

File IO

Chapter 17

Motivation

- A program's binary remains static and unalterable
 - All data generated and stored into RAM is lost on completion of the program
- There is a need to store data for prolong periods of time
 - Documents
 - Configuration Files
 - Saves
 - Etc.
- Storing data as a file allows for data to be stored
 - Through multiple executions of a program
 - Through multiple power cycles of the computer

File IO Steps

1. Declare File Stream
2. Open the file
3. Check if the file has successfully opened
4. Read/Write to the File
5. Close the File

Reading a File

1. Declare a File stream

```
ifstream file_stream_name;
```

i.e.

```
ifstream inFile;
```

- This declares an object that is suitable for reading in a file
 - There are strong similarities between ifstream and cin
- Include the following library to use ifstream
 - fstream

Reading a File

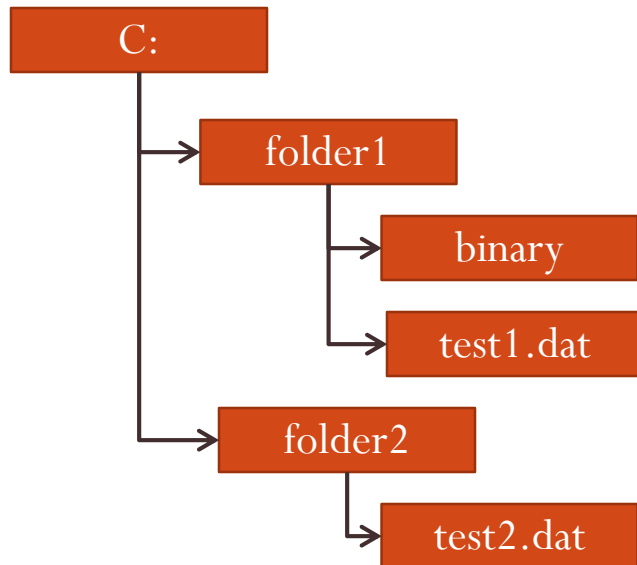
2. Open a file for reading

```
inFile.open("C:\\pathToText\\myFile.dat");
```

- Absolute Paths
 - Specifies a complete path from harddrive to the textfile
 - Regardless of where the program's current path is it will be able to find the file
- Relative Paths
 - Location relative to the program's current path
 - Often used when a file is in the same directory as the program
- Double backslash used because of '\ ' is an escape character

Relative Paths

- A program starts with its current path being that of where its binary is located
- Single Period “.” specifies current directory
- Double Period “..” specifies directory immediately above



Absolute Location of Program's Binary:

C:\\folder1\\binary

Location of test1.dat Relative to Binary:

\\.\\test1.dat

Location of test2.dat Relative to Binary:

..\\.\\folder2\\.\\test2.dat

Reading a File

3. Check if the file opened

```
if(!inFile.is_open())
```

```
{
```

```
    ... provide feedback to user and possibly exit ...
```

```
}
```

- A file can fail to open for the following reasons
 - Doesn't exist
 - Locked by another resource
 - You do not have proper permissions to read the file

Reading a File

4. Read the Contents of the File

- 4 Standard Methods of Reading a File
 - Line by Line
 - Word by Word
 - Character by Character
 - Random Access
 - Typically used for binary files
 - Outside the scope of the class
- We will cover the first 3 methods

What is a file?

- A file is a stream of bytes which can either be in a readable text form or an unreadable binary

T	h	i	s		i	s		t	e	x	t	\n	i	n		a	\n	f	i	l	e	.
---	---	---	---	--	---	---	--	---	---	---	---	----	---	---	--	---	----	---	---	---	---	---

Byte Stream

This is text
in a
file.

- Essentially a 1D Array of Characters
- Newlines are stored as ‘\n’ characters
- There is no character specifying the end of a document
- Files are read from left to right

A Text File

Reading a File Line by Line

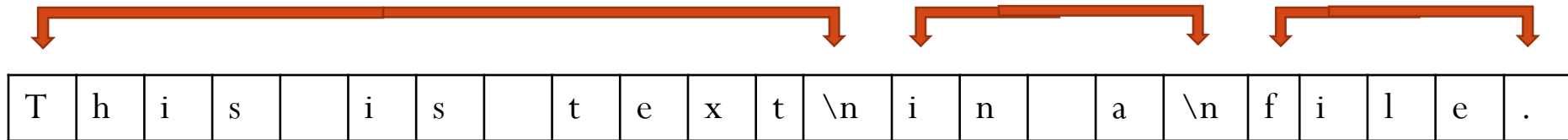
- Getline() can be used to read up to a '\n'
- Used for reading lines of text

```
string sentence;  
getline(inFile, sentence);
```

First Getline Call

Second Call

Third Call



Byte Stream

Reading a File Line by Line

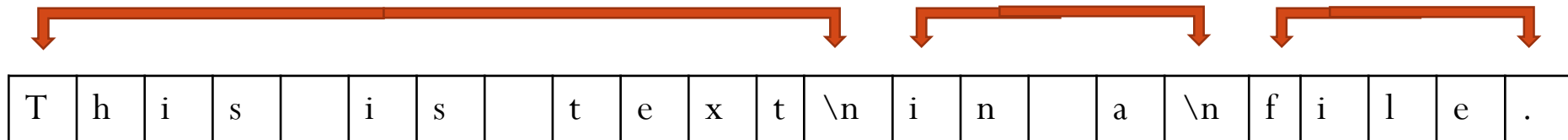
- Reading the complete file
- When the end of the file is reached the `good()` method will return false

```
string sentence;  
while(inFile.good())  
{  
    getline(inFile, sentence);  
    cout << sentence;  
}
```

First Getline Call

Second Call

Third Call



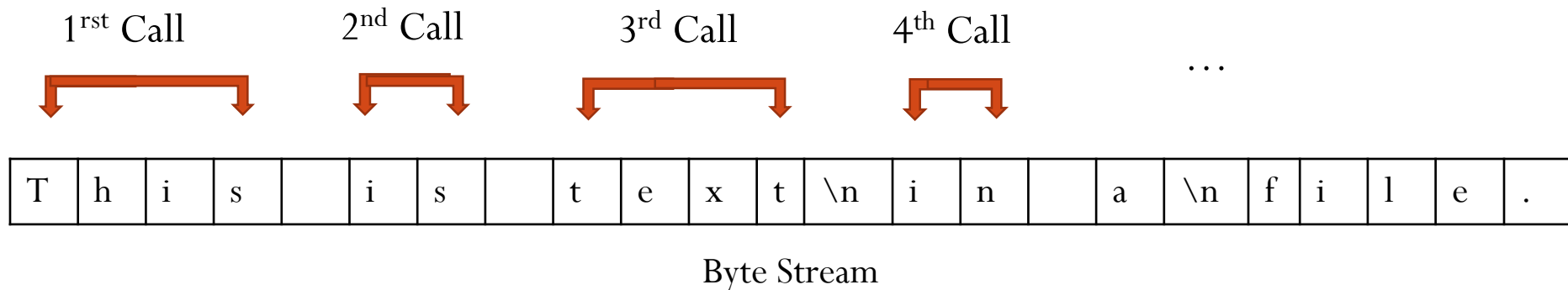
Byte Stream

Reading A File Word by Word

- Files can be read using cin syntax as shown below

```
string sentence;  
inFile >> sentence;
```

- This method allows for reading of numerical data and storing it directly into data types such as int, float, and double



Reading a File Word by Word

- If the file has mixed data types in a regular format we can use word by word to store the numerical data into a data type

```
apples .24  
oranges .88  
bananas .50
```

```
string fruit;  
double price;  
while(inFile.good())  
{  
    inFile >> fruit >> price;  
    cout << fruit << " " << price << endl;  
}
```

A Text File

Word by Word Pitfalls

- If the data read from the file does not match the datatype which extraction operator is trying to store the value in problems will occur
 - There is no easy way to test the datatype of the next data in the file.
- It is common for there to be an empty line at the end of the file. This typically creates problems.

Reading a File Letter by Letter

- Reads just a single character
- Often used for parsing
- More predictable behavior than other methods

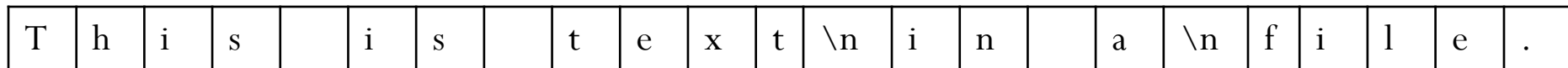
```
char letter;  
inFile.get(letter);
```

1st Call

2nd Call

3rd Call

...



T	h	i	s		i	s		t	e	x	t	\n	i	n		a	\n	f	i	l	e	.
---	---	---	---	--	---	---	--	---	---	---	---	----	---	---	--	---	----	---	---	---	---	---

Byte Stream

Reading a File Letter by Letter

- Below is a routine to read the entire file character by character

```
char letter;  
while(inFile.get(letter))  
    cout << letter;
```

- The get() method will return a 0 when it reaches the end of the file exiting the while loop
- Newline characters ‘\n’ are also read and printed in the above example

Notes on Reading Text Files

- Generation of text files should be done in a program like notepad to ensure the file is ascii
 - MS Word will not generate ascii files!!!!
- All data, even numeric can be read as a string or character
- Handling of the end of file can be quirky
 - If the file ends with a newline character an empty string maybe read
 - It is not uncommon to have the last word in the file read twice
 - Often more robust code must be used to make the behavior more predictable

Reading a File

5. Close the file

```
inFile.close();
```

- Reading from a hard drive is slow so the data is buffered in the ifstream
- To free up the associated memory you should close the file
- Closing the file also prevents any accidental modifications to the file
- After the file is closed you may reuse the string for another file

Best Practices

- Using the getline method is typically the best method to read in a file for the following reasons
 - All datatypes can be read in as a string
 - Data in a string can be tested before conversion, thus safer than word by word which does not test
 - Empty strings at the end of the file are easy to dealt with

C++ Type Conversion

- Convert numbers in a string to an int, float, or double
- Example using stringstream

```
int num1;  
float num2;  
string buffer = "100 101.1";  
stringstream ss(buffer);  
ss >> num1 >> num2;
```

- This is require the library *sstream* to be included

StringStream More Detail

```
string buffer = "100 101.1";  
stringstream ss(buffer);
```

On initialization:

1	0	0		1	0	1	.	1
---	---	---	--	---	---	---	---	---



```
ss >> num1;
```

After First Read:

1	0	0		1	0	1	.	1
---	---	---	--	---	---	---	---	---



```
ss >> num2;
```

After Second Read:

1	0	0		1	0	1	.	1
---	---	---	--	---	---	---	---	---



stringstream for Generating Strings

- stringstream can be used to generate a string using mixed datatypes

```
int num1 = 100;
float num2 = 101.1;
string buffer;
stringstream ss;
// Places integer and float into stream
ss << num1 << " " << num2;
// Converts stream to string
buffer = ss.str();
```

Exercise 11.1

- Generate a file in notepad with several items which follows the pattern below:

```
item_name    price_per_item    quantity
```

- Read in the file and calculate the cost of each item by price * quantity and print it to the console
- Calculate the Total Cost and print it to the console

Writing a File

1. Declare File Stream

`ofstream name_of_stream;` General format

`ofstream outFile;` Example

2. Open File

`outFile.open(" ... location of file ...");`

- If the file doesn't exist it will be created
- If the file does exist it will be truncated

3. Check if File Opened

`if(!outFile.is_open())`

...

- Though less likely to fail than reading a file, it is still possible

Writing a File

- Writing to the file can be accomplished using cout syntax but using your declared stream as cout, i.e.

```
outFile << "this text will be outputed" << endl;
```

```
outFile << someVar << endl;
```

- Even iomanip functions such as setw() work the same for outFile

Writing a File

5. Close the file

```
outFile.close();
```

- Rational for closing the file is the same as reading a file

Example 11.2

- Using the code from Example 11.1, dump the calculated data to a file instead of just the console.

Final Notes

- Writing a file does necessary have to truncate the data opening it up in another mode can allow for appending
- Other methods exist that can be useful for File IO manipulation
 - See your book or Cplusplus.com for information on these
- Fetching data from a file is trivial however parsing the data into useful variables is typically a harder problem