

Chapter 10- Interactive programs: File IO

Valerie Colber, MBA, PMP, SCPM

Text file input

- Another form of interactive input is text files.
- Text files are like typing responses to console prompts, but doing so in advance and storing them in a file.
- It's remarkably easy to convert console input to text file input.
- Actually, you can have both file and console input in the same program.
- For example, you could use console input for a user to type the name of a text file, and then get the rest of the input from that text file.)

Text file input

- But there are two important differences between console input and text file input.
- Prompts are not necessary for text file input -- remember, the responses are typed in advance and stored in a text file.
- So the user has to know the program pretty well in order to know what to reply *before* the questions are asked!
- Another difference is that the *name* of a file has to be specified, and your program has to "open" the file.

Text file input

- You do not have to deal with "error handling" when using console I/O, but you do with file I/O because things can go wrong with files.
- For example, a specified file may not exist, or it may not be complete.
- So we will learn the basics of file error handling -- enough to get by.
- The coding requirements for text file input are these:
 1. type the proper include statement
 2. Replace **cin** with its text file equivalent, **fin**
 3. "opening" a file
 4. add error handling.

Console vs. File input

Console
input

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

cin

text file
input

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

ifstream fin;

fin.open ("data.txt");

fin.close();

programmer:

```
string name = "Fred";
```

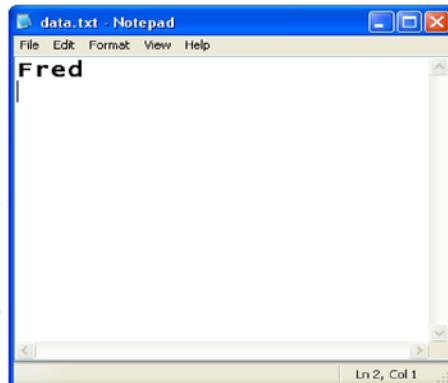
Console:

```
string name;  
cout << "What is your name? ";  
getline(cin, name);
```



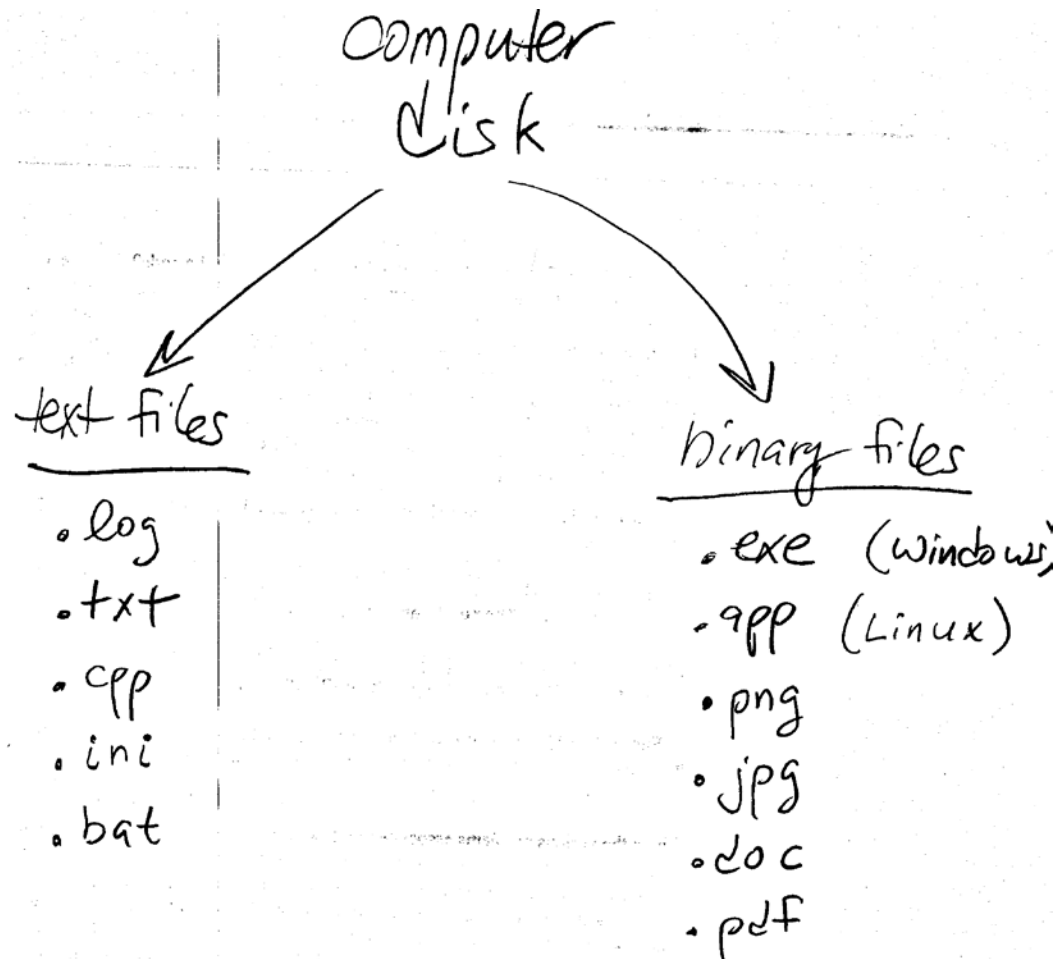
text file:

```
string name;  
getline(fin, name);
```



3 sources of input

Different files



Include statement for File IO

#include <fstream>

File In

- Actually, this does not have to be named **fin** -- it just has to be any valid C++ identifier.
- Here's what to insert in the data code block in the body of "main", before any other reference to **fin** (*remember sequential processing!*):

ifstream fin;

- The above statement reserves the identifier **fin** to be used for reading one or more text files.
- In the same way that the same variable identifier cannot be declared more than once, the same is true of **fin** .

Opening a file for text input

- "Opening" a
- file makes an exclusive, read-only attachment of a specified file to your program.
- This attachment continues until your program "closes" the file or ends.
- Here is how to open a specified file:
fin.open("data.txt");
- Then after the last reference to **fin** , when you are finished reading from the file, use this statement to close the file:
fin.close();
- In the above, the name of the file to be opened is **data.txt**.
- It is expected to exist in the *working* folder.
- Not closing a file before program execution completion may cause data loss or corruption and prevents opening another file using the identifier **fin** during program execution.

Opening a file for text input

- You can specify that a file be located in any other folder by including the folder specification in front of the filename, like this (*note the forward slash / -- some systems allow a backslash instead, but it has to be written as a double-backslash: *):

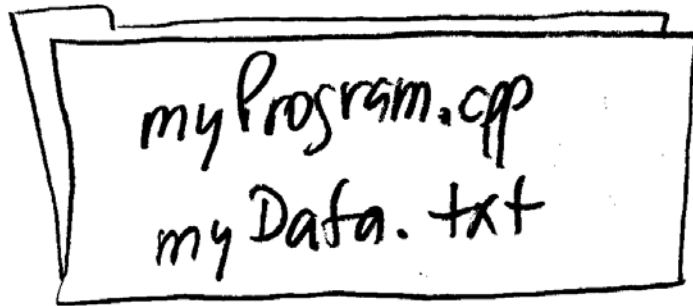
`fin.open("e:/Programming class/data.txt");`

- The filename does *not* have to end in **.txt**.
- It can be anything, including **.ini** or no filename extension at all.
- The only requirement is that it be a text file -- one that is editable and viewable using any text editor.
- The other type of file is "binary" -- we will deal with those in chapter 11.

In this course we will...

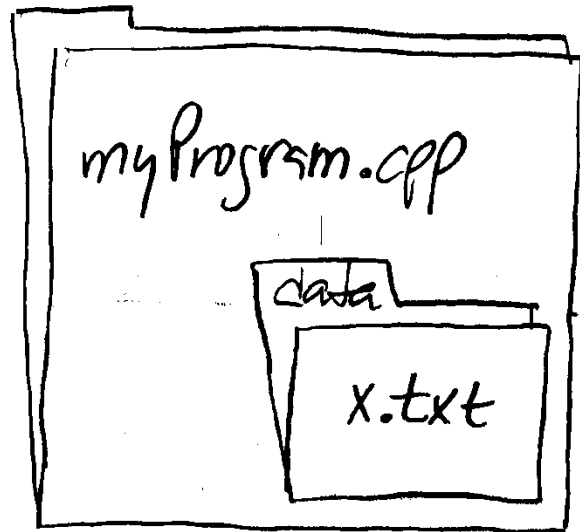
- We will NEVER specify the full pathname for a file!
- We will assume the file exists in the same directory/folder where the program exists.
- That way a program can be execute on any computer regardless of structure.

Where's that file?



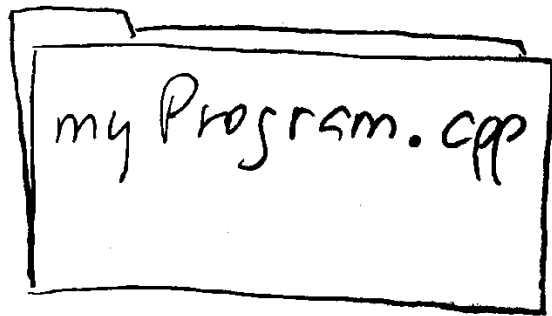
```
fin.open("myData.txt");
```

Where's that file?

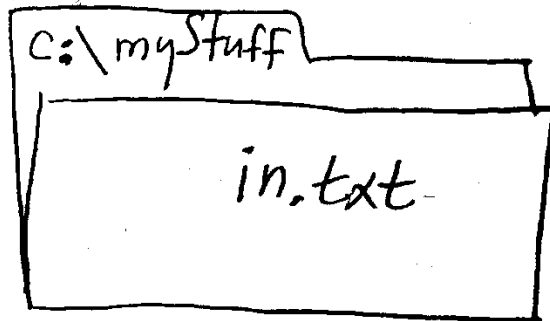


```
fin.open("data/x.txt");
```

Where's that file?



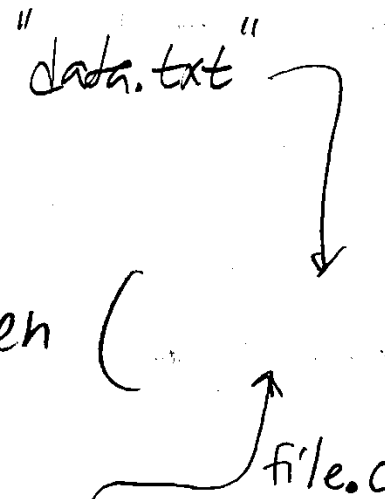
```
fin.open("c:/myStuff/in.txt");
```



Literal string versus string variable

`fin.open ("data.txt");`

`file.c_str()`



`string file = "data.txt";`

OR

`string file;`

`cout << "Enter file name: ";`

`getline (cin, file);`

Opening a file for text input

- The filename can be specified using a string variable instead of a filename written in quotes, like this:

```
string fileName = "e:/Programming class/data.txt";  
    fin.open(fileName.c_str());
```

- Note the peculiar use of `.c_str()` in the file open statement.
- This is a peculiarity of C++ in the way that text variables get used inside parentheses sometimes.

Opening a file for text input

- Since you can use a text variable to specify the filename in an "open" statement, that makes it possible to prompt the user to type the name of a file:

```
string fileName;
```

```
cout << "What file do you want to use for input? ";
```

```
getline(cin, fileName);
```

```
fin.open(fileName.c_str());
```

- Note that you may include a folder specification for a file when you type it in response to a prompt.
- Use either a slash / or a backslash \ to separate parent and child folders -- you do *not* have to use double-backslash as you would do if you typed the filename in your code.

Opening a file for text input

- To read a file a second time, close it and open it again. Here's how to reopen the same file, or use **fin** to open a different file:

```
fin.close();
```

```
fin.clear();
```

```
cout << "What file do you want to use for input next?"  
    ;
```

```
getline(cin, fileName);
```

```
fin.open(fileName.c_str());
```

fin.open alternatives

```
// open a specifically named file, data.txt, in local folder
fin.open("data.txt");
// open a specifically named file, data.txt, in some other folder
fin.open("e:/Programming class/data.txt");
// open a file whose name is stored in a string variable
string fileName = "e:/Programming class/data.txt";
fin.open(fileName.c_str());
// open a file whose name is entered by the user
string fileName;
cout << "What file do you want to use for input? ";
getline(cin, fileName);
fin.open(fileName.c_str());
// open a 2nd file (storing the filenames in a String variable)
fin.close(); // be sure to close 1st file before opening 2nd
fin.clear();
cout << "What file do you want to use for input next? ";
getline(cin, fileName);
fin.open(fileName.c_str());
```

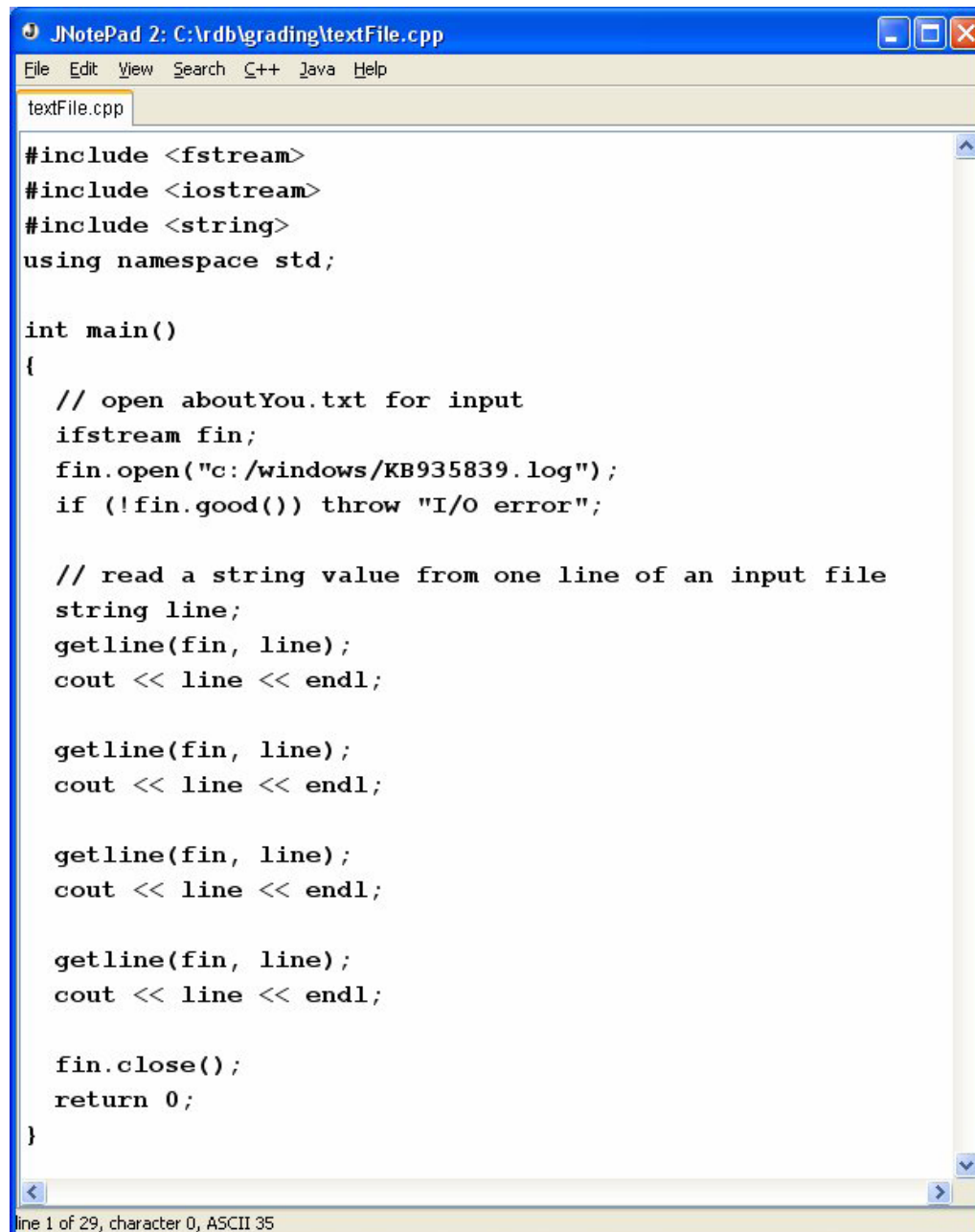
Error Handling

- File error handling is *not* required in C++.
- The purpose of writing error handling code is to tell the computer what to do *in case* there is a problem with file I/O.
- There does not actually have to *be* a problem in order to use error handling -- there just has to be the *possibility* of a problem.
- To accommodate possible file I/O errors, insert this statement after **fin.open** :

if (!fin.good()) throw "I/O error";

- This causes the program to terminate in case the file does not exist.
- **throw "I/O error";** THIS DOES NOT OUTPUT that text!!! That text is a code that your program during execution *throws* to the operating system before suffering an execution error so that the operation system may inform the user in a less alarming fashion.
- Some operating systems, such as Windows, may require that the user press CTRL-C in order to return to the command prompt after such errors.

textfile.cpp



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

int main()
{
    // open aboutYou.txt for input
    ifstream fin;
    fin.open("c:/windows/KB935839.log");
    if (!fin.good()) throw "I/O error";

    // read a string value from one line of an input file
    string line;
    getline(fin, line);
    cout << line << endl;

    getline(fin, line);
    cout << line << endl;

    getline(fin, line);
    cout << line << endl;

    getline(fin, line);
    cout << line << endl;

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

line 1 of 29, character 0, ASCII 35

aboutYou.txt (input file)

46

3.9

George Washington

M

Try rewriting itsAboutYouFile.cpp

- To read that input from aboutYou.txt data input file

What to note about textfile.cpp

- Of course, itsAboutYou2.cpp program produces no output.
- But by leaving out this detail, we focus on text file input coding.
- Note how similar it is to console input -- one uses **cin** and the other uses **fin** .
- Also note the differences -- text file input requires that the filename be specified.
- In file input, the statement that "reads" an input value also skips to the next input value in the file – that is, it reads it sequentially.
- There is no need to write any code to advance the program to the next input value, because this happens automatically.
- There is also no going back or skipping forward -- not easily, anyway.
- The most straightforward way to go back to a previous input is to close and reopen the text file.

See example
readFile.cpp

Text file input and output

Input Output options

- Input sources: programmer, console keyboard, text file
- Output destinations: console screen, created text file, appended text file
- Input sources and output destinations can be used in *any* combination.
- For example, a program can use keyboard input to enter an input filename, read input values from a text file, calculate results and print them to the console screen and append them to an output text file.
- This next visual shows several ways to do the simple area calculation, with various input/output possibilities.

Multiple inputs on the same line

- The purpose of the statement **`fin.ignore(1000, 10);`** is to skip to the next line in the input file.
- But if you leave this statement out, you can put multiple inputs on the same line -- separated by one or more spaces.

46 3.9 George Washington
M

text filename: **aboutYou.txt**

NOT RECOMMENDED!

Multiple inputs on the same line

- Multiple inputs on one line make it easy for the user to make a mistake!
- By removing the first two **ignore** statements, you can do one of two things to separate the first three inputs:
 - either put them on the same line separated by spaces, or
 - leave them on separate lines.
- But note that there is never an **ignore** statement following **getline**.
- **getline** automatically reads to the end of a line.
- In the example above, the "M" could not be moved to the line above, because **getline** is used to read to the end of that line.
- All of this applies to console input using **cin**!
- By leaving out **ignore**, you can stage multiple inputs on the same line before pressing ENTER.
- Or you can still press ENTER between each input -- it becomes the user's choice.

Text File Output

- Sending output to a text file is a lot like sending it to the console.
- The include is the same as for text file input.
- The error handling code is the same as for text file input.
- Also, the name of the output text file must be specified as it must for text file input.

fout

- As with **fin**, this does not have to be named **fout** -- it just has to be any valid C++ identifier.
- Here's what to insert somewhere in the body of "main", before any other reference to **fout**:

ofstream fout;

- This reserves the identifier **fout** to be used for writing one or more text files.

Creating a text file

- When specifying a filename for output, the file may either already exist or not.
- If you specify a file that already exists, the file will be discarded and overwritten without warning.
- If the file does not already exist, this process creates it.
- In most operating systems it will appear in the file listing as soon as it is opened for output, but its size may be reported as zero until it is closed (or the program ends).

- Here is how to open a specified file:

```
fout.open("data.txt");
```

- Then after the last reference to **fout**, when you are finished writing to the file, use this statement to close the file:

```
fout.close();
```

Creating a text file

- In the example, **data.txt** will be created (or recreated) in the *working* folder.
- As with input files, you can specify that a file be located in any other folder by including the folder specification in front of the filename.
- Also, the filename does *not* have to end in **.txt**.
- The text file that is written by your program will be editable and viewable using any text editor.
- Overwriting of files is useful if you are keeping score in a game, or saving the state of a program for later restart at the same position.
- For example, a tic-tac-toe game may save the number of wins, losses, and ties in a text file.
- Upon starting the game, the program could read the file and display the win-loss history.
- While playing the game, the win-loss history is updated by new wins and losses.
- Then when the program ends, it can then replace the old win-loss history file by overwriting it.

See example
keepingScoreFile.cpp

What to note about keepingScoreFile.cpp

- The keepingScore.cpp program uses a text variable **scoreFile** to store the name of the text file.
- This is an alternative to typing the exact same name twice, in the two file opening statements (one for input and another for output).
- It is good programming practice to store a recurring value in a variable and use the variable instead of repeating the value.

What to note about keepingScoreFile.cpp

- Blank lines are used to separate blocks of code that are related, making it easier for humans to read -- it makes no difference to the compiler.
- Variables are declared just before they are used, instead of putting all of them at the top of "main" -- this is by programmer preference.
- The **.close()** statements appear as soon as they possibly can.
- This is not important for **fout.close()**, because the program ends after that anyway.
- But it is important that **fin.close()** appear before attempting to open **fout**, because otherwise the file remains exclusively attached to **fin** and the **fout** would fail.

What to note about keepingScoreFile.cpp

- Two new integers, **newWins** and **newLosses**, are used because it is not possible to directly *add* a value from a file to an existing variable.
- You have to read the value from the file separately, and then add it.
- Both **cin** and **fin** are used because there is input from both the keyboard and a file.
- Note that prompts are associated with the keyboard inputs, but not with the file inputs.
- Both **cout** and **fout** are used because there is output to both the console and a file.

What to note about keepingScoreFile.cpp

- Some of the console output statements skip to the next line after printing and some do not.
- Why???
- The file output statements skip to the next line after printing - is this important?
- What would happen if they did not skip to the next line?
- This code listing is rather long and complicated.
- If there were any typing mistakes, what would the compiler errors look like?
- Try making some errors and see for yourself -- you could leave out includes, semicolons, open statements, `c_str()`, etc.
- Note that some errors get by the compiler and do not show up until you run the program.

fout.open alternatives

```
// create or overwrite a specifically named file, data.txt, in local folder
fout.open("data.txt");

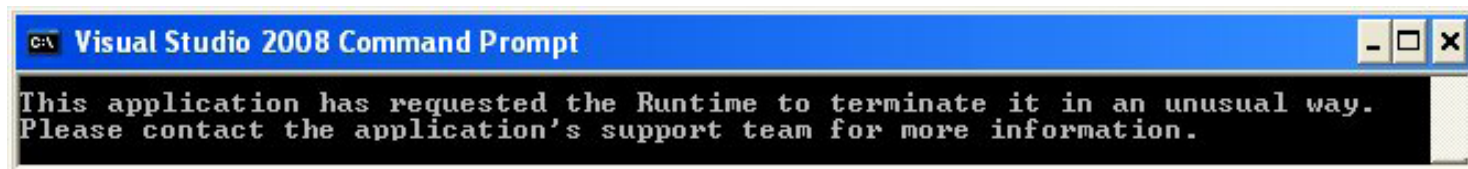
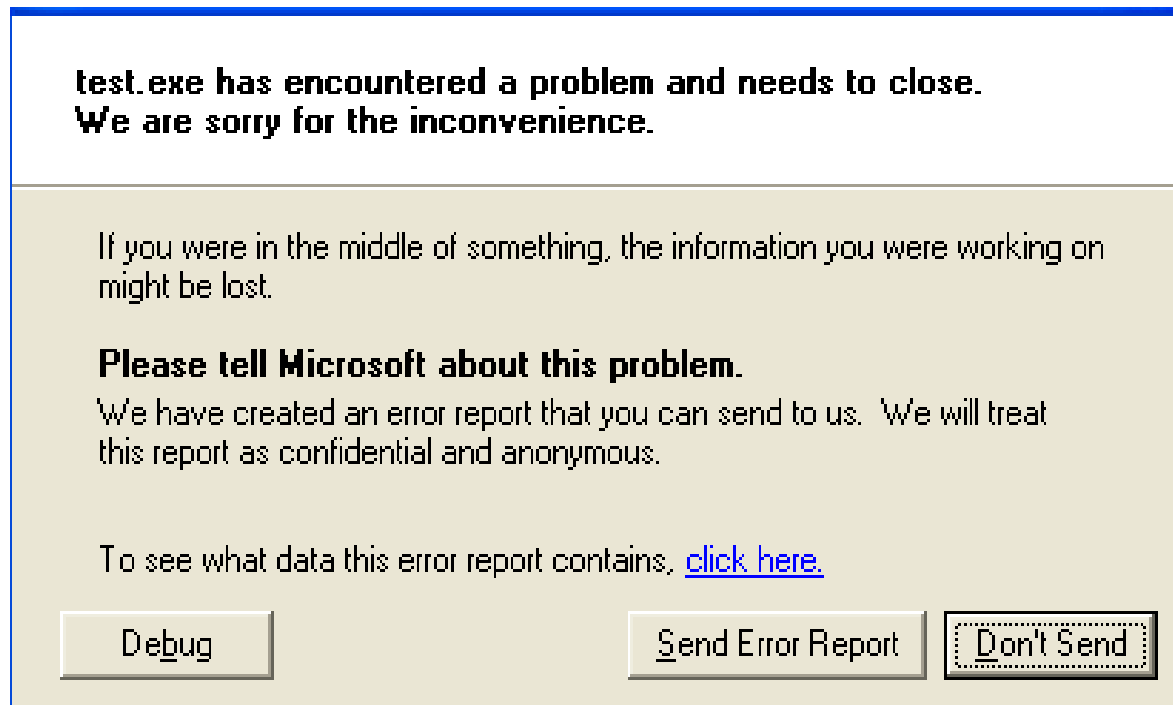
// create or overwrite a specifically named file, data.txt, in some other folder
fout.open("e:/Programming class/data.txt");

// create or overwrite a file whose name is stored in a string variable
string fileName = "e:/Programming class/data.txt";
fout.open(fileName.c_str());

// create or overwrite a file whose name is entered by the user
string fileName;
cout << "What file do you want to use for input? ";
getline(cin, fileName);
fout.open(fileName.c_str());

// create or overwrite a 2nd file (storing the filenames in a String variable)
fout.close(); // be sure to close 1st file before creating 2nd
fout.clear();
cout << "What file do you want to use for input next? ";
getline(cin, fileName);
fout.open(fileName.c_str());
```


How to tell if you are trying to open an input file that does not exist



Appending to a text file

- By default, any existing text file is discarded and overwritten when you open it for output.
- But it is possible for your program to *append*, or add to an existing file instead of overwriting it.
- This is useful if you are building a database of transactions, for example, where each new transaction gets saved to a file.
- Here is how to open an output text file for appending instead of overwriting:

`fout.open("data.txt", ios::app);`

- In the above, if the file does not already exist, it is created as it would have been without the specification to append.

classRoster.cpp example

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

int main()
{
    //data
    string name; //user name
    ofstream fout; //represents output file

    // read a student's name from the keyboard
    cout << "What is your name? ";
    getline(cin, name);

    // write the student's name to classRoster.txt
    fout.open("classRoster.txt", ios::app);
    if (!fout.good()) throw "I/O error";
    fout << name << endl;
    fout.close();
}
```

classRoster.cpp example

- Note the change in the file open statement that specifies that the output file be appended to, instead of overwritten.
- When you compile and run the classRoster.cpp program, and note that the output file is created the first time you run the program -- you do not need to create it in advance!
- This applies to any and all of the **fout.open** code blocks!

References

- INTRODUCTION TO PROGRAMMING USING C++ by Robert Burns