ) {

      // user enters last name, first name and balance
      cout << "Enter lastname, firstname, balance\n? ";
      cin >> setw( 15 ) >> lastName;
      cin >> setw( 10 ) >> firstName;
      cin >> balance;

      // set record lastName, firstName and balance values
      client.setLastName( lastName );
      client.setFirstName( firstName );
      client.setBalance( balance );

      // seek position in file of user-specified record
      outCredit.seekp( ( client.getAccountNumber() - 1 ) *
        sizeof( ClientData ) );

      // write user-specified information in file
      outCredit.write(
        reinterpret_cast< const char * >( &client ),
        sizeof( ClientData ) );
      // enable user to specify another account number
      cout << "Enter account number\n? ";
      cin >> accountNumber;
      client.setAccountNumber( accountNumber );

    } // end while
    return 0;
} // end main
```

Open file for raw (binary) writing.

Position **outCredit** to the proper location in the file (based on the account number).

Get account number, put into object. It has not yet been written to file.

Write **ClientData** object to file at specified position.

```
Enter account number (1 to 100, 0 to end input)
? 37
Enter lastname, firstname, balance
? Barker Doug 0.00
Enter account number
? 29
Enter lastname, firstname, balance
? Brown Nancy -24.54
Enter account number
? 96
Enter lastname, firstname, balance
? Stone Sam 34.98
Enter account number
? 88
Enter lastname, firstname, balance
? Smith Dave 258.34
Enter account number
? 33
Enter lastname, firstname, balance
? Dunn Stacey 314.33
Enter account number
? 0
```

Notice that accounts can be created in any order.

LECTURE 2

Read Data Sequentially from Random Access File

Reading Data Sequentially from a Random-Access File

`read` - similar to **`write`**

- Reads raw bytes from file into memory
- **`inFile.read( reinterpret_cast<char *>( &number ),`**
  `sizeof( int ) );`
  - **`&number`**: location to store data
  - **`sizeof(int)`**: how many bytes to read
- Do not use **`inFile >> number`** with raw bytes
  - >> expects **`char *`**

Upcoming program

- Output data from a random-access file
- Go through each record sequentially
  - If no data (accountNumber == 0) then skip

seqRand.cpp

```cpp
// display single record
void outputLine( ostream &output, const ClientData &record ){
    output << left << setw( 10 ) << record.getAccountNumber()
        << setw( 16 ) << record.getLastName().data()
        << setw( 11 ) << record.getFirstName().data()
        << setw( 10 ) << setprecision( 2 ) << right << fixed
        << showpoint << record.getBalance() << endl;
} // end outputLine

int main(){
    ifstream inCredit( "credit.dat", ios::in );
     // exit program if ifstream cannot open file
     if ( !inCredit ) {
       cerr << "File could not be opened." << endl;
       exit( 1 );
       } // end if
       cout << left << setw( 10 ) << "Account" << setw( 16 )
         << "Last Name" << setw( 11 ) << "First Name" << left
         << setw( 10 ) << right << "Balance" << endl;
     ClientData client; // create record
     // read first record from file
     inCredit.read( reinterpret_cast< char * >( &client ),
       sizeof( ClientData ) );
     // read all records from file
     while ( inCredit && !inCredit.eof() ) {
       // display record
       if ( client.getAccountNumber() != 0 )
         outputLine( cout, client );
       // read next from file
       inCredit.read( reinterpret_cast< char * >( &client ),
         sizeof( ClientData ) );
       } // end while
     return 0;
} // end main
```

Read **`sizeof(ClientData)`** bytes and put into object **`client`**. This may be an empty record.

Loop exits if there is an error reading (**`inCredit == 0`**) or EOF is found (**`inCredit.eof() == 1`**)

Output non-empty accounts. Note that **`outputLine`** takes an **`ostream`** argument. We could easily output to another file (opened with an **`ofstream`** object, which derives from **`ostream`**).

```cpp
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdlib>  // exit
#include "clientData.h"  // ClientData class definition
using namespace std;
```

```
Account    Last Name        First Name     Balance
29         Brown            Nancy           -24.54
33         Dunn             Stacey          314.33
37         Barker           Doug              0.00
88         Smith            Dave            258.34
96         Stone            Sam              34.98
```

LECTURE 2

Case Study

Example: A Transaction-Processing Program

- Instant access for bank accounts
  - Use random access file (data in **`client.dat`**)
- Give user menu
  - Option 1: store accounts to **`print.txt`**

```
Account    Last Name        First Name    Balance
29         Brown            Nancy          -24.54
33         Dunn             Stacey         314.33
37         Barker           Doug             0.00
88         Smith            Dave           258.34
96         Stone            Sam             34.98
```

  - Option 2: update record

```
Enter account to update (1 - 100): 37
37         Barker           Doug             0.00

Enter charge (+) or payment (-): +87.99
37         Barker           Doug            87.99
```

Example: A Transaction-Processing Program

Menu options (continued)

- Option 3: add new record

```
Enter new account number (1 - 100): 22
Enter lastname, firstname, balance
? Johnston Sarah 247.45
```

- Option 4: delete record

```
Enter account to delete (1 - 100): 29
Account #29 deleted.
```

To open file for reading and writing

- Use **`fstream`** object
- "Or" file-open modes together

```
fstream inOutCredit( "credit.dat", ios::in | ios::out );
```

Demo Program: bank package

bank.cpp+clientData.cpp

Go Notepad++!!!

```cpp
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdlib>      // exit prototype
#include "clientData.h"  // ClientData class definition
using namespace std;

enum Choices { PRINT = 1, UPDATE, NEW, DELETE, END };
int enterChoice();
void printRecord( fstream& );
void updateRecord( fstream& );
void newRecord( fstream& );
void deleteRecord( fstream& );
void outputLine( ostream&, const ClientData & );
int getAccount( const char * const );

// enable user to input menu choice
int enterChoice(){
    // display available options
    cout << "\nEnter your choice" << endl
      << "1 - store a formatted text file of accounts" << endl
      << "    called \"print.txt\" for printing" << endl
      << "2 - update an account" << endl
      << "3 - add a new account" << endl
      << "4 - delete an account" << endl
      << "5 - end program\n? ";
   int menuChoice;
   cin >> menuChoice; // receive choice from user
   return menuChoice;
 } // end function enterChoice
```

Go to front of file, read account data, and print record if not empty.

Note that **`outputLine`** takes an **`ostream`** object (base of **`ofstream`**). It can easily print to a file (as in this case) or **`cout`**.

```cpp
// create formatted text file for printing
void printRecord( fstream &readFromFile ){
   // create text file
   ofstream outPrintFile( "print.txt", ios::out );
   // exit program if ofstream cannot create file
   if ( !outPrintFile ) {
     cerr << "File could not be created." << endl;
     exit( 1 );
   } // end if
   outPrintFile << left << setw( 10 ) << "Account" << setw( 16 )
      << "Last Name" << setw( 11 ) << "First Name" << right
      << setw( 10 ) << "Balance" << endl;
  // set file-position pointer to beginning of record file
   readFromFile.seekg( 0 );
   // read first record from record file
   ClientData client;
   readFromFile.read( reinterpret_cast< char * >( &client ),
     sizeof( ClientData ) );

   // copy all records from record file into text file
   while ( !readFromFile.eof() ) {
     // write single record to text file
     if ( client.getAccountNumber() != 0 )
       outputLine( outPrintFile, client );
     // read next record from record file
     readFromFile.read( reinterpret_cast< char * >( &client ),
       sizeof( ClientData ) );
   } // end while
 } // end function printRecord
```

Output to **`print.txt`**. First, print the header for the table.

```cpp
void updateRecord( fstream &updateFile ){
    // obtain number of account to update
    int accountNumber = getAccount( "Enter account to update" );

    // move file-position pointer to correct record in file
    updateFile.seekg(
      ( accountNumber - 1 ) * sizeof( ClientData ) );

    // read first record from file
    ClientData client;
    updateFile.read( reinterpret_cast< char * >( &client ),
      sizeof( ClientData ) );
    // update record
    if ( client.getAccountNumber() != 0 ) {
      outputLine( cout, client );
      // request user to specify transaction
      cout << "\nEnter charge (+) or payment (-): "
      double transaction; // charge or payment
      cin >> transaction;
      // update record balance
      double oldBalance = client.getBalance();
      client.setBalance( oldBalance + transaction );
      outputLine( cout, client );
      // move file-position pointer to correct record in file
      updateFile.seekp(
        ( accountNumber - 1 ) * sizeof( ClientData ) );

      // write updated record over old record in file
      updateFile.write(
        reinterpret_cast< const char * >( &client ),
        sizeof( ClientData ) );
    } // end if
    // display error if account does not exist
    else
      cerr << "Account #" << accountNumber
        << " has no information." << endl;
  } // end function updateRecord
```

This is **`fstream`** (I/O) because we must read the old balance, update it, and write the new balance.

```cpp
void newRecord( fstream &insertInFile ){
    // obtain number of account to create
    int accountNumber = getAccount( "Enter new account number" );
    // move file-position pointer to correct record in file
    insertInFile.seekg(
      ( accountNumber - 1 ) * sizeof( ClientData ) );
    // read record from file
    ClientData client;
    insertInFile.read( reinterpret_cast< char * >( &client ),
      sizeof( ClientData ) );
    // create record, if record does not previously exist
    if ( client.getAccountNumber() == 0 ) {
      char lastName[ 15 ];
      char firstName[ 10 ];
      double balance;
      // user enters last name, first name and balance
      cout << "Enter lastname, firstname, balance\n? ";
      cin >> setw( 15 ) >> lastName;
      cin >> setw( 10 ) >> firstName;
      cin >> balance;

      // use values to populate account values
      client.setLastName( lastName );
      client.setFirstName( firstName );
      client.setBalance( balance );
      client.setAccountNumber( accountNumber );
      // move file-position pointer to correct record in file
      insertInFile.seekp( ( accountNumber - 1 ) *
        sizeof( ClientData ) );
      // insert record in file
      insertInFile.write(
        reinterpret_cast< const char * >( &client ),
        sizeof( ClientData ) );
    } // end if
    // display error if account previously exists
    else
      cerr << "Account #" << accountNumber
        << " already contains information." << endl;
} // end function newRecord
```

This is **`fstream`** because we read to see if a non-empty record already exists. If not, we write a new record.

```cpp
void deleteRecord( fstream &deleteFromFile ){
   // obtain number of account to delete
   int accountNumber = getAccount( "Enter account to delete" );
   // move file-position pointer to correct record in file
   deleteFromFile.seekg(
     ( accountNumber - 1 ) * sizeof( ClientData ) );
   // read record from file
   ClientData client;
   deleteFromFile.read( reinterpret_cast< char * >( &client ),
     sizeof( ClientData ) );
   // delete record, if record exists in file
   if ( client.getAccountNumber() != 0 ) {
     ClientData blankClient;
     // move file-position pointer to correct record in file
     deleteFromFile.seekp( ( accountNumber - 1 ) *
       sizeof( ClientData ) );
     // replace existing record with blank record
     deleteFromFile.write(
       reinterpret_cast< const char * >( &blankClient ),
       sizeof( ClientData ) );
     cout << "Account #" << accountNumber << " deleted.\n";
   } // end if
   // display error if record does not exist
   else
     cerr << "Account #" << accountNumber << " is empty.\n";
 } // end deleteRecord

// display single record
void outputLine( ostream &output, const ClientData &record ){
   output << left << setw( 10 ) << record.getAccountNumber()
        << setw( 16 ) << record.getLastName().data()
        << setw( 11 ) << record.getFirstName().data()
        << setw( 10 ) << setprecision( 2 ) << right << fixed
        << showpoint << record.getBalance() << endl;

} // end function outputLine
```

```cpp
 // obtain account-number value from user
int getAccount( const char * const prompt ){
   int accountNumber;
   // obtain account-number value
   do {
     cout << prompt << " (1 - 100): ";
     cin >> accountNumber;
   } while ( accountNumber < 1 || accountNumber > 100 );
   return accountNumber;
} // end function getAccount
```

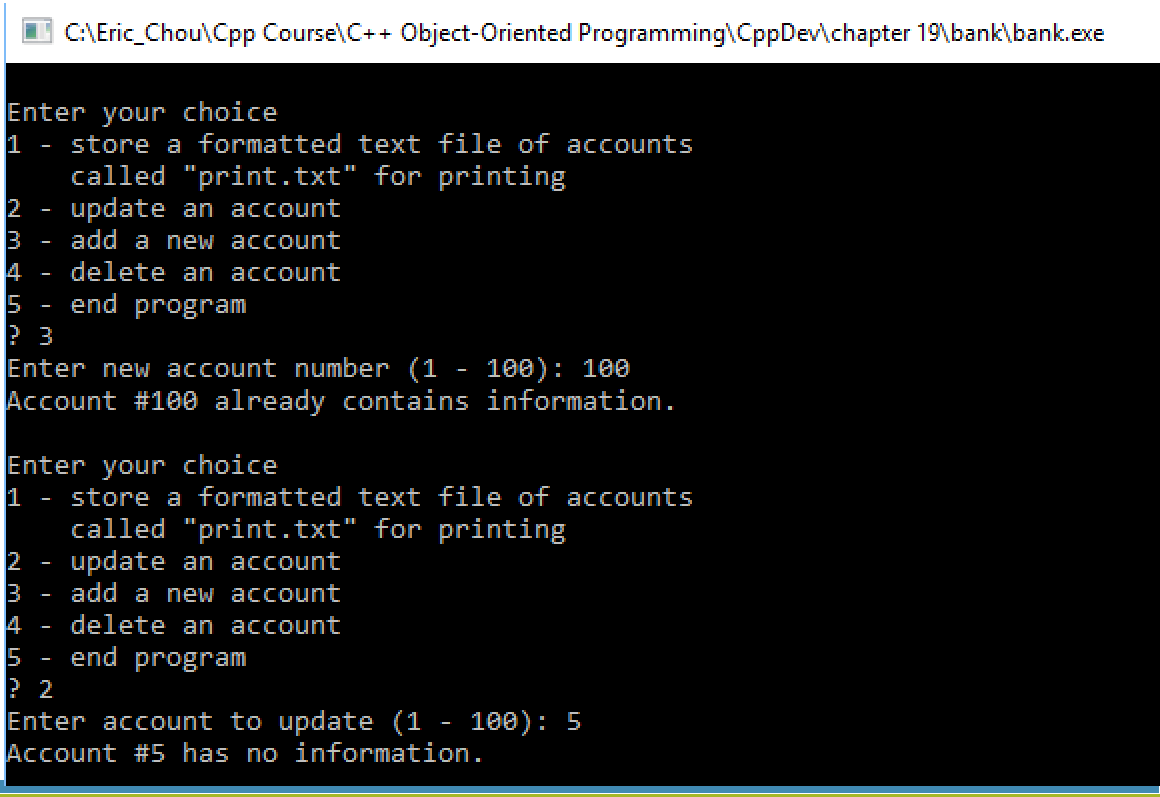
`fstream` because we read to check if the account exits. If it does, we write blank data (erase it). If it does not exist, there is no need to delete it.

`outputLine` is very flexible, and can output to any **`ostream`** object (such as a file or **`cout`**).

```cpp
int main(){
    int choice;
    // open file for reading and writing
    fstream inOutCredit( "credit.dat", ios::in | ios::out );
    // exit program if fstream cannot open file
    if ( !inOutCredit ) {
      cerr << "File could not be opened." << endl;
      exit ( 1 );
    } // end if

    // enable user to specify action
    while ( ( choice = enterChoice() ) != END ) {
      switch ( choice ) {
        // create text file from record file
        case PRINT:
          printRecord( inOutCredit );
          break;
        // update record
        case UPDATE:
          updateRecord( inOutCredit );
          break;
        // create record
        case NEW:
          newRecord( inOutCredit );
          break;
        // delete existing record
        case DELETE:
          deleteRecord( inOutCredit );
          break;
        // display error if user does not select valid choice
        default:
          cerr << "Incorrect choice" << endl;
          break;
      } // end switch
      inOutCredit.clear(); // reset end-of-file indicator
    } // end while
    return 0;
} // end main
```

Open file for reading and writing (**`fstream`** object needed).

Displays menu and returns user's choice.


C:\Eric_Chou\Cpp Course\C++ Object-Oriented Programming\CppDev\chapter 19\bank\bank.exe

```
Enter your choice
1 - store a formatted text file of accounts
    called "print.txt" for printing
2 - update an account
3 - add a new account
4 - delete an account
5 - end program
? 3
Enter new account number (1 - 100): 100
Account #100 already contains information.

Enter your choice
1 - store a formatted text file of accounts
    called "print.txt" for printing
2 - update an account
3 - add a new account
4 - delete an account
5 - end program
? 2
Enter account to update (1 - 100): 5
Account #5 has no information.
```

LECTURE 2

Read/Write of Objects

Input/Output of Objects

I/O of objects

- Chapter 8 (overloaded **>>**)
- Only object's data transmitted
  - Member functions available internally
- When objects stored in file, lose type info (class, etc.)
  - Program must know type of object when reading
- One solution
  - When writing, output object type code before real object
  - When reading, read type code
    - Call proper overloaded function (**`switch`**)

Read/Write Class Objects from/to File in C++

Given a file “aaa.txt” in which every line has values same as instance variables of a class.

Read the values into the class’s object and do necessary operations.

Write a object to a file and, Then, read it in again.

Demo Program: readobject.cpp

Go Notepadd++!!!

```cpp
//C++ program to write and read object using read and write function.
#include <iostream>
#include <fstream>

using namespace std;

//class student to read and write student details
class student
{
    private:
        char name[30];
        int age;
    public:
        void getData(void)
        { cout<<"Enter name:"; cin.getline(name,30);
          cout<<"Enter age:"; cin>>age;
        }

        void showData(void)
        {
        cout<<"Name:"<<name<<",Age:"<<age<<endl;
        }
};
```

```cpp
int main(){
    student s;
    ofstream file;
    //open file in write mode
    file.open("aaa.txt",ios::out);
    if(!file){
      cout<<"Error in creating file.."<<endl;
      return 0;
    }
    cout<<"\nFile created successfully."<<endl;
    //write into file
    s.getData();    //read from user
    file.write((char*)&s,sizeof(s));    //write into file
    file.close();   //close the file
    cout<<"\nFile saved and closed succesfully."<<endl;
    //re open file in input mode and read data
    //open file1
    ifstream file1;
    //again open file in read mode
    file1.open("aaa.txt",ios::in);
    if(!file1){
        cout<<"Error in opening file..";
        return 0;
    }
    //read data from file
    file1.read((char*)&s,sizeof(s));
    //display data on monitor
    s.showData();
    //close the file
    file1.close();
    return 0;
}
```