Object-Oriented Programming
(in C++)

File Processing

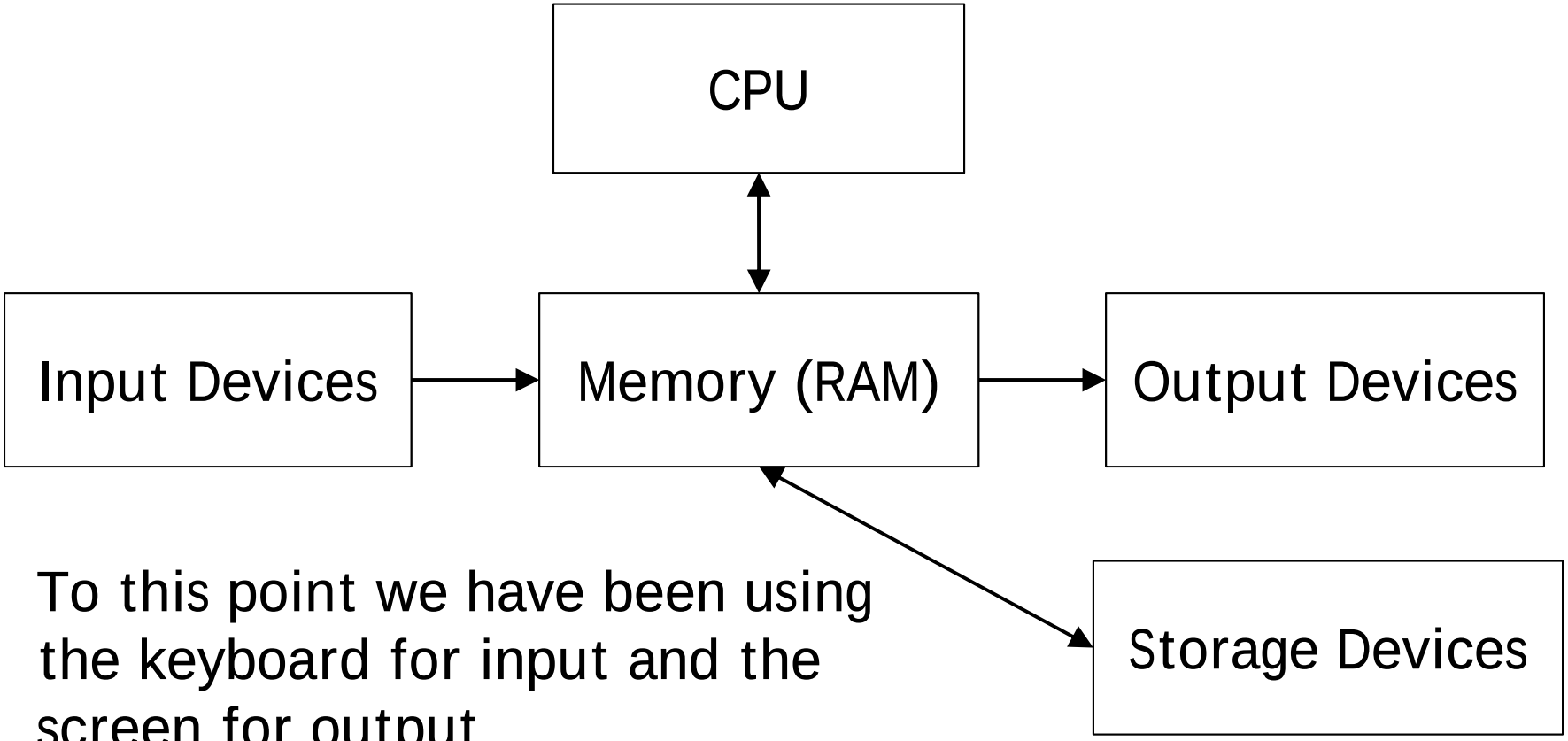
(Chapters 8 and 17)

Professor Yi-Ping You (游逸平)
Department of Computer Science
http://www.cs.nctu.edu.tw/~ypyou/

Outline

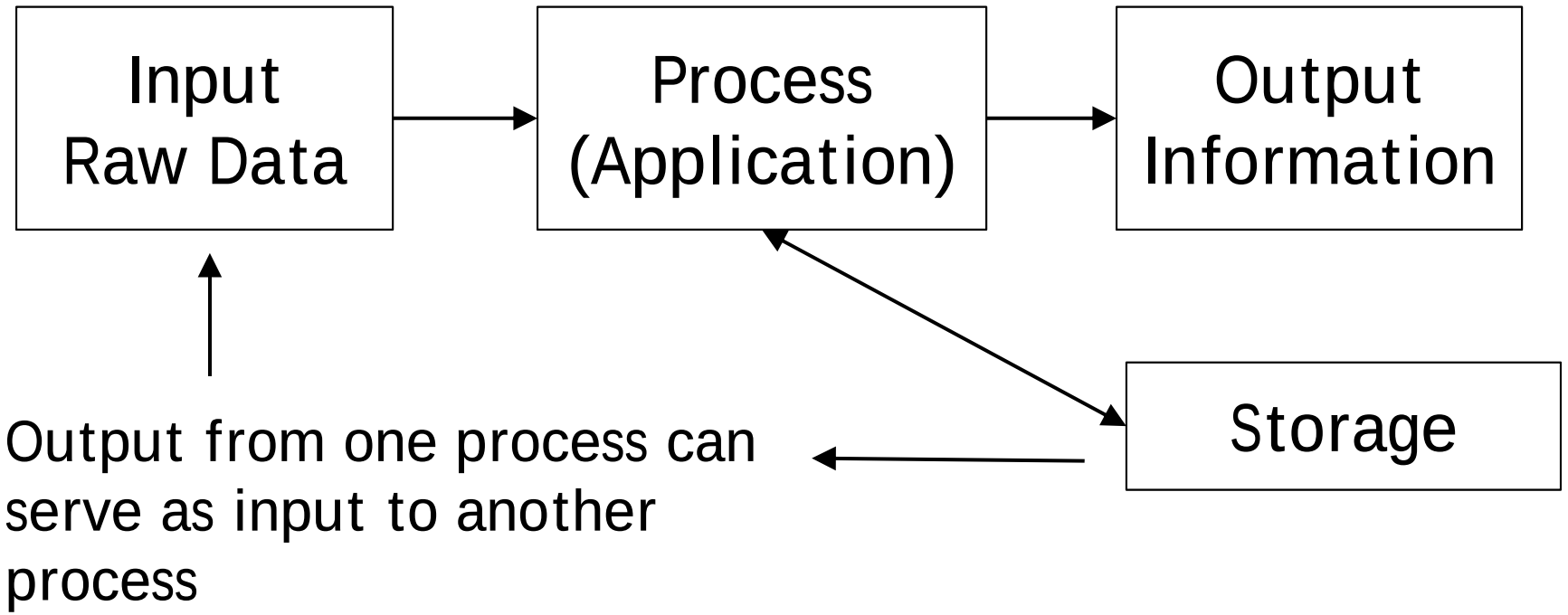
- Introduction to Storages
- Data Hierarchy
- Introduction to Files
- Sequential-Access Files
  - Writing to a file
  - Reading from a file
- File-Position Pointers
- Formatted and Unformatted I/O
- Random-Access Files
  - Case Study: Credit Processing Program

Hardware Model



- To this point we have been using the keyboard for input and the screen for output
  - Both of these are temporary

Information Process Cycle



- Storage is referred to as secondary storage, and it is permanent storage
  - Data is permanently stored in files

Files

- Files
  - Used for data persistence
    - Permanent retention of large amounts of data
  - Stored on secondary storage devices
    - Hard disks
    - CDs, DVDs
    - Flash drives
    - Tapes

Outline

- Introduction to Storages
- Data Hierarchy
- Introduction to Files
- Sequential-Access Files
  - Writing to a file
  - Reading from a file
- File-Position Pointers
- Formatted and Unformatted I/O
- Random-Access Files
  - Case Study: Credit Processing Program

The Data Hierarchy (1/3)

- Bits ("binary digits")
  - Can assume one of two values, `0` or `1`
  - Smallest data item that computers support
    - Computer circuitry performs simple bit manipulations
    - Ultimately all data items are composed of bits
- Characters
  - Include decimal digits, letters and special symbols
    - Composed of 0s and 1s
  - A character set is the set of all characters used on a particular computer
    - `char`s are stored in bytes (8 bits)
    - `wchar_t` may be 8, 16, or 32 bits wide (compiler dependent)

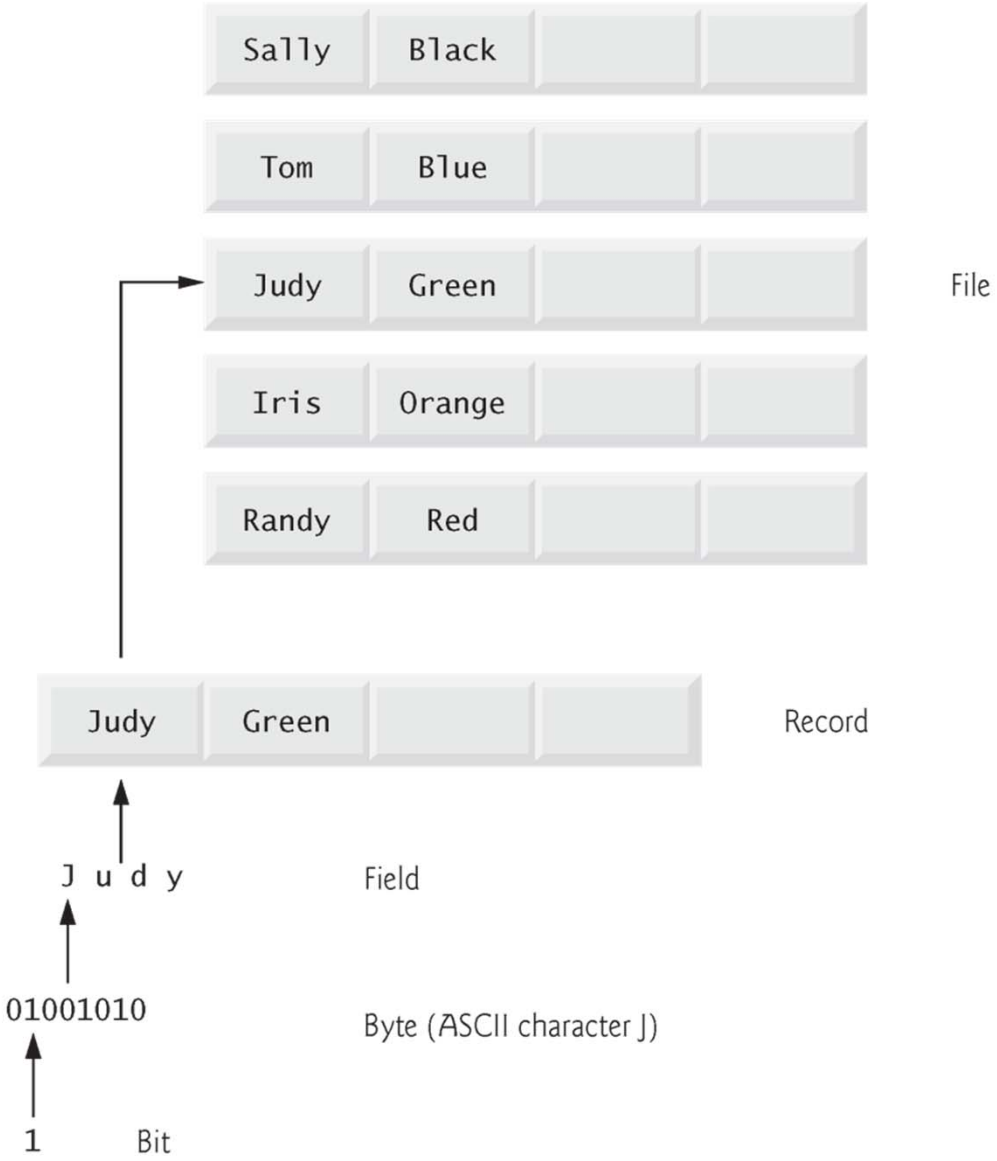
The Data Hierarchy (2/3)

- Fields
  - Composed of characters
  - Conveys some meaning
  - Example
    - Uppercase and lowercase letters can represent a name
- Records
  - Composed of several fields
  - Represented as a class in C++
  - Example
    - An employee's record might include id#, name, address, etc.
  - A record key is a field unique to each record

The Data Hierarchy (3/3)

- File
  - A good view of a file is as a list of records - an open-ended array of records
  - Example
    - A payroll file contains one record for each employee
- Database
  - Composed of a group of related files
  - Managed by a collection of programs called a database management system (DBMS)
    - E.g., MySQL, IBM DB2, Oracle, etc.

Data Hierarchy: An Example



Outline

- Introduction to Storages
- Data Hierarchy
- Introduction to Files
- Sequential-Access Files
  - Writing to a file
  - Reading from a file
- File-Position Pointers
- Formatted and Unformatted I/O
- Random-Access Files
  - Case Study: Credit Processing Program

File Organization

- Sequential-access files
  - Records are stored in order by a record-key field
  - Records do not all have to be the same length
    - Fixed length records or variable-length records
  - Great for data you read or write all at once
- Random-access files
  - Lengths of records are all identical (i.e., random access files require fixed length records)
  - Best for data you read and write in pieces (or rewrite)

Files and Streams

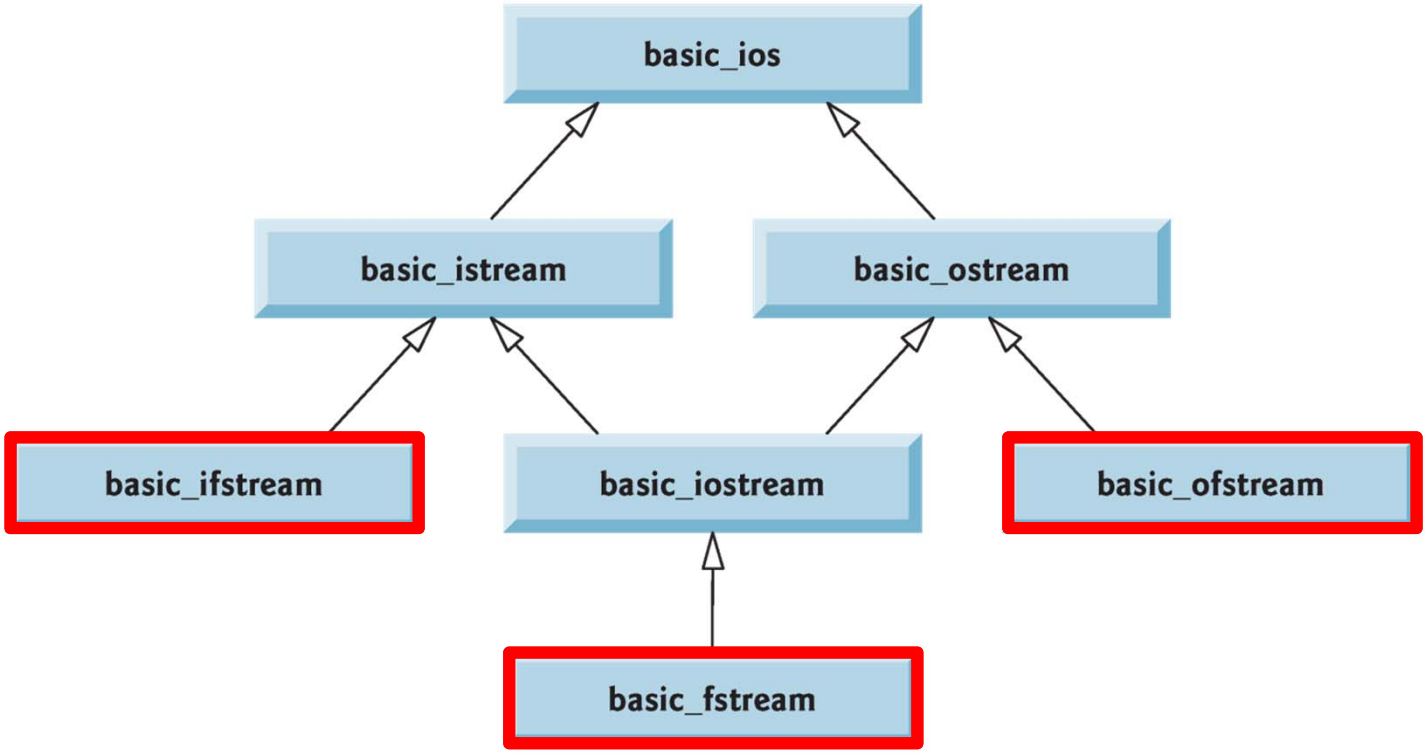
- Files
  - A file is a collection of data that the system maintains in persistent storage (**physically**)
  - Viewed by C++ as a sequence of bytes
    - A sequence of ASCII characters (e.g., a text file)
    - A sequence of pixel values that form a 24-bit color photograph
  - Ends either with an **end-of-file** marker or **at a system-recorded byte number**

- When a file is opened, an object is created
  - A stream is associated with the object

File I/O

- Communication between a program and a file is performed through stream objects
  - `<fstream>` header file
    - Contains stream class templates for file input and output
    - Files are opened by creating objects of stream template specializations

Stream-I/O Template Hierarchy



Outline

- Introduction to Storages
- Data Hierarchy
- Introduction to Files
- Sequential-Access Files
  - Writing to a file
  - Reading from a file
- File-Position Pointers
- Formatted and Unformatted I/O
- Random-Access Files
  - Case Study: Credit Processing Program

Creating a Sequential File

- File structure
  - The programmer must structure files
    - C++ does not impose structures on files
    - The concept of a “record” doe not exist in a C++ file
- Sequential File
  - Records are stored in order by a record-key field
  - Records do not all have to be the same length
    - Fixed length records or variable-length records

Opening a File for Output (1/2)

- Creating an `ofstream` object
  - Constructor takes two arguments
    - A filename : `const char *`
      - If the file doe not exist, it is first created
    - A file-open mode: `ios_base::openmode`
      - `ios::out` – the default mode
        - » Overwrites preexisting data in the file
      - `ios::app`
        - » Appends data to the end of the file

```
ofstream outClientFile("clients.txt", ios::out);
ofstream outClientFile("clients.txt");
```

Opening a File for Output (2/2)

- Calls member function **open** on existing object
  - Takes same arguments as the constructor
    - A filename and a file-open mode

```
ofstream outputFile;
outputFile.open("clients.txt", ios::out);
outputFile.open("clients.txt");
outputFile.open("output.txt", ios::app);
```

Use caution when opening an existing file for output (`ios::out`), especially when you want to preserve the file's contents, which will be discarded without warning

File Open Modes

Mode	Description
`ios::app`	Append 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 (this also is the default action for `ios::out`).
`ios::binary`	Open a file for binary (i.e., nontext) input or output.

- app comes from 'append'
  - All output will be added (appended) to the end of the file. In other words you cannot write anywhere else in the file but at the end
- `ate` comes form 'at end'
  - It sets the stream position at the end of the file when you open it but you are free to move it around (seek) and write wherever it pleases you

Writing to the File

- Use the stream insertion operator **<<** to write a set of **formatted** data to the **text file**, the same as **cout**

```
outClientFile << x << y << endl;
```

  - `operator<<` should be overloaded to handle the data
  - Because `ios::binary` flag is not the opening mode, the output file is a text file, which can be viewed by any text editor

Checking `ofstream` Object Status

- Operator member function `operator!`
  - Returns `true` if the file is not opened successfully

```
if (!outClientFile) {
  cerr << "File could not be opened" << endl;
  exit(1);
}
```

- Member function `is_open()`
  - Returns `true` if the file is opened successfully

```
if (outClientFile.is_open() == false) {
  ...
}
```

Closing the File

- Member function `close`
  - Releases the file resource
  - Implicitly performed by `ofstream`'s destructor

```
outClientFile.close();
```

Creating a Sequential File: An Example

```
3  #include <iostream>
4  #include <string>
5  #include <fstream> // file stream
6  #include <cstdlib>
7  using namespace std;
8
9  int main()
10 {
11    // ofstream constructor opens file
12    ofstream outClientFile( "clients.txt", ios::out );
13
14    // exit program if unable to create file
15    if ( !outClientFile ) // overloaded ! operator
16    {
17       cerr << "File could not be opened" << endl;
18       exit( 1 );
19    } // end if
20
21    cout << "Enter the account, name, and balance." << endl
22       << "Enter end-of-file to end input.\n? ";
23
24    int account; // customer's account number
25    string name; //customer's name
26    double balance; // amount of money customer owes company
27
28    // read account, name and balance from cin, then place in file
29    while ( cin >> account >> name >> balance )
30    {
31       outClientFile << account << ' ' << name << ' ' << balance << endl;
32       cout << "? ";
33    } // end while
34 } // end main
```

```
Enter the account, name, and balance.
Enter end-of-file to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

Computer system	Keyboard combination
UNIX/Linux/Mac OS X	<*Ctrl-d*> (on a line by itself)
Microsoft Windows	<*Ctrl-z*> (sometimes followed by pressing *Enter*)

```
100 Jones 24.98
200 Doe 345.67
300 White 0
400 Stone -42.16
500 Rich 224.62
```

clients.txt

Outline

- Introduction to Storages
- Data Hierarchy
- Introduction to Files
- Sequential-Access Files
  - Writing to a file
  - Reading from a file
- File-Position Pointers
- Formatted and Unformatted I/O
- Random-Access Files
  - Case Study: Credit Processing Program

Reading Data from a Sequential File (1/2)

- Creating an `ifstream` object
  - Opens a file for input
  - Constructor takes two arguments
    - A filename
    - A file-open mode
      - `ios::in` – the default mode
        - » Can only read from the file, this prevents unintentional modification of the file's contents
  - Can also use member function **open** on an existing object
    - Takes same arguments as the constructor

Reading Data from a Sequential File (2/2)

- Use the stream extraction operator **>>** to read set of formatted data from the file, the same as `cin`

```
inputFile >> x >> y;
```

  - **`operator>>`** should be overloaded to handle the data
- Operator member function `operator void *`
  - Converts the stream to a pointer when it is used as a condition
    - The null pointer if the input fails

Reading from a Sequential File: An Example

```
3  #include <iostream>
4  #include <fstream> // file stream
5  #include <iomanip>
6  #include <string>
7  #include <cstdlib>
8  using namespace std;
9
10 void outputLine( int, const string, double ); // prototype
11
12 int main()
13 {
14    // ifstream constructor opens the file
15    ifstream inClientFile( "clients.txt", ios::in );
16
17    // exit program if ifstream could not open file
18    if ( !inClientFile )
19    {
20       cerr << "File could not be opened" << endl;
21       exit( 1 );
22    } // end if
23
24    int account; // customer's account number
25    string name; // customer's name
26    double balance; //amount of money customer owes company
27
28    cout << left << setw( 10 ) << "Account" << setw( 13 )
29       << "Name" << "Balance" << endl << fixed << showpoint;
30
31    // display each record in file
32    while ( inClientFile >> account >> name >> balance )
33       outputLine( account, name, balance );
34 } // end main
35
36 // display single record from file
37 void outputLine( int account, const string name, double balance )
38 {
39    cout << left << setw( 10 ) << account << setw( 13 ) << name
40       << setw( 7 ) << setprecision( 2 ) << right << balance << endl;
41 } // end function outputLine
```

100 Jones 24.98
200 Doe 345.67
300 White 0
400 Stone -42.16
500 Rich 224.62

clients.txt

```
Account    Name           Balance
100        Jones            24.98
200        Doe             345.67
300        White             0.00
400        Stone           -42.16
500        Rich            224.62
```

Outline

- Introduction to Storages
- Data Hierarchy
- Introduction to Files
- Sequential-Access Files
  - Writing to a file
  - Reading from a file
- File-Position Pointers
- Formatted and Unformatted I/O
- Random-Access Files
  - Case Study: Credit Processing Program

File-Position Pointer

- The byte number of the **next** byte to be read or written

Repositioning the File-Position Pointer

- Member functions `seekg` ("seek get") and `seekp` ("seek put") (of `ifream` and `oftream`, respectively)
  - Repositions the file-position pointer to the specified location
    - Takes desired offset argument as a `long`
  - A second argument can specify the seek direction
    - `ios::beg` – the default
      - Positioning relative to the beginning
    - `ios::cur`
      - Positioning relative to the current position
    - `ios::end`
      - Positioning relative to the end

Repositioning the File-Position Pointer: Examples (1/3)

```
fileObject.seekp( 0 );
```

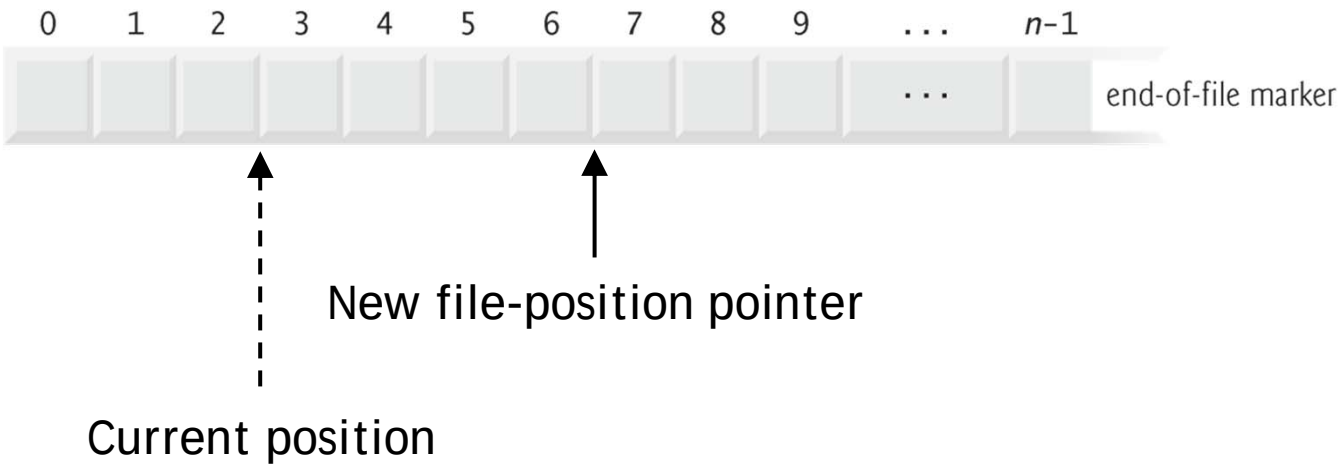
File-position pointer

```
fileObject.seekp( 3 );
```

File-position pointer

Repositioning the File-Position Pointer: Examples (1/3)

```
fileObject.seekp( 4, ios::cur );
```



Repositioning the File-Position Pointer: Examples (1/3)

```
fileObject.seekp( 1, ios::end );
```


```
fileObject.seekp( 0, ios::end );
```

File-Position Pointer

- Member functions `tellg` and `tellp` (of `istream` and `ostream`, respectively)
  - Returns current position of the file-position pointer as type `long`
  - Example

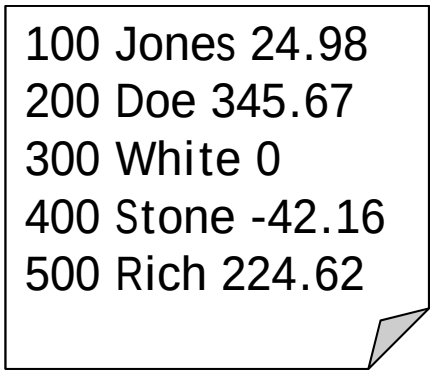
```
long location = fileObject.tellg();
```

Updating Sequential Files

- Updating a record in a sequential file
  - The new record could be longer than the old record
    - If it is, it could overwrite the next sequential record
    - You would have to rewrite every record into another file
      - Copy over all records before this one
      - Write new version of this record
      - Copy over all records after this one
    - This might be acceptable if you are updating many records

Updating Sequential Files: An Example

- If the name “White” needs to be changed to “Worthington”, the old name cannot be overwritten without corrupting the file



clients.txt

Outline

- Introduction to Storages
- Data Hierarchy
- Introduction to Files
- Sequential-Access Files
  - Writing to a file
  - Reading from a file
- File-Position Pointers
- Formatted and Unformatted I/O
- Random-Access Files
  - Case Study: Credit Processing Program

Recall: Streams

- "High-level", formatted I/O
  - Bytes are grouped into meaningful units
    - Integers, floating-point numbers, characters, etc.
  - Satisfactory for most I/O other than high-volume file processing
- "Low-level", unformatted I/O
  - Individual bytes are the items of interest
  - High-speed, high-volume
  - Not particularly convenient for programmers

Binary vs. Text Files

- Text File
  - A very special kind of variable-length **sequential file** that uses special end of line markers (carriage returns/line feeds) at the end of each record (line) in the file
  - Human readable, could be opened by any text editor
- Binary File
  - Contains binary information
  - More compact and more efficient to access
  - Only applications that are aware of the binary file's record format can easily access the file
  - Example:
    - Object and executable files
    - Image/video files

Text File: An Example

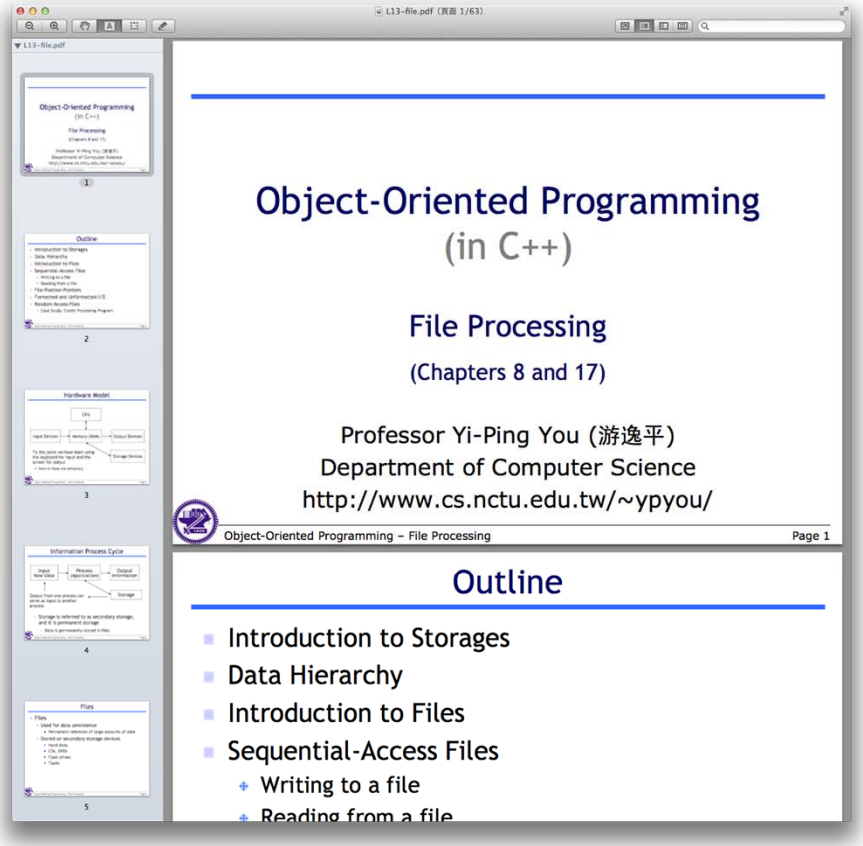
poem

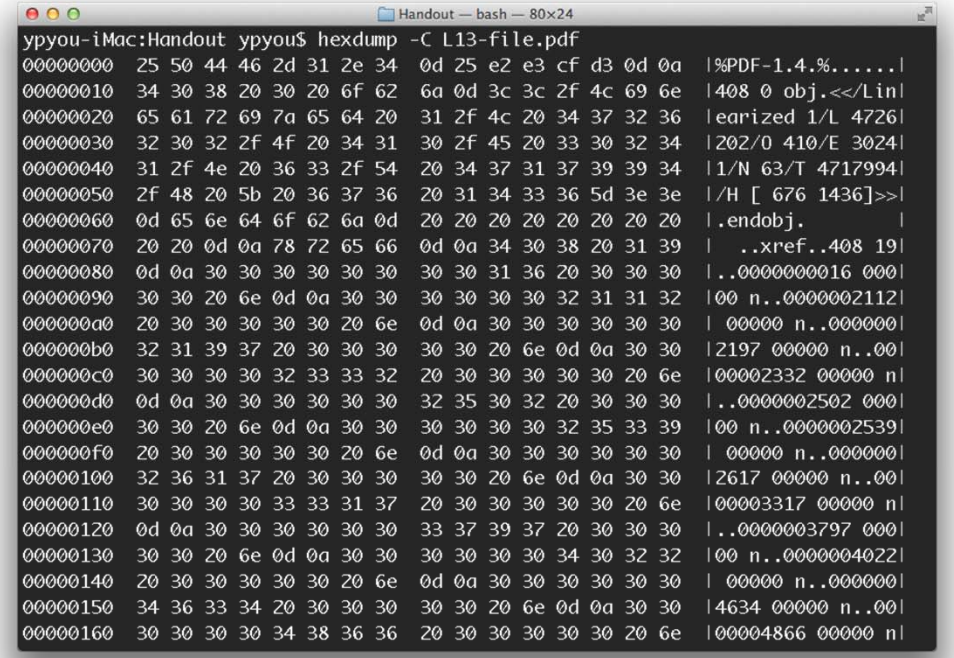
```
A Fairy Song by William Shakespeare

Over hill, over dale,
Thorough bush, thorough brier,
Over park, over pale,
Thorough flood, thorough fire!
I do wander everywhere,
Swifter than the moon's sphere;
And I serve the Fairy Queen,
To dew her orbs upon the green;
The cowslips tall her pensioners be;
In their gold coats spots you see;
Those be rubies, fairy favours;
In those freckles live their savours;
I must go seek some dewdrops here,
And hang a pearl in every cowslip's ear.
```

桌面 — bash — 80×24

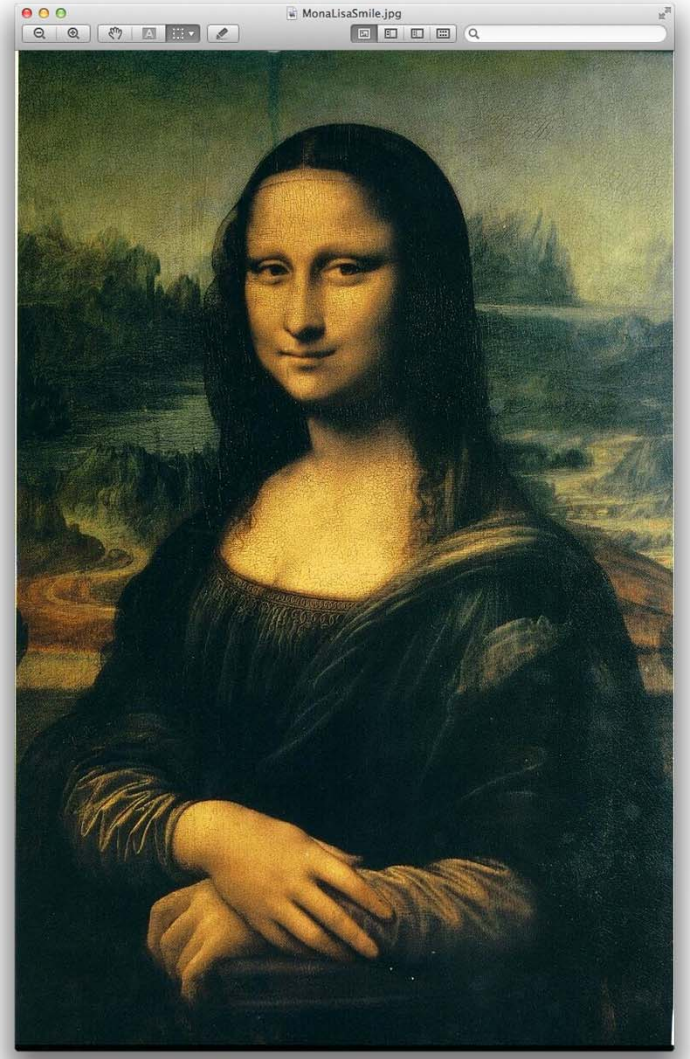
```
ypyou-iMac:Desktop ypyou$ hexdump -C poem.txt
00000000  41 20 46 61 69 72 79 20  53 6f 6e 67 20 62 79 20  |A Fairy Song by |
00000010  57 69 6c 6c 69 61 6d 20  53 68 61 6b 65 73 70 65  |William Shakespe|
00000020  61 72 65 0a 0a 4f 76 65  72 20 68 69 6c 6c 2c 20  |are..Over hill, |
00000030  6f 76 65 72 20 64 61 6c  65 2c 0a 54 68 6f 72 6f  |over dale,.Thoro|
00000040  75 67 68 20 62 75 73 68  2c 20 74 68 6f 72 6f 75  |ugh bush, thorou|
00000050  67 68 20 62 72 69 65 72  2c 0a 4f 76 65 72 20 70  |gh brier,.Over p|
00000060  61 72 6b 2c 20 6f 76 65  72 20 70 61 6c 65 2c 0a  |ark, over pale,.|
00000070  54 68 6f 72 6f 75 67 68  20 66 6c 6f 6f 64 2c 20  |Thorough flood, |
00000080  74 68 6f 72 6f 75 67 68  20 66 69 72 65 21 0a 49  |thorough fire!.I|
00000090  20 64 6f 20 77 61 6e 64  65 72 20 65 76 65 72 79  | do wander every|
000000a0  77 68 65 72 65 2c 0a 53  77 69 66 74 65 72 20 74  |where,.Swifter t|
000000b0  68 61 6e 20 74 68 65 20  6d 6f 6f 6e 27 73 20 73  |han the moon's s|
000000c0  70 68 65 72 65 3b 0a 41  6e 64 20 49 20 73 65 72  |phere;.And I ser|
000000d0  76 65 20 74 68 65 20 46  61 69 72 79 20 51 75 65  |ve the Fairy Que|
000000e0  65 6e 2c 0a 54 6f 20 64  65 77 20 68 65 72 20 6f  |en,.To dew her o|
000000f0  72 62 73 20 75 70 6f 6e  20 74 68 65 20 67 72 65  |rbs upon the gre|
00000100  65 6e 3b 0a 54 68 65 20  63 6f 77 73 6c 69 70 73  |en;.The cowslips|
00000110  20 74 61 6c 6c 20 68 65  72 20 70 65 6e 73 69 6f  | tall her pensio|
00000120  6e 65 72 73 20 62 65 3b  0a 49 6e 20 74 68 65 69  |ners be;.In thei|
00000130  72 20 67 6f 6c 64 20 63  6f 61 74 73 20 73 70 6f  |r gold coats spo|
00000140  74 73 20 79 6f 75 20 73  65 65 3b 0a 54 68 6f 73  |ts you see;.Thos|
00000150  65 20 62 65 20 72 75 62  69 65 73 2c 20 66 61 69  |e be rubies, fai|
00000160  72 79 20 66 61 76 6f 75  72 73 3b 0a 49 6e 20 74  |ry favours;.In t|
```

Binary File: An Example (PDF File)





Binary File: Another Example (Image File)




```
桌面 — bash — 80×24
ypyou-iMac:Desktop ypyou$ hexdump -C MonaLisaSmile.jpg
00000000  ff d8 ff e0 00 10 4a 46  49 46 00 01 01 01 00 60  |......JFIF.....`|
00000010  00 60 00 00 ff db 00 43  00 08 06 06 07 06 05 08  |.`.....C........|
00000020  07 07 07 09 09 08 0a 0c  14 0d 0c 0b 0b 0c 19 12  |................|
00000030  13 0f 14 1d 1a 1f 1e 1d  1a 1c 1c 20 24 2e 27 20  |........... $.' |
00000040  22 2c 23 1c 1c 28 37 29  2c 30 31 34 34 34 1f 27  |",#..(7),01444.'|
00000050  39 3d 38 32 3c 2e 33 34  32 ff db 00 43 01 09 09  |9=82<.342...C...|
00000060  09 0c 0b 0c 18 0d 0d 18  32 21 1c 21 32 32 32 32  |........2!.!2222|
00000070  32 32 32 32 32 32 32 32  32 32 32 32 32 32 32 32  |2222222222222222|
*
00000090  32 32 32 32 32 32 32 32  32 32 32 32 32 32 ff c2  |22222222222222..|
000000a0  00 11 08 04 9a 03 08 03  01 22 00 02 11 01 03 11  |.........".......|
000000b0  01 ff c4 00 1a 00 00 03  01 01 01 01 00 00 00 00  |................|
000000c0  00 00 00 00 00 00 00 01  02 03 04 05 06 ff c4 00  |................|
000000d0  19 01 01 01 01 01 01 01  00 00 00 00 00 00 00 00  |................|
000000e0  00 00 00 01 02 03 04 05  ff da 00 0c 03 01 00 02  |................|
000000f0  10 03 10 00 00 01 eb f8  7f b8 f8 8c 61 4d ab 01  |............aM..|
00000100  88 0d 0c 8b 20 b5 49 68  24 9a 21 31 0c 90 96 3b  |.... .Ih$.!1...;|
00000110  18 38 49 35 54 d0 d0 c9  2d 59 52 03 68 2f a3 9b  |.8I5T...-YR.h/..|
00000120  33 bb 93 3d 02 a2 a5 4d  82 8d 54 a6 91 50 dc a9  |3..=...M..T..P..|
00000130  74 24 28 40 da 63 6e a2  0b 52 b1 a0 40 0c 64 d0  |t$(@.cn..R..@.d.|
00000140  12 36 03 a5 4d b8 2a 58  e5 82 29 93 3a 2b 23 48  |.6..M.*X..).:+#H|
00000150  74 e8 d1 71 7a ba c9 6e  cc 27 b1 d7 2c f6 2b 38  |t..qz..n.'..,.+8|
00000160  ce db 31 3b ca af 8b fb  5f 8c e7 64 a9 b9 06 02  |..1;...._..d....|
```

Unformatted Output: Member Function `write`

- Member function `write`
  - Writes a number of bytes from a location in memory to the stream
    - If the stream is associated with a file, the writing is at the "put" file-position pointer
  - First argument
    - A `const char *` , a pointer to a byte
      - Unfortunately, most pointers that we pass to the function are not of type `const char *`, so we must convert the pointer to type `const char*`
  - Second argument
    - A `size_t` specifying the number of bytes to write

Formatted vs Unformatted Output

```
#include <iostream>
#include <fstream>
#include <cstdlib> // exit function
using namespace std;

int main() {
  ofstream outputFile("clients.txt", ios::binary);
  if ( !outputFile ) {
    cerr << "File could not be opened" << endl;
    exit( 1 );
  }

  int number = 16384; // 2^14
  outputFile << number; // formatted output
  // writes 00110001001101100110011001111000 00110100 to the file
                1       6       3       8       4

  outputFile.write(
    reinterpret_cast<const char *>(&number),
    sizeof(number)
  ); // unformatted output (writing the binary version of the integer)
  // writes 00000000000000001000000000000000 to the file
}
```

```
tmp — bash — 80×7
Yi-Pingteki-MacBook-Air:tmp ypyou$ g++ binary.cpp
Yi-Pingteki-MacBook-Air:tmp ypyou$ ./a.out
Yi-Pingteki-MacBook-Air:tmp ypyou$ hexdump -C clients.txt
00000000  31 36 33 38 34 00 40 00  00                       |16384.@..|
00000009
Yi-Pingteki-MacBook-Air:tmp ypyou$
```

An ASCII character

ASCII Table

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

Operator `reinterpret_cast`

- Casts a pointer of one type to an unrelated type

```
reinterpret_cast<const char *>(&number)
```

- Is performed at compile time
  - Does not change the value of the object pointed to
- The operation result is a simple binary copy of the value from one pointer to the other

Outline

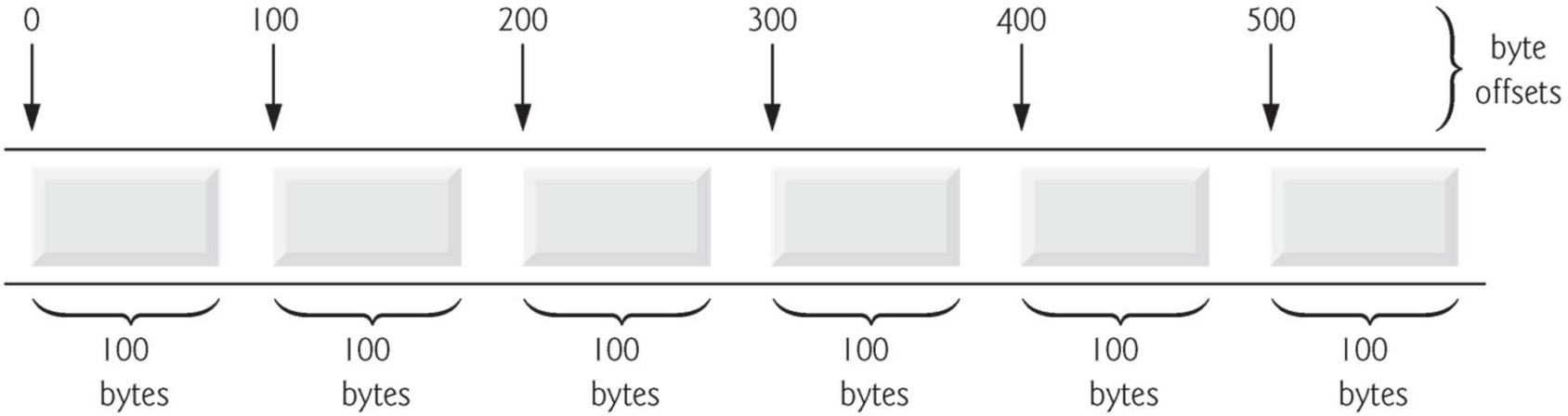
- Introduction to Storages
- Data Hierarchy
- Introduction to Files
- Sequential-Access Files
  - Writing to a file
  - Reading from a file
- File-Position Pointers
- Formatted and Unformatted I/O
- Random-Access Files
  - Case Study: Credit Processing Program

Random-Access Files

- Necessary for instant-access applications
  - Such as transaction-processing systems
  - E.g., airline reservation, banking system, and ATM
- A record can be inserted, deleted or modified without affecting other records
- Various techniques can be used
  - The easiest method requires that all records be of the **same length**, arranged in the order of the record keys
    - Program can calculate the exact location of any record
      - Base on the record size and record key

C++ View of a Random-Access File

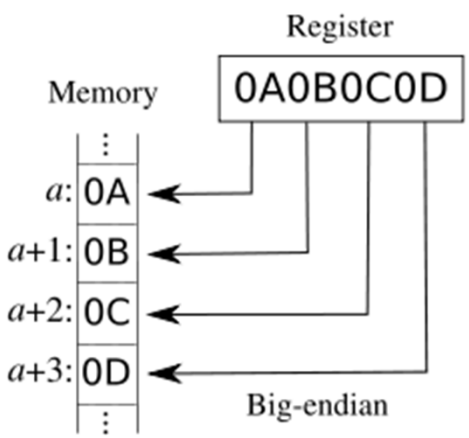
- Each block is usually an object of a class

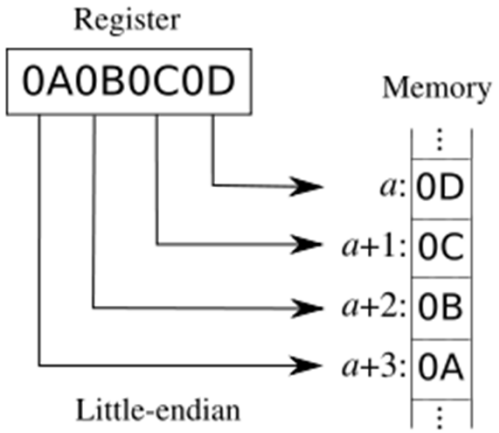


Portability Tip

- A program that reads unformatted data (written by `write`) must be compiled and executed on a system compatible with the program that wrote the data, because different systems may represent internal data differently
  - increasing numeric significance with increasing memory addresses, known as *little-endian*
  - its opposite, most-significant byte first, called *big-endian*

Big-endian vs Little-endian





Examples of storing the value **0x0A0B0C0D** (168496141) in memory

Outline

- Introduction to Storages
- Data Hierarchy
- Introduction to Files
- Sequential-Access Files
  - Writing to a file
  - Reading from a file
- File-Position Pointers
- Formatted and Unformatted I/O
- Random-Access Files
  - Case Study: Credit Processing Program

Case Study: Credit Processing Program

- Problem:
  - Create a credit-processing program capable of storing at most 100 fixed-length records for a company that can have up to 100 customers
  - Each record should consist of
    - an account number that acts as the record key
    - a last name,
    - a first name, and
    - a balance
  - The program should be able to update an account, insert a new account, delete an account, and insert all the account records into a formatted text file for printing

ClientData.h

```
3  #ifndef CLIENTDATA_H
4  #define CLIENTDATA_H
5
6  #include <string>
7  using namespace std;
8
9  class ClientData
10 {
11 public:
12    // default ClientData constructor
13    ClientData( int = 0, string = "", string = "", double = 0.0 );
14
15    // accessor functions for accountNumber
16    void setAccountNumber( int );
17    int getAccountNumber() const;
18
19    // accessor functions for lastName
20    void setLastName( string );
21    string getLastName() const;
22
23    // accessor functions for firstName
24    void setFirstName( string );
25    string getFirstName() const;
26
27    // accessor functions for balance
28    void setBalance( double );
29    double getBalance() const;
30 private:
31    int accountNumber;
32    char lastName[ 15 ];
33    char firstName[ 10 ];
34    double balance;
35 }; // end class ClientData
36
37 #endif
```

ClientData.cpp (1/2)

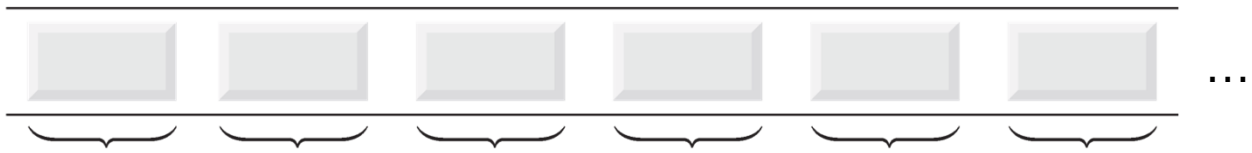
```
3  #include <string>
4  #include "ClientData.h"
5  using namespace std;
6
7  // default ClientData constructor
8  ClientData::ClientData( int accountNumberValue,
9     string lastNameValue, string firstNameValue, double balanceValue )
10 {
11    setAccountNumber( accountNumberValue );
12    setLastName( lastNameValue );
13    setFirstName( firstNameValue );
14    setBalance( balanceValue );
15 } // end ClientData constructor
16
17 // get account-number value
18 int ClientData::getAccountNumber() const
19 {
20    return accountNumber;
21 } // end function getAccountNumber
22
23 // set account-number value
24 void ClientData::setAccountNumber( int accountNumberValue )
25 {
26    accountNumber = accountNumberValue; // should validate
27 } // end function setAccountNumber
28
29 // get last-name value
30 string ClientData::getLastName() const
31 {
32    return lastName;
33 } // end function getLastName
34
35 // set last-name value
36 void ClientData::setLastName( string lastNameString )
37 {
38    // copy at most 15 characters from string to lastName
39    int length = lastNameString.size();
40    length = ( length < 15 ? length : 14 );
41    lastNameString.copy( lastName, length );
42    lastName[ length ] = '\0'; // append null character to lastName
43 } // end function setLastName
```

ClientData.cpp (2/2)

```
45  // get first-name value
46  string ClientData::getFirstName() const
47  {
48     return firstName;
49  } // end function getFirstName
50
51  // set first-name value
52  void ClientData::setFirstName( string firstNameString )
53  {
54     // copy at most 10 characters from string to firstName
55     int length = firstNameString.size();
56     length = ( length < 10 ? length : 9 );
57     firstNameString.copy( firstName, length );
58     firstName[ length ] = '\0'; // append null character to firstName
59  } // end function setFirstName
60
61  // get balance value
62  double ClientData::getBalance() const
63  {
64     return balance;
65  } // end function getBalance
66
67  // set balance value
68  void ClientData::setBalance( double balanceValue )
69  {
70     balance = balanceValue;
71  } // end function setBalance
```

Credit Processing Program

```
 3  #include <iostream>
 4  #include <fstream>
 5  #include <cstdlib>
 6  #include "ClientData.h" // ClientData class definition
 7  using namespace std;
 8
 9  int main()
10  {
11     ofstream outCredit( "credit.dat", ios::out | ios::binary );
12
13     // exit program if ofstream could not open file
14     if ( !outCredit )
15     {
16        cerr << "File could not be opened." << endl;
17        exit( 1 );
18     } // end if
19
20     ClientData blankClient; // constructor zeros out each data member
21
22     // output 100 blank records to file
23     for ( int i = 0; i < 100; i++ )
24        outCredit.write( reinterpret_cast< const char * >( &blankClient ),
25           sizeof( ClientData ) );
26  } // end main
```



Credit Processing Program: Writing Data Randomly (1/3)

```
 3  #include <iostream>
 4  #include <fstream>
 5  #include <cstdlib>
 6  #include "ClientData.h" // ClientData class definition
 7  using namespace std;
 8
 9  int main()
10  {
11     int accountNumber; // customer's account number
12     string lastName; // customer's last name
13     string firstName; // customer's first name
14     double balance; // amount of money customer owes company
15
16     fstream outCredit( "credit.dat", ios::in | ios::out | ios::binary );
17
18     // exit program if fstream cannot open file
19     if ( !outCredit )
20     {
21        cerr << "File could not be opened." << endl;
22        exit( 1 );
23     } // end if
24
25     cout << "Enter account number (1 to 100, 0 to end input)\n? ";
26
27     // require user to specify account number
28     ClientData client;
29     cin >> accountNumber;
```

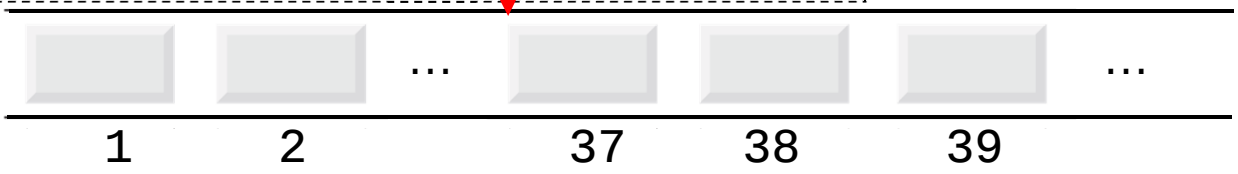
client

Credit Processing Program: Writing Data Randomly (2/3)

```
31      // user enters information, which is copied into file
32      while ( accountNumber > 0 && accountNumber <= 100 )
33      {
34         // user enters last name, first name and balance
35         cout << "Enter lastname, firstname, balance\n? ";
36         cin >> lastName;
37         cin >> firstName;
38         cin >> balance;
39
40         // set record accountNumber, lastName, firstName and balance values
41         client.setAccountNumber( accountNumber );
42         client.setLastName( lastName );
43         client.setFirstName( firstName );
44         client.setBalance( balance );
45
46         // seek position in file of user-specified record
47         outCredit.seekp( ( client.getAccountNumber() - 1 ) *
48            sizeof( ClientData ) );
49
50         // write user-specified information in file
51         outCredit.write( reinterpret_cast< const char * >( &client ),
52            sizeof( ClientData ) );
53
54         // enable user to enter another account
55         cout << "Enter account number\n? ";
56         cin >> accountNumber;
57      } // end while
58   } // end main
```

```
Enter account number (1 to 100, 0 to end input)
? 37
Enter lastname, firstname, balance
? Barker Doug 0.00
```

client



Credit Processing Program: Writing Data Randomly (3/3)

```
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
```

Credit Processing Program: Reading Data Randomly (1/2)

```
 3  #include <iostream>
 4  #include <iomanip>
 5  #include <fstream>
 6  #include <cstdlib>
 7  #include "ClientData.h" // ClientData class definition
 8  using namespace std;
 9
10  void outputLine( ostream&, const ClientData & ); // prototype
11
12  int main()
13  {
14     ifstream inCredit( "credit.dat", ios::in | ios::binary );
15
16     // exit program if ifstream cannot open file
17     if ( !inCredit )
18     {
19        cerr << "File could not be opened." << endl;
20        exit( 1 );
21     } // end if
22
23     cout << left << setw( 10 ) << "Account" << setw( 16 )
24        << "Last Name" << setw( 11 ) << "First Name" << left
25        << setw( 10 ) << right << "Balance" << endl;
```

```
Account   Last Name       First Name    Balance
```

Credit Processing Program: Reading Data Randomly (2/2)

```
27       ClientData client; // create record
28
29       // read first record from file
30       inCredit.read( reinterpret_cast< char * >( &client ),
31          sizeof( ClientData ) );
32
33       // read all records from file
34       while ( inCredit && !inCredit.eof() )
35       {
36          // display record
37          if ( client.getAccountNumber() != 0 )
38             outputLine( cout, client );
39
40          // read next from file
41          inCredit.read( reinterpret_cast< char * >( &client ),
42             sizeof( ClientData ) );
43       } // end while
44    } // end main
45
46    // display single record
47    void outputLine( ostream &output, const ClientData &record )
48    {
49       output << left << setw( 10 ) << record.getAccountNumber()
50          << setw( 16 ) << record.getLastName()
51          << setw( 11 ) << record.getFirstName()
52          << setw( 10 ) << setprecision( 2 ) << right << fixed
53          << showpoint << record.getBalance() << endl;
54    } // end function outputLine
```

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