Laboratory 6C
Summary of Report Formatting

CS-102

Storing Records in Sequential Files

- Here's an example of a file that we have recently used where our records were made up of two types of fields: a text field & a floating point field.

violet 713.79
blue 637.86
cyan 599.58
green 565.65
yellow 516.88
orange 483.54
red 428.27

- We could read both those fields using **cin >>** because there was no space in the names of any of the colors.
- Another way of saying this is that the presence of the space became a good delimiter for our use.

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

int main()
{
   ifstream infile;
   double freq;
   string color;
   infile.open("colors.txt");

   infile >> color;
   while (!infile.eof())
   {
      infile >> freq;
      cout <<"The frequency of "<< color <<" is "<< freq <<" THz"<< endl;
      infile >> color;
   }
   infile.close();
   return 0;
}
```

All we needed to read this file was the following.

Storing Records in Sequential Files

- Suppose now we have a situation where there may be a space in the names of the text field:

baby blue 637.86

sea green 565.65

ruby red 428.27

- Now we have a problem as cin will no longer work for us.

Storing Records in Sequential Files

- Although cin works well with numeric data, it has problems with reading strings.
  - If the strings have spaces in them, cin will read the text up to the first space, and then stop.
  - This problem can be overcome using the **getline** function.
- **getline(datafile, str, delimiter)** takes three arguments
  - **datafile** is the name of the source of the data, which up till now, most commonly is **cin**.
  - **str** is the name of the string object.
  - **delimiter** is the name of the character that is used to denote the end of a line, which is most commonly **'\n'** .

Storing Records in Sequential Files

- Now we can re-write our datafile as follows:

baby blue#637.86

sea green#565.65

ruby red#428.27

- In other words, instead of using the delimiter, space, for the text, we use the delimiter, **#** ?
- We could then use the following code to read in file of data records.

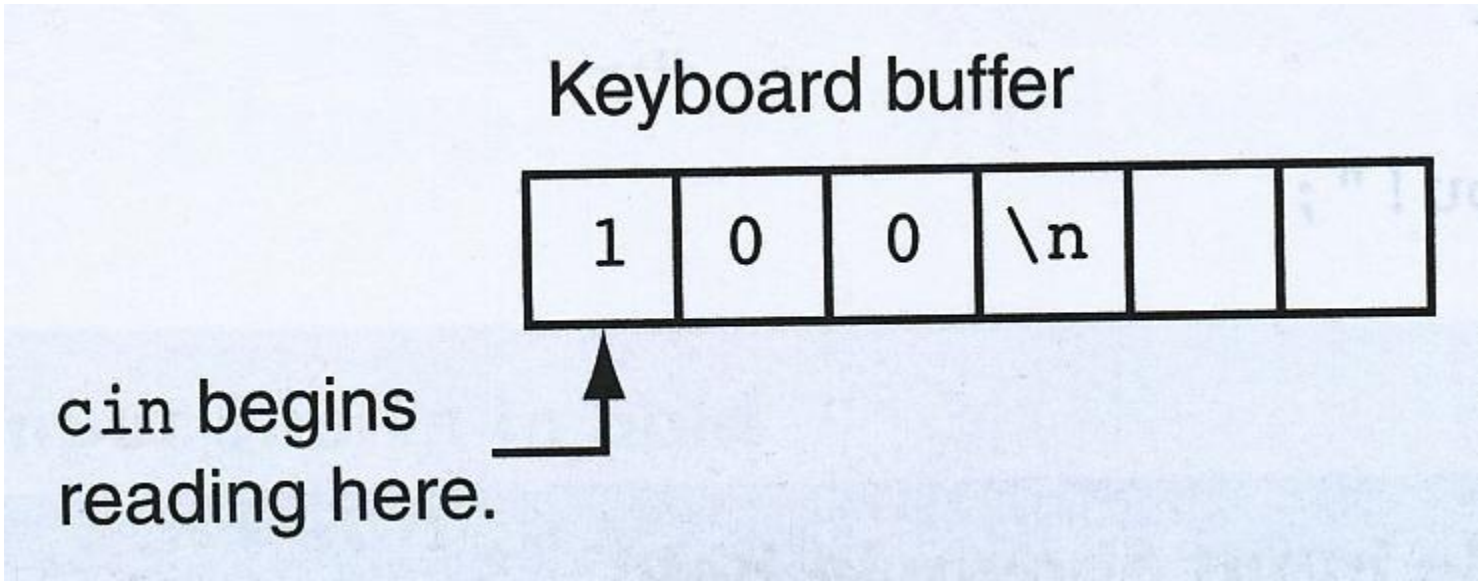
```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

int main()
{
   fstream infile;
   double freq;
   string color;
   infile.open("colors.txt");

   getline(infile, color, '#');
   while (!infile.eof())
   {
      infile >> freq;
      cout <<"The frequency of " << color << " is "<< freq <<" THz"<< endl;
      infile.ignore();
      getline(infile, color, '#');
   }
   infile.close();
   return 0;
}
```

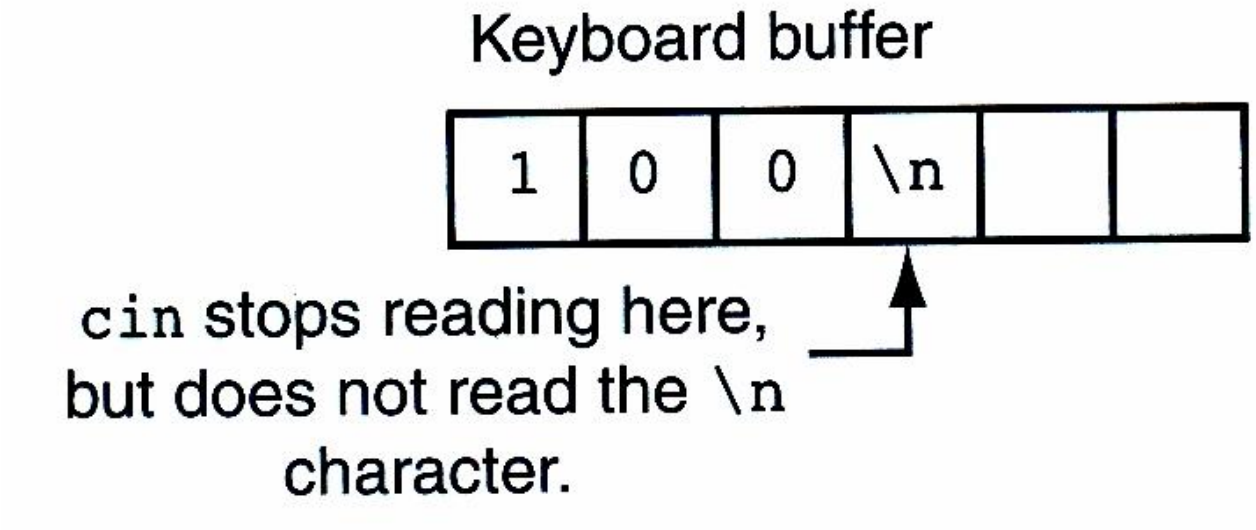
All we needed to read this file was the following.

Storing Records in Sequential Files

- Note: A brief word about the **infile.ignore();** command.
  - As you may recall, the problem with the >> operator is that it doesn't read the **'\0'** which is saved at the end of a string.
  - So, in order to remove that character from the input buffer after a number has been read, but before the next string is read, we need to use the **ignore()** method to rid the buffer of this character.
  - You would use the name of the datafile (in this case it's **infile**) followed by **.ignore()** in order to accomplish this.
- Now we have all we need to read records stored in a single line of a text file which combine both strings with possible spaces in them and numeric datatypes.

What happens with Keyboard Buffer





Laboratory 6C – Part 1

- Create a menu driven application that uses information in files to keep track of your stock portfolio. The data is stored in a sequential file. An example file (“stocks.txt”) is as follows:

```
Amgen#200 50.75 75.625
Delta#100 111.75 97.25
Novell#500 10.375 12.25
PPG Inc#100 56.75 54.5
Timken#300 34.625 15.125
HP#100 100.50 115.25
```

- Each line in the file must have the format :

  Company name#number_of_shares purchase_price current_price

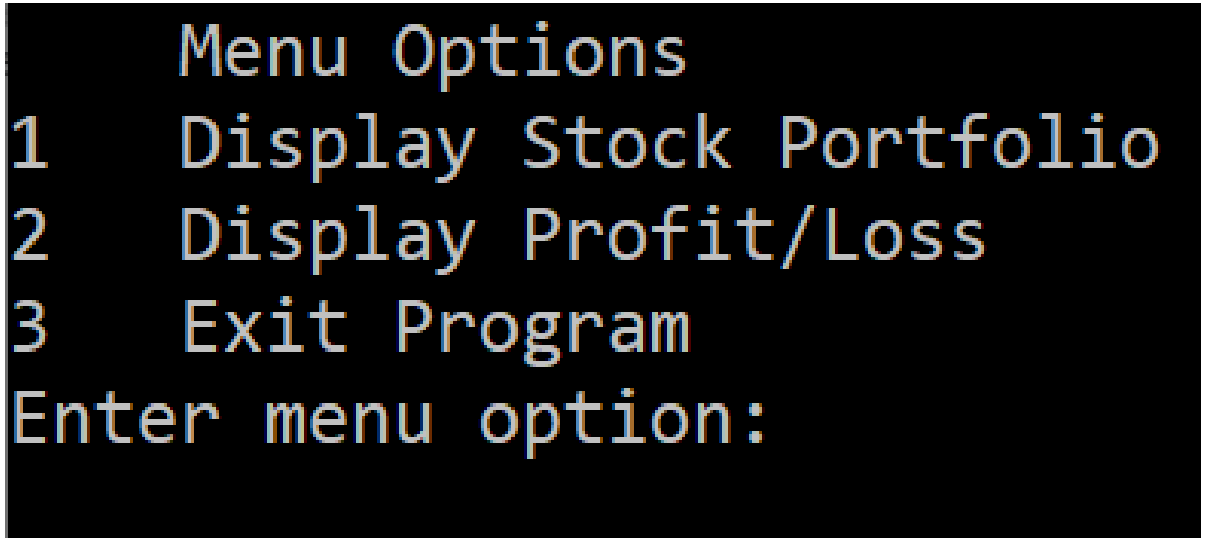
- Note that there must be a delimiter after the company name. In this case we are using the ‘#’ character.
  - Also notice that there are spaces between the numbers.
  - These spaces are the delimiter when using the stream extraction operator **>>**

Laboratory 6C

- Part 1: You will start by creating a function called displayMenu() which will display the menu.
- Part 2: You will create a function called displayStocks() which receives the name of the file to open.
  - It will create an input file object,
  - open the file for input,
  - read the file,
  - Display the stock name, the number of shares, the purchase price and the current value
- Part 3: You will create a function called totalValue() which will:
  - calculate and display the profit or loss on each stock,
  - and display the total value.
- **Note the formatting! Don’t forget to close the file!**

Laboratory 6C – Part 1

- Main will prompt for the filename (“stocks.txt”) and have the loop to process the options and call the function.
  - You do not need any file objects in main.
  - Handle the files in the functions (parts 2 and 3)
  - You may use stubs for now in those functions until you write them in parts 2 & 3
- **Testing – show the following output for part 1:**
  - Using the file “stocks.txt” shown above, here is the main menu.



Laboratory 6C-1

- Save your program as: *YourName*-Lab06C-1.cpp
- If you are doing Lab06C synchronously, show your work for parts 1, 2 and 3, to the Instructor for proper credit.
- If you are doing Lab06C asynchronously, submit your program to Canvas.

Laboratory 6C – Part 2

- When you run Option 1, you should see the following:

Stock Name	# Shares	Purchase Price	Current Price
Amgen	200	50.75	75.62
Delta	100	111.75	97.25
Novell	500	10.38	12.25
PPG Inc	100	56.75	54.50
Timken	300	34.62	15.12
HP	100	100.50	115.25

Laboratory 6C-2

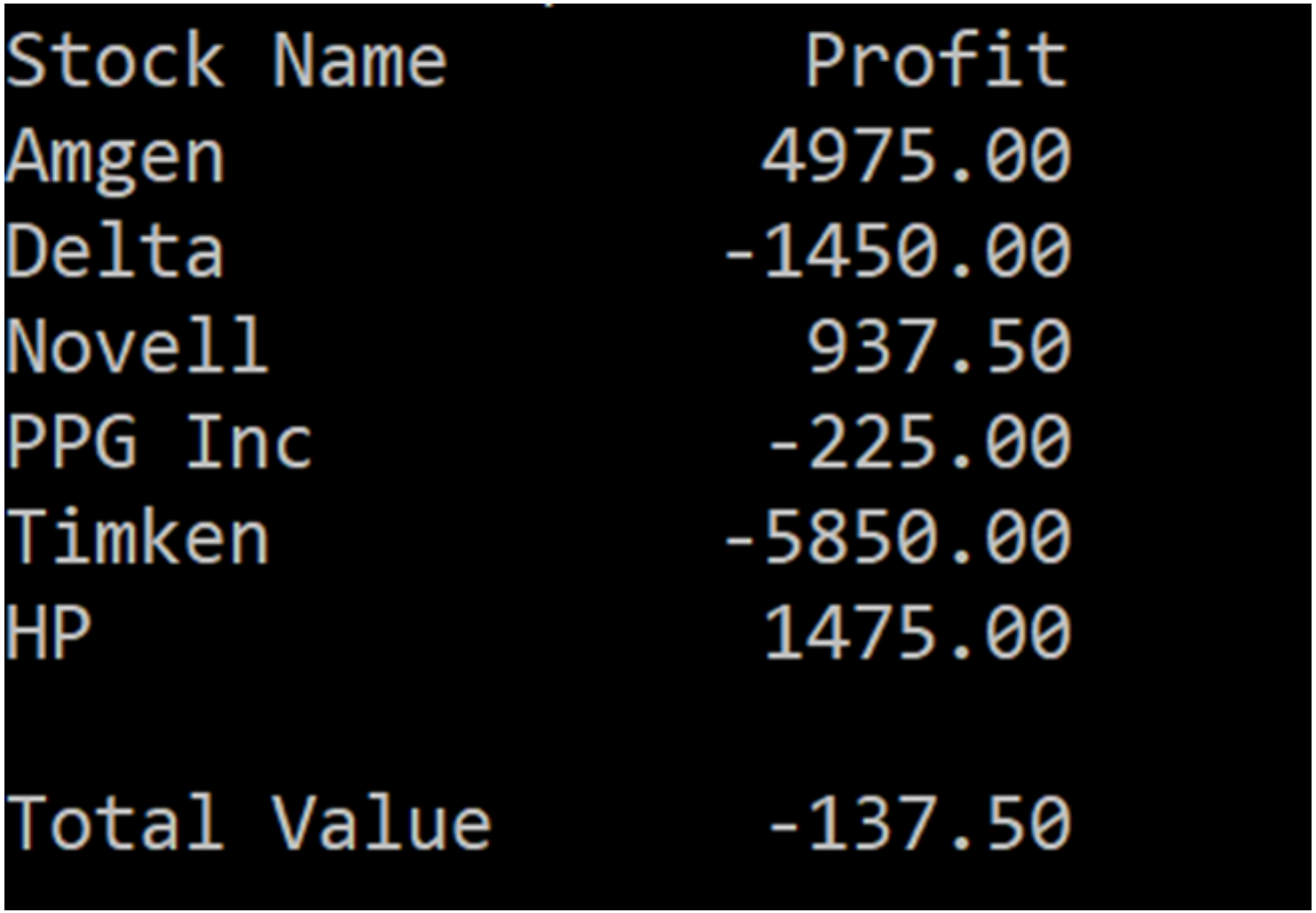
- Save your program as: *YourName*-Lab06C-2.cpp
- If you are doing Lab06C synchronously, show your work for parts 1, 2 and 3, to the Instructor for proper credit.
- If you are doing Lab06C asynchronously, submit your program to Canvas.

Laboratory 6C-3

- Save your program as: *YourName*-Lab06C-3.cpp
- If you are doing Lab06C synchronously, show your work for parts 1, 2 and 3, to the Instructor for proper credit.
- If you are doing Lab06C asynchronously, submit your program to Canvas.

Laboratory 6C – Part 3

- When you run Option 2, you should see the following:



Laboratory 6C – Part 4

- Write the program that will create the file that contains the data found in: stocks.txt, to begin with.

Amgen#200 50.75 75.625
Delta#100 111.75 97.25
Novell#500 10.375 12.25
PPG Inc#100 56.75 54.5
Timken#300 34.625 15.125
HP#100 100.50 115.25

- Call your new file: stocks2.txt.
  - By giving it a different name than stocks.txt, you won't accidently destroy the data you already have in stocks.txt.
- Again, pay particularly close attention to the space in your resulting file.

Laboratory 6C – Part 4

- When you have finished *YourName*-Lab06C-4.cpp and generated stocks2.txt:
- If you are doing Lab06C synchronously, show your work for part 4, to the Instructor for proper credit.
- If you are doing Lab06C asynchronously, submit your program and text file to Canvas.

Laboratory 6C – Part 5

- Now add a $ in front of your prices in the output of Part 2.
  - There should be no space between the $ and the amount that follows.
- This will require the use of StringStream.
- **Remember that when you are done using a StringStream variable, it must be cleared before using it again.**
  - **If strs is a stringstream object then you would need to do the following in your display output loop before you re-used the variable strs again.**
    - **strs.str(string());**
    - **strs.clear();**
- Call your result: *YourName*-Lab06C-5.cpp .
- Shown on the next page is what your output would look like using StringStream.

Laboratory 6C – Part 5

Stock Name	# Shares	Purchase Price	Current Price
Amgen	200	$50.75	$75.62
Delta	100	$111.75	$97.25
Novell	500	$10.38	$12.25
PPG Inc	100	$56.75	$54.50
Timken	300	$34.62	$15.12
HP	100	$100.50	$115.25

Laboratory 6C – Part 5

- When you have finished *YourName*-Lab06C-5.cpp:
  - If you are doing Lab06C synchronously, show your work for part 5, to the Instructor for proper credit.
  - If you are doing Lab06C asynchronously, submit your program to Canvas.