

Object Oriented Programming

File Handling LECTURE-32, 33



1

FILE HANDLING



2

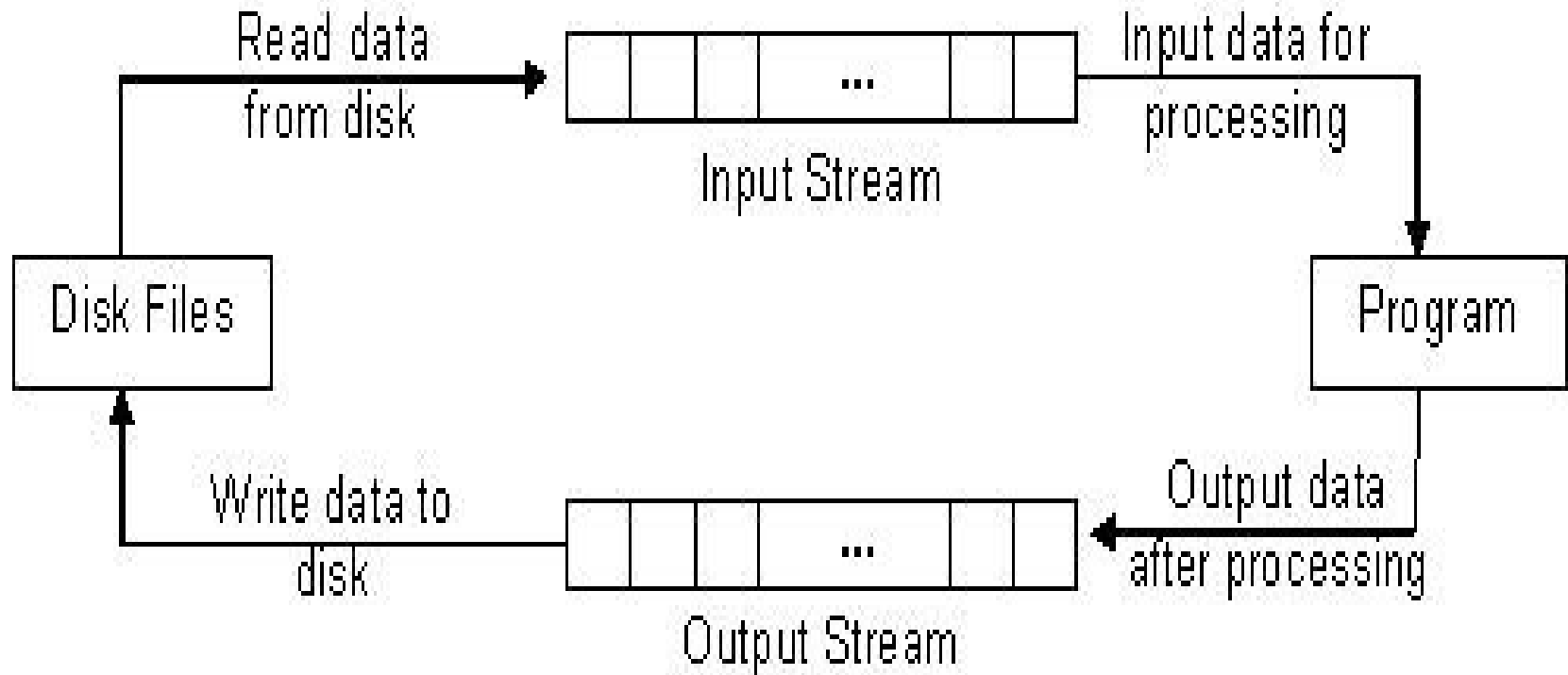
Introduction

- ☞ All programs we looked earlier:
 - ☐ input data from the keyboard.
 - ☐ output data to the screen.
- ☞ Output would be lost as soon as we exit from the program.
- ☞ How do we store data permanently?
 - ☐ We can use secondary storage device.
 - ☐ Data is packaged up on the storage device as data structures called **files**.

Streams Usage

- We've used streams already
 - cin
 - Input from stream object connected to keyboard
 - cout
 - Output to stream object connected to screen
- Can define other streams
 - To or from files
 - Used similarly as cin, cout

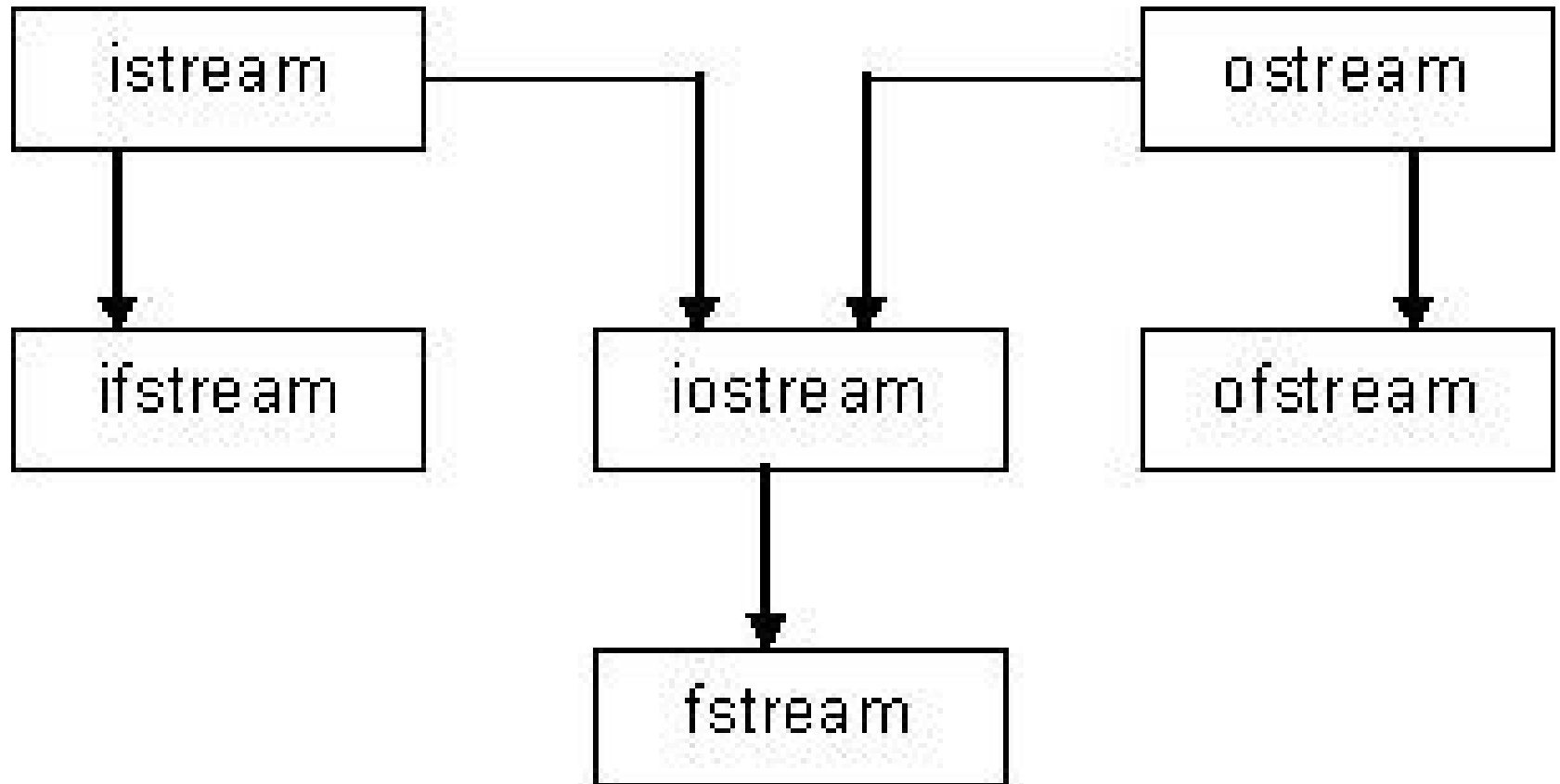
File input and output streams



Streams

- ☛ **File Input Stream** – reads data from disk file to the program.
- ☛ **File output Stream** – writes data to the disk file from the program.
- ☛ The I/O system of C++ contains:
 - ☐ **ifstream** – provides input operations on files
 - ☐ **ofstream** – provides output operations on files
 - ☐ **fstream** – supports for simultaneous input and output operations on files

Stream Classes



The Data Hierarchy

- From smallest to largest
 - Bit (binary digit)
 - 1 or 0
 - Everything in computer ultimately represented as bits
 - 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**)

The Data Hierarchy (contd.)

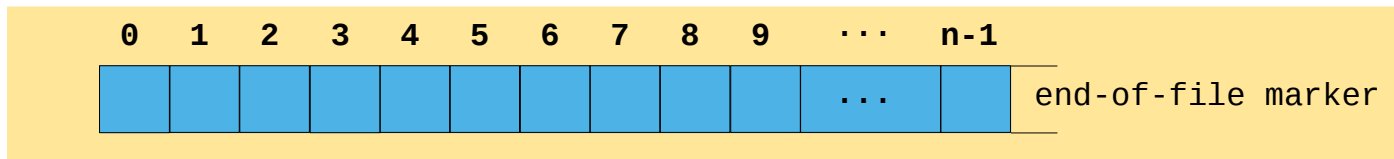
- From smallest to largest (continued)
 - Field: group of characters with some meaning
 - e.g., Your name
 - Record: group of related fields
 - **struct** or **class** in C++
 - In payroll system, could be name, S#, address, wage
 - Each field associated with same employee
 - Record key: field used to uniquely identify record
 - File: group of related records
 - Payroll for entire company
 - Database: group of related files
 - Payroll, accounts-receivable, inventory...

General File I/O Steps

- Declare a file name variable
- Associate the file name variable with the disk file name
- Open the file
- Use the file
- Close the file

Files and Streams

- C++ views file as sequence of bytes
 - Ends with *end-of-file* marker



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

flag value	opening mode
app	(append) Set the stream's position indicator to the end of the stream before each output operation.
ate	(at end) Set the stream's position indicator to the end of the stream on opening.
binary	(binary) Consider stream as binary rather than text.
in	(input) Allow input operations on the stream.
out	(output) Allow output operations on the stream.
trunc	(truncate) Any current content is discarded, assuming a length of zero on opening.

Creating a Sequential-Access File

- Operations

- Overloaded **operator!**

- **!outClientFile**

- Returns nonzero (true) if some error

- Opening non-existent file for reading, no permissions, disk full etc.

- Writing to file (just like **cout**)

- **outClientFile << myVariable**

- Closing file

- **outClientFile.close()**

- Automatically closed when destructor called

```
// Create a sequential file.
```

```
#include <iostream>
```

```
#include <fstream>
```

```
using namespace std;
```

```
int main() {
```

```
    // ofstream constructor opens file
```

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

```
    // exit program if unable to create file
```

```
    if ( !outClientFile ) { // overloaded ! operator
```

```
        cout << "File could not be opened" << endl;
```

```
        exit( 1 );
```

```
    } // end if
```

```
cout << "Enter the account, name, and balance." << endl  
    << "Enter \'N\' to end input.\n? ";
```

```
int account;
```

```
char name[ 30 ], ch='y';
```

```
double balance;
```

```
// read account, name and balance from cin, then place in file
```

```
while (ch == 'y') {
```

```
    cin >> account >> name >> balance;
```

```
    outClientFile << account << ' ' << name << ' ' << balance  
        << endl;
```

```
    cout << "? ";
```

```
    cin>>ch;
```

```
} // end while
```

```
return 0; // ofstream destructor closes file
```

```
} // end main
```


Enter the account, name, and balance.

Enter 'N' to end input.

100 Jones 24.98

? Y

200 Doe 345.67

? Y

300 White 0.00

? Y

400 Stone -42.16

? Y

500 Rich 224.62

? N

Reading Data from a Sequential-Access File

- Reading files
 - `ifstream inClientFile("filename", ios::in);`
 - Overloaded !
 - **!inClientFile** tests if file was opened properly
- Upcoming example
 - Credit manager program
 - List accounts with zero balance, credit, and debit

```
const int ZERO = 0, CREDIT = 1, DEBIT = 2, END = 3;
```

```
int main()
```

```
{
```

```
    // ifstream constructor opens the file
```

```
    ifstream inClientFile( "clients.txt", ios::in );
```

```
    // exit program if ifstream could not open file
```

```
    if ( !inClientFile )
```

```
    {
```

```
        cout << "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:
```

```
            cout << "\nAccounts with zero balances:\n";
```

```
            break;
```

```
        case CREDIT:
```

```
            cout << "\nAccounts with credit balances:\n";
```

```
            break;
```

```
        case DEBIT:
```

```
            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 ) )
        cout << account << ' ' << name << ' ' << balance << endl;

    // 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

```

// getRequest function definition

```
int getRequest()
{
    int choice;

    cout<<"Enter 0 to see zero balance accounts"<<endl;
    cout<<"Enter 1 to see credit balance accounts"<<endl;
    cout<<"Enter 2 to see debit balance accounts"<<endl;
    cout<<"Enter 3 to end the program"<<endl;

    cin>>choice;
    return choice;
}
```

// shouldDisplay function definition

```
bool shouldDisplay(int req, double bal)
{
    if( (req == ZERO && bal == 0) || (req == CREDIT && bal < 0) ||
        (req == DEBIT && bal > 0) )
        return true;
    else return false;
}
```

Reading Data from a Sequential-Access File

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

Reading Data from a Sequential-Access File

- Examples
 - **`fileObject.seekg(0)`**
 - Goes to start 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 from current position
 - **`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**
 - **int location = fileObject.tellg()**

Storage in Sequential-Access File

- **"1234567"** (**char ***) vs **1234567** (**int**)
 - **char *** takes 7 bytes (1 for each character)
 - **int** takes fixed number of bytes
 - 123 same size in bytes as 1234567
- **<<** operator
 - **outFile << number**
 - Outputs **number** (**int**) as a **char ***
 - Variable number of bytes

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

Data gets overwritten

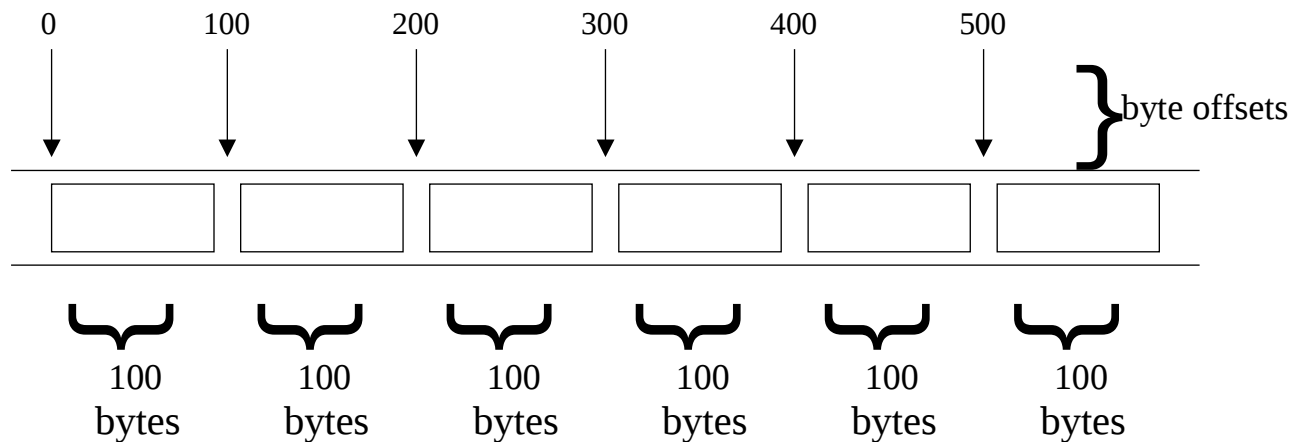
300 Worthington 0.00ones 32.87

Random-Access Files

- Instant access
 - Want to locate record quickly
 - Airline reservations, Banking system, ATMs
 - Sequential files must search through each one
- Random-access files are solution
 - Instant access
 - Update/delete items without changing other data

Random-Access Files

- C++ imposes no structure on files
 - Programmer must create random-access files
 - Fixed-length records
 - Calculate position in file from record size and key



<< operator vs. `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** when using raw, unformatted data
 - Use **ios::binary** for raw writes/reads

Creating a Random-Access File

- Problem statement
 - Credit processing program
 - 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