FILE HANDLING IN C++

Outline of what we will cover

Objectives: these sections only!

1. Defining File Streams sec 5-1
2. Reading Data Files sec 5-2
3. Generating a Data File sec 5-3
4. Numerical Technique: Linear Modeling* sec 5-6
5. Problem Solving Applied: Ozone Measurements* sec 5-7

6. **Be sure to run youtube video(s) in the online syllabus (watch this one after you understand a bit of file access)**

Objectives

Develop problem solutions in C++ that:

- 1. Open and close data files for input and output.
- 2. Read data from files using common looping structures.
- 3. Check the state of an input stream(incoming data).
- 4. Recover from input stream errors(prevent our program from
- crashing (if there is an input error from a file).

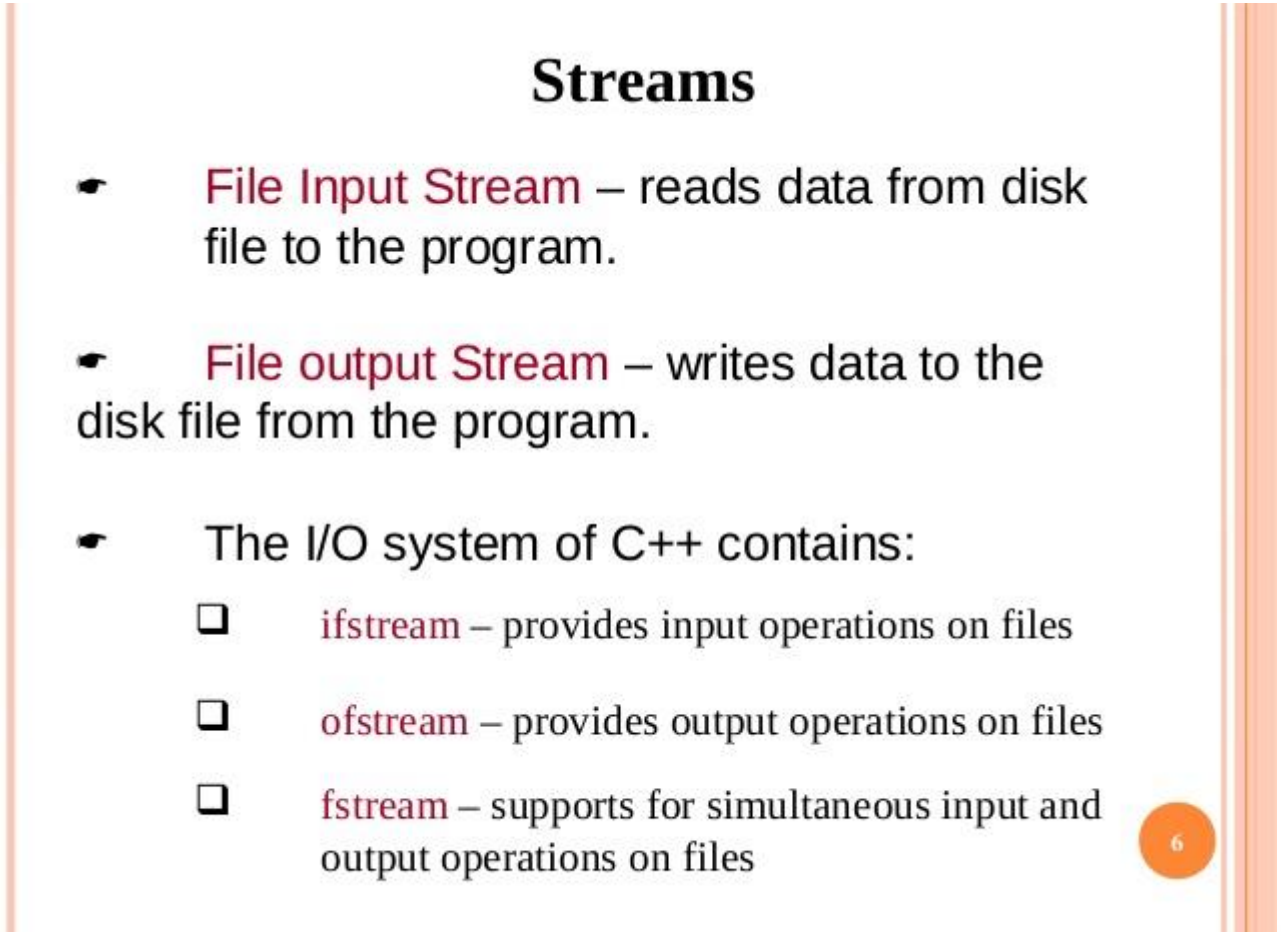
- 5. As an important aside for engineering we will
- Apply the numerical technique of linear modeling.

Example: Standard Input and Output Streams

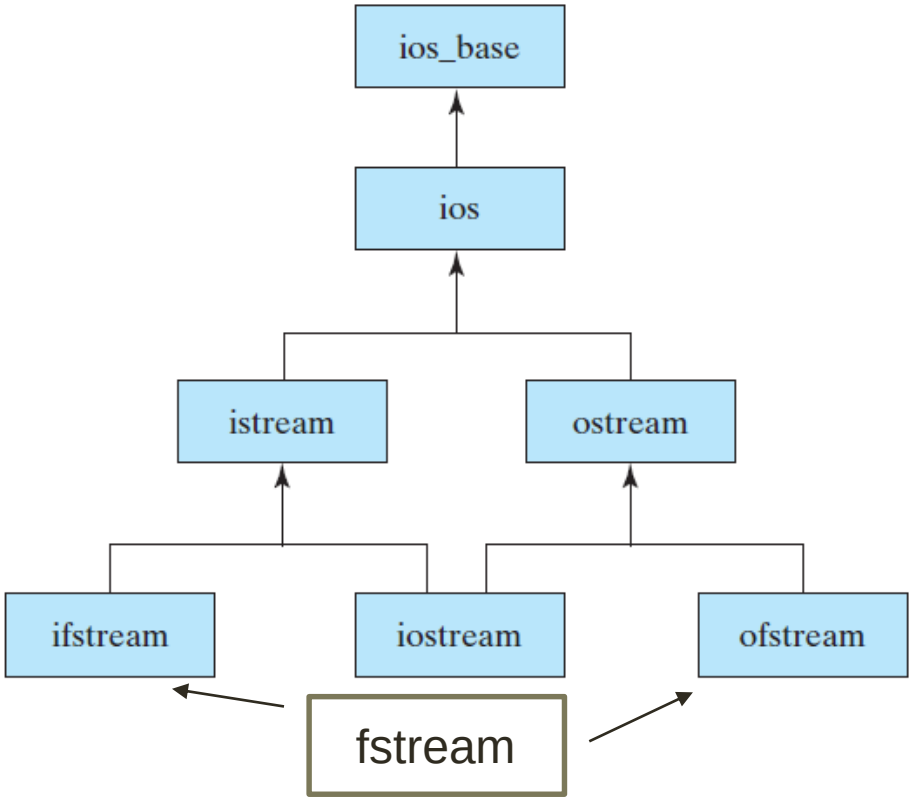
- C++ defines the `cin` and `cout` objects to represent the standard input (keyboard) and standard output (console).
- C++ also defines the standard error stream, `cerr`. Which streams output to a **standard error** output device. (**our screens could be elsewhere**)
- Standard input is a kind of general input stream class; standard output and error streams are special kinds of general output stream.

Defining File Streams

- Standard C++ library also defines
  special kinds of streams for **file** input and output.
  - #include <ifstream>
  - #include <ofstream> **but #include <fstream> supports both!**



Stream Class Hierarchy: a bit confusing diagram in the text
but input/output operations will be clear to you by the examples we will study. For now just take the concept
of "File stream classes" as all the functions and sub-functions provided by the C++ Standard Library for input and output to
files. Many will be obvious once we use them. We already have
used the "iostream" class objects **"cin" and "cout"** connected to the standard input (keyboard) and output devices (screen).



Example using **Data files** (saved as text files):we will construct a program to analyze a file with two sets of numbers and check all values if they are negative values and then **create** another data file with only the positive values from the first one. The data originally is sets of two numbers from some kind of sensor.

CPP FILES ARE FOUND AT THE ONLINE COURSE SYLABUS SOURCE CODE SECTION. WE CAN AND WILL CREATE DATA FILES WITH NOTEPAD (all are text files). File Sensor.dat (3rd ed) or Sensor.txt (4th ed) used is organized as number below but **best save file as "'sensor" with no extension (automatically a txt file!) we use ".txt" in our program but not when we save to avoid some problems with our system. Avoid for now ".dat"!**

```
 0  45
 1  48
 2   56
 3  -1
 4   10
-5    8
-6   -4
 6   12
```

<- be sure to use the enter key here to advance to the next (blank) line in your file! Our program will read these numbers and weed out negatives as it creates on output a data file only with the positive set of numbers. All files have to be in the same directory as the source (.cpp) file of the project if we do not specifically specify the director(s) the data files are in or going to be created. NOTE: program uses the new stream object **"cerr"** which sends output to the **standard error** output device **(screen)**

```
/*-------------------------------------------------------------*/
/* Program 5_1                                                 */
/* This program reads data pairs from the                      */
/* the file sensor.txt and writes valid data pairs             */
/* to the file checkedSensor.txt. Valid data pairs             */
/* may not be negative. Invalid data is written to             */
/* to standard error(cerr)                                     */
#include<iostream> //Required for cerr
#include<fstream> //Required for ifstream, ofstream
using namespace std;
int main()
{
  //Define file streams for input and output. One of several ways to do this!
  ifstream fin("sensor.txt");
  ofstream fout("checkedSensor.txt");
  //Check for possible errors.
  if(fin. fail())
  {
    cerr << "could not open input file sensor.txt\n";
    exit(1);
  }
  if(fout. fail())
  {
    cerr << "could not open output file checkedSensor.txt\n";
    exit(1);
  }
  //All files are open.
  double t, motion;
  int count(0);
  fin >> t >> motion;
  while(!fin.eof())
  {
    ++count;
    //Write valid data to output file.
    if(t >= 0 && motion >= 0)
    {
      fout << t << " " << motion << endl;
    }
    //Write invalid data to standard error output.
    else
    {
      cerr << "Bad data encountered on line"
          << count << endl
          << t << " " << motion << endl;
    }
    //Input next data pair.
    fin >> t >> motion;
  }//end while
  //close all files.    fin.close();    fout.close();    return 0;      }
```

```
/*-------------------------------------------------------*/
/* Program 5_1                                           */
/* This program reads data pairs from the                */
/* the file sensor.txt and writes valid data pairs       */
/* to the file checkedSensor.txt. Valid data pairs       */
/* may not be negative. Invalid data is written to       */
/* to standard error(cerr)                               */
#include<iostream> //Required for cerr
#include<fstream> //Required for ifstream, ofstream
using namespace std;
int main()
{
//Define file streams for input and output.One of several ways to do this!
  ifstream fin("sensor.txt");// CLASS ifstream   OBJECT FIN
  ofstream fout("checkedSensor.txt"); // CLASS ofstream   OBJECT fout
  //Check for possible errors.
  if(fin. fail())
  {
    cerr << "could not open input file sensor.txt\n";
    exit(1);
  }
  if(fout. fail())
  {
    cerr << "could not open output file checkedSensor.txt\n";
    exit(1);
  }
  //All files are open.
```

```
 double t, motion;
  int count(0);
  fin >> t >> motion;
  while(!fin.eof())
  { ++count;
    //Write valid data to output file.
    if(t >= 0 && motion >= 0)
    {
      fout << t << " " << motion << endl;
    }
    //Write invalid data to standard error output.
    else
    {
      cerr << "Bad data encountered on line"
           << count << endl
           << t << " " << motion << endl;
    }
    //Input next data pair.
    fin >> t >> motion;
  }//end while
  //close all files.    fin.close();    fout.close();
return 0;       }
```

HAND IN LABORATORY TASK: LAB #18 Read files

Create the NAME.txt file and HAND IN this last program WITH COPY OF it and the output file created (5.1 cpp File at the syllabus-source)

Sensor.txt Be sure to hit return on last set of number. Use these

0 45
1 48
2 -56
3 -11
4 10
-5 8
-6 -4
6 12 <- hit return (Enter) key here!

Copy both the data file accessed and created. Make changes to see the consequences for your self.

The lines ifstream fin("name. txt"); opens the file for getting data or input to the program.

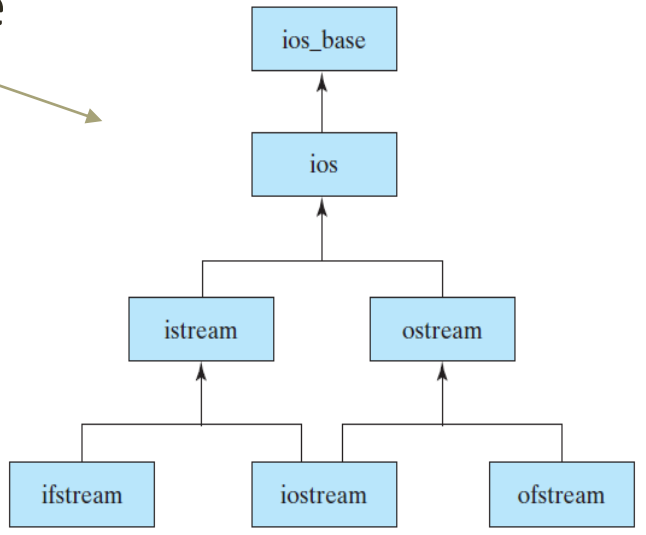
The line ofstream fout ("checkedSensor.txt"); creates and opens the file for data from the program or output from the program to the file

Next are Alternative ways to open
files for input and output using the file stream classes (defined structures for IO to files) specifically classes we have encountered
ifstream and ofstream
we define "objects" (names that are linked to the files) in these classes as we saw before ! Study carefully examples to follow

The ifstream Class

The ifstream (input file stream) class is ***derived*** from istream (input stream class).

- Thus ifstream ***inherits*** the input operator and member functions eof() and fail() defined in istream. eof() function was used in the last program in the loop command "while"
- The fail() function was used twice in the last program to be sure the files were available at the default place. See the two if statements that used it
- The ifstream class also defined specialized methods specific to working with files.



Input File Streams (alternative ways open files for input)

In the program we used `#include<fstream>` and the following:

- `ifstream fin("sensor.txt");` opens the file for input
- then used (object "fin")
- `if(fin. fail())` to test for the file and
- `fin >> t >> motion;` to get the data
- We can setup data files used for input by using an **ifstream object as** fin above but a different name "sensor1" below associated with it and can be defined This way!

```
ifstream sensor1;  creates object "sensor1" but does not open file.
            sensor1 can do what fin did before
sensor1.open("sensor1.txt"); //just opens the file then we check if the file
is open as before.  IE object sensor1 calls open() function!
```

- Or just check open and check alternatively as follows.

```
ifstream sensor1("sensor1.txt");   will open the file(similar to "fin" above)
 if(!sensor1)  // checks if file is open alternate to calling fail()!
HERE "sensor1" is the object in the ifstream class
Now we can use the object name for input to the program like "cin" for keypad
or "fin" above for files.

sensor1 >>t>>motion;  // reading the data in the file(similar to "fin" above)

NOTE: always show a few lines of input on the screen to be sure you are
reading the file correctly ie  use  cout! To see data on the screen since the
file object does not do that!
E.g.sensor1>>t>>motion;
    cout>>t>>motion;//lets us see the data every time it is read from a file
```

Avoiding Bugs

- If opening the named file fails, the fail error bit is set, and all statements to read from the file will be ignored.
  - NO error message will be generated but the program will continue to execute. So we used as before the “cerr” To get a message to us, see below!
  - So we Checked to be sure that the open was successful using the
    - fail() method of the istream class returns false. (ifstream inherits functions from the istream!) “a bit technical here”
    - So we used cerr To get a message to us! And then use exit(1) to stop the program. in the last program we had lines like
    - If (fin.fail()) { // as before
    - cerr << ”could not open input file\n”;
    - exit(1);
    - }
    - USING **THE CLASS OBJECT** ALTERNATE METHOD TO DO THIS IS NEXT!

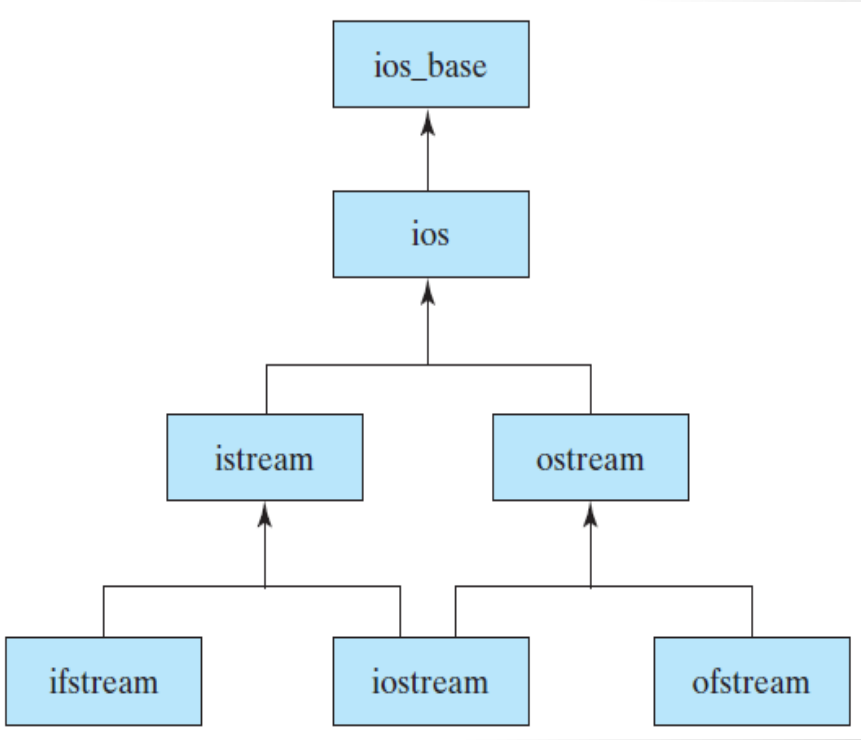
Input File Example using an ifstream object name (sensor1 here) to check file opening (ALTERNATE APPOACHES!)

```
ifstream sensor1;
sensor1.open("sensor1.txt");
if ( sensor1.fail() ) //open failed-preferred
{ cerr << “File sensor1.txt could not be opened";
   exit(1); //end execution of the program
}

Alternative 3:
ifstream sensor1(“sensor1.txt”);
if( !sensor1)  // open failed- a more direct way
{ cerr << “File sensor1.txt could not be opened";
   exit(1); //end execution of the program
}
   getting dizzy?   Stick to a method your comfortable with
and don’t worry but be sure to know these variations exist
```

The ofstream Class

- The ofstream (output file stream) class is ***derived*** from ostream (output stream class).
  - Thus ofstream ***inherits*** the output operator and member functions ostream.
  - The ofstream class also defines specialized methods specific to working with files.



Output File Modes (Alternate methods)

- WE USED IN THE FIRST PROGRAM
- `ofstream fout("checkedSensor.txt");`
- `Which defined the object fout and created and opened the file for output from the program to the file`
- If we use just define our object with
- ofstream balloon: // Defines the object balloon then WE USE->
- balloon.open ("balloon.txt") which calls on the function open() a member function of the ofstream class. And a file for output in this way will either be created if it doesn't already exist or overwrite a previously existing file with the same name! (this will DESTROY the existing file be careful if you want the data avoid this!).

IF we wish to simply append (add new data) i.e. new content to the previously existing file, we open the file in append mode:

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

- Rarely errors happen when we open a file to output so we don't check for errors but be **careful about previously existing ones!!**

Output File Streams alternative ways to output to a file data

- In our first program we used
- `ofstream fout("checkedSensor.txt");`
- `if(fout.fail()) to check for the file`
- And
- `fout << t << " " << motion << endl`
- To write data to the file (**note the space inserted**)

- Now using the class approach Each data files used for output must have an ofstream object associated with it:

```
ofstream balloon;
balloon.open("balloon.txt");
```

- Or just –

```
ofstream balloon ("balloon.txt");
```

In the last twp statements we initialized the object **balloon** to link to the data file **balloon.txt to write data to it!**

Now just like "cout" for screens or "fout" for files we can use the object name for output to the file

Illustration to write to the file ballon.txt

```
Balloon <<time <<" " << height << " "<< velocity/3600<< endl;
```

The latter values all go into the associated file (balloon.txt).

Closing open input or output files

- The close(), (ie.close the file access), methods for both input and output stream objects may be called when the program is done with a file if needed. For the previous 2 examples

- sensor1.close();
- Balloon.close()
  - i.e. each object has an associated file it reads(sensor1.txt) or writes to (Balloon.txt)
  - The files will be closed automatically when the program exits with a return 0 use of exit().

Accessing a file with its name!
very import for engineering and science files

- When you use a **string class** to represent the file name, you must use the special function c_str() method to provide the appropriate type for use in **input and output**. Illustrated as follows for an output file
- **string** filename; (“filename” is the string variable you will put the real file name into)
- Used as follows to access a file for output here.
- cout << “Enter the name of the file you want to open for output”
- cin >> filename;
- Balloon.open (filename.c_str());

Hand in HW #11 40 PTS (5 EACH) ANSWER CAREFULLY

Read and Study section 5-1 and carefully pick the correct answer to the following quesitions. Use QUESTION NUMBER AND letter answers in the HW.

1. The class name used to declare a user defined input file is …

A. inFile B. ifstream C. iostream D. istream

2. Which include statement must you use to define and use files A. #include<ifstream>

B.#include <filestream> C. #include <cstream> D. #include <fstream>

3. Which of the following statements correctly declares the input file object MyInput and initializes the file object to read the file *exam.txt*? A. ifstream MyInput("exam.txt");

B MyInput ifstream("exam.txt"); C ifstream MyInput(exam.txt); D.MyInput ifstream (exam.txt);

4. What is the next step in using a file, after the file has been declared with the following statement? ofstream AnsFile; A. You use the file to input a variable.

B. You use the file by outputting a value to it. C.You search for a file named AnsFile.

D. You open the file, and associate it with a specific file name.

5. Which of the following statements correctly opens the output file object AnsFile to store answers into file *myanswers.txt* ? A. AnsFile.open("*myanswers.txt*");

B. open.AnsFile("*myanswers.txt*"); C. open.AnsFile(*myanswers.txt*); D. AnsFile.open(*myanswers.txt*);

6. Which of the following statements will properly store the number from variable *value* into the output file object AnsFile? A. AnsFile << *value*; B. AnsFile << cout << *value*;

C. cout << AnsFile << *value*; D. store.AnsFile(*value*);

7. Read a number from the input file object MyInput into variable *value*

A. MyInput >> value; B. MyInput >> cin >> value; C. cin >> MyInput >> value; D. value = read.MyInput();

8. Which of the following statements will close the output file object AnsFile from the previous question? A. close.AnsFile; B."*myanswers.txt*".close; C.AnsFile.close();

D. AnsFile.close ("*myanswers.txt* ");

Reading Data Files

How do we know how the data is setup in a file we want to use?.

We have to know certain things about the file before we proceed. Such information is usually available by those who created the file.

We need to know

1. The **name of the file**
2. **The order and data type of values stored**, so we can declare the appropriate variables (identifiers) correctly.
3. **How much data is in the file.** We usually know the structure of the data files which can be of THREE file types.

Knowing the structure helps us decide how we can set up our programs to identify and get all the data.

WE WILL COVER THE THREE TYPE BUT BE SURE TO STUDY THE PROGRAMS TO COME THAT DEMONSTRATE HOW TO GET THE DATA! USE THEM AS A MODEL FOR YOUR WORK

Data File Formats(The 3 types!)

- .Three common structures: There are three common
- structures used to do this. NAMELY
- A. **The first line** contains the number of lines (records) in the file. Similar to the counter control we saw before. First value becomes the final counter.
- `for(int i=1; i<=counter; ++i)`
- B. The file has a special value not normal for the file , like -99 or 0 **as the last value** to let us know we are finished reading the records. Known as a trailer signal or sentinel signal. Similar to the while loop technique of the last chapter.
- `while(exam_score>=0)`
- C. End-of-file loop. File stream lets us know when we reached the end with the eof() function. Like the end of data loop from keypad while(! cin.eof()) we use value of object returned
- E.g. while(!sensor.eof()) would loop to the end of file!

Type A: Data file with Specified Number of Lines: First entry has that number and thereafter data is in pairs, containing time a motion sensor reading.

STUDY PROGRAM THAT FOLLOWS!

```
sensor1.txt
10
0.0 132.5
0.1 147.2
0.2 148.3
0.3 157.3
0.4 163.2
0.5 158.2
0.6 169.3
0.7 148.2
0.8 137.6
```

```
// SOME CODE TO open input file
ifstream sensor1("sensor1.txt");
//read the number of data entries
int numEntries;
sensor1 >> numEntries;
//read every row
double t, y;
for (int i = 0; I < numEntries; i++) {
      sensor1 >> t >> y;
      //do something with the data
}
```

```
/* Program chapter5_2                                  */
/*program generates a summary report from a data file that has the number of data points in the first record.*/
#include <iostream> //Required for cerr, cin, cout.
#include <fstream> //Required for ifstream, ofstream.
#include <string> //Required for string.
using namespace std;
int main()
{int num_data_pts, k; // Declare and initialize objects.
double time, motion, sum = 0, max, min;
string filename;
ifstream sensor1;
ofstream report;
cout << "Enter the name of the input file"; // Prompt user for name of input file. Here sensor1.txt
cin >> filename;
sensor1.open(filename.c_str()); // Open input file.
if (sensor1.fail()) // check if file exists
{ cerr << "Error opening input file" << filename << endl;
exit(1);
}
report.open("sensor1Report.txt"); // Open output report file.
sensor1 >> num_data_pts; // Read first value.
sensor1 >> time >> motion;  // Read first data pair from input file
cout << time <<motion;   // echo file data to console in case of problems
max = min = motion;  //and set motion value to initial max and min
sum += motion; // build a sum of the motion value-> sum=sum+motion. (sum was 0)
// Read remaining data and  find max and min values of the motion and compute summary information.
for (k = 1; k<num_data_pts; k++)
{sensor1 >> time >> motion;
cout << time <<motion;   // echo file data to console in case of problems
sum += motion;
if (motion > max)
{max = motion;
}
if (motion < min)
{min = motion;
}
} // end of the for loop which has found the maz and min and sum of the motion data
// Set format flags.
report.setf(ios::fixed); // start to prepare a report file- note use of format flags see chap 2
report.setf(ios::showpoint);
report.precision(2);  //  manipulator chap 2
report << "Number of sensor readings:" << num_data_pts << endl;//build report file and  summary information.
report << "Average reading: " << sum / num_data_pts << endl;
report << "Maximum reading: " << max << endl;
report << "Minimum reading: " << min << endl;
sensor1.close(); // Close files and exit program.
report.close();
return 0;
} //end main
```

```
/* Program chapter5_2                                              */
/*program generates a summary report from a data file that has
the number of data points in the first record.*/
#include <iostream> //Required for cerr, cin, cout.
#include <fstream> //Required for ifstream, ofstream.
#include <string> //Required for string.
using namespace std;
int main()
{ int num_data_pts, k; // Declare and initialize objects.
  double time, motion, sum=0, max, min;
  string filename;
  ifstream sensor1;
  ofstream report;
  cout << "Enter the name of the input file"; // Prompt user for
name of input file.
  cin >> filename;   ?? Use extension .txt for data file here
  sensor1.open(filename.c_str()); // Open input file.
  if( sensor1.fail() )\\ check if file exists
  { cerr << "Error opening input file" << filename << endl;
    exit (1);
  }
  report.open ("sensor1Report.txt"); // Open output report file.
  sensor1 >> num_data_pts; // Read first value.
  sensor1 >> time >> motion;  // Read first data pair from input
file
```

```
max=min=motion;  //and set motion value to initial max and min
  sum += motion; // build a sum of the motion value-> sum=sum+motion.
(sum was 0)
  // Read remaining data and  find max and min values of the motion and
compute summary information.
  for (k=1; k<num_data_pts; k++)
   { sensor1 >> time >> motion;
     sum += motion;
     if (motion > max)
      { max = motion;
      }
     if (motion < min)
      { min = motion;
      }
   } // end of the for loop which has found the maz and min and sum of
the motion data
  // Set format flags.
  report.setf(ios::fixed); // start to prepare a report file- note use of
format flags see chap 2
  report.setf(ios::showpoint);
  report.precision(2);  //  manipulator chap 2
  report << "Number of sensor readings: << num_data_pts << endl; "//build
report file and  summary information.
  report << "Average reading: " << sum/num_data_pts << endl;
  report << "Maximum reading: " << max << endl;
  report << "Minimum reading: "  << min << endl;
  sensor1.close(); // Close files and exit program.
  report.close();  return 0;} //end main
```

Data generated in the report file looks like

- Number of sensor readings 10
- Average reading: 149.77
- Maximum reading: 169.30
- Minimum reading: 132.20

Type B:Trailer/Sentinel Signal: similar data as Type A but last values are unique

```
sensor2.txt
0.0 132.5
0.1 147.2
0.2 148.3
0.3 157.3
0.4 163.2
0.5 158.2
0.6 169.3
0.7 148.2
0.8 137.6
-99 -99
```

```
//open input file
ifstream sensor2("sensor2.txt");
double time, motion;
//read first values
sensor1 >> time >> motion;
//do something with the data
//enter loop
do { //do something with the data
//  get next set of data
sensor1 >> time >> motion
} while (time>=0 );//last value?
```

```
/* Program chapter5_3   This program generates a summary report from a data file that has a trailer record
 with  negative values.*/                                    */
#include <iostream> //Required for cin, cout, cerr
#include <fstream> //Required for ifstream, ofstream
#include <string> //Required for string.
using namespace std;
int main()
{ int num_data_pts(0), k; // Declare and initialize objects.
  double time, motion, sum(0), max, min;
  string filename;
  ifstream sensor2;   //define input and output objects
  ofstream report;
  cout << "Enter the name of the input file"; // Prompt user for name of input file.
  cin >> filename;
  sensor2.open(filename.c_str()); // Open input file
  if(sensor2.fail())   // bad name we are out of here!
  {    cerr << "Error opening input file\n";
       exit(1);
  }
  report.open("sensor2Report.txt"); // Open output report file.
  sensor2 >> time >> motion; // get first data set
  max = min = motion; // Initialize min and mzx using first motion data point.
  do                 // Update summary data until trailer record read.
    { sum += motion;
      if (motion > max)
       {  max = motion;
       }
      if (motion < min)
       {  min = motion;
       }
     num_data_pts++; // counting data points -> num_data_pts=num_data_pts+1;
     sensor2 >> time >> motion;
    } while (time >= 0);  //is it the end of the set yer?
. report.setf(ios::fixed); // Set format flags for output report
  report.setf(ios::showpoint);
  report.precision(2);
  report << "Number of sensor readings: " << num_data_pts << endl // Print summary information.
  report << "Average reading: " << sum/num_data_pts << endl
  report << "Maximum reading: " << max << endl
  report << "Minimum reading: " << min << endl;  // see text example  will that work as shown for this part????

 // Close files and exit program.
  sensor2.close();
  report.close();
  return 0;
  } //end main
```

NOTE: we get the about the same report as the last example

```
/* Program chapter5_3   This program generates a summary report
from a data file that has a trailer record
 with  negative values.*/                                    */
#include <iostream> //Required for cin, cout, cerr
#include <fstream> //Required for ifstream, ofstream
#include <string> //Required for string.
using namespace std;
int main()
{ int num_data_pts(0), k; // Declare and initialize objects.
  double time, motion, sum(0), max, min;
  string filename;
  ifstream sensor2;   //define input and output objects
  ofstream report;
  cout << "Enter the name of the input file"; // Prompt user for
name of input file.
  cin >> filename;  // use extension “txt”
  sensor2.open(filename.c_str()); // Open input file
  if(sensor2.fail())   // bad name we are out of here!
  {    cerr << "Error opening input file\n";
       exit(1);
  }
  report.open("sensor2Report.txt"); // Open output report file.
  sensor2 >> time >> motion; // get first data set
```

```
 max = min = motion; // Initialize min and mzx using first motion
data point.
  do     // Update summary data until trailer record read.
    { sum += motion;
      if (motion > max)
       {  max = motion;
       }
      if (motion < min)
       {  min = motion;
       }
      num_data_pts++; // counting data points ->
       num_data_pts=num_data_pts+1;
      sensor2 >> time >> motion;
    } while (time >= 0);  //is it the end of the set yer?
. report.setf(ios::fixed); // Set format flags for output report.
  report.setf(ios::showpoint);
  report.precision(2);
  report <<"Number of sensor readings: " << num_data_pts << endl.
  report << "Average reading: " << sum/num_data_pts << endl
  report << "Maximum reading: " << max << endl
  report << "Minimum reading: " << min << endl;
  sensor2.close(); // Close files and exit program.
  report.close();
  return 0;
  } //end main NOTe we get the about the same report as last cpp
```

EOF-based File: no initial value and no end value we depend on the system knowing the file has while we read it.

```
sensor3.txt
0.0 132.5
0.1 147.2
0.2 148.3
0.3 157.3
0.4 163.2
0.5 158.2
0.6 169.3
0.7 148.2
0.8 137.6
```

```
sum=count=0; // consider this example
data1 >>x // input object gets x
while (!data.eof())
       { ++count;
        sum=+=x;
        data >> x;
       }
// another approach for next program
double t, y;
ifstream sensor3("sensor3.txt");//open input
//read first  row
sensor1 >> t >> y; // maybe do something
while (! sensor3.eof()) {
   //do something with the data
       sensor1 >> t >> y;?? try get next set
}      // if last attempt failed then while
       // loop will end
```

```
/* Program chapter5_4   This program generates a summary report from a data file that does not have a header
 record or a trailer record.*/
#include <iostream> //Required for cin, cout, cerr
#include <fstream> //Required for ifstream, ofstream.
#include <string> //Required for string.
using namespace std;
int main()
{  int num_data_pts(0), k; // Declare and initialize objects (variables and i0 objects pointing to files).
  double time, motion, sum(0), max, min;
  string filename;
  ifstream sensor3;
  ofstream report;
  cout << "Enter the name of the input file"; // Prompt user for name of input file.
  cin >> filename;
  sensor3.open(filename.c_str()); // Open file and check if it exists.
    if(sensor3.fail())
    { cerr << "Error opening input file\n";
      exit(1);
     }
  report.open("sensor3Report.txt"); // open report file.
  sensor3 >> time >> motion; // initial input  read the first data point.
  while ( !sensor3.eof() ) // While not at the end of the file, read and accumulate information
  { num_data_pts++;
    if (num_data_pts == 1)
    {  max = min = motion;
    }
    sum += motion;
    if (motion > max)
    {      max = motion;
    }
    if (motion < min)
    {      min = motion;
    }
    sensor3 >> time >> motion; // input next go back to while
  } // end while
  // Set format flags.
  report.setf(ios::fixed);
  report.setf(ios::showpoint);
  report.precision(2);
  // Print summary information.
  report << "Number of sensor readings: " << num_data_pts << endl   // will the use of report output work here?
        << "Average reading:  << sum/num_data_pts << endl
        << "Maximum reading: "  << max << endl
        << "Minimum reading: " << min << endl;
  // Close file and exit program.
  sensor3.close();
  report.close();
  return 0;
} //end main
/*-----------------------------------------------------*/
```

```
/* Program chapter5_4   This program generates a summary report
from a data file that does not have a header
 record or a trailer record.*/
#include <iostream> //Required for cin, cout, cerr
#include <fstream> //Required for ifstream, ofstream.
#include <string> //Required for string.
using namespace std;
int main()
{  int num_data_pts(0), k; // Declare and initialize objects
(variables and i0 objects pointing to files).
  double time, motion, sum(0), max, min;
  string filename;
  ifstream sensor3;
  ofstream report;
  cout << "Enter the name of the input file"; // Prompt user for
name of input file.
  cin >> filename;
  sensor3.open(filename.c_str()); // Open file and check if it
exists.
  if(sensor3.fail())
    { cerr << "Error opening input file\n";
      exit(1);
     }
  report.open("sensor3Report.txt"); // open report file.
  sensor3>>time>>motion;//initialinput  read the first data point.
```

```
 while ( !sensor3.eof() ) // While not at the end of the file,
read and accumulate information
  { num_data_pts++;
    if (num_data_pts == 1)
    {  max = min = motion;
    }
    sum += motion;
    if (motion > max)
    {       max = motion;
    }
    if (motion < min)
    {       min = motion;
    }
    sensor3 >> time >> motion; // input next go back to while
  } // end while
  // Set format flags.
  report.setf(ios::fixed);
  report.setf(ios::showpoint);
  report.precision(2);  // Print summary information.
  report << "Number of sensor readings: " << num_data_pts << endl
         << "Average reading:  << sum/num_data_pts << endl
        << "Maximum reading: "  << max << endl
        << "Minimum reading: " << min << endl;
sensor3.close(); // Close file and exit program.
  report.close();  return 0;} //end main
```

A note on Writing to Files

- After opening an output file, writing to a file is no different from writing to standard output.
- Must decide on a file format.
  - Sentinels can be hard to choose to avoid conflict with valid data.
  - Knowing in advance how many lines/records are in the file is sometimes difficult or impractical.
  - Usually best to use eof-style file format where file structure contains only valid information.

LAB #19 HAND IN LABORATORY TASK:

reading files with eof()

Consider the following data of **time, motion (sensor) and temperature**

0.0 132.5 24.6
0.1 147.2 19.3
0.2 148.3 88.3
0.3 157.3 23.7
0.4 163.2 08.5
0.5 158.2 54.7
0.6 169.3 39.9
0.7 148.2 43.6
0.8 137.6 76.9

Construct a **data file** with this data. Use notepad but not a word processor which would put control characters in the file. Then using the programing model of getting data from a file with **eof()** function read the three values in the sets Produce similar to previous examples a **report file** Nicely worded

1. The total number of records
2. The average motion sensor reading and temperature
3. The maximum motion sensor reading and temperature values
4. The minimum motion sensor and temperature values

Hand in your **program file**, the **data file** and the **report file (use notepad for the last two to print these files)**

Recall where you have to place the data file to make your program find it?
Name the file what you want.
Copy the contents of the report file and place it in the usual output place when Handing in your work.

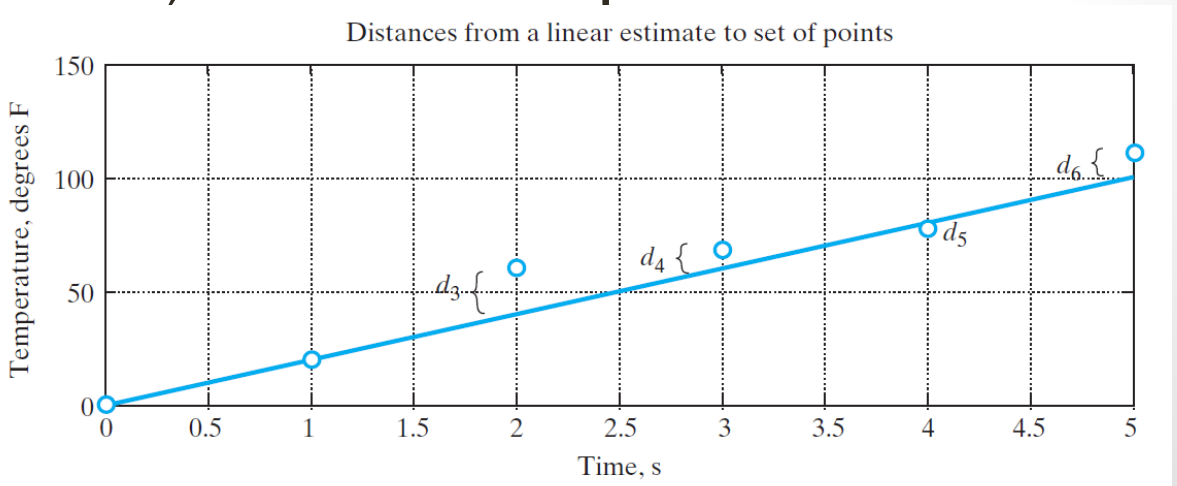
EXTRA Credit write a program that creates the data file! +1 on final average! Or add this Extension to the main program as the first part.

Numerical technique "Linear Modeling"

- Linear modeling also called Linear regression is the name given to the process that determines the linear equation (y = mx + b) that is the **best fit** to a set of data points. Assuming the data looks close to a straight line!
  - **Linear regression does this by minimizing the squared distance**
  - **between the line created and the original data points.**
  - Data table below is for points on graph (circles) whose connection looks like a straight line so we use this technique.

Time(s)	Temp (°F)
0	0
1	20
2	60
3	68
4	77
5	110

Vertical distance from data points to linear curve are each squared and these squared values are summed The created line has the smallest sum of these squared values, called the **"least-squares"** distance



Solving for Linear Model Parameters

Once we set up an expression representing values of x,y of the new Linear curve and there distance from each of the data points x_k y_k we use the curve y=mx+b along with the data points for our Expression and then take derivative of the sum expression to minimize. The results yield the equations below that define the line y=mx+b

If you want to learn more see on the web: https://en.wikipedia.org/wiki/Linear_regression

$$m = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} y_k - n \cdot \sum_{k=1}^{n} x_k y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2},$$

$$b = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} x_k y_k - \sum_{k=1}^{n} x_k^2 \cdot \sum_{k=1}^{n} y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2}.$$

The large Σ (SIGMA), you may recall Represents a summation

- **$\Sigma x_k = x_1 + x_2 \ldots x_n$**
- **Similarly, the sums Are constructed**
- **And the calculation as shown**
- **Yields the m and b of the**
- **y=mx+b curve**
- **Which is normally plotted along**
- **with The original data points as**
- **in the figure on the last slide.**
- **NOTE: Denominators are same for m**
- **and b! Evaluate all for this EG case !**

Given 3 points: x_1,y_1 ; x_2,y_2 ; x_3 y_3

WHAT DOES EACH m and b look like!

IN CLASS NOW!!!! $\Sigma x_k = x_1 + x_2 + x_3$ etc.

Problem Solving Applied: Ozone Measurements

be sure to read section 5.7

This illustration will show that we easily can construct summation in loops, as you have seen before when Taking averages for example.

Linear regression Applied: Ozone Measurements

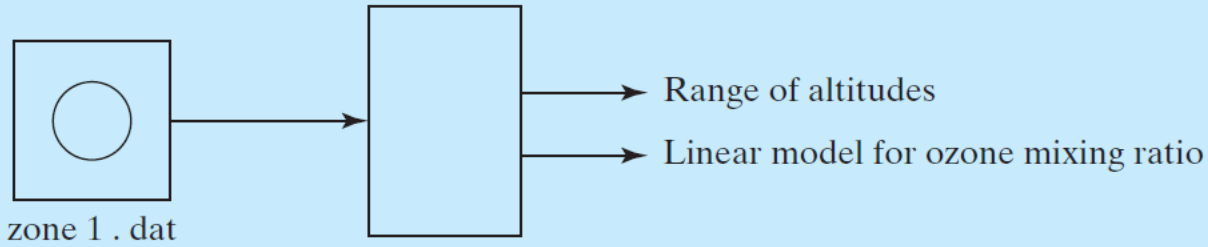
Data from spacecraft gave ozone data (Ozone Mixing Ratio in ppmv -parts per million volume) for different altitudes (km) and we want to get values for the ozone at other altitudes in between.

1. PROBLEM STATEMENT

Use the least-squares technique to determine a linear model for estimating the ozone mixing ratio at a specified altitude.

2. INPUT/OUTPUT DESCRIPTION

The following I/O diagram shows that the data file *zone1.dat* is the input and that the output is the range of altitudes and the linear model:



Problem Solving Applied: Ozone Measurements*

3. HAND EXAMPLE

Assume that the data consist of the following four data points:

Altitude (km)	Ozone Mixing Ratio (ppmv)
20	3
24	4
26	5
28	6

We now need to evaluate equations (5.1) and (5.2), which are repeated here for convenience:

$$m = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} y_k - n \cdot \sum_{k=1}^{n} x_k y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2}, \tag{5.1}$$

$$b = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} x_k y_k - \sum_{k=1}^{n} x_k^2 \cdot \sum_{k=1}^{n} y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2}. \tag{5.2}$$

Problem Solving Applied: Ozone Measurements*

To evaluate these equations using the hand-example data, we need to compute the following group of sums:

$$\sum_{k=1}^{4} x_k = 20 + 24 + 26 + 28 = 98,$$

$$\sum_{k=1}^{4} y_k = 3 + 4 + 5 + 6 = 18,$$

$$\sum_{k=1}^{4} x_k y_k = 20 \cdot 3 + 24 \cdot 4 + 26 \cdot 5 + 28 \cdot 6 = 454,$$

$$\sum_{k=1}^{4} x_k^2 = (20)^2 + (24)^2 + (26)^2 + (28)^2 = 2436.$$

Using these sums, we can now compute the values of m and b:

$$m = 0.37;$$

$$b = -4.6.$$

Problem Solving Applied: Ozone Measurements

4. ALGORITHM DEVELOPMENT

We first develop the decomposition outline because it divides the solution into a series of sequential steps.

Decomposition Outline

1. Read data file values and compute corresponding sums and ranges.
2. Compute slope and *y*-intercept.
3. Print range of altitudes and linear model.

Refining our solution to C++ we will need a counter=0
Then variables for the sums, sumx sumy sumxy sumx2 (squared)
Then a while loop to do get the data x,y and build the sums as we get them.
Also counting the points and then do the computation for the slope and
y intercept. Print all the goodies

More refinements on the code

Looking at the formulas below to solve for m and b which include various sums and I already have some "sum" Variables. THUS, I can put Together a little code by matching my variables to the Formulas as follows

```
denominator = sumx*sumx - count*sumx2;
m = (sumx*sumy - count*sumxy)/denominator;
b = (sumx*sumxy - sumx2*sumy)/denominator;
// used in program to follow.
```

$$m = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} y_k - n \cdot \sum_{k=1}^{n} x_k y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2},$$

$$b = \frac{\sum_{k=1}^{n} x_k \cdot \sum_{k=1}^{n} x_k y_k - \sum_{k=1}^{n} x_k^2 \cdot \sum_{k=1}^{n} y_k}{\left(\sum_{k=1}^{n} x_k\right)^2 - n \cdot \sum_{k=1}^{n} x_k^2}.$$

Other aspects of Linear Regression

Once we have our line we are concerned on how well does the line
Fit the original data. Various statistical calculations are used to
Evaluate our line for its ability to be the best fit.
When you take your physics laboratory course you will be using
Software that will do the linear regression calculation for you from data
You collect in some experiments. The software will plot the line and then
Give you a number known as the **Correlation** which if it has the value
1 is a perfect fit but mostly is some fraction. The closer the fraction to 1
the better the fit.
Other statistical calculations related to the **Correlation** are
The **mean**
The **variance**
The **deviation**
The **standard deviation**

You will return to these in later courses.
The web has many references to these concepts if you want to Explore them.
THE LEAST SQUARE LINEAR REGRESSION PROGRAM FOLLOWS.

```
/* Program chapter5_8 */
/* This program computes a linear model for a set of altitude and ozone mixing ratio values. */
#include <iostream> //Required for cin, cout
#include <fstream> //Required for ifstream
#include <string> //Required for string
using namespace std;
int main()
{// Declare and initialize objects.
  int count(0);
  double x, y, first, last, sumx(0), sumy(0), sumx2(0),
  sumxy(0), denominator, m, b;
  string filename;
  ifstream zone1;
  cout << "Enter name of input file:";
  cin >> filename;
  // Open input file.
  zone1.open(filename.c_str());
  if(zone1.fail())
  {    cerr << "Error opening input file\n";
    exit(1);
  }
  zone1 >> x >> y; // While not at the end of the file, read and accumulate information
  while ( !zone1.eof() )
  { ++count;
    if (count == 1)
      first = x;
    sumx += x;
    sumy += y;
    sumx2 += x*x;
    sumxy += x*y;
    zone1 >> x >> y;
  }
  last = x;
  denominator = sumx*sumx - count*sumx2; // Compute slope and y-intercept.
  m = (sumx*sumy - count*sumxy)/denominator;
  b = (sumx*sumxy - sumx2*sumy)/denominator;
  // Set format flags
  cout.setf(ios::fixed);
  cout.precision(2);
  // Print summary information.
  cout << "Range of altitudes in km: \n";
  cout << first << " to " << last << endl << endl;
  cout << "Linear model: \n";
  cout << "ozone-mix-ratio = " << m << " altitude + "
  << b << endl;
  // Close file and exit program.
  zone1.close();
  return 0;
} // end of main
```

```
/* Program chapter5_8 */
/* This program computes a linear model for a set of altitude and
ozone mixing ratio values. */
#include <iostream> //Required for cin, cout
#include <fstream> //Required for ifstream
#include <string> //Required for string
using namespace std;
int main()
{// Declare and initialize objects.
  int count(0);
  double x, y, first, last, sumx(0), sumy(0), sumx2(0),
  sumxy(0), denominator, m, b;
  string filename;
  ifstream zone1;
  cout << "Enter name of input file:";
  cin >> filename;
  // Open input file.
  zone1.open(filename.c_str());
  if(zone1.fail())
  {    cerr << "Error opening input file\n";
    exit(1);
  }
  zone1 >> x >> y; Get first data set
 // While not at the end of the file, read and accumulate
information follows
```

```
while ( !zone1.eof() )
  { ++count;
    if (count == 1)
      first = x;
    sumx += x;
    sumy += y;
    sumx2 += x*x;
    sumxy += x*y;
    zone1 >> x >> y;
  }
  last = x;
  denominator = sumx*sumx - count*sumx2; // Compute slope and y-
intercept.
  m = (sumx*sumy - count*sumxy)/denominator;
  b = (sumx*sumxy - sumx2*sumy)/denominator;
  cout.setf(ios::fixed); // Set format flags
  cout.precision(2);
  cout << "Range of altitudes in km: \n"; // Print summary
  cout << first << " to " << last << endl << endl;
  cout << "Linear model: \n";
  cout << "ozone-mix-ratio = " << m << " altitude + "
  << b << endl;
  // Close file and exit program.
  zone1.close();     return 0;  } // end of main
```

HAND IN HW #12 (4 PTS EACH 44 TOTAL PTS

Do chap 5 exam practice 1-5, 8-11 Memory snapshot only for exam practice 8 to 11 ANSWER CAREFULLY 12-13

12. Which of the following repetition constructs will properly repeat the loop body while not at the end-of-file for input file object DataFile

A. while(! DataFile B. while (! eof()) C, while (! DataFile**.**eof())

D. while (! eof**.**DataFile())

13. Given the name of the file you want to open has been stored in a string object filename, which of the following will statements will correctly open the file.

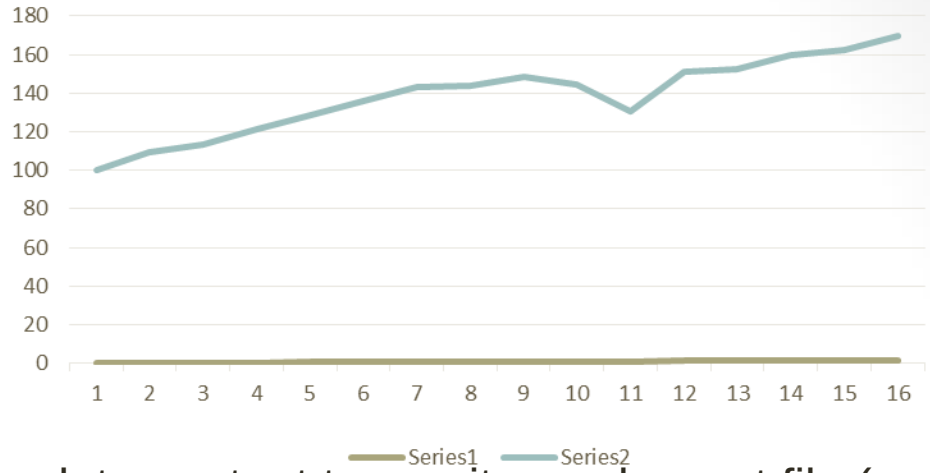
A. inFile.open(filename); B.inFile.open(c_str.filename());

C. inFile.open(filename.c_str()); D. inFile.open("filename");

HAND IN LABORATORY TASK: LAB #20 each step to help you organize is numbered!

Cruise missile
fired from AGM-84
Time Velocity

0	100.0
0.1	109.4
0.2	113.1
0.3	121.4
0.4	128.8
0.5	135.7
0.6	143.3
0.7	144.1
0.8	148.6
0.9	144.7
1	130.7
1.1	151.0
1.2	152.2
1.3	159.9
1.4	162.6
1.5	169.7



Obtain a linear fit to the data. output to monitor and report file (part 6)
1. create a data file read data into your program.
2. Calculate Slope and
3. initial velocity (y intercept) of best fit
4. Obtain the velocity at 0.85 seconds from the linear fit.
5. In the output to the screen show EACH OF THE FOLLOWING
Slope =m=?
Initial velocity (intercept, V_0 =?)
The straight line formula of Velocity vs time. Velocity =m* time + V_0 .
Value of the Velocity at 0.85 seconds =?
6. Also send all info derived above
To a recording data file.
Attach printout of the file and
screen to Your program to hand in